z/OS

C/C++ User's Guide

SC09-4767-00

z/OS

C/C++ User's Guide

SC09-4767-00

Note!

Before using this information and the product it supports, be sure to read the information in “Notices” on page 723.

First Edition (March 2001)

This edition applies to Version 1 Release 1 Modification 0 of z/OS C/C++ (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition replaces SC09-2361-06. 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.

Technical changes in the text since the last release of this book are indicated by a vertical line (|) to the left of the change.

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 in any one of the following methods:

- Electronically to the network ID listed below. Be sure to include your entire network address if you want a reply.

  Internet: torrcf@ca.ibm.com
  IBMLink: toribm(torrcf)

- By FAX, use the following number:

  United States and Canada: (416) 448-6161
  Other countries: (+1) 416-448-6161

- By mail, to the following address:

  IBM Canada Ltd. Laboratory
  Information Development
  2G/KB7/1150/TOR
  1150 Eglinton Avenue East
  Toronto, Ontario, Canada M3C 1H7

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

Part 1. Introduction

This part discusses the z/OS C/C++ information library, and presents introductory concepts on the z/OS C/C++ product. Specifically, it discusses the following:

Chapter 1. About This Book

This edition of *z/OS C/C++ User's Guide* is intended for users of the IBM z/OS C/C++ compiler with the z/OS Language Environment product. It provides you with information about implementing (compiling, linking, and running) programs that are written in C and C++. It contains guidelines for preparing C and C++ programs to run on the z/OS operating system.

To use this, or any other z/OS C/C++ book, you must have a working knowledge of the C and C++ programming languages. You should also know the operating system, and the related products as appropriate. This includes the z/OS Language Environment product and z/OS UNIX System Services (z/OS UNIX).

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 runtime 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 runtime options • Compiling, IPA Linking, binding, and running z/OS C/C++ programs • Using precompiled headers • 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 • Error messages and return codes

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

Book Title and Number	Key Sections/Chapters in the Book
z/OS C/C++ Language Reference, SC09-4764	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++ Run-Time Library Reference, SA22-7821	Reference information for: • C header files • C 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, SC09-4763	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
z/OS C/C++ Reference Summary, SX09-1319	Summary tables for: • Character set, trigraphs, digraphs, and keywords • Escape sequences, storage classes • Predefined and derived types, type qualifiers • Operator precedence, redirection symbols • `fprintf()` format, type characters, and flag characters • `fscanf()` format and type characters • __amrc structure • Hardware exceptions and signals • Compiler return codes • Compiler options • #pragma directives • Library functions • Utilities

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

Book Title and Number	Key Sections/Chapters in the Book
OS/390 C/C++ IBM Open Class Library User's Guide, SC09-2363	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, tutorials, 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
OS/390 C/C++ IBM Open Class Library Reference, SC09-2364	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
APAR and BOOKS files (Shipped with Program materials)	Partitioned data set CBC.SCBCDOC 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 runtime 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*, SC09-4763 • *z/OS Language Environment Customization*, SA22-7564 • *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 • *z/OS C/C++ Language Reference*, SC09-4764 • *z/OS C/C++ Reference Summary*, SX09-1319 • *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 • *OS/390 C/C++ IBM Open Class Library User's Guide*, SC09-2363 • *OS/390 C/C++ IBM Open Class Library Reference*, SC09-2364
Coding and binding programs with interlanguage calls	• *z/OS C/C++ Programming Guide*, SC09-4765 • *z/OS C/C++ Language Reference*, SC09-4764 • *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 • 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`)
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++ Programming Guide*, SC09-4765 • *z/OS Language Environment Programming Guide*, SA22-7561 • *z/OS Language Environment Debugging Guide*, GA22-7560 • *z/OS UNIX System Services Messages and Codes*, SA22-7806 • *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
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*, SA22-7806
Using sockets library functions in the z/OS UNIX environment	• *z/OS C/C++ Run-Time Library Reference*, SA22-7821
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.
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.SCBCDOC(APAR) on z/OS C/C++ product tape

Table 2. Publications by Task (continued)

Tasks	Books
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
Quick reference	• *z/OS C/C++ Reference Summary*, SX09-1319

Note: For information on using the prelinker, see “Appendix A. Prelinking and Linking z/OS C/C++ Programs” on page 461. As of 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++ Programming Guide*, SC09-4765
- *z/OS C/C++ Reference Summary*, SX09-1319
- *OS/390 C/C++ IBM Open Class Library User’s Guide*, SC09-2363
- *z/OS C Curses*, SA22-7820
- *z/OS C/C++ Compiler and Run-Time Migration Guide*, SC09-4763
- *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*, SC09-4763 at no charge. Feature code 8009 includes the remaining books.

Softcopy Books

The z/OS C/C++ publications are supplied in PDF and BookMaster format on the following CD: *IBM Online Library Omnibus Edition z/OS Collection*, SK2T-6700. 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:
- *z/OS C/C++ Language Reference*, SC09-4764
- *z/OS C/C++ User's Guide*, SC09-4767
- *z/OS C/C++ Programming Guide*, SC09-4765
- *OS/390 C/C++ IBM Open Class Library User's Guide*, SC09-2363
- *OS/390 C/C++ IBM Open Class Library Reference*, SC09-2364

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.SCBCSAM` or `CBC.SCLBSAM`. The labels have the form CBC*xyyy* or CLB*xyyy*, where *x* refers to a publication:
- R and X refer to *z/OS C/C++ Language Reference*, SC09-4764
- G refers to *z/OS C/C++ Programming Guide*, SC09-4765
- U refers to *z/OS C/C++ User's Guide*, SC09-4767
- A refers to *OS/390 C/C++ IBM Open Class Library User's Guide*, SC09-2363

Examples labelled as CBC*xyyy* appear in *z/OS 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 *OS/390 C/C++ IBM Open Class Library User's Guide*.

An exception applies to the example names for the Collection Class Library which do not follow a naming convention. These examples are in *OS/390 C/C++ IBM Open Class Library Reference*, SC09-2364.

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.

How to Read the Syntax Diagrams

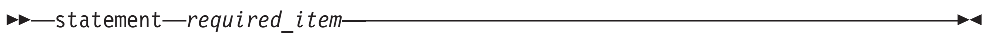
This book describes the syntax for commands, directives, and statements, using the following structure:

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

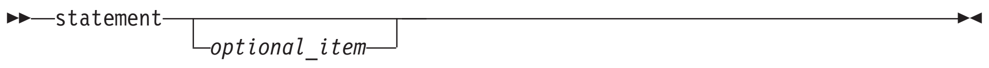
  A double right arrowhead indicates the beginning of a command, directive, or statement. A single right arrowhead indicates that it is continued on the next line. In the following diagrams, "statement" represents a command, directive, or statement.

```
►►──statement──────────────────────────────────────────────────────►◄
```

- Required items are on the horizontal line (the main path).

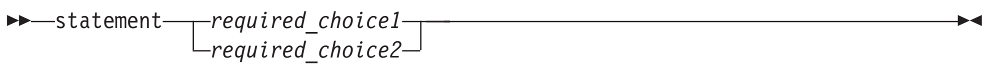


- Optional items are below the main path.

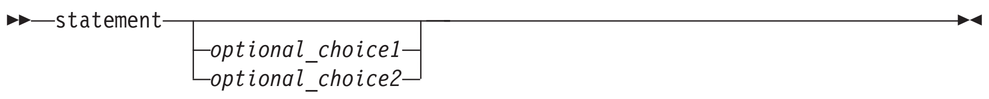


- If you can choose from two or more items, they are vertical in a stack.

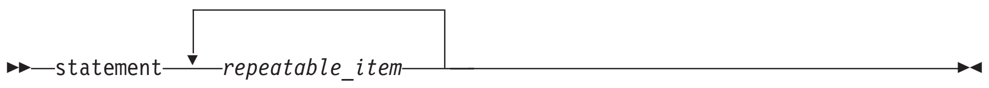
  If you *must* choose one of the items, one item of the stack is on the main path.



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



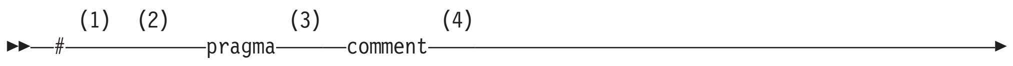
- An arrow that returns to the left above the main line indicates an item that you can repeat.



  A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.
- Keywords are not italicized, and should be entered exactly as shown (for example, `pragma`). You must spell keywords exactly as shown in the syntax diagram. Variables are in lowercase italics (in hardcopy), for example, *identifier*. They represent user-supplied names or values.
- If the syntax diagram shows punctuation marks, parentheses, arithmetic operators, or other nonalphanumeric characters, you must enter them as part of the syntax.

Note: You do not always require the white space between tokens. You should, however, include at least one blank space between tokens unless otherwise specified.

The following syntax diagram example shows the syntax for the `#pragma comment` directive.



```
    (5)             (6)                                                      (9)  (10)
►─(──────────┬─compiler──────────────────────────────────────────────────┬─)─────────►◄
             ├─date──────────────────────────────────────────────────────┤
             ├─timestamp─────────────────────────────────────────────────┤
             └──┬─copyright─┬─┬────────────────────────────────────────┬─┘
                └─user──────┘ │     (7)                        (8)    │
                              └─,─────────"──token_sequence──"────────┘
```

Notes:

1 This is the start of the syntax diagram.

2 The symbol `#` must appear first.

3 The keyword `pragma` must follow the `#` symbol.

4 The keyword `comment` must follow the keyword `pragma`.

5 An opening parenthesis must follow the keyword `comment`.

6 The comment type must be entered only as one of the following: `compiler`, `date`, `timestamp`, `copyright`, or `user`.

7 If the comment type is `copyright` or `user`, and an optional character string is following, a comma must be present after the comment type.

8 A character string must follow the comma. The character string must be enclosed in double quotation marks.

9 A closing parenthesis is required.

10 This is the end of the syntax diagram.

The following examples of the `#pragma comment` directive are syntactically correct according to the diagram above:

```
#pragma comment(date)
#pragma comment(user)
#pragma comment(copyright,"This text will appear in the module")
```

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 C/C++ Productivity Tools for z/OS product
- A set of utilities for C/C++ application development

IBM offers the C language on other platforms, such as the AIX®, OS/2, OS/400®, VM/ESA®, VSE/ESA, and Windows operating systems. The AIX, OS/2, OS/400, and Windows operating systems also offer the C++ language.

Changes for z/OS V1R1 and OS/390 V2R10

Note: z/OS V1R1 C/C++ is functionally equivalent to OS/390 V2R10 C/C++. This section describes changes that were introduced in OS/390 V2R10.

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

Extra Performance Linkage (XPLINK)
: Extra Performance Linkage (XPLINK) is a new call linkage between functions that has the potential for a significant performance increase when used in an environment of frequent calls between small functions. XPLINK makes subroutine calls more efficient by removing nonessential instructions from the main path. When all functions are compiled with the XPLINK option, pointers can be used without restriction, which makes it easier to port new applications to S/390.

GOFF
: The Generalized Object File Format (GOFF) is the strategic object module format for S/390. It extends the capabilities of object modules to contain more information than current object modules. It removes the limitations of the previous object module format and supports future enhancements. GOFF makes re-binding easier and more efficient. It is required for XPLINK.

IPA Level 2
: Under IPA Level 1, many optimizations such as constant propagation and pointer analysis are performed at the intraprocedural (subprogram) level. With IPA Level 2, these optimizations are performed across the entire program, which can result in significant improvement in the generated code.

Addition of @STATIC Map into Compiler Listing
: The @STATIC Map displays offset information for file scope read/write static variables.

This release has introduced the following compiler option:

`COMPACT`	During optimizations performed during code generation, for both NOIPA and IPA, choices must be made between those optimizations which tend to result in faster but larger code and those which tend to result in smaller but slower code. The `COMPACT` | `NOCOMPACT` option controls these choices. When the `COMPACT` option is used, the compiler favors those optimizations which tend to limit the growth of the code. This feature gives you the flexibility to choose between faster but larger code or slower and smaller code.

The IBM System Object Model™ (SOM) is no longer supported in the C++ compiler and the IBM Open Class™ Library. The SOM-enabled class library DLLs have been stabilized at the V2R9 level and continue to be shipped as a run-time environment only. You cannot use the V2R10 Compiler to build SOM® applications.

The Model Tool is no longer available.

The option_override #pragma directive defines function-specific options that override those specified by the command line options when performing optimization for code and data in that subprogram. This enables finer control of program optimization. In V2R10 we have added support for the `COMPACT` and `SPILL` options. The subprogram-specific `SPILL` option is not a new option, however, the maximum spill area size has been increased for this release to `1073741823` bytes or $2^{30}-1$ bytes.

z/OS Language Environment® Downward Compatibility

z/OS V1R1 Language Environment provides downward compatibility support. Assuming that you have met the required programming guidelines and restrictions, described in *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 the “TARGET” on page 179 for details on this compiler option.

For example, a company may use z/OS V1R1 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 z/OS. 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 V1R1 and by the toleration PTFs does not change Language Environment’s upward compatibility. That is, applications coded and link-edited with one release of z/OS Language Environment will continue to run on later releases of 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 Release 6), but in earlier releases of OS/390 the user is required to copy header files and link-edit syslib datasets from the deployment release of OS/390. Starting with OS/390 Release 10, the Release 10 header files and syslib datasets can be used.

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, but incorporates support for object-oriented concepts. For a detailed description of the differences between z/OS C++ and z/OS C, refer to the *z/OS 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 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 to improve 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 (prelinker) and program management binder. The z/OS C++ compiler always ensures that C++ programs are reentrant.
- Locale-based internationalization support derived from the *IEEE POSIX 1003.2-1992* standard. Also derived from the *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.
- When used with z/OS UNIX and z/OS Language Environment, support for the following standards at the system level:
  - A subset of the extended multibyte and wide character functions as defined by the *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
- Additional optimization capabilities through the `INLINE` compile-time option
- 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 provides you with the following:

- An implementation based on the definition of the language that is contained in the *Draft Proposal International Standard for Information Systems – Programming Language C++ (X3J16/92-00091)*. The z/OS C/C++ compiler also supports a *subset* of the *International Standard for the C++ Programming Language (ISO/IEC 14882-1998)* specification. The following items in the standard are **not** supported by z/OS C++:
  - New cast syntax and semantics
    - `dynamic_cast`
    - `static_cast`
    - `reinterpret_cast`
    - `const_cast`
  - Explicit specifier
  - Mutable specifier
  - Namespace
  - Run-Time Type Identification (RTTI)
  - ANSI Template (supports the specification in X3J16/92-00091 as mentioned above)
  - The `bool` built-in boolean data type
  - Run-time standard exceptions
    - `bad_alloc`
    - `bad_exception`
    - `bad_cast`
  - Universal Character Names
  - ANSI C++ Standard Class Library (which includes the Standard Template Library)
- C++ template support and exception handling.

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 localedef Utility to read the locale definition file and produce a locale object that the locale-specific library functions can use.
- The DSECT Conversion Utility to convert descriptive assembler DSECTs into z/OS C/C++ data structures.

z/OS Language Environment provides the following utilities:

- The Object Library Utility (C370LIB) to update partitioned data set (PDS and PDS/E) 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 an 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.

Class Libraries

z/OS C/C++ provides a base set of class libraries, called C/C++ IBM Open Class, which is consistent with that available in other members of the VisualAge® C++ Version 3.0 product family. These class libraries are:

- The I/O Stream Class Library

  The I/O Stream Class Library lets you perform input and output (I/O) operations independent of physical I/O devices or data types that are used. You can code sophisticated I/O statements easily and clearly, and define input and output for your own data types. You can improve the maintainability of programs that use input and output by using the I/O Stream Class Library.
- The Complex Mathematics Class Library

  The Complex Mathematics Class Library lets you manipulate and perform standard arithmetic on complex numbers. Scientific and technical fields use complex numbers.
- The Application Support Class Library

  The Application Support Class Library provides the basic abstractions that are needed during the creation of most C++ applications, including String, Date, Time, and Decimal.
- The Collection Class Library

  The Collection Class Library implements a wide variety of classical data structures such as stack, tree, list, hash table, and so on. Most programs use collections. You can develop programs without having to define every collection. Programmers can start programming by using a high level of abstraction, and later replace an abstract data type with the appropriate concrete implementation. Each abstract data type has a common interface for all of its implementations. The Collection Class Library provides programmers 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.

All of the libraries that are described above are thread-safe.

All of the libraries that are described above are available in both static and DLL formats. z/OS C/C++ packages the Application Support Class and Collection Class libraries together in a single DLL. For compatibility, separate side-decks are available for the Application Support Class and Collection Class libraries, in addition to the side-deck available for the combined library.

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 feature(s) or to use the DLL Rename Utility.

The DLLs for the Open Class libraries are compiled without `XPLINK`. If you use these DLLs with `XPLINK` applications, the performance gain you realize in your application code by using `XPLINK` may be offset partially or completely (depending on the frequency of use of DLL functions) by the cost of switching to the non-`XPLINK` environment when crossing the boundary between your application code and the class library code in the DLL. If you use these DLLs with `XPLINK` applications, you may notice reduced performance. There are two ways to avoid this problem:

- Use the static library instead of the DLL. This static library has both the `XPLINK` and `NOXPLINK` versions of the objects.
- For the Application Support Class Library or Collection Class Library, recompile the source code that is shipped with z/OS C/C++. For build instructions, refer to the CBC.SCLDBLD readme file.

Class Library Source

The Class Library Source consists of the following:

- Application Support Class Library source code
- Collection Class Library source code (C++ native)
- Instructions for building the Application Support Class and Collection Class Libraries in C++ native (static and DLL) versions
- Class Library Language Environment message file source
- Instructions for building the Class Library Language Environment message files

The Debug Tool

z/OS C/C++ supports program development by using theDebug 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*, SA22-7802.

For further information, see "IBM C/C++ Productivity Tools for z/OS".

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

With the *IBM C/C++ Productivity Tools for z/OS* 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 z/OS 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 z/OS 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 z/OS Performance Analyzer included in the C/C++ Productivity Tools for z/OS product.

z/OS Language Environment

z/OS C/C++ exploits the C/C++ runtime environment and library of runtime 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 runtime libraries, and Base Routines and Common Services, as shown below. z/OS Language Environment establishes a common runtime environment and common runtime services for language products, user programs, and other products.

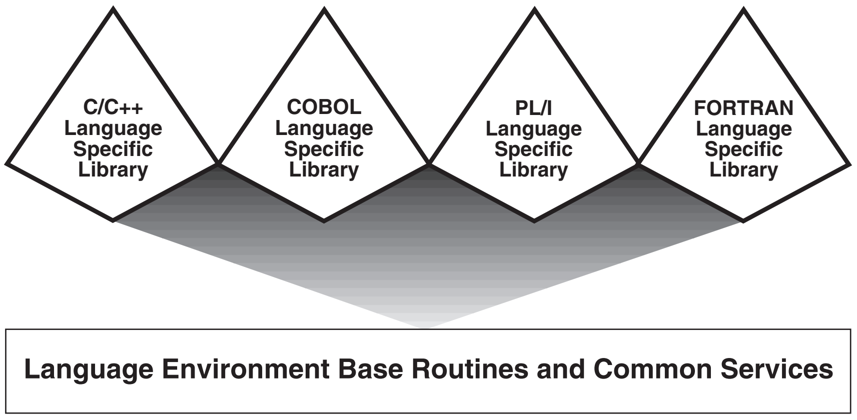

C/C++ Language Specific Library
COBOL Language Specific Library
PL/I Language Specific Library
FORTRAN Language Specific Library
Language Environment Base Routines and Common Services

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.
- Runtime 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.

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` compile-time options, you must use 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

The prelinker provided with z/OS Language Environment combines the object modules comprising an 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 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:

- C programming language support as defined by ISO/ANSI C
- C++ programming language support
- 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 applications
dbx	Provides an environment to debug and run programs
gencat	Merges the message text source files Messagefile (usually *.msg) into a formatted message Catalogfile (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 C/C++ 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
- z/OS UNIX, which provides 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.

To 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)` runtime 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 ANSI. Refer to *z/OS C/C++ Run-Time Library Reference* for more information about requirements for each function.

You can invoke an 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)` runtime option.

Input and Output

The C/C++ runtime 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++ runtime library supports the following I/O interfaces:

C Stream I/O
This is the default and the ANSI-defined I/O method. This method processes all input and output by character.

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 an z/OS C/C++ extension to the ANSI 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 that are 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 others in the network using TCP/IP.

In addition, the C++ I/O Stream Library supports 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++ runtime 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 the reverse order).

- Relative-record data sets (RRDS). Use RRDS for data in which each item has a particular number (for example, a telephone system with a record associated with each 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 gigabytes of contiguous virtual storage space. A program can use this storage as a buffer (1 gigabyte = 2^{30} bytes).

Additional I/O Features

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

- 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 PDS/Es 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 to better utilize the low-level services available on your operating system. SPC offers a number of advantages:

- You can develop applications that you can execute 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++ runtime 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 an 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 independently 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 V4 R1 is not supported for z/OS C++ applications. z/OS C++ code preprocessed on CICS/ESA V4 R1 cannot run under CICS/ESA V3 R3.
- 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).

  The DB2 program uses SQL statements that are embedded in the program. The SQL translator (DB2 preprocessor) translates the embedded SQL into host language statements that perform the requested functions. The z/OS C/C++ compilers compile the output of the SQL translator. The DB2 program processes a request, and processing returns to the application.
- 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 person 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 double-byte character set (DBCS)
  - 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. There is no distinction between compiled Java and interpretted Java as far as calls to C or C++.

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 in `LANGLVL(EXTENDED)` mode.
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 “FLOAT” on page 108.

Feature	Description
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 the 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*
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.

Feature	Description
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 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 a G2, G3, G4, and 2003 processor and code compiled with `ARCH(3)` runs on a G5 or G6 processor, and follow-on models. For information on which machines and architectures support the above options, refer to "ARCHITECTURE" on page 81. Use the `TUNE` compiler option to optimize your application for a selected 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 information on which machines and architectures support the above options, refer to "TUNE" on page 190.

Chapter 3. About Prelinking, Linking, and Binding

When describing the process to build an application, this document refers to the *bind step*. Unless otherwise stated, the earlier prelink and link steps can be substituted for the bind step. The prelink and link steps are described in detail in "Appendix A. Prelinking and Linking z/OS C/C++ Programs" on page 461.

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 *z/OS 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.

Prior to OS/390 C/C++ Version 2 Release 4, this book 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 S/370 operating system. The Prelinker, shipped with the z/OS C/C++ RunTime 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*.

Note that for problem determination, ensure that you have applied the latest binder maintenance. 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++ RunTime Library and the Linkage-Editor (supplied through the binder) you have to apply the latest maintenance for the RunTime Library as well as the binder.

Prior to OS/390 C/C++ Version 2 Release 4, examples in this book showed how to use the prelinker and linkage-editor. As of OS/390 C/C++ Version 2 Release 4, these examples show how to use the binder, and the concept of binding is discussed throughout the book.

If you still need to use the prelinker and linkage-editor, see "Appendix A. Prelinking and Linking z/OS C/C++ Programs" on page 461.

TARGET Option Considerations

If you are using the `TARGET` compiler option to build an application to run on OS/390 V1R3, you must use the prelinker and linker instead of the binder.

File Format Considerations

You can use the binder in place of the prelinker and linkage-editor, with the following exceptions:

- **CICS**

  Prior to CICS 1.3, PDSEs are not supported. From CICS Transaction Server 1.3 onwards, there is support in CICS for PDSEs. Please refer to the *CICS Transaction Server for OS/390 Release Guide*, GC34-5701, where there are several references to PDSEs, and a list of prerequisite APAR fixes.
- **MTF**

  MTF does not support PDSEs. If your program targets MTF, you cannot use the binder.
- **IPA Restrictions**

  Object files that are generated by the IPA Compile step using the compiler option `IPA(NOLINK,OBJECT)` may be given as input to the binder. Such an object file is a combination of an IPA object module, and a regular compiler object module. The binder processes the regular compiler object module, ignores the IPA object module, and no IPA optimization is done.

  Object files that are generated by the IPA Compile step using compiler option `IPA(NOLINK,NOOBJECT)` should not be given as input to the binder. These are IPA only object files, and do not contain a regular compiler object module.

  IPA Link can process load module (PDS) input files, but not program object (PDSE) input files.

Part 2. User’s Reference

This part reviews the basic steps for compiling, binding, and running z/OS C/C++ programs under the z/OS operating system. It also describes the options available to you at compile, IPA link, bind, and run time.

Chapter 4. z/OS C Example

This chapter outlines the basic steps for compiling, binding, and running a C example program under z/OS batch, TSO, or the z/OS shell.

If you have not yet compiled a C program, some concepts in this chapter may be unfamiliar. Refer to “Chapter 9. Compiling” on page 283, “Chapter 12. Binding z/OS C/C++ Programs” on page 353, and “Chapter 14. Running a C or C++ Application” on page 401 for a detailed description on compiling, binding, and running a C program.

This chapter describes steps to bind an C example program. It does not describe the prelink and link steps. If you are using the prelinker, see “Appendix A. Prelinking and Linking z/OS C/C++ Programs” on page 461.

The example program that this chapter describes is shipped with the z/OS C compiler in the data set CBC.SCBCSAM.

Example of a C Program

The following example shows a simple z/OS C program that converts temperatures in Celsius to Fahrenheit. You can either enter the temperatures on the command line or let the program prompt you for the temperature.

In this example, the main program calls the function `convert()` to convert the Celsius temperature to a Fahrenheit temperature and to print the result.

CBC3UAAM

```
#include <stdio.h>                 1

#include "cbc3uaan.h"              2

void convert(double);              3

int main(int argc, char **argv)    4
{
    double c_temp;                 5

    if (argc == 1) {  /* get Celsius value from stdin */

       printf("Enter Celsius temperature: \n");  6

       if (scanf("%f", &c_temp) != 1) {
          printf("You must enter a valid temperature\n");
       }
       else {
          convert(c_temp);         7
       }
    }
```

Figure 2. Celsius-to-Fahrenheit Conversion (Part 1 of 2)

```
    else { /* convert the command-line arguments to Fahrenheit */
       int i;

       for (i = 1; i < argc; ++i) {
           if (sscanf(argv[i], "%f", &c_temp) != 1)
              printf("%s is not a valid temperature\n",argv[i]);
           else
              convert(c_temp);    7
       }
    }
}

void convert(double c_temp) {     8
   double f_temp = (c_temp * CONV + OFFSET);
   printf("%5.2f Celsius is %5.2f Fahrenheit\n",c_temp, f_temp);
}
```

Figure 2. Celsius-to-Fahrenheit Conversion (Part 2 of 2)

CBC3UAAN

```
/**************************************************************
 *  User include file:  cbc3uaan.h                            *   9
 **************************************************************/

#define CONV   (9./5.)
#define OFFSET 32
```

Figure 3. User #include File for the Conversion Program

1 This preprocessor directive includes the system file that contains the declarations of standard library functions, such as the `printf()` function used by this program.

The compiler searches the system libraries for the file STDIO. For more information about searches for include files, see “Search Sequences for Include Files” on page 316.

2 This preprocessor directive includes a user file that defines constants that are used by the program.

The compiler searches the user libraries for the file CBC3UAAN.

If the compiler cannot locate the file in the user libraries, it searches the system libraries.

3 This is a function prototype declaration. This statement declares `convert()` as an external function having one parameter.

4 The program begins execution at this entry point.

5 This is the automatic (local) data definition to `main()`.

6 This `printf` statement is a call to a library function that allows you to format your output and print it on the standard output device. The `printf()` function is declared in the standard I/O header file `stdio.h` included at the beginning of the program.

7 This statement contains a call to the `convert()` function, which was declared earlier in the program as receiving one double value, and not returning a value.

8 This is a function definition. In this example, the declaration for this function appears immediately before the definition of the `main()` function. The code for the function is in the same file as the code for the `main()` function.

9 This is the user include file containing the definitions for `CONV` and `OFFSET`.

If you need more details on the constructs of the z/OS C language, see the *z/OS C/C++ Language Reference* and the *z/OS C/C++ Run-Time Library Reference*.

Compiling, Binding, and Running the z/OS C Example

In general, you can compile, bind, and run z/OS C programs under z/OS batch, TSO, or the z/OS shell. You cannot run the IPA Link step under TSO, or under z/OS batch by using the ISPF interface. For more information, see “Chapter 9. Compiling” on page 283, “Chapter 12. Binding z/OS C/C++ Programs” on page 353, and “Chapter 14. Running a C or C++ Application” on page 401.

This book uses the term *user prefix* to refer to the high-level qualifier of your data sets. For example, in `PETE.TESTHDR.H`, the user prefix is `PETE`. Under TSO, your prefix is set or queried by the `PROFILE` command.

Under z/OS Batch

Copy the IBM-supplied sample program and header file into your data set. For example, if your user prefix is `PETE`, store the sample program (`CBC3UAAM`) in `PETE.TEST.C(CTOF)` and the header file in `PETE.TESTHDR.H(CBC3UAAN)`. You can use the IBM-supplied cataloged procedure EDCCBG to compile, bind, and run the example program as follows:

```
//DOCLG       EXEC   PROC=EDCCBG,INFILE='PETE.TEST.C(CTOF)',
//            CPARM='LSEARCH(''''PETE.TESTHDR.+'''')'
//GO.SYSIN    DD DATA,DLM=@@
19
@@
```

Figure 4. JCL to Compile, Bind, and Run the Example Program Using the EDCCBG Procedure

In Figure 4, the `LSEARCH` statement describes where to find the user include files. The `GO.SYSIN` statement indicates that the input that follows it is given for the execution of the program.

XPLINK Under z/OS Batch

Figure 5 shows the JCL for building with `XPLINK`.

```
//DOCLG       EXEC   PROC=EDCXCBG,INFILE='PETE.TEST.C(CTOF)',
//            CPARM='LSEARCH(''''PETE.TESTHDR.+'''')'
//GO.SYSIN    DD DATA,DLM=@@
19
@@
```

Figure 5. JCL to Build with XPLINK

Non-XPLINK and XPLINK Under TSO

Copy the IBM-supplied sample program and header file into your data set. For example, if your user prefix is PETE, store the sample z/OS C program (CBC3UAAM) in PETE.TEST.C(CTOF) and the header file in PETE.TESTHDR.H(CBC3UAAN).

Use the following set of TSO commands to compile, bind, and run the example program:

1. Ensure that the z/OS Language Environment runtime libraries SCEERUN and SCEERUN2, and the z/OS C compiler are in the STEPLIB, LPALST, or LNKLST concatenation.
2. Compile the z/OS C source. You can use the REXX EXEC CC to invoke the z/OS C compiler under TSO as follows:

   ```
   %CC TEST.C(CTOF) (LSEARCH(TESTHDR.H)

   -- or, for XPLINK --

   %CC TEST.C(CTOF) (LSEARCH(TESTHDR.H) XPLINK
   ```

   The REXX EXEC CC compiles CTOF with the default compiler options and stores the resulting object module in PETE.TEST.C.OBJ(CTOF).

   The compiler searches for user header files in the PDS PETE.TESTHDR.H, which you specified at compile time by the LSEARCH option.

   For more information see “Compiling Under TSO” on page 293.
3. Perform a bind:

   ```
   CXXBIND OBJ(TEST.C.OBJ(CTOF)) LOAD(TEST.C.LOAD(CTOF))

   -- or, for XPLINK --

   CXXBIND OBJ(TEST.C.OBJ(CTOF)) LOAD(TEST.C.LOAD(CTOF)) XPLINK
   ```

   CXXBIND binds the object module PETE.TEST.C.OBJ(CTOF) to create an executable module CTOF in the PDSE PETE.TEST.C.LOAD, with the default bind options. See “Chapter 12. Binding z/OS C/C++ Programs” on page 353 for more information.
4. Run the program:

   ```
   CALL TEST.C.LOAD(CTOF)
   ```

   CALL runs CTOF from PETE.TEST.C.LOAD with the default runtime options in effect. See “Chapter 14. Running a C or C++ Application” on page 401 for more information.

Non-XPLINK and XPLINK Under the z/OS UNIX Shell

Put the source in the HFS.

1. Ensure that the z/OS Language Environment runtime libraries SCEERUN and SCEERUN2, and the z/OS C compiler are in the STEPLIB, LPALST, or LNKLST concatenation.
2. From the z/OS shell type:

   ```
   cp "//'cbc.scbcsam(cbc3uaam)'" cbc3uaam.c
   cp "//'cbc.scbcsam(cbc3uaan)'" cbc3uaan.h
   ```
3. Compile and bind:

```
c89 -o ctof cbc3uaam.c

-- or, for XPLINK --

c89 -o ctof -Wc,xplink -Wl,xplink cbc3uaam.c
```

4. Run the program:

   ```
   ./ctof
   ```

Chapter 5. z/OS C++ Examples

This chapter outlines the basic steps for compiling, binding, and running z/OS C++ example programs under z/OS batch, TSO, or the z/OS shell.

If you have not yet compiled a C++ program, some concepts in this chapter may be unfamiliar. Refer to “Chapter 9. Compiling” on page 283, “Chapter 12. Binding z/OS C/C++ Programs” on page 353, and “Chapter 14. Running a C or C++ Application” on page 401 for a detailed description on compiling, binding, and running an C++ program.

The example programs that this chapter describes are shipped with the z/OS C++ compiler. Example programs with the names `CBC3Uxxx` are shipped in the data set CBC.SCBCSAM. Example programs with the names `CLB3xxxx` are shipped in the data set CBC.SCLBSAM.

Example of a C++ Program

The following example shows a simple z/OS C++ program that prompts you to enter a birth date. The program output is the corresponding biorhythm chart.

The program is written in object-oriented fashion. A class that is called `BioRhythm` is defined. It contains an object `birthDate` of class `BirthDate`, which is derived from the class `Date`. An object that is called `bio` of the class `BioRhythm` is declared.

The example contains 2 files. File `CBC3UBRH` contains the classes that are used in the main program. File `CBC3UBRC` contains the remaining source code. The example files `CBC3UBRC` and `CBC3UBRH` are shipped with the z/OS C++ compiler in data sets CBC.SCBCSAM(CBC3UBRC) and CBC.SCBCSAM(CBC3UBRH).

If you need more details on the constructs of the z/OS C++ language, see the *z/OS C/C++ Language Reference* or the *z/OS C/C++ Run-Time Library Reference*.

CBC3UBRH

```
//
// Sample Program: Biorhythm
// Description   : Calculates biorhythm based on the current
//                 system date and birth date entered
//
// File 1 of 2-other file is CBC3UBRC
class Date {
  public:
    Date();
    int DaysSince(const char *date);
  protected:
    int curYear, curDay;
    static const int dateLen;
    static const int numMonths;
    static const int numDays[];
};
const int Date::dateLen = 10;
const int Date::numMonths = 12;
const int Date::numDays[Date::numMonths] = {
  31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};

class BirthDate : public Date {
  public:
    BirthDate();
    BirthDate(const char *birthText);
    int DaysOld() { return(DaysSince(text)); }

  private:
    char text[dateLen+1];
};
```

Figure 6. Header File for the Biorhythm Example (Part 1 of 2)

```
class BioRhythm {
  friend static ostream& operator<<(ostream&, BioRhythm&);

  public:
    BioRhythm(char *birthText) : birthDate(birthText) {
      age = birthDate.DaysOld();
    }
    BioRhythm() : birthDate() {
      age = birthDate.DaysOld();
    }
    ~BioRhythm() {}

    int AgeInDays() {
      return(age);
    }
    double Physical()  {
      return(Cycle(pCycle));
    }
    double Emotional() {
      return(Cycle(eCycle));
    }
    double Intellectual() {
      return(Cycle(iCycle));
    }
    int ok() {
      return(age >= 0);
    }

  private:
    int age;
    double Cycle(int phase) {
      return(sin(fmod(age, phase) / phase * M_2PI));
    }
    BirthDate birthDate;
    static const int pCycle;
    static const int eCycle;
    static const int iCycle;
};

const int BioRhythm::pCycle=23;     // Physical cycle - 23 days
const int BioRhythm::eCycle=28;     // Emotional cycle - 28 days
const int BioRhythm::iCycle=33;     // Intellectual cycle - 33 days

static ostream& operator<<(ostream&,BioRhythm&);
```

Figure 6. Header File for the Biorhythm Example (Part 2 of 2)

CBC3UBRC

```
//
// Sample Program: Biorhythm
// Description   : Calculates biorhythm based on the current
//                 system date and birth date entered
//
// File 2 of 2-other file is CBC3UBRH

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <iostream.h>
#include <iomanip.h>

#include "cbc3ubrh.h"  //BioRhythm class and Date class

int main(void) {
  BioRhythm bio;
  int code;

  if (!bio.ok()) {
    cerr << "Error in birthdate specification - format is yyyy/mm/dd\n";
    code = 8;
  }
  else {
    cout << bio;  // write out birthdate for bio
    code = 0;
  }
  return(code);
}

static ostream& operator<<(ostream& os, BioRhythm& bio) {
  os << "Total Days  : " << bio.AgeInDays() << "\n";
  os << "Physical    : " << bio.Physical() << "\n";
  os << "Emotional   : " << bio.Emotional() << "\n";
  os << "Intellectual: " << bio.Intellectual() << "\n";

return(os);
}

Date::Date() {
  time_t lTime;
  struct tm *newTime;

  time(&lTime);
  newTime = localtime(&lTime);
  cout << "local time is " << asctime(newTime) << endl;


  curYear = newTime->tm_year + 1900;
  curDay  = newTime->tm_day + 1;
}
```

Figure 7. z/OS C++ Biorhythm Example Program (Part 1 of 2)

```
BirthDate::BirthDate(const char *birthText) {
  strcpy(text, birthText);
}

BirthDate::BirthDate() {
  cout << "Please enter your birthdate in the form yyyy/mm/dd\n";
  cin >> setw(dateLen+1) >> text;
}

Date::DaysSince(const char *text) {

  int year, month, day, totDays, delim;
  int daysInYear = 0;
  int i;
  int leap = 0;

  int rc = sscanf(text, "%4d%c%2d%c%2d",
                  &year, &delim, &month, &delim, &day);
  --month;
  if (rc != 5 || year  < 0 || year  > 9999 ||
                month < 0 || month >   11 ||
                day   < 1 || day   >   31 ||
               (day   >  numDays[month]&& month != 1)) {
    return(-1);
  }
  if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
     leap = 1;

  if (month == 1 && day > numDays[month]) {
    if (day > 29)
      return(-1);
    else if (!leap)
      return (-1);
  }

  for (i=0;i<month;++i) {
    daysInYear += numDays[i];
  }
  daysInYear += day;

  // correct for leap year
  if (leap == 1 &&
     (month > 1))
     ++daysInYear;

  totDays = (curDay - daysInYear) + (curYear - year)*365;

  // now, correct for leap year
  for (i=year+1; i < curYear; ++i) {
    if ((i % 4 == 0 && i % 100 != 0) || i % 400 == 0) {
      ++totDays;
    }
  }
  return(totDays);
}
```

Figure 7. z/OS C++ Biorhythm Example Program (Part 2 of 2)

Compiling, Binding, and Running the z/OS C++ Example

In general, you can compile, bind, and run z/OS C++ programs under z/OS batch, TSO, or the z/OS shell. You cannot run the IPA Link step under TSO, or under z/OS batch by using the ISPF interface. For more information, see "Chapter 9. Compiling" on page 283, "Chapter 12. Binding z/OS C/C++ Programs" on page 353, and "Chapter 14. Running a C or C++ Application" on page 401.

This book uses the term *user prefix* to refers to the high-level qualifier of your data sets. For example, in `CEE.SCEERUN`, the user prefix is `CEE`.

Under z/OS Batch

Copy the IBM-supplied sample program and header file into your data set. For example, if your user prefix is `PETE`, store the sample program (`CBCUBRC`) in `PETE.TEST.C(CBC3UBRC)`, and the header file (`CBCUBRC`) in `PETE.TESTHDR.H(CBC3UBRH)`. You can use the IBM-supplied cataloged procedure CBCCBG to compile, bind, and run the source code as follows:

```
//*
//* COMPILE, BIND AND RUN
//*
//DOCLG    EXEC  CBCCBG,
//         INFILE='PETE.TEST.C(CBC3UBRC)',
//         CPARM='OPTFILE(DD:CCOPT)'
//COMPILE.CCOPT DD *
          LSEARCH('PETE.TESTHDR.H')
          SEARCH('CEE.SCEEH.+','CBC.SCLBH.+')
/*
//* ENTER TODAY'S DATE IN THE FORM YYYY/MM/DD
//GO.SYSIN DD *
  1997/10/19
/*
```

Figure 8. JCL to Compile, Bind, and Run the Example Program Using the CBCCBG Procedure

In Figure 8, the `LSEARCH` statement describes where to find the user include files, and the `SEARCH` statement describes where to find the system include files. The `GO.SYSIN` statement indicates that the input that follows it is given for the execution of the program.

XPLINK Under z/OS Batch

The following example shows how to compile, bind, and run a program with `XPLINK` using the CBCXCBG procedure:

```
//*
//* COMPILE, BIND AND RUN
//*
//DOCLG    EXEC  CBCXCBG,
//         INFILE='PETE.TEST.C(CBC3UBRC)',
//         CPARM='OPTFILE(DD:CCOPT)'
//COMPILE.CCOPT DD *
        LSEARCH('PETE.TESTHDR.H')
        SEARCH('CEE.SCEEH.+','CBC.SCLBH.+')
/*
//* ENTER TODAY'S DATE IN THE FORM YYYY/MM/DD
//GO.SYSIN DD *
  1997/10/19
/*
```

Figure 9. JCL to Compile, Bind, and Run the Example Program with XPLINK Using the CBCXCBG Procedure

For more information on compiling, binding, and running, see “Chapter 9. Compiling” on page 283, “Chapter 12. Binding z/OS C/C++ Programs” on page 353, and “Chapter 14. Running a C or C++ Application” on page 401.

Non-XPLINK and XPLINK Under TSO

Copy the IBM-supplied sample program and header file into your data set. For example, if your user prefix is `PETE`, store the sample program (`CBCUBRC`) in `PETE.TEST.C(CBC3UBRC)`, and the header file (`CBCUBRH`) in `PETE.TESTHDR.H(CBC3UBRH)`.

Use the following set of TSO commands to compile, bind, and run the example program:

1. Ensure that the z/OS Language Environment runtime libraries `SCEERUN` and `SCEERUN2`, the z/OS class library DLLs, and the z/OS C++ compiler are in the `STEPLIB`, dynamic LPA, or Link List concatenation.
2. Compile the z/OS C++ source. You can use the REXX EXEC `CXX` to invoke the z/OS C++ compiler under TSO as follows:

   ```
   CXX 'PETE.TEST.C(CBC3UBRC)' ( LSEARCH('PETE.TESTHDR.H') OBJECT(BIO.TEXT)
            SEARCH('CEE.SCEEH.+','CBC.SCLBH.+')

   -- or, for XPLINK --

    CXX 'PETE.TEST.C(CBC3UBRC)' ( LSEARCH('PETE.TESTHDR.H') OBJECT(BIO.TEXT)
    SEARCH('CEE.SCEEH.+','CBC.SCLBH.+') XPLINK
   ```

   `CXX` compiles `CBC3UBRC` with the specified compiler options and stores the resulting object module in `PETE.BIO.TEXT(CBC3UBRC)`.

   The compiler searches for user header files in the PDS `PETE.TESTHDR.H` , which you specified at compile time with the `LSEARCH` option. The compiler searches for system header files in the PDS `CEE.SCEEH.+` and `CBC.SCLBH.+`, which you specified at compile time with the `SEARCH` option.

   For more information see “Compiling Under TSO” on page 293.
3. Bind:

   ```
   CXXBIND OBJ(BIO.TEXT(CBC3UBRC)) LOAD(BIO.LOAD(BIORUN))

   -- or, for XPLINK --

    CCXXBIND OBJ(BIO.TEXT(CBC3UBRC)) LOAD(BIO.LOAD(BIORUN)) XPLINK
   ```

CXXBIND binds the object module PETE.BIO.TEXT(CBC3UBRC), and creates an executable module BIORUN in PETE.BIO.LOAD PDSE with the default bind options.

Note: To avoid a bind error, the dataset PETE.BIO.LOAD must be a PDSE, not a PDS.

For more information see "Chapter 12. Binding z/OS C/C++ Programs" on page 353.

4. Run the program:

```
CALL BIO.LOAD(BIORUN)
```

CALL runs the module BIORUN from the PDSE PETE.BIO.LOAD with the default runtime options.

For more information see "Running an Application under TSO" on page 404.

Non-XPLINK and XPLINK Under the z/OS UNIX Shell

Put the source in the HFS. From the z/OS shell type:

```
cp "//'cbc.scbcsam(cbc3ubrc)'" cbc3ubrc.C
cp "//'cbc.scbcsam(cbc3ubrh)'" cbc3ubrh.h
```

In this example, the current working directory is used, so make sure that you are in the directory you want to use. Use the pwd command to display the current working directory, the mkdir command to create a new directory, and the cd command to change directories.

1. Ensure that the z/OS Language Environment runtime libraries SCEERUN and SCEERUN2, the z/OS class library DLLs, and the z/OS C++ compiler are in the STEPLIB, dynamic LPA, or Link List concatenation.
2. Compile and bind:

```
c++ -o bio cbc3ubrc.C

-- or, for XPLINK --

c++ -o bio -Wc,xplink -Wl,xplink cbc3ubrc.C
```

Note: You can use c++ to compile source that is stored in a data set.

3. Run the program:

```
./bio
```

Example of a C++ Template Program

A class template or generic class is a blueprint that describes how members of a set of related classes are constructed.

The following example shows a simple z/OS C++ program that uses templates to perform simple operations on linked lists. This program consists of ten files that are described and illustrated below.

The main program, CLB3ATMP.CXX (see "CLB3ATMP.CXX" on page 53) has two class templates: List (in the file CLB3ALST.C that uses CLB3ALST.H) and Iterator (in the file CLB3AITR.C that uses CLB3AITR.H). List is a template of a linked list, and Iterator is a template that walks a List class.

CLB3ALST.C

```
#include "clb3alst.h"
template <class Item> void List<Item>::append(Item item) {
  GetNode();
  cur->node = item;
}

template <class Item> void List<Item>::GetNode() {
  if (cur) {
    cur->next = new Node<Item>;
    cur = cur->next;
  }
  else {
    cur = new Node<Item>;
    head = cur;
  }
  cur->next = 0;
  return;
}
```

Figure 10. Template of a Linked List

CLB3ALST.H

```
                ??=ifndef _CBCLIST_H_
                ??=ifdef __COMPILER_VER__
                  ??=pragma filetag ("IBM-1047")
                ??=endif
                ??=define _CBCLIST_H_  1
                #pragma nomargins nosequence
                #pragma checkout (suspend)

template <class Item> struct Node {
  Item node;
  struct Node<Item> *next;
};

template <class Item> class List {
  public:

    List() :cur(0), head(0) {}

      ~List() {}
    void append(Item item);
    Node<Item> *cur, *head;

  private:
    void GetNode();
};
                #pragma checkout (resume)
                #endif
```

Figure 11. Header file for CBC3ALST.C

CLB3AITR.C

```
#include "clb3aitr.h"
template <class Item> Item& Iterator<Item>::operator++() {
  node = cur->node;
  cur  = cur->next;
  return(node);
}

template <class Item> int Iterator<Item>::eol() {
  return(cur == 0);
}

template <class Item> void Iterator<Item>::reset() {
  cur = head;
}
```

Figure 12. Template of an Iterator

CLB3AITR.H

```
                  ??=ifndef _CBCITER_H_
                  ??=ifdef __COMPILER_VER__
                    ??=pragma filetag ("IBM-1047")
                  ??=endif
                  ??=define _CBCITER_H_    1
                  #pragma nomargins nosequence
                  #pragma checkout (suspend)
#include "clb3alst.h"
template <class Item> class Iterator {
  public:
    Iterator(List<Item>& list)
            :cur(list.head), head(list.head) {}
    Item& operator++();
    int eol();
    void reset();

  private:
    Node<Item> *cur;
    Node<Item> *head;
    Item node;
};
                  #pragma checkout (resume)
                  #endif
```

Figure 13. Header file for CLB3AITR.C

There are two template functions, `max(T,T)` (in the file CLB3AMAX.C which uses CLB3AMAX.H), and `min(T,T)` (in the file CLB3AMIN.C which uses CLB3AMIN.H). `max(T,T)` returns the maximum object of two objects, and `min(T,T)` returns the minimum object of two objects.

CLB3AMAX.H

```
template <class T> T& max(T a, T b);
```

CLB3AMAX.C

```
template <class T> T& max(T a, T b) {
  if (a > b) return(a);
  else return(b);
}
```

CLB3AMIN.H

```
template <class T> T& min(T a, T b);
```

CLB3AMIN.C

```
template <class T> T& min(T a, T b) {
  if (a < b) return(a);
  else return(b);
}
```

There is one simple class, `String`, defined in the file CLB3ASTR.H.

CLB3ASTR.H

```
                ??=ifndef _CBCSTR_H_
                ??=ifdef __COMPILER_VER__
                  ??=pragma filetag ("IBM-1047")
                ??=endif
                ??=define _CBCSTR_H_    1
                #pragma nomargins nosequence
                #pragma checkout (suspend)
#include <iostream.h>
#include <iomanip.h>
class String {
  friend static ostream& operator<<(ostream&, String&);
  friend static istream& operator>>(istream&, String&);
  public:
    String() {
      str = new char[1];
      str[0] = '\0';
    }
    String(const char *s) {
      const int len = strlen(s);
      str = new char[len+1];
      memcpy(str, s, len+1);
    }

     ~String() {
      delete str;
    }
    void replace(const char *s) {
      const int len = strlen(s);
      char *newStr = new char[len+1];
      delete str;
      str = newStr;
      memcpy(str, s, len+1);
    }
    int operator >(String& rhs) {
      return(strcmp(str, rhs.string()));
    }
    int operator <(String& rhs) {
      return(!strcmp(str, rhs.string()));
    }
    const char *string() {
      return(str);
    }
  private:
    char *str;
};
                #pragma checkout (resume)
                #endif
```

Figure 14. Definition of the String Class

CLB3ATMP.CXX

```
#include "clb3amax.h"
#include "clb3amin.h"
#include "clb3alst.h"
#include "clb3aitr.h"
#include "clb3astr.h"
#include <string.h>
#include <iostream.h>
#include <iomanip.h>

template <class Item> class IOList {
  public:
    IOList() : list() {}
    void write();
    void read(const char *msg);
    void append(Item item) {
      list.append(item);
    }
  private:
    List<Item> list;
};

template <class Item> void IOList<Item>::write() {
  Iterator<Item> iter(list);
  while (!iter.eol()) {
    cout << ' ' << ++iter;
  }
  cout << endl;
}

template <class Item> void IOList<Item>::read(const char *msg) {
  Item item;
  cout << msg << endl;
  while (cin >> item) {
    list.append(item);
  }
}

ostream& operator<<(ostream& os, String& str) {
  os << str.string() << endl;
  return(os);
}

istream& operator>>(istream& is, String& str) {
  char tmpStr[80];
  cin.width(79);
  is >> tmpStr;
  str.replace(tmpStr);
  return(is);
}
```

Figure 15. z/OS C++ Template Program (Part 1 of 2)

```
int main() {
  IOList<String> stringList;
  IOList<int>    intList;

  char line1[]= "This program will read in a list of";
  char line2[]= "strings, integers and real numbers";
  char line3[]= "and then print them out";

  stringList.append(line1);
  stringList.append(line2);
  stringList.append(line3);
  stringList.write();
  intList.read("Enter some integers (/* to terminate)");
  intList.write();

  String name1 = "Bloe, Joe";
  String name2 = "Jackson, Joseph";

  cout << min(name1, name2) << " comes before "
       << max(name1, name2) << endl;

  int num1 = 23;
  int num2 = 28;

  cout << min(num1, num2) << " comes before "
       << max(num1, num2) << endl;

  return(0);
}
```

Figure 15. z/OS C++ Template Program (Part 2 of 2)

Compiling, Binding, and Running the C++ Template Example

This section describes the commands to compile, bind and run the template example under z/OS batch,TSO, and the z/OS shell.

Under z/OS Batch

To compile, bind, and run the template example program under z/OS batch, follow these steps:

1. Ensure that z/OS Language Environment runtime library and the z/OS C++ compiler are in `STEPLIB`, `LPALST`, or the `LNKLST` concatenation.
2. Use the following JCL to compile, bind, and run the template example. In the example JCL, change `<userhlq>` to your own user prefix.

CBC3UNCL

```
//Jobcard info
//PROC JCLLIB ORDER=(CBC.SCBCPRC,
//   CEE.SCEEPROC)
//*
//* Compile MAIN program, creating an object deck and a TEMPLATE PDS
//* of the source code.  The TEMPLATE PDS of source code will be
//* written to the default TEMPLATE PDS '<userhlq>.TEMPINC'
//*
//MAINCC  EXEC CBCC,          * Compile main program
//        INFILE='CBC.SCLBSAM(CLB3ATMP)',
//        OUTFILE='<userhlq>.SAMPLE.OBJ(CLB3ATMP),DISP=SHR ',
//        CPARM='OPTF(DD:COPTS)'
//COPTS   DD *
  SEARCH('CEE.SCEEH.+','CBC.SCLBH.+')
  LSEARCH('CBC.SCLBSAM.+') TEMPINC
/*
//*
//* Compile PDS of TEMPLATE source code.  Direct template source file
//* creation to this PDS with the TEMPINC option.  Then, if any
//* TEMPLATE compilation creates new members, they will be created
//* in this PDS.  The compiler will detect this and automatically
//* compile the newly created members as part of this step.
//*
//TMPCC   EXEC CBCC,          * Compile PDS of templates
//        INFILE='<userhlq>.TEMPINC',
//        OUTFILE='<userhlq>.TEMPINC.OBJ,DISP=SHR ',
//        CPARM='OPTF(DD:COPTS)'
//COPTS   DD *
  SEARCH('CEE.SCEEH.+','CBC.SCLBH.+')
  LSEARCH('CBC.SCLBSAM.+') TEMPINC
/*
//*
//* Make the PDS of template objects have long named aliases used
//* for autocall by using the EDCLIB utility with the DIR command.
//*
//GENLIB  EXEC EDCLIB,        * Create Template Library
//        OPARM='DIR',
//        LIBRARY='<userhlq>.TEMPINC.OBJ'
//*
//* Bind the program --- specify the template library on the
//* bind autocall library.
//*
//BIND    EXEC CBCB,          * Bind main program
//        INFILE='<userhlq>.SAMPLE.OBJ(CLB3ATMP)',
//        OUTFILE='<userhlq>.SAMPLE.LOAD(CLB3ATMP),DISP=SHR'
//BIND.SYSLIB DD
//             DD
//             DD
//             DD DSN=<userhlq>.TEMPINC.OBJ,DISP=SHR
//GO      EXEC CBCG,
//        INFILE='<userhlq>.SAMPLE.LOAD',
//        GOPGM=CLB3ATMP
//GO.SYSIN DD *
 1 2 5 3 7 8 3 2 10 11
/*
```

Figure 16. JCL to Compile, Bind and Run the Template Example

Under TSO

To compile, bind, and run the example program under TSO, follow these steps:

1. Ensure that z/OS Language Environment runtime library, the z/OS Class Library DLLs, and the z/OS C++ compiler are in `STEPLIB`, `LPALST`, or the `LNKLST` concatenation.
2. Compile the source files:
   a. `CXX 'CBC.SCLBSAM(CLB3ATMP)' (LSEARCH('CBC.SCLBSAM.+') SEARCH('CEE.SCEEH.+','CBC.SCLBH.+') OBJ(SAMPLE.OBJ(CLB3ATMP))`

      Compiles `CLB3ATMP` with the default compiler options, and stores the object module in *userhlq*`.SAMPLE.OBJ(CLB3ATMP)`, where *userhlq* is your user prefix. The template instantiation files are written to the PDS *userhlq*`.TEMPINC`.

   b. `CXX TEMPINC (LSEARCH('CBC.SCLBSAM.+') SEARCH('CEE.SCEEH.+','CBC.SCLBH.+')`

      Compiles the PDS `TEMPINC` and creates the corresponding objects in the PDS *userhlq*`.TEMPINC.OBJ`.

   See "Compiling Under TSO" on page 293 for more information.
3. Create a library from the PDS *userhlq*`.TEMPINC.OBJ`:

   ```
   C370LIB DIR LIB(TEMPINC.OBJ)
   ```

   For more information see "Creating an Object Library Under TSO" on page 412.
4. Bind the program:

   ```
   CXXBIND OBJ(SAMPLE.OBJ(CLB3ATMP)) LIB(TEMPINC.OBJ) LOAD(SAMPLE.LOAD(CLB3ATMP))
   ```

   Binds the *userhlq*`.SAMPLE.OBJ(CLB3ATMP)` text deck using the *userhlq*`.TEMPINC.OBJ` library and the default bind options. This step creates the executable module *userhlq*`.SAMPLE.LOAD(CLB3ATMP)`.

   Note: To avoid a binder error, the dataset *userhlq*`.SAMPLE.LOAD` must be a PDSE.

   For more information see "Binding Under TSO Using CXXBIND" on page 369.
5. Run the program:

   ```
   CALL SAMPLE.LOAD(CLB3ATMP)
   ```

   Executes the module *userhlq*`.SAMPLE.LOAD(CLB3ATMP)` using the default runtime options. For more information see "Running an Application under TSO" on page 404.

Under the z/OS UNIX Shell

To compile, bind, and run the template example program under the z/OS shell, follow these steps:

1. Ensure that z/OS Language Environment runtime library and the z/OS C++ compiler are in `STEPLIB`, `LPALST`, or the `LNKLST` concatenation.
2. Perform a series of `oput` commands for all files that are used, as follows:

   ```
   cp "//'cbc.sclbsam(clb3atmp)'" clb3atmp.C
   cp "//'cbc.sclbsam.h(clb3astr)'" clb3astr.h
   cp "//'cbc.sclbsam.h(clb3aitr)'" clb3aitr.h
   cp "//'cbc.sclbsam.h(clb3amin)'" clb3amin.h
   cp "//'cbc.sclbsam.h(clb3amax)'" clb3amax.h
   cp "//'cbc.sclbsam.h(clb3alst)'" clb3alst.c
   cp "//'cbc.sclbsam.c(clb3aitr)'" clb3aitr.c
   cp "//'cbc.sclbsam.c(clb3alst)'" clb3alst.c
   cp "//'cbc.sclbsam.c(clb3amax)'" clb3amax.c
   cp "//'cbc.sclbsam.c(clb3amin)'" clb3amin.c
   ```

Note: You must use the correct suffixes: `C` for the main source file, `c` for the template definition files, and `h` for all header files.

3. Then, to compile and bind:

   ```
   c++ -o clb3atmp clb3atmp.C
   ```

   This command compiles `clb3atmp.C` and then compiles the `./tempinc` directory (which is created if it doesn't already exist). It then binds using all the objects in the `./tempinc` directory. An archive file, or `C370LIB` object library is not created.

4. Run the program:

   ```
   ./clb3atmp
   ```

Chapter 6. Compiler Options

This chapter describes the options that you can use to alter the compilation of your program.

Specifying Compiler Options

You can override your installation default options when you compile your z/OS C/C++ program, by specifying an option in one of the following ways:

- In the option list when you invoke the IBM-supplied REXX EXECs.
- In the `CPARM` parameter of the IBM-supplied cataloged procedures, when you are compiling under z/OS batch.

  See "Chapter 9. Compiling" on page 283, and "Appendix D. Cataloged Procedures and REXX EXECs" on page 529 for details.
- In your own JCL procedure, by passing a parameter string to the compiler.
- In an options file. See "OPTFILE | NOOPTFILE" on page 152 for details.
- For z/OS C, in a `#pragma options` preprocessor directive within your source file. See "Specifying z/OS C Compiler Options Using #pragma Options" on page 62 for details.

  Compiler options that you specify on the command line or in the `CPARM` parameter of IBM-supplied cataloged procedures can override compiler options that are used in `#pragma options`. The exception is `CSECT`, where the `#pragma csect` directive takes precedence.
- In the utilities `c89`, `cc`, or `c++`, by using the `-Wc` option to pass options to the compiler.
- In the ISPF panels that are used to invoke the z/OS C/C++ compiler in foreground and background.

The following compiler options are inserted in your object module to indicate their status:

AGGRCOPY	
ALIAS	(C compile and IPA Link step only)
ANSIALIAS	(C compile and C++ compile only)
ARCHITECTURE	
ARGPARSE	
COMPACT	
COMPRESS	
CONVLIT	
CSECT	
CVFT	(C++ compile only)
DLL	
EXECOPS	
EXPORTALL	(C compile and C++ compile only)
FLOAT	
GOFF	
GONUMBER	
IGNERRNO	
INITAUTO	
INLINE	(C compile and IPA Link step only)
IPA	

LANGLVL	
LIBANSI	
LOCALE	
LONGNAME	
OPTIMIZE	
PLIST	
REDIR	(C compile and IPA Link step only)
RENT	(C compile and IPA Link step only)
ROCONST	(C and C++ compile only)
ROSTRING	(C and C++ compile only)
ROUND	
SERVICE	
SPILL	
START	
STRICT	
STRICT_INDUCTION	
TARGET	
TEST	
TUNE	
UPCONV	(C compile only)
XPLINK	

IPA Considerations

The following sections explain what you should be aware of if you request Interprocedural Analysis (IPA) through the `IPA` option. Refer to the *z/OS C/C++ Programming Guide* for an overview of IPA before you use the `IPA` compiler option.

IPA Compiles Versus Compiles with IPA Optimization

The `IPA(OBJONLY)` compilation is an intermediate level of optimization. This results in a modified regular compile, not an IPA Compile step. Unlike the IPA Compile step, no IPA information is written to the object file.

During compilation, this step performs the same IPA-specific compile-time optimizations as the IPA Compile step, performs the requested non-IPA optimizations, and then generates optimized object code and data.

The object file may be used by an IPA Link step, a prelink/link, or a bind. If it is used as input to an IPA Link step, no IPA link-time optimizations can be performed for this compilation unit because no IPA information is available.

Applicability of Compiler Options under IPA

You should keep the following points in mind when specifying compiler options for the IPA Compile or IPA Link step:

- Many compiler options do not have any special effect on IPA. For example, the `PPONLY` option processes source code, then terminates processing prior to IPA Compile step analysis.
- Compiler options that affect the way the compiler generates a regular object module have the same effect on how the IPA compile step generates an object module with `IPA (OBJECT)`.
- Compiler options for `IPA(OBJONLY)` compiles are the same as for `NOIPA` compiles.
- In some situations, you must specify a compiler option on the IPA Compile step if you want the benefit of the option on the IPA Link step. In some situations, you must specify the option again on the IPA Link step.

- Some compiler options have special behavior or restrictions other than what is described above.
- `#pragma` directives in your source code, and compiler options you specify for the IPA Compile step, may conflict across compilation units.

  `#pragma` directives in your source code, and compiler options you specify for the IPA Compile step may conflict with options you specify for the IPA Link step.

  IPA will detect such conflicts and apply default resolutions with appropriate diagnostic messages. The Compiler Options Map section of the IPA Link step listing displays the conflicts and their resolutions.

  To avoid problems, use the same options and suboptions on the IPA Compile and IPA Link steps. Also, if you use `#pragma` directives in your source code, specify the corresponding options for the IPA Link step.
- If you specify a compiler option that is irrelevant for a particular IPA step, IPA ignores it and does not issue a message.

In this chapter, the description of each compiler option includes its effect on IPA processing.

Interactions between Compiler Options and IPA Suboptions

During IPA Compile step processing, IPA handles conflicts between `IPA` suboptions and certain compiler options that affect code generation.

If you specify a compiler option for the IPA Compile step, but do not specify the corresponding suboption of the `IPA` option, the compiler option may override the `IPA` suboption. Table 3 shows how the `OPT`, `LIST`, and `GONUMBER` compiler options interact with the `OPT`, `LIST`, and `GONUMBER` suboptions of the `IPA` option. The `xxxx` indicates the name of the option or suboption. `NOxxxx` indicates the corresponding negative option or suboption.

Table 3. Interactions Between Compiler Options and IPA Suboptions

Compiler Option	Corresponding IPA Suboption	Value used in IPA Object
no option specified	no suboption specified	NOxxxx
no option specified	NOxxxx	NOxxxx
no option specified	xxxx	xxxx
NOxxxx	no option specified	NOxxxx
NOxxxx	NOxxxx	NOxxxx
NOxxxx	xxxx	xxxx
xxxx	no option specified	xxxx
xxxx	NOxxxx	xxxx [1]
xxxx	xxxx	xxxx

Note: [1]An informational message is produced that indicates that the suboption `NOxxxx` is promoted to `xxxx`.

Using Special Characters

Under TSO

When HFS file names contain the special characters blank, backslash, and double quote, a backslash (\) must precede these characters.

Two backslashes must precede suboptions that contain these special characters:

left parenthesis (, right parenthesis), comma, backslash, blank, double quote, less than <, and greater than >

For example:

```
def(errno=\\(*__errno\\(\\)\\))
```

Note: Under TSO, a backslash \ must precede special characters in file names and options.

Under the z/OS Shell

The z/OS shell imposes its own parsing rules. The c89 utility escapes special compiler and runtime characters as needed to invoke the compiler, so you need only be concerned with shell parsing rules. See "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555 for more information.

Under z/OS Batch

When invoking the compiler directly (not through a cataloged procedure), you should type a single quote (') within a string as two single quotes (''), as follows:

```
//COMPILE EXEC PGM=CBCDRVR,PARM='OPTFILE(''USERID.OPTS'')'
```

If you are using the same string to pass a parameter to a JCL PROC, use four single quotes (''''), as follows:

```
//COMPILE EXEC CBCC,CPARM='OPTFILE(''''USERID.OPTS'''')'
```

Special characters in HFS file names that are referenced in `DD` cards do not need a preceding backslash. For example, the special character blank in the file name `obj 1.o` does not need a preceding backslash when it is used in a `DD` card:

```
//SYSLIN DD PATH='u/user1/obj 1.o'
```

A backslash must precede special characters in HFS file names that are referenced in the `PARM` statement. The special characters are: backslash, blank, and double quote. For example, a backslash precedes the special character blank in the file name `obj 1.o`, when used in the `PARM` keyword:

```
//STEP1 EXEC PGM=CBCDRVR,PARM='OBJ(/u/user1/obj\ 1.o)'
```

Specifying z/OS C Compiler Options Using #pragma Options

You can use the `#pragma options` preprocessor directive to override the default values for compiler options. Compiler options that are specified on the command line or in the `CPARM` parameter of the IBM-supplied cataloged procedures can override compiler options that are used in `#pragma options`. The exception is `CSECT`, where the `#pragma csect` directive takes precedence. For complete details on the `#pragma options` preprocessor directive, see the *z/OS C/C++ Language Reference*.

The `#pragma options` preprocessor directive must appear before the first z/OS C statement in your input source file. Only comments and other preprocessor directives can precede the `#pragma options` directive. Only the options that are listed below can be specified in a `#pragma options` directive. If you specify a compiler option that is not in the following list, the compiler generates a warning message, and does not use the option.

AGGREGATE	ALIAS
ANSIALIAS	ARCHITECTURE
CHECKOUT	DECK
GONUMBER	HWOPTS[1]

IGNERRNO	INLINE
LIBANSI	MAXMEM
OBJECT	OPTIMIZE
RENT	SERVICE
SPILL	START
TEST	TUNE
XREF	UPCONV

[1]The compiler accepts the `HWOPTS` option, but you should use the `ARCHITECTURE` option instead.

Notes:

1. When you specify conflicting attributes explicitly, or implicitly by the specification of other options, the last explicit option is accepted. The compiler usually does not issue a diagnostic message indicating that it is overriding any options.
2. When you compile your program with the `SOURCE` compiler option, an options list in the listing indicates the options in effect at invocation. The values in the list are the options that are specified on the command line, or the default options that were specified at installation. These values do not reflect options that are specified in the `#pragma options` directive.

Specifying Compiler Options under z/OS UNIX

The `c89` and `c++` utilities specify most compiler options when they call the z/OS C/C++ compiler. Therefore, `#pragma options` and other #pragma directives that are overridden by command line options are not used. For example, if you compile using `c89`, and have `#pragma langlvl (EXTENDED)` in your source, `c89` uses `LANGLVL(ANSI)`. This is because `c89` specifies ANSI explicitly when it calls the compiler.

To change compiler options, use the corresponding `c89` or `c++` option. For example, use `-I` to set the search option that specifies where to search for #include files. If there is no corresponding `c89` or `c++` option, use `-Wc`. For example, specify `-Wc,expo` to export all functions and variables.

For a complete description of `c89`, `c++` and related utilities, refer to "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555 or to the *Z/OS UNIX System Services Command Reference*.

Compiler Option Defaults

You can use various options to change the compilation of your program. You can specify compiler options when you invoke the compiler or, in a C program, in a `#pragma options` directive in your source program. Options that you specify when you invoke the compiler override installation defaults and compiler options that are specified through a `#pragma options` directive.

The compiler option defaults that are supplied by IBM can be changed to other selected defaults when z/OS C/C++ is installed. To find out the current defaults, compile a program with only the `SOURCE` compiler option specified. The compiler listing shows the options that are in effect at invocation. The listing does not reflect options that are specified through a `#pragma options` directive in the source file.

The `c89`, `cc`, and `c++` utilities that run in the z/OS UNIX shell specify certain compile options in order to support POSIX standards. For a complete description of these utilities, refer to "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555 or to the *Z/OS UNIX System Services Command Reference*. For some options, these utilities specify values that are different than the supplied defaults in MVS Batch or TSO environments. However, for many options, they specify the same value as in MVS Batch or TSO. There are some options that the above Utilities do not specify explicitly. In those cases, the default value is the same as in Batch or TSO. An option that you specify explicitly using the above z/OS UNIX utilities override the setting of the same option if it is specified using a `#pragma options` directive. The exception is `CSECT`, where the `#pragma csect` directive takes precedence.

In effect, invoking the compiler with the `c89`, `cc`, and `c++` utilities overrides the default values for many options, compared to running the compiler in MVS Batch or TSO. For example, the `c89` utility specifies the `RENT` option, while the compiler default in MVS batch or TSO is `NORENT`. Any overrides of the defaults by the `c89`, `cc`, or `c++` utilities are noted in the `DEFAULT` category for the option. As the compiler defaults can always be changed during installation, you should always consult the compiler listing to verify the values passed to the compiler. See "Using the z/OS C Compiler Listing" on page 206 and "Using the z/OS C++ Compiler Listing" on page 244 for more information.

The `c89` utilities remap the following options to the values shown. Note that these values are set for a regular (non-IPA) compile. These values will change when if you invoke IPA compile, IPA link, or specify certain other options. For example, specifying the c89 -V option changes the settings of many of the compiler listing options. See "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555 for more information and also refer to the default information provided for each compiler option.

The `c89` options remapped are as follows:

```
LOCALE(POSIX)
LANGLVL(ANSI)
OE
LONGNAME
RENT
OBJECT(file_name.o)
NOGENPCH(./)
NOUSEPCH(./)
NOLIST(/dev/fd1)
NOSOURCE(/dev/fd1)
NOPPONLY(NOCOMMENTS,NOLINES,/dev/fd1,2048)
FLAG(W)
DEFINE(errno=\\(*__errno\\(\\)\\))
DEFINE(_OPEN_DEFAULT=1)
```

The `cc` options remapped are as follows:

```
NOANSIALIAS
LOCALE(POSIX)
LANGLVL(COMMONC)
OE
LONGNAME
RENT
OBJECT(file_name.o)
NOGENPCH(./)
NOUSEPCH(./)
NOLIST(/dev/fd1)
NOSOURCE(/dev/fd1)
NOPPONLY(NOCOMMENTS,NOLINES,/dev/fd1,2048)
```

```
FLAG(W)
DEFINE(errno=\\(*__errno\\(\\)\\))
DEFINE(_OPEN_DEFAULT=0)
DEFINE(_NO_PROTO=1)
```

The c++ options remapped are as follows:

```
LOCALE(POSIX)
LANGLVL(ANSI)
OE
OBJECT(file_name.o)
NOGENPCH(./)
NOUSEPCH(./)
NOINLRPT(/dev/fd1)
NOLIST(/dev/fd1)
NOSOURCE(/dev/fd1)
NOPPONLY(NOCOMMENTS,NOLINES,/dev/fd1,2048)
FLAG(W)
DEFINE(errno=\\(*__errno\\(\\)\\))
DEFINE(_OPEN_DEFAULT=1)
```

Note that the `locale` option is set according to the environment when the `cc`, `c89`, and `c++` utilities are invoked. The current execution locale is determined by the values associated with environment variables `LANG` and `LC_ALL`. The following list shows the order of precedence for determining the current execution locale:

- If you specify `LC_ALL`, the current execution locale will be the value associated with `LC_ALL`.
- If `LC_ALL` was not specified but `LANG` was specified, the current execution locale will be the value associated with `LANG`.
- If neither of the two environment variables is specified, the current execution locale will default to ″C″.
- If the current execution locale is ″C″, the compiler will be invoked with `LOCALE(POSIX)` otherwise it will be invoked with the current execution locale.

Note that for `SEARCH`, the *value* is determined by the following:

- Additional include search directories identified by the `c89 -I` options. Refer to "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555 for more information.
- z/OS UNIX environment variable settings: `{_INCDIRS}`, `{_INCLIBS}`, and `{_CSYSLIB}`. They are normally set during compiler installation to reflect the compiler and runtime include libraries. Refer to "Environment Variables" on page 567 for more information.

Refer to "SEARCH | NOSEARCH" on page 167 for more information on `SEARCH`.

For the remainder of the compiler options, the `c89, cc,`or `c++` utilities default matches the `C/C++` default. Some of these are explicitly specified by `c89, cc,`or `c++`. Therefore if the installation changes the default options, you may find that `c89, cc,`or `c++` continues to use the default options. You can use the `{_OPTIONS}` (`_C89_OPTIONS, _CC_OPTIONS, _CXX_OPTIONS`) environment variable to override these settings if necessary. Note that certain options are required for the correct execution of `c89, cc,`or `c++`.

Summary of Compiler Options

Most compiler options have a positive and negative form. The negative form is the positive with `NO` before it. For example, `NOXREF` is the negative form of `XREF`. Table 4 lists the compiler options in alphabetical order, their abbreviations, and the defaults that are supplied by IBM. Suboptions inside square brackets are optional. Table 5 on page 69 lists options that are retained for compatibility with previous versions of the compiler. Use these options only in existing code. For each of these options, there is a replacement option in Table 4 that you should use for new programs.

The C, C++, and IPA Link columns, which are shown in the table below, indicate where the option is accepted by the compiler but this acceptance does not necessarily cause an action; for example, IPA LINK accepts the `MARGINS` option but ignores it. ″C″ refers to a C language compile step. ″C++″ refers to a C++ language compile step. These options are accepted regardless of whether these are for NOIPA, IPA (OBJONLY), or IPA(NOLINK).

Table 4. Compiler Options, Abbreviations, and IBM Supplied Defaults

Compiler Option (Abbreviated Names are underlined)	IBM Supplied Default	C	C++	IPA Link	More Information
AGGRCOPY (OVERLAP | NOOVERLAP)	AGGRC(NOOVERL)	✓	✓	✓	See 77
AGGREGATE |NOAGGREGATE	NOAGG	✓		✓	See 78
ALIAS[(name)] | NOALIAS	NOALI	✓			See 79
ANSIALIAS | NOANSIALIAS	ANS	✓	✓	✓	See 80
ARCHITECTURE(*n*)	TARGET(OSV2R10 and above): ARCH(2) TARGET(OSV2R9 and below): ARCH(0)	✓	✓	✓	See 81
ARGPARSE| NOARGPARSE	ARG	✓	✓	✓	See 83
ATTRIBUTE[(FULL)] | NOATTRIBUTE	ATT		✓	✓	See 84
CHECKOUT(subopts) | NOCHECKOUT	NOCHE	✓		✓	See 85
COMPACT | NOCOMPACT	NOCOMPACT	✓	✓	✓	See 88
COMPRESS | NOCOMPRESS	NOCOMPRESS	✓	✓	✓	See 89
CONVLIT | NOCONVLIT	NOCONV (, WCHAR)	✓	✓	✓	See 90
CSECT | NOCSECT	NOCSE	✓	✓	✓	See 92
CVFT | NOCVFT	CVFT		✓		See 95
DEFINE(name1[= | =def1], name2[= | =def2],...)	no default user definitions	✓	✓	✓	See 96
DIGRAPH | NODIGRAPH	NODIGR	✓	✓	✓	See 98
DLL(CBA | NOCBA) | NODLL(CBA |NCBA)	NODLL(NOCBA)	✓		✓	See 99
DLL(CBA | NOCBA)	DLL(NOCBA)		✓	✓	See 99
EVENTS[(filename)] | NOEVENTS	NOEVENT	✓	✓	✓	See 101
EXECOPS | NOEXECOPS	EXEC	✓	✓	✓	See 102
EXH | NOEXH	EXH		✓		See 103
EXPMAC | NOEXPMAC	NOEXP	✓	✓	✓	See 104
EXPORTALL | NOEXPORTALL	NOEXPO	✓	✓	✓	See 105
FASTTEMPINC | NOFASTTEMPINC	NOFASTT		✓		See 106
FLAG(severity) | NOFLAG	FL(I)	✓	✓	✓	See 107

Table 4. Compiler Options, Abbreviations, and IBM Supplied Defaults (continued)

Compiler Option (Abbreviated Names are underlined)	IBM Supplied Default	C	C++	IPA Link	More Information
FLOAT(subopts)	FLOAT(HEX, FOLD, NOMAF, NORRM, NOAFP)	✓	✓	✓	See 108
GENPCH[(filename)] | NOGENPCH[(filename)]	NOGENP	✓	✓	✓	See 112
GOFF | NOGOFF	NOGOFF	✓	✓	✓	See 114
GONUMBER | NOGONUMBER	NOGONUM	✓	✓	✓	See 115
HALT(num)	HALT(16)	✓	✓	✓	See 116
IGNERRNO | NOIGNERRNO	NOIGNER	✓	✓	✓	See 117
INFO[(subopts)] | NOINFO	NOIN		✓		See 118
INITAUTO | NOINITAUTO	NOINITA	✓	✓	✓	See 119
INLINE(subopts) | NOINLINE [(subopts)]	NOOPT: NOINL(AUTO, NOREPORT, 100, 1000) OPT: INL(AUTO, NOREPORT, 100, 1000)	✓		✓	See 120
INLRPT[(filename)] | NOINLRPT[(filename)]	NOINLR	✓	✓	✓	See 124
IPA[(subopts)] | NOIPA[(subopts)]	NOIPA(NOLINK, OBJECT, OPT, NOLIST, NOGONUMBER, NOATTRIBUTE, NOXREF, LEVEL(1),NOMAP, DUP, ER, NONCAL, NOUPCASE, NOCONTROL)	✓	✓	✓	See 125
LANGLVL(ANSI|SAA |SAAL2|COMPAT|EXTENDED|COMMONC)	LANG(EXTENDED)	✓	✓	✓	See 130
LIBANSI | NOLIBANSI	NOLIB	✓	✓	✓	See 132
LIST[(filename)] | NOLIST [(filename)]	NOLIS	✓	✓	✓	See 133
LOCALE[(name)] | NOLOCALE	NOLOC	✓	✓	✓	See 135
LONGNAME | NOLONGNAME	C:NOLO C++: LO	✓	✓	✓	See 137
LSEARCH(subopts) | NOLSEARCH	NOLSE	✓	✓	✓	See 138
MARGINS | NOMARGINS	NOMAR		✓		See 143
MARGINS(m,n) | NOMARGINS	V-format: NOMAR F-format: MAR(1,72)	✓		✓	See 143
MAXMEM | NOMAXMEM	MAXM(2097152)	✓	✓	✓	See 145
MEMORY | NOMEMORY	MEM	✓	✓	✓	See 146
NESTINC(num) | NONESTINC	NONEST	✓	✓	✓	See 147
OBJECT[(filename)] | NOOBJECT [(filename)]	OBJ	✓	✓	✓	See 148
OE[(filename)] | NOOE[(filename)]	NOOE	✓	✓	✓	See 150
OFFSET | NOOFFSET	NOOF	✓	✓	✓	See 151
OPTFILE[(filename)] | NOOPTFILE[(filename)]	NOOPTF	✓	✓	✓	See 152
OPTIMIZE[(level)] | NOOPTIMIZE	NOOPT	✓	✓	✓	See 154
PHASEID | NOPHASEID	NOPHASEID	✓	✓	✓	See 157
PLIST(HOST | OS)	PLIST(HOST)	✓	✓	✓	See 157

Table 4. Compiler Options, Abbreviations, and IBM Supplied Defaults (continued)

Compiler Option (Abbreviated Names are underlined)	IBM Supplied Default	C	C++	IPA Link	More Information
PORT(PPS | NOPPS) | NOPORT(PPS | NOPPS)	NOPORT(NOPPS)		✓		See 158
PPONLY[(subopts)] | NOPPONLY [(subopts)]	NOPP	✓	✓	✓	See 160
REDIR | NOREDIR	RED	✓	✓	✓	See 162
RENT| NORENT	NORENT	✓		✓	See 163
ROCONST | NOROCONST	NOROC	✓	✓	✓	See 164
ROSTRING | NOROSTRING	NOROS	✓	✓	✓	See 165
ROUND(opt)	ROUND(N)	✓	✓	✓	See 166
SEARCH(opt1,opt2,...) | NOSEARCH	For C++, SE(//'CEE.SCEEH.+, //'CBC>SCLBH.+') For C, SE(//'CEE.SCEEH.+')	✓	✓	✓	See 167
SERVICE(string) | NOSERVICE	NOSERV	✓	✓	✓	See 168
SEQUENCE | NOSEQUENCE	NOSEQ		✓		See 170
SEQUENCE(m,n) | NOSEQUENCE	V-format: NOSEQ F-format: SEQ(73,80)	✓		✓	See 170
SHOWINC | NOSHOWINC	NOSHOW	✓	✓	✓	See 171
SOURCE[(filename)] | NOSOURCE[(filename)]	NOSO	✓	✓	✓	See 172
SPILL(size) | NOSPILL[(size)]	SPILL(128)	✓	✓	✓	See 173
SRCMSG | NOSRCMSG	NOSRCM		✓		See 174
SSCOMM | NOSSCOMM	NOSS	✓		✓	See 175
START | NOSTART	STA	✓	✓	✓	See 176
STRICT | NOSTRICT	STRICT	✓	✓	✓	See 177
STRICT_INDUCTION | NOSTRICT_INDUCTION	NOSTRICT_INDUC	✓	✓	✓	See 178
TARGET(suboption)	TARG(LE, CURRENT)	✓	✓	✓	See 179
TEMPINC[(filename)] | NOTEMPINC[(filename)]	For PDS: TEMP(TEMPINC) For HFS: TEMP(./tempinc)		✓		See 185
TERMINAL | NOTERMINAL	TERM	✓	✓	✓	See 186
TEST[(subopts)] | NOTEST[(subopts)]	C: NOTEST (HOOK, SYM, BLOCK, LINE, PATH) C++: NOTEST(HOOK)	✓	✓	✓	See 186
TUNE(n)	TUN(3)	✓	✓	✓	See 190
UNDEFINE(name)	no default	✓	✓	✓	See 192
UPCONV | NOUPCONV	NOUPC	✓		✓	See 193
USEPCH[(filename)] | NOUSEPCH[(filename)]	NOUSEP	✓	✓	✓	See 194
WSIZEOF| NOWSIZEOF	NOWSIZEOF	✓	✓	✓	See 195
XPLINK | NOXPLINK	NOXPL	✓	✓		See 196
XREF | NOXREF	NOXR	✓	✓	✓	See 198

Compatibility Options

Table 5 lists options that are compatible with previous versions of the compiler. Use these options only if they already exist in your code. For new programs, use the replacement options that are listed for each of the compatibility options.

Note: Some parameters such as the output data set may differ between the old option and its replacement option. Read the description of the replacement option before you use it.

Table 5. Compatibility Compiler Options, Abbreviations, and IBM Supplied Defaults

Compiler Option (Abbreviated names are underlined)	IBM Supplied Default	C	C++	IPA Link	Replacement Option	More Information
DECK | NODECK	NODECK	✓		✓	OBJECT	See 199 and 148
HWOPTS(STRING | NOSTRING) | NOHWOPTS	NOHWO	✓		✓	ARCH	See 200 and 81
OMVS[(filename)] | NOOMVS	NOOMVS	✓		✓	OE	See 201and 150
SYSLIB(pdsname-list)| NOSYSL	NOSYSL	✓	✓	✓	SEARCH	See 167 and 202
SYSPATH[(path1,path2,...)]| NOSYSPATH	NOSYS		✓		SEARCH	See 167 and 203
USERLIB(pdsname-list)| NOUSERL	NOUSERL	✓	✓	✓	LSEARCH	See 138 and 204
USERPATH[(path1,path2,...)] | NOUSERPATH	NOUSER		✓		LSEARCH	See 138 and 205

Compiler Options for File Management

These options specify the data set or HFS directory where the compiler stores output files, and direct the compiler's search for include files.

Table 6. Compiler Options for File Management

Option	Description	C Compile	C++ Compile	IPA Link	More Information
DECK	Produces an object module, and stores it in the data set associated with `SYSPUNCH`. **Use OBJECT instead of DECK.**	✓		✓	See 199 and 148
FASTTEMPINC	Defers generating object code until the final version of all template definitions have been determined. Then, a single compilation pass generates the final object code, resulting in improved compilation time when recursive templates are used in an application.		✓		See 106
GENPCH	Generates precompiled header files.	✓	✓	✓	See 112
IPA(CONTROL)	Indicates the name of the control file that contains additional directives for the IPA Link step. This option only affects the IPA Link step.	✓	✓	✓	See 129
LSEARCH	Specifies the libraries or disks to be scanned for user include files.	✓	✓	✓	See 138
MEMORY	Improves compile-time performance by using a MEMORY file in place of a work file, if possible.	✓	✓	✓	See 146
OBJECT	Produces an object module, and stores it in the file that you specify, or in the data set associated with `SYSLIN`.	✓	✓	✓	See 148

Table 6. Compiler Options for File Management (continued)

Option	Description	C Compile	C++ Compile	IPA Link	More Information
OE	Specifies that file names used in compiler options and include directives should be interpreted as HFS file names when the file name provided is ambiguous. Also specifies that POSIX.2 standard rules for include file searching should be used.	✓	✓	✓	See 150
OMVS	**The options OMVS and OE perform the same function. Use the OE option instead of the OMVS option.** Specifies that file names used in compiler options and include directives should be interpreted as HFS file names when the file name provided is ambiguous. Also specifies that POSIX.2 standard rules for include file searching should be used.	✓		✓	See 201 and 150
OPTFILE	Directs the compiler to look for compiler options in the file specified.	✓	✓	✓	See 152
SEARCH	Specifies the libraries or disks to be scanned for system include files.	✓	✓	✓	See 167
SYSLIB	Specifies the PDSs to be scanned for system include files. **Use SEARCH instead of SYSLIB**.	✓	✓	✓	See 167 and 202
SYSPATH	Specifies search paths to be scanned for system include files. **Use SEARCH instead of SYSPATH.**		✓		See 167 and 203
TEMPINC	Places template instantiation files in the PDS or HFS directory specified.		✓		See 185
USEPCH	Instructs the compiler to use precompiled header files.	✓	✓	✓	See 194
USERLIB	Specifies the PDSs to be scanned for your own include files. **Use LSEARCH instead of USERLIB.**	✓	✓	✓	See 138 and 204
USERPATH	Specifies search paths to be scanned for your own include files. **Use LEARCH instead of USERPATH.**		✓		See 138 and 205

Options That Control the Preprocessor

These options specify how the preprocessor runs.

Table 7. Summary of Compiler Options for Preprocessor

Option	Description	C Compile	C++ Compile	IPA Link	More Information
CONVLIT	Turns on string literal codepage conversion.	✓	✓	✓	See 90
DEFINE	Defines preprocessor macro names.	✓	✓	✓	See 96
DIGRAPH	Allows you to use additional digraphs in both C and C++ applications.	✓	✓	✓	See 98
LOCALE	Specifies the locale to be used at compile time.	✓	✓	✓	See 135
PPONLY	Specifies that only the preprocessor is to be run and not the compiler.	✓	✓	✓	See 160

Table 7. Summary of Compiler Options for Preprocessor (continued)

Option	Description	C Compile	C++ Compile	IPA Link	More Information
UNDEFINE	Removes any value its argument may have.	✓	✓	✓	See 192

Options That Control the Source Code

These options allow you to control your z/OS C/C++ source code.

Table 8. Summary of Compiler Options Used for Source Code Control

Option	Description	C Compile	C++ Compile	IPA Link	More Information
HALT	Specifies that the compiler stop processing files when it returns an error severity level of *n* or above.	✓	✓	✓	See 116
LANGLVL	Specifies the language standard to be used.	✓	✓	✓	See 130
MARGINS	Identifies position of source to be scanned by the compiler.	✓	✓	✓	See 143
NESTINC	Specifies the number of nested include files to be allowed.	✓	✓	✓	See 147
SEQUENCE	Specifies the columns used for sequence numbers.	✓	✓	✓	See 170
SSCOMM	Allows comments to be specified by two slashes (//). The IPA Link step accepts but ignores this option.	✓		✓	See 175
UPCONV	Preserves *unsignedness* during z/OS C/C++ type conversions. The IPA Link step accepts but ignores this option.	✓		✓	See 193

Options That Control the Compiler Listing

These options control whether the compiler produces a listing, and the kind of information that goes into the listing.

Table 9. Compiler Options That Control Listings

Option	Description	C Compile	C++ Compile	IPA Link	More Information
AGGREGATE	Lists structures and unions, and their sizes. The IPA Link step accepts but ignores this option.	✓		✓	See 78
ATTRIBUTE	For C++ compile, generates a cross reference section showing attributes for each symbol and External Symbol Cross Reference section. For IPA Link, it also generates the Storage Offset Listing if IPA objects were created using the C compiler with `XREF`, `IPA(ATTR)`, or `IPA(XREF)` options and the symbols for the current partition were not coalesced.		✓	✓	See 84
EXPMAC	Lists all expanded macros. You must use the `SOURCE` option with `EXPMAC`.	✓	✓	✓	See 104
INLINE(,REPORT,,)	Generates a report on the status of inlined functions.	✓		✓	See 120

Table 9. Compiler Options That Control Listings (continued)

Option	Description	C Compile	C++ Compile	IPA Link	More Information
INLRPT	Generates a report on the status of inlined functions.	✓	✓	✓	See 124
IPA(MAP)	Generates the following listing sections for the IPA Link step: Object File Map, Source File Map, Compiler Options Map, Global Symbols Map, Partition Map. This option only affects the IPA Link step.	✓	✓	✓	See 125
LIST	Includes the object module in the compiler listing, in assembler-like code.	✓	✓	✓	See 133
OFFSET	Lists offset addresses relative to entry points of functions. The `LIST` option must be used with `OFFSET`.	✓	✓	✓	See 151
SHOWINC	Lists include files if `SOURCE` option specified.	✓	✓	✓	See 171
SOURCE	Lists source file.	✓	✓	✓	See 172
XREF	For C/C++, generates a cross reference listing showing file/line definition, reference and modification information for each symbol. Also generates the External Symbol Cross Reference and Static Map. If you specify the `XREF` option for the IPA Link step, it generates an External Symbol Cross Reference listing section for each partition and Static Map. The IPA Link step creates a Storage Offset listing section if you created your IPA objects with the C compiler and the `XREF`, `IPA(ATTR)`, or `IPA(XREF)` option, and if IPA did not coalesce the symbols for the current partition.	✓	✓	✓	See 198

Options That Control the IPA Object

These options control the content of the IPA object that is produced by the IPA Compile step.

Table 10. Compiler Options for IPA Object Control

Option	Description	C Compile	C++ Compile	IPA Link	More Information
IPA(ATTRIBUTE)	Saves information about symbol storage offsets in the IPA object file.	✓	✓	✓	See 126
IPA(GONUMBER)	Saves source line numbers in the IPA object file without generating line number tables. This option can only be specified for the IPA Compile step, if a combined conventional/IPA object file is requested.	✓	✓	✓	See 126
IPA(LIST)	Saves source line numbers in the IPA object file without generating a Pseudo Assembly listing. This option can only be specified for the IPA Compile step, if a combined conventional/IPA object file is requested.	✓	✓	✓	See 126

Table 10. Compiler Options for IPA Object Control (continued)

Option	Description	C Compile	C++ Compile	IPA Link	More Information
IPA(OBJECT)	Indicates whether a conventional (non-IPA)/IPA object is to be produced during the IPA Compile step.	✓	✓	✓	See 126
IPA(OPTIMIZE)	Generates information in the IPA object file that the compiler option `OPT` needs during IPA Link processing. `IPA(OPTIMIZE)` is the default setting. If you specify `IPA(NOOPTIMIZE)`, IPA will change the option to `IPA(OPTIMIZE)` and issue an informational message.	✓	✓	✓	See 126
IPA(XREF)	Saves information about symbol storage offsets in the IPA object file.	✓	✓	✓	See 126

Options That Control the IPA Link Step

These options control the IPA Link step.

Table 11. Compiler Options for IPA Link Control

Option	Description	C Compile	C++ Compile	IPA Link	More Information
IPA (0|1|2)	Indicates the level of IPA optimization that the IPA Link step should perform after it links the object files into the call graph.	✓	✓	✓	See 129
IPA(LINK)	Instructs the compiler to perform IPA Link processing.	✓	✓	✓	See 129
IPA(NCAL)	Indicates whether library searches are performed during the IPA Link step to locate an object file or files that satisfy unresolved symbol references within the current set of object information. This suboption controls both explicit searches triggered by the LIBRARY IPA Link control statement, and the implicit SYSLIB search that occurs at the end of IPA Link step input processing.	✓	✓	✓	See 129
IPA(UPCASE)	Determines whether an additional automatic library call pass is made for SYSLIB if unresolved references remain at the end of standard IPA Link step processing. Symbol matching is not case-sensitive in this pass.	✓	✓	✓	See 129

Options for Debugging and Diagnosing Errors

These options help you to detect and correct errors in your z/OS C/C++ program.

Table 12. Compiler Options for Debugging and Diagnostics

Option	Description	C Compile	C++ Compile	IPA Link	More Information
CHECKOUT	Gives informational messages for possible programming errors. The IPA Link step accepts but ignores this option.	✓		✓	See 85

Table 12. Compiler Options for Debugging and Diagnostics (continued)

Option	Description	C Compile	C++ Compile	IPA Link	More Information
EVENTS	Produces an events file that contains error information and source file statistics. The IPA Link step accepts but ignores this option.	✓	✓	✓	See 101
FLAG	Specifies the lowest severity level to be listed.	✓	✓	✓	See 107
GONUMBER	Generates line number tables for Debug Tool and error trace backs. The `TEST` option turns on `GONUMBER`.	✓	✓	✓	See 115
INFO	Generates informational messages.		✓		See 118
IPA(DUP)	Indicates whether a message and a list of duplicate symbols are written to the console during the IPA Link step. This option only affects the IPA Link step.	✓	✓	✓	See 125
IPA(ER)	Indicates whether a message and a list of unresolved symbols are written to the console during the IPA Link step. This option only affects the IPA Link step.	✓	✓	✓	See 125
PHASEID	Causes each compiler module (phase) to issue an informational message which identifies the compiler phase module name, product identifier, and build level.	✓	✓	✓	See 157
SERVICE	Places a string in the object module, which is displayed in the traceback if the application fails abnormally.	✓	✓	✓	See 168
SRCMSG	Adds source code lines to diagnostic messages.		✓		See 174
TERMINAL	Directs diagnostic messages to be displayed on the terminal.	✓	✓	✓	See 186
TEST	Generates information that the Debug Tool needs to debug your program.	✓	✓	✓	See 186
XPLINK (BACKCHAIN)	Generates a prolog that saves information about the calling function in the called function's stack frame. This facilitates debugging using storage dumps, at a cost in execution time.	✓	✓	✓	See 196
XPLINK (STOREARGS)	Generates code to store arguments that are normally passed in registers, into the argument area. This facilitates debugging using storage dumps, at a cost in execution time.	✓	✓	✓	See 196

Options That Control Object Code Generation

These options are used to control how your z/OS C/C++ object code is produced.

Table 13. Summary of Compiler Options Used for Object Code Control

Option	Description	C Compile	C++ Compile	IPA Link	More Information
AGGRCOPY	Instructs the compiler whether the source and destination in structure assignments can overlap. (They cannot overlap according to ANSI Standard C rules.)	✓	✓	✓	See 77

Table 13. Summary of Compiler Options Used for Object Code Control (continued)

Option	Description	C Compile	C++ Compile	IPA Link	More Information
ALIAS	Generates ALIAS binder control statements for each required entry point.	✓			See 79
ANSIALIAS	Specifies whether type-based aliasing is to be used during optimization.	✓	✓	✓	See 80
ARCHITECTURE	Specifies the architecture for which the executable program instructions are to be generated.	✓	✓	✓	See 81
CSECT	Instructs the compiler to generate csect names in the output object module.	✓	✓	✓	See 92
COMPACT	Controls choices made between those optimizations which tend to result in faster but larger code and those which tend to result in smaller but slower code.	✓	✓	✓	See 88
COMPRESS	Suppresses the generation of function names in the function control block, thereby reducing the size of your application's load module.	✓	✓	✓	See 89
CVFT	Shrinks the size of the writeable static area (WSA) and reduces the size of construction virtual function tables (CVFT), which in turn reduces the load module size.		✓		See 95
DLL	Generates object code for DLLs or DLL applications.	✓	✓	✓	See 99
EXH	Controls the generation of C++ exception handling code.		✓		See 103
EXPORTALL	Exports all externally defined functions and variables.	✓	✓	✓	See 105
FLOAT	Switches floating-point representation between IEEE and hexadecimal.	✓	✓	✓	See 108
GOFF	Instructs the compiler to produce an object file in the Generalized Object File Format.	✓	✓	✓	See 114
HWOPTS	Generates code for different hardware features. Use ARCH instead of this option.	✓		✓	See 200
IGNERRNO	Informs the compiler that your application is not using errno, allowing the compiler to explore additional optimization opportunities for library functions in LIBANSI.	✓	✓	✓	See 117
INITAUTO	Directs the compiler to generate code to initialize automatic variables. Automatic variables require storage only while the function in which they are declared are active.	✓	✓	✓	See 119
INLINE	Inlines user functions into source and helps maximize optimization.	✓		✓	See 120
IPA	Instructs the compiler to perform Interprocedural Analysis (IPA) processing.	✓	✓	✓	See 125
IPA(LEVEL)	Indicates the level of IPA optimization that the IPA Link step should perform.	✓	✓	✓	See 125

Table 13. Summary of Compiler Options Used for Object Code Control *(continued)*

Option	Description	C Compile	C++ Compile	IPA Link	More Information
LIBANSI	Indicates whether functions with the name of an ANSI C library function are in fact ANSI C library functions.	✓	✓	✓	See 132
LONGNAME	Provides support for external names of mixed case and up to 1024 characters long.	✓	✓	✓	See 137
MAXMEM	Limits the amount of memory used for local tables of specific, memory intensive optimization.	✓	✓	✓	See 145
OBJECT	Produces an object module, and stores it in the file that you specify, or in the data set associated with `SYSLIN`.	✓	✓	✓	See 148
OPTIMIZE	Improves runtime performance by introducing optimizations during code generation.	✓	✓	✓	See 154
RENT	Generates reentrant code. The IPA Link step accepts but ignores this option.	✓		✓	See 163
ROCONST	Informs the compiler that the `const` qualifier is respected by the program so that variables defined with the `const` keyword are not be overridden (for example, by a casting operation).	✓	✓	✓	See 164
ROSTRING	Informs the compiler that string literals are read-only.	✓	✓	✓	See 165
ROUND	Sets the rounding mode for binary floating point numbers.	✓	✓	✓	See 166
SPILL	Specifies the size of the spill area to be used for compilation.	✓	✓	✓	See 173
START	Generates a `CEESTART` whenever necessary.	✓	✓	✓	See 176
STRICT	Affects the precision of floating point calculations	✓	✓	✓	See 177
STRICT_INDUCTION	Instructs the compiler to disable loop induction variable optimizations.	✓	✓	✓	See 178
TARGET	Generates an object module for the targeted operating system or runtime library.	✓	✓	✓	See 179
TUNE	Specifies the architecture for which the executable program will be optimized.	✓	✓	✓	See 190
WSIZEOF	Causes the size of operator to return the widened size for function return types	✓	✓	✓	See 195
XPLINK	Instructs the compiler to generate extra performance linkage for function calls.	✓	✓		See 196

Options That Control Program Execution

These options control the execution of your program

Table 14. Summary of Compiler Options for Program Execution

Option	Description	C Compile	C++ Compile	IPA Link	More Information
ARGPARSE	Parses arguments provided on the invocation line.	✓	✓	✓	See 83
EXECOPS	Allows you to specify runtime options on the invocation line.	✓	✓	✓	See 102
PLIST	Specifies that the original operating system parameter list should be available.	✓	✓	✓	See 157
REDIR	Allows redirection of `stderr`, `stdin`, and `stdout` from the invocation line.	✓	✓	✓	See 162

Portability Options

These options allow you to port your C++ code to the z/OS C++ compiler.

Table 15. Summary of Compiler Options for Portability

Option	Description	C Compile	C++ Compile	IPA Link	More Information
PORT	Adjusts the error recovery action that the compiler takes when it encounters an ill-formed `#pragma pack` directive.		✓		See 158

Description of Compiler Options

The following sections describe the compiler options and their usage. Compiler options are listed alphabetically. Syntax diagrams show the abbreviated forms of the compiler options.

AGGRCOPY

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`AGGRCOPY(NOOVERLAP)`						

CATEGORY: Object Code Control

```
►►──AGGRC──┬─OVERL───┬──────────────────────────────────────────────►◄
           └─NOOVERL─┘
```

The AGGRCOPY option instructs the compiler whether the source and destination assignments for structures can overlap in accordance with ANSI Standard C rules. For example, in the assignment `a = b;`, where a and b are structs, a is the destination and b is the source.

In the case of structure assignments, the compiler can generate faster code if no overlap is assumed. The OVERLAP suboption specifies that the source and destination in a structure assignment might overlap. The NOOVERLAP option specifies that they do not, and that the compiler can assume this when generating code.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. The AGGRCOPY option affects the regular object module if you requested one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

The IPA Link step accepts the AGGRCOPY option, but ignores it.

The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition. Only compatible subprograms are included in a given partition.

The value of the AGGRCOPY option for a partition is set to the value of the first subprogram that is placed in the partition. During IPA inlining, subprograms with different AGGRCOPY settings may be combined in the same partition. When this occurs, the resulting partition is always set to AGGRCOPY(OVERLAP).

AGGREGATE | NOAGGREGATE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOAGGREGATE	NOAGGREGATE -V sets AGGREGATE	NOAGGREGATE -V sets AGGREGATE				

CATEGORY: Listing

```
►►──┬─AGG───┬──────────────────────────────────────────────────────►◄
    └─NOAGG─┘
```

The AGGREGATE option instructs the compiler to include a layout of all struct or union type variables in the compiler listing. Depending on the struct or union declaration, the maps are generated as follows:

- If the `struct` or `union` declaration has a tag, two maps are created: one contains the packed layout, and the other contains the unpacked layout. Each layout map contains the offsets and lengths of the structure members and the union members.
- If the `struct` or `union` declaration does not have a tag, one map is generated for the variable name that is specified on the `struct` or `union` declaration.

You can specify this option using the `#pragma option` directive for C.

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c89` or `cc` commands.

Effect on IPA Compile Step

The `AGGREGATE` option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the `AGGREGATE` option, but ignores it.

ALIAS | NOALIAS

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓			✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOALIAS`	`NOALIAS`	`NOALIAS`				

CATEGORY: Object Code Control

```
►►──┬─ALI──┬─────────┬─┬──────────────────────────────────────►◄
    │      ├─()──────┤ │
    │      └─(name)──┘ │
    └─NOALI────────────┘
```

The `ALIAS` option generates `ALIAS` control statements that help the binder locate modules in a load library. The suboption *name* is assigned to the `NAME` control statement.

`ALIAS`*(name)* If you specify `ALIAS`*(name)*, the compiler generates the following:

- Control statements in the object module.
- A `NAME` control statement in the form `NAME` *name* `(R)`. `R` indicates that the binder should replace the member in the library with the new member.

The compiler generates one `ALIAS` control statement for every external entry point that it encounters during compilation. These control statements are then appended to the object module.

`ALIAS`	If you specify `ALIAS` with no suboption, the compiler selects an existing CSECT name from the program, and nominates it on the `NAME` statement.
`ALIAS()`	If you use an empty set of parentheses, `ALIAS()`, or specify `NOALIAS`, the compiler does not generate a `NAME` control statement.
`NOALIAS`	If you specify `NOALIAS`, the compiler does not generate a `NAME` control statement. `NOALIAS` has the same effect as `ALIAS()`.

If you specify the `ALIAS` option with `LONGNAME`, the compiler does not generate an `ALIAS` control statement.

For complete details on `ALIAS` and `NAME` control statements, see *z/OS DFSMS Program Management*.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

If you specify the `ALIAS` option on the IPA Compile step, the `ALIAS` option is ignored.

Effect on IPA Link Step

If you specify the `ALIAS` option on the IPA Link step, the IPA Link step generates an unrecoverable error.

ANSIALIAS | NOANSIALIAS

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`ANSIALIAS`	`ANSIALIAS`	`NOANSIALIAS`	`ANSIALIAS`			

CATEGORY: Source Code Analysis

```
►►──┬─ANS───┬──────────────────────────────────────────────────────────►◄
    └─NOANS─┘
```

The `ANSIALIAS` option specifies whether type-based aliasing is to be used during optimization. That is, the optimizer assumes that pointers can only be used to access objects of the same type. Type-based aliasing improves optimization.

The following are not subject to type-based aliasing:

- Types that differ only in reference to whether they are signed or unsigned. For example, a pointer to a *signed int* can point to an *unsigned int*.
- Character pointer types can point to any type.

- Types that differ only in their *const* or *volatile* qualification. For example, a pointer to a *const int* can point to an *int*.

If you specify `NOANSIALIAS`, the optimizer makes worst-case aliasing assumptions. It assumes that a given pointer of a given type can point to an external object or any object whose address is taken, regardless of type.

For more information on type-based aliasing and an example that illustrates the ANSI aliasing rule, refer to *z/OS C/C++ Language Reference*.

Notes:

1. This option only takes effect if the `OPTIMIZE` option is in effect.
2. If you specify `LANGLVL(COMMONC)`, the `ANSIALIAS` option is automatically turned off. If you want `ANSIALIAS` turned on, you must explicitly specify it. Using `LANGLVL(COMMONC)` and `ANSIALIAS` together may have undesirable effects on your code at a high optimization level. See “LANGLVL” on page 130 for more information on `LANGLVL(COMMONC)`.
3. A comment that indicates the `ANSIALIAS` option setting is generated in your object module to aid you in diagnosing your program.
4. Although type-based aliasing does not apply to the `volatile` and `const` qualifiers, these qualifiers are still subject to other semantic restrictions. For example, casting away a `const` qualifier might lead to an error at run time. See “CHECKOUT | NOCHECKOUT” on page 85 to see how to obtain more diagnostic information.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

The `ANSIALIAS` option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the `ANSIALIAS` option, but ignores it.

ARCHITECTURE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`TARGET(OSV2R10 and above): ARCH(2)` `TARGET(OSV2R9 and below): ARCH(0)`						

CATEGORY: Object Code Control

```
►►──ARCH──(──n──)──────────────────────────────────────────────►◄
```

The ARCH option selects the instruction set available during the code generation of your program based on the specified machine architecture. Specifying a higher ARCH level generates code that uses newer and faster instructions instead of the sequences of common instructions. A subparameter specifies the group to which a model number belongs. Note that your application will not run on a lower architecture processor than what you specified using the ARCH option. Use the ARCH level that matches the lowest machine architecture where your program will run.

Use the ARCH option in cooperation with the TUNE option. For more information on the interaction between ARCH and TUNE, see "TUNE" on page 190.

If you specify a group which does not exist or is not supported, the compiler uses the default, and issues a warning message.

Current groups of models that are supported include the following:

0 Is the default value. It produces code that is executable on all models.

1 Produces code that uses instructions available on the following models:
- 9021-520, 9021-640, 9021-660, 9021-740, 9021-820, 9021-860, and 9021-900
- 9021-xx1 and 9021-xx2
- 9672-Rx1, 9672-Rx2 (G1), 9672-Exx, and 9672-Pxx

Specifically, these ARCH(1) machines and their follow-ons add the *C Logical String Assist* hardware instructions. These instructions are exploited by the compiler, when practical, for a faster and more compact implementation of some functions, for example, strcmp().

2 Produces code that uses instructions available on the following models:
- 9672-Rx3 (G2), 9672-Rx4 (G3), 9672-Rx5 (G4), and 2003

Specifically, these ARCH(2) machines and their follow-ons add the Branch Relative instruction (Branch Relative and Save - BRAS), and the halfword Immediate instruction set (for example, Add Halfword Immediate - AHI) which may be exploited by the compiler for faster processing.

3 Produces code that uses instructions available on the 9672-xx6 (G5), 9672-xx7 (G6), and follow-on models.

Specifically, these ARCH(3) machines and their follow-ons add a set of facilities for IEEE floating-point representation, as well as 12 additional floating-point registers and some new floating-point support instructions may be exploited by the compiler.

Note that ARCH(3) is required for execution of a program that specifies the FLOAT(IEEE) compiler option. However, if the program is executed on a physical processor that does not actually provide these ARCH(3) facilities, any program check (operation or specification exception), resulting from an attempt to use features associated with IEEE floating point or the additional floating point registers, will be intercepted by the underlying OS/390 V2R6 or higher operating system, and simulated by software. There will be a significant performance degradation for the simulation.

Notes:

1. Code that is compiled at ARCH(1) runs on machines in the ARCH(1) group and later machines, including those in the ARCH(2) and ARCH(3)groups. It may not run on earlier machines. Code that is compiled at ARCH(2) may not run on ARCH(1) or earlier machines and code that is compiled at ARCH(3) may not run on ARCH(2) or earlier machines.
2. For the above system machine models, x indicates any value. For example, 9672-Rx4 means 9672-RA4 through to 9672-RX4, not just 9672-RX4.

You can specify this option using the #pragma option directive for C.

Effect on IPA Compile Step

If you specify the ARCHITECTURE option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition.

If you specify the ARCH option on the IPA Link step, it uses the value of that option for all partitions. The IPA Link step Prolog and all Partition Map sections of the IPA Link step listing display that value.

If you do not specify the option on the IPA Link step, the value used for a partition depends upon the value that you specified for the IPA Compile step for each compilation unit that provided code for that partition. If you specified the same value for each compilation unit, the IPA Link step uses that value. If you specified different values, the IPA Link step uses the lowest level of ARCH.

The level of ARCH for a partition determines the level of TUNE for the partition. For more information on the interaction between ARCH and TUNE, see "TUNE" on page 190.

The Partition Map section of the IPA Link step listing, and the object module display the final option value for each partition. If you override this option on the IPA Link step, the Prolog section of the IPA Link step listing displays the value of the option.

The Compiler Options Map section of the IPA Link step listing displays the option value that you specified for each IPA object file during the IPA Compile step.

ARGPARSE | NOARGPARSE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
ARGPARSE	ARGPARSE	ARGPARSE	ARGPARSE			

CATEGORY: Program Execution

```
►►──┬─ARG───┬──────────────────────────────────────────────────►◄
    └─NOARG─┘
```

The ARGPARSE option specifies that the arguments supplied on the invocation line are parsed and passed to the main() routine in the C argument format, commonly argc and argv. argc contains the argument count, and argv contains the tokens after the command processor has parsed the string.

If you specify NOARGPARSE , arguments on the invocation line are not parsed. argc has a value of 2, and argv contains a pointer to the string.

Note: If you specify NOARGPARSE, you cannot specify REDIR. The compiler will turn off REDIR with a warning since the whole string on the command line is treated as an argument and put into argv .

This option has no effect under CICS.

Effect on IPA Compile Step

If you specify ARGPARSE for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

If you specify this option for both the IPA Compile and the IPA Link steps, the setting on the IPA Link step overrides the setting on the IPA Compile step. This applies whether you use ARGPARSE and NOARGPARSE as compiler options, or specify them using the #pragma runopts directive on the IPA Compile step.

If you specified ARGPARSE on the IPA Compile step, you do not need to specify it again on the IPA Link step to affect that step. The IPA Link step uses the information generated for the compilation unit that contains the main() function. If it cannot find a compilation unit that contains main(), it uses the information generated by the first compilation unit that it finds.

ATTRIBUTE | NOATTRIBUTE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓	✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOATTRIBUTE			NOATTRIBUTE	NOATTRIBUTE	NOATTRIBUTE	NOATTRIBUTE

CATEGORY: Listing

```
►►──┬─ATT──┬────────┬─┬──────────────────────────────────────────►◄
    │      └─(FULL)─┘ │
    └─NOATT───────────┘
```

The ATTRIBUTE option produces a Cross Reference listing that shows the attributes for each symbol, an External Symbol Cross Reference section, and Static Map section.

The ATTRIBUTE(FULL) option produces a listing of all identifiers that are found in your code, even those that are not referenced. The compiler writes the listing produced by ATTRIBUTE or ATTRIBUTE(FULL) to a listing file.

The NOATTRIBUTE option suppresses the attribute listing.

In the z/OS UNIX System Services environment, this option is turned on by specifying -V when using the cxx command.

Effect on IPA Compile Step

During the IPA Compile step, the compiler saves symbol storage offset information in the IPA object file as follows:

- For C, if you specify the XREF, IPA(ATTRIBUTE), or IPA(XREF) options or the #pragma option (XREF)
- For C++, if you specify the ATTR, XREF, IPA(ATTRIBUTE), or IPA(XREF) options

If regular object code/data is produced using the IPA(OBJECT) option, the cross reference sections of the compile listing will be controlled by the ATTR and XREF options.

Effect on IPA Link Step

If you specify the ATTR or XREF options for the IPA Link step, it generates External Symbol Cross Reference and Static Map listing sections for each partition.

The IPA Link step creates a Storage Offset listing section if during the IPA Compile step you requested the additional symbol storage offset information for your IPA objects.

CHECKOUT | NOCHECKOUT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities Note that this changes if the -v flag is set.					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOCHECKOUT	NOCHECKOUT	NOCHECKOUT				

CATEGORY: Debug/Diagnostic

```
               ┌─,───────────┐
               ▼             │
►►──┬─CHE──(───subopts───────┴──)──┬────────────────────────────────────────►◄
    └─NOCHE────────────────────────┘
```

where:

subopts is one of the suboptions that are shown in Table 16.

The CHECKOUT option specifies that the compiler is to produce informational messages that indicate possible programming errors. The messages can help z/OS C programmers to debug their programs.

You can specify CHECKOUT with or without suboptions. If you include suboptions, you can specify any number with commas between them. If you do not include suboptions, the compiler uses the default for CHECKOUT at your installation.

This table lists the CHECKOUT suboptions, their abbreviations, and the messages they generate.

Note: Default CHECKOUT suboptions are underlined.

Table 16. CHECKOUT Suboptions, Abbreviations, and Descriptions

CHECKOUT Suboption	Abbreviated Name	Description
ACCURACY | NOACCURACY	AC | NOAC	Assignments of `long` values to variables that are not `long`
CAST | NOCAST	CA | NOCA	Potential violation of ANSI type-based aliasing rules in explicit pointer type castings. Implicit conversions, for example, those due to assignment statements, are already checked with a warning message for incompatible pointer types. See “ANSIALIAS | NOANSIALIAS” on page 80 for more information on ANSI type-based aliasing. Also see “DLL | NODLL” on page 99 for DLL function pointer casting restrictions.
ENUM | NOENUM	EN | NOEN	Usage of enumerations
EXTERN | NOEXTERN	EX | NOEX	Unused variables that have external declarations
GENERAL | NOGENERAL	GE | NOGE	General checkout messages
GOTO | NOGOTO	GO | NOGO	Appearance and usage of `goto` statements
INIT | NOINIT	I | NOI	Variables that are not explicitly initialized
PARM | NOPARM	PAR | NOPAR	Function parameters that are not used

Table 16. CHECKOUT Suboptions, Abbreviations, and Descriptions (continued)

CHECKOUT Suboption	Abbreviated Name	Description
PORT | NOPORT	POR | NOPOR	Nonportable usage of the z/OS C language
PPCHECK | NOPPCHECK	PPC | NOPPC	All preprocessor directives
PPTRACE | NOPPTRACE	PPT | NOPPT	Tracing of include files by the preprocessor
TRUNC | NOTRUNC	TRU | NOTRU	Variable names that are truncated by the compiler
ALL	ALL	Turns on all of the suboptions for `CHECKOUT`
NONE	NONE	Turns off all of the suboptions for `CHECKOUT`

You can specify the `CHECKOUT` option on the invocation line and on the `#pragma options` preprocessor directive. When you use both methods at the same time, the options are merged. If an option on the invocation line conflicts with an option in the `#pragma options` directive, the option on the invocation line takes precedence. The following examples illustrate these rules.

Source file:
```
#pragma options (NOCHECKOUT(NONE,ENUM))
```

Invocation line:
```
CHECKOUT (GOTO)
```

Result:
```
CHECKOUT (NONE,ENUM,GOTO)
```

Source file:
```
#pragma options (NOCHECKOUT(NONE,ENUM))
```

Invocation line:
```
CHECKOUT (ALL,NOENUM)
```

Result:
```
CHECKOUT (ALL,NOENUM)
```

Note: If you used the `CHECKOUT` option and did not receive an informational message, ensure that the setting of the `FLAG` option is `FLAG(I)`.

The `NOCHECKOUT` option specifies that the compiler should not generate informational error messages. Suboptions that are specified in a `#pragma options(NOCHECKOUT(subopts))` directive, or `NOCHECKOUT(subopts)` apply if `CHECKOUT` is specified on the command line.

You can turn the `CHECKOUT` option off for certain files or statements of your source program by using a `#pragma checkout(suspend)` directive. Refer to the *z/OS C/C++ Language Reference* for more information regarding this pragma directive.

You can specify this option using the `#pragma option` directive for C.

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c89` or `cc` commands. The suboptions, when `-V` is specified, become `ALL`, `NOEXTERN`, `NOPPCHECK`, `NOPPTRACE`.

Effect on IPA Compile Step
The CHECKOUT option is used for source code analysis, and has the same effect on IPA Compile step processing as it does on a regular compilation.

Effect on IPA Link Step
The IPA Link step accepts the CHECKOUT option, but ignores it.

COMPACT | NOCOMPACT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOCOMPACT						

CATEGORY: Object Code Control

```
►►──┬─COMPACT───┬──────────────────────────────────────────────►◄
    └─NOCOMPACT─┘
```

During optimizations performed during code generation, for both NOIPA and IPA, choices must be made between those optimizations which tend to result in faster but larger code and those which tend to result in smaller but slower code. The COMPACT option influences these choices. When the COMPACT option is used, the compiler favours those optimizations which tend to limit the growth of the code. Because of the interaction between various optimizations, including inlining, code compiled with the COMPACT option may not always generate smaller code and data. To determine the final status of inlining, generate and check the inline report as subprograms may not be inlined or inline other subprograms when COMPACT is specified.

To evaluate the use of the COMPACT option for your application:
- Compare the size of the objects generated with COMPACT and NOCOMPACT
- Compare the size of the modules generated with COMPACT and NOCOMPACT
- Compare the execution time of a representative workload with COMPACT and NOCOMPACT

If the objects and modules are smaller with an acceptable change in execution time, then you can consider using COMPACT.

As new optimizations are added to the compiler, the behavior of the COMPACT option may change. You should re-evaluate the use of this option for each new release of the compiler and when the user changes the application code.

You can specify this option for a specific subprogram using the #pragma option_override(subprogram_name, OPT(COMPACT)) directive.

Effect on IPA(OBJONLY) Compilation

During a compilation with IPA compile-time optimizations active, any subprogram-specific COMPACT option specified by #pragma option_override(subprogram_name, OPT(COMPACT)) directives will be retained.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

If you specify the COMPACT option for the IPA Link step, it sets the Compilation Unit values of the COMPACT option that you specify. The IPA Link step Prolog listing section will display the value of this option.

If you do not specify COMPACT option in the IPA Link step, the setting from the IPA Compile step for each Compilation Unit will be used.

In either case, subprogram-specific COMPACT options will be retained.

The IPA Link step merges and optimizes your application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition. Only compatible subprograms are included in a given partition. Compatible subprograms have the same COMPACT setting.

The COMPACT setting for a partition is set to the specification of the first subprogram that is placed in the partition. Subprograms that follow are placed in partitions that have the same COMPACT setting. A NOCOMPACT subprogram is placed in a NOCOMPACT partition, and a COMPACT subprogram is placed in a COMPACT partition.

The option value that you specified for each IPA object file on the IPA Compile step appears in the IPA Link step Compiler Options Map listing section.

The Partition Map sections of the IPA Link step listing and the object module END information section display the value of the COMPACT option. The Partition Map also displays any subprogram-specific COMPACT values.

COMPRESS | NOCOMPRESS

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOCOMPRESS						

CATEGORY: Object Code Control

```
►►──┬─COMPRESS───┬──────────────────────────────────────────────────►◄
    └─NOCOMPRESS─┘
```

Use the COMPRESS option to suppress the generation of function names in the function control block thereby reducing the size of your application's load module. Note that the function names are used by the dump service to provide you with meaningful diagnostic information when a program encounters a fatal program error. They are also used by tools such as the Debug Tool and Performance Analyzer. Without these function names, the reports generated by these services and tools may not be complete.

Note that if COMPRESS and TEST are in effect at the same time, the compiler issues a warning message and ignores the COMPRESS option.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

If you specify the COMPRESS option for the IPA Link step, it uses the value of the option that you specify. The IPA Link step Prolog listing section will display the value of the option that you specify.

If you do not specify COMPRESS option in the IPA Link step, the setting from the IPA Compile step will be used.

The IPA Link step merges and optimizes your application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition. Only compatible subprograms are included in a given partition. Compatible subprograms have the same COMPRESS setting.

The COMPRESS setting for a partition is set to the specification of the first subprogram that is placed in the partition. Subprograms that follow are placed in partitions that have the same COMPRESS setting. A NOCOMPRESS mode subprogram is placed in a NOCOMPRESS partition, and a COMPRESS mode subprogram is placed in a COMPRESS partition.

The option value that you specified for each IPA object file on the IPA Compile step appears in the IPA Link step Compiler Options Map listing section.

The Partition Map sections of the IPA Link step listing and the object module END information section display the value of the COMPRESS option.

CONVLIT | NOCONVLIT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOCONVLIT (, WCHAR)`						

CATEGORY: Preprocessor

```
►►──┬─CONV───┬──┬───────────────────────────────────────┬──────────────►◄
    └─NOCONV─┘  └─(─┬──────────┬──┬─, WCHAR───┬──)─┘
                    └─codepage─┘  └─, NOWCHAR─┘
```

The `CONVLIT` option changes the assumed codepage for character and string literals within the compilation unit. You can use an optional suboption to specify the codepage that you want to use for string literals. If you specify `NOCONV` or `CONV` without a suboption, the default codepage, or the codepage specified by the `LOCALE` option is used.

You can also specify a suboption with the `NOCONV` option. The result of the following specifications is the same:
- `NOCONV(IBM-1027) CONV`
- `CONV(IBM-1027)`

The `CONVLIT` option affects all the source files that are processed within a compilation unit, including user header files and system header files. All string literals within a compilation unit are converted to the specified codepage unless you use `#pragma convlit(suspend)` and `#pragma convlit(resume)` to exclude sections of code from conversion. See the *z/OS C/C++ Language Reference* for more information on `#pragma convlit`.

The `CONVLIT` option only affects string literals within the compilation unit. The following determines the codepage that the rest of the program uses:
- If you specified a `LOCALE`, the remainder of the program will be in the codepage that you specified with the `LOCALE` option.
- If you did not specify a `LOCALE`, the remainder of the program will be in the default codepage IBM-1047.

The `CONVLIT` option does not affect the following types of string literals:
- literals in the #include directive
- literals in the #pragma directive
- literals used to specify linkage, for example, `extern "C"`

The `CONVLIT(, WCHAR)` suboption instructs the compiler to change the codepage for wide character constants and string literals declared with the L” or L″″ prefix. Although optional, the `WCHAR` suboption is positional, and must appear as the second suboption to the `CONVLIT` option. The default is `WCHAR`. Only wide character constants and string literals made up of single byte character set (SBCS) characters are converted. If there are any shift-out (SO) and shift-in (SI) characters in the literal, the compilation will end with an error message.

If you specify either `PPONLY` with `CONVLIT` , the compiler ignores `CONVLIT`.

If you specify the CONVLIT option, the codepage appears after the locale name and locale code set in the Prolog section of the listing. The option appears in the END card at the end of the generated object module.

Note: Although you can continue to use the __STRING_CODE_SET__ macro, you should use the CONV option instead. If you specify both the macro and the option, the compiler uses the option regardless of the order in which you specify them.

Effect on IPA Compile Step

The CONVLIT option only controls processing for the IPA step for which you specify it.

During the IPA Compile step, the compiler uses the code page that is specified by the CONVLIT option to convert the character string literals.

Effect on IPA Link Step

The IPA Link step accepts the CONVLIT option, but ignores it.

CSECT | NOCSECT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
For NOGOFF, NOCSECT For GOFF, CSECT()						

CATEGORY: Object Code Control

```
►►──┬─CSE───┬──┬─────────────────┬──────────────────────────────────►◄
    └─NOCSE─┘  └─(──qualifier──)─┘
```

The CSECT option ensures that the code, static data, and test sections of your object module are named. Use this option, or the #pragma CSECT directive, if you will be using SMP/E to service your product, and to aid in debugging your program. See the *z/OS C/C++ Language Reference* for further information on the #pragma CSECT directive.

The NOCSECT option does not name the code, static, or test data sections of your object module.

The qualifier suboption of the CSECT option allows the compiler to generate long CSECT names.

For NOGOFF, if the LONGNAME compiler option is not in effect when you specify CSECT(qualifier), the compiler turns it on, and issues an informational message. For GOFF, both NOLONGNAME and LONGNAME options are supported.

The CSECT option names sections of your object module differently depending on whether you specified CSECT with or without a qualifier.

The CSECT option with no qualifier

If you specify the CSECT option without the qualifier suboption, the CSECT option names the code, static data, and test sections of your object module as *csectname*, where *csectname* is one of the following:

- The member name of your primary source file, if it is a PDS member
- The low-level qualifier of your primary source file, if it is a sequential data set
- The source file name with path information and the right-most extension information removed, if it is an HFS file.
- For NOGOFF, if the NOLONGNAME option is in effect, then the csectname is truncated to 8 characters long starting from the left. For GOFF, the full csectname is always used.

code CSECT
: Is named with *csectname* name in uppercase.

data CSECT
: Is named with *csectname* in lower case.

test CSECT
: When you use the TEST option together with the CSECT option, the debug information is placed in the test CSECT. The test CSECT is the static CSECT name with the prefix $. If the static CSECT name is eight characters long, the rightmost character is dropped and the compiler issues an informational message except in the case of GOFF. The test CSECT name is always truncated to eight characters.

  For example, if you compile /u/cricket/project/mem1.ext.c:

  - with the options NOGOFF and CSECT, the test CSECT will have the name $mem1.ex
  - with the options GOFF and CSECT, the test CSECT will have the name $mem1.ext

The CSECT option with the qualifier suboption

If you specify the CSECT option with the qualifier suboption, the CSECT option names the code, static data, and test sections of your object module as *qualifier#basename#suffix*, where:

qualifier
: Is the suboption you specified as a qualifier

basename
: Is one of the following:
  - The member name of your primary source file, if it is a PDS member
  - There is no basename, if your primary source file is a sequential data set or instream JCL
  - The source file name with path information and the right-most extension information removed, if it is an HFS file

suffix
: Is one of the following:

  C For code CSECT

  S For static CSECT

  T For test CSECT

For example, if you compile `/u/cricket/project/mem1.ext.c` with the options `TEST` and `CSECT(example)`, the compiler constructs the CSECT names as follows:

```
example#mem1.ext#C
example#mem1.ext#S
example#mem1.ext#T
```

The qualifier suboption of the `CSECT` option allows the compiler to generate long CSECT names. If the compiler option `LONGNAME` is not in effect when you specify the `CSECT(qualifier)`, the compiler turns it on, and issues an informational message.

For example, if you compile `/u/cricket/project/reallylongfilename.ext.c` with the options `TEST` and `CSECT(example)`, the compiler constructs the CSECT names as follows:

```
example#reallylongfilename.ext#C
example#reallylongfilename.ext#S
example#reallylongfilename.ext#T
```

When you specify `CSECT(qualifier)`, the code, data, and test CSECTs are always generated. The test CSECT has content only if you also specify the `TEST` option.

If you use `CSECT("")` or `CSECT()`, the CSECT name has the form *basename#suffix*, where *basename* is:

- `@Sequential@` for a sequential data set
- `@InStream@` for instream JCL

Notes:

1. If the qualifier suboption is longer than 8 characters you must use the binder.
2. The qualifier suboption takes advantage of the binder's capabilities, and may not generate names acceptable to the z/OS Language Environment Prelinker.
3. The # that is appended as part of the #C, #S, or #T suffix is not locale-sensitive.
4. The string that is specified as the qualifier suboption has the following restrictions:
   - Leading and trailing blanks are removed
   - You can specify a string of any length. However if the complete CSECT name exceeds 1024 bytes, it is truncated starting from the left.
5. If the source file is either sequential or instream in your JCL, you must use the `#pragma csect` directive to name your CSECT. Otherwise, you may receive an error message at bind time.

Effect on IPA Compile Step

The `CSECT` option has the same effect on the IPA Compile step (if you specify the `OBJECT` suboption of the `IPA` option) as it does on a regular compilation.

Effect on IPA Link Step

For the IPA Link step, this option has the following effects:

1. If you specify the `CSECT` option, the IPA Link step names all of the CSECTs that it generates.

   The IPA Link step determines whether the IPA Link control file contains CSECT name prefix directives. If you did not specify the directives, or did not specify enough CSECT entries for the number of partitions, the IPA Link step automatically generates CSECT name prefixes for the remaining partitions, and issues an error diagnostic message each time.

The form of the CSECT name that IPA Link generates depends on whether the CSECT or `CSECT(`*`qualifier`*`)` format is used. See “IPA Link Step Control File” on page 339 for details.

2. If you do not specify the `CSECT` option, but you have specified CSECT name prefix directives in the IPA Link control file, the IPA Link step names all CSECTs in a partition. If you did not specify enough CSECT entries for the number of partitions, the IPA Link step automatically generates a CSECT name prefix for each remaining partition, and issues a warning diagnostic message each time.
3. If you do not specify the `CSECT` option, and do not specify CSECT name prefix directives in the IPA Link control file, the IPA Link step does not name the CSECTs in a partition.

The IPA Link step ignores the information that is generated by `#pragma csect` on the IPA Compile step.

CVFT | NOCVFT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`CVFT`						

CATEGORY: Object Code Control

```
►►──┬─CVFT───┬────────────────────────────────────────────────►◄
    └─NOCVFT─┘
```

The `NOCVFT` option benefits your application’s performance by shrinking the size of the writeable static area (WSA). It reduces the size of construction virtual function tables (CVFT), which in turn reduces the load module size. Use `NOCVFT` if none of the constructors in your application calls virtual functions from within the class hierarchy, either directly or indirectly.

The `NOCVFT` option relieves certain constructors from tracking which virtual function to call at different stages of the construction process. This tracking by the constructor would require that the constructor maintain its own CVFT. Only constructors that call virtual functions within a class hierarchy that uses virtual inheritance are affected.

Consider the following example:

```
struct  A {
          virtual int f() { return 0; }    // line a
};

struct B : virtual A {
          virtual int f() { return 1; }    // line b
          B() { cout << f() << endl; }
};

struct C : virtual B {
          virtual int f() { return 2; }    // line c
};

...
```

In the above example, if an instance of `C` is constructed, the ANSI C++ standard requires that `1` (the number one) be printed. That is, the function `B::f()` at `line b` should be called during the construction of `C`. After `C` is constructed, a call to `f()` should invoke `C::f()` at `line c`. To support the ANSI behavior and call the right function, the z/OS C++ compiler needs to keep extra information during object construction. This extra information can require a lot of memory if an application uses a lot of virtual inheritance.

The `NOCVFT` option breaks the above ANSI C++ behavior. In that case, the virtual function called by your application is always the same one that would be called if the object is fully constructed. In the above example, this is `C::f()`, and `2` is printed during the construction of an instance of `C` (the function at `line c`). The `CVFT` option preserves the ANSI C++ behavior.

The `CVFT` option is shown on the listing prolog and the text deck end card.

Effect on IPA Compile Step

The `CVFT` option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step issues a diagnostic message if you specify the `CVFT` option for that step.

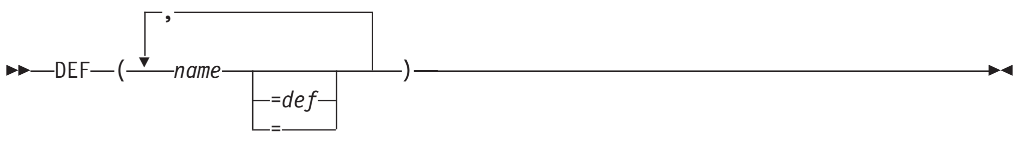
DEFINE

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities** **Note you can override with appropriate `-D` or `-U` options.**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**

no default user definitions	`DEFINE(errno=(*__errno()))`
	
	`c89` also specifies
	`DEFINE(_OPEN_DEFAULT=1)`
		`DEFINE(errno=(*__errno()+))`
		
		`cc` also specifies
		`DEFINE(_OPEN_DEFAULT=0)`
		
		and
		`DEFINE(_NO_PROTO=1)`
			`DEFINE(errno=(*__errno()))`
			
			`c++` also specifies
			`DEFINE(_OPEN_DEFAULT=1)`			

CATEGORY: Preprocessor



The `DEFINE` option defines preprocessor macros that take effect before the compiler processes the file. You can use this option more than once.

DEFINE(*name*)
: is equal to the preprocessor directive `#define` *name* `1`.

DEFINE(*name*=*def*)
: is equal to the preprocessor directive `#define` *name def*.

DEFINE(*name*=)
: is equal to the preprocessor directive `#define` *name*.

If the suboptions that you specify contain special characters, see "Using Special Characters" on page 61 for information on how to escape special characters.

Note: There is no command-line equivalent of function-like macros that take parameters such as the following:

```
#define max(a,b)  ((a)>(b)?(a):(b))
```

In the z/OS UNIX System Services environment, variables can be set by specifying `-D` when using the `c89`, `cc`, or `c++` commands.

Effect on IPA Compile Step

The `DEFINE` option is used for source code analysis, and has the same effect on an IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts but ignores the `DEFINE` option.

DIGRAPH | NODIGRAPH

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NODIGRAPH			NODIGRAPH			

CATEGORY: Source Code Analysis or Preprocessor

```
►►──┬─DIGR───┬────────────────────────────────────────────────►◄
    └─NODIGR─┘
```

The DIGRAPH option allows you to use additional digraphs in both C and C++ applications. In addition, it allows you to use additional keywords in C++ applications only. A digraph is a combination of keys that produces a character not available on some keyboards. Table 17 shows the digraphs that z/OS C/C++ supports:

Table 17. Digraphs

Key Combination	Character Produced
<%	{
%>	}
<:	[
:>	]
%:	#
%%[1]	#
%:%:	##
%%%%[1]	##

Table 18 shows additional keywords that z/OS C++ supports:

Table 18. Additional Keywords

Keyword	Characters produced
bitand	&
and	&&
bitor	|
or	||
xor	^

1. The digraphs %% and %%%% are not digraphs in the C Standard. For compatibility with z/OS C++, however, they are supported by z/OS C. Use the %: and %:%: digraphs instead of %% and %%%% whenever possible.

Table 18. Additional Keywords (continued)

Keyword	Characters produced
compl	~
and_eq	&=
or_eq	|=
xor_eq	^=
not	!
not_eq	!=

Note: Digraphs are not replaced in string literals, comments, or character literals. For example:

```
char * s = "<%%>";   // stays "<%%>"

switch (c) {
  case '<%' : ...    // stays '<%'
  case '%>' : ...    // stays '%>'
}
```

Effect on IPA Compile Step

The DIGRAPH option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step issues a diagnostic message if you specify the DIGRAPH option on that step.

DLL | NODLL

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NODLL(NOCBA) for C compile and IPA Link step. DLL(NOCBA) for C++ Compile.						

CATEGORY: Object Code Control

```
►►──┬─DLL───┬──┬─(CBA)───┬──────────────────────────────────────────────►◄
    └─NODLL─┘  └─(NOCBA)─┘
```

The DLL option instructs the compiler to produce DLL code. The DLL code can export or import functions and external variables.

The DLL option has two suboptions:

NOCALLBACKANY
: is the default. If you specify NOCALLBACKANY, no changes will be made to the function pointer in your compile unit. The abbreviation for NOCALLBACKANY is NOCBA.

CALLBACKANY
: If you specify CALLBACKANY, all calls through function pointers will accommodate function pointers created by applications compiled without the DLL option. This accommodation accounts for the incompatibility of function pointers created with and without the DLL compiler option. The CALLBACKANY suboption is not supported when the XPLINK option is used. When function pointers having their origins (that is, where the address of a function is taken and assigned to a function pointer) in XPLINK code in the same or another DLL, or non-XPLINK NODLL code in another DLL, or non-XPLINK DLL code in another DLL, are passed to exported XPLINK functions, the compiler inserts code to check whether or not the function pointers received as actual arguments are valid (useable directly) XPLINK function pointers, and converts them if required. This provides results that are similar in many respects to the function pointer conversion provided when DLL(CALLBACKANY) is specified for non-XPLINK code. Other function pointers that have their origins in non-XPLINK code, including function pointer parameters passed to non-exported functions or otherwise acquired, are not converted automatically by XPLINK compiled code. Use of such function pointers will cause the application to fail. The abbreviation for CALLBACKANY is CBA.

Note: You should write your code according to the rules listed in the chapter ″Building Complex DLLs″ in the *z/OS C/C++ Programming Guide*, and compile with the NOCALLBACKANY suboption. Use the suboption CALLBACKANY only when you have calls through function pointers and C code compiled without the DLL option. CALLBACKANY causes **all** calls through function pointers to incur overhead due to internally-generated calls to library routines that determine whether the function pointed to is in a DLL (in which case internal control structures need to be updated), or not. This overhead is unnecessary in an environment where all function pointers were created either in C++ code or in C code compiled with the DLL option.

For information on how to create or use DLLs, and on when to use the appropriate DLL options and suboptions, see the *z/OS C/C++ Programming Guide*.

Notes:

1. Code compiled with the z/OS C++ compiler, and code compiled with the XPLINK compiler option, is always DLL code. You can not specify NODLL for these cases.
2. You must use the LONGNAME and RENT options with the DLL option. If you use the DLL option without RENT and LONGNAME, the z/OS C compiler automatically turns them on. However, when the XPLINK option is used, though RENT and LONGNAME are the default options, both NOLONGNAME and NORENT are allowed.
3. In code compiled with the XPLINK compiler option, function pointers are compared using the address of the descriptor. No special considerations, such as dereferencing, are required to initialize the function pointer prior to comparison.
4. In code compiled without the XPLINK compiler option, you cannot cast a non-zero integer const type to a DLL function pointer type as shown in the following example:

```
void (*foo)();

void main() {
            /* ... */

        if (foo != (void (*)()) (50L) ) {
            /* do something other than calling foo */
        }
}
```

The above conditional expression will cause an abend at execution time because the function pointer (with value 50L) needs to be dereferenced to perform the comparison. The compiler will check for this type of casting problem if you use the CHECKOUT(CAST) option along with the DLL option. See "CHECKOUT | NOCHECKOUT" on page 85 for more information on obtaining diagnostic information for C applications.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. The CALLBACKANY option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

The IPA Link step accepts the DLL compiler option, but ignores it.

The IPA Link step uses information from the IPA Compile step to classify an IPA object module as DLL or non-DLL as follows:

- C code that is compiled with the DLL option is classified as DLL.
- C++ code is classified as DLL
- C code that is compiled with the NODLL option is classified as non-DLL.

Each partition is initially empty and is set as DLL or non-DLL, when the first subprogram (function or method) is placed in the partition. The setting is based on the DLL or non-DLL classification of the IPA object module which contained the subprogram. Procedures from IPA object modules with incompatible DLL values will not be inlined. This results in reduced performance. For best performance, compile your application as all DLL code or all non-DLL code.

The IPA Link step allows you to input a mixture of IPA objects that are compiled with DLL(CBA) and DLL(NOCBA). The IPA Link step does not convert function pointers from the IPA Objects that are compiled with the option DLL(NOCBA).

You should only export subprograms (functions and C++ methods) or variables that you need for the interface to the final DLL. If you export subprograms or variables unnecessarily (for example, by using the EXPORTALL option), you severely limit IPA optimization. Global variables are not coalesced, and unreachable or 100% inlined code is not pruned.

EVENTS | NOEVENTS

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOEVENTS	NOEVENTS	NOEVENTS				

CATEGORY: Debug/Diagnostic

```
►►──┬─EVENT───┬──┬──────────────────────────────────────────────┬──────────────►◄
    └─NOEVENT─┘  └─(──┬─Sequential filename─────────────┬──)─┘
                      ├─Partitioned data set────────────┤
                      ├─Partitioned data set (member)───┤
                      ├─Hierarchical filename───────────┤
                      └─Hierarchical directory──────────┘
```

The EVENTS option creates an events file that contains error information and source file statistics. The compiler writes the events data to the DD:SYSEVENT ddname, if you allocated one before you called the compiler. Otherwise, it allocates a data set, and the name is the file name with SYSEVENT as the lowest-level qualifier.

If you specified a suboption, the compiler uses the data set that you specified, and ignores the DD:SYSEVENT.

If the source file is an HFS file, and you do not specify the events file name as a suboption, the compiler writes the events file in the current working directory. The events file name is the name of the source file with the extension .err.

The compiler ignores #line directives when the EVENTS option is active, and issues a warning message.

For a description of the events file's layout, see "Appendix I. Layout of the Events File" on page 719.

Effect on IPA Compile Step

The EVENT option has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the EVENT option, but ignores it.

EXECOPS | NOEXECOPS

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
EXECOPS	EXECOPS	EXECOPS	EXECOPS			

CATEGORY: Object Code Control and Program Execution

```
►►──┬─EXEC───┬──────────────────────────────────────────────────────►◄
    └─NOEXEC─┘
```

The EXECOPS option allows you to control whether runtime options will be recognized at run time without changing your source code. It is equivalent to including a #pragma runopts (EXECOPS) directive in your source code.

If this option is specified on both the command line and in a #pragma runopts directive, the option on the command line takes precedence.

Effect on IPA Compile Step

If you specify EXECOPS for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

If you specify the EXECOPS option for the IPA Compile step, you do not need to specify it again on the IPA Link step. The IPA Link step uses the information generated for the compilation unit that contains the main() function. If it cannot find a compilation unit that contains main(), it uses information generated for the first compilation unit that it finds.

If you specify this option on both the IPA Compile and the IPA Link steps, the setting on the IPA Link step overrides the setting on the IPA Compile step. This situation occurs whether you use EXECOPS and NOEXECOPS as compiler options, or specify them by using the #pragma runopts directive on the IPA Compile step.

EXH | NOEXH

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
EXH						

CATEGORY: Object Code Control

```
►►──┬─EXH───┬─────────────────────────────────────────────────────►◄
    └─NOEXH─┘
```

The EXH option controls the generation of C++ exception handling code.

The NOEXH option suppresses the generation of the exception handling code, which results in code that runs faster, but will not be ANSI-compliant if the program uses exception handling.

If you compile a source file with NOEXH, active objects on the stack are not destroyed if the stack collapses in an abnormal fashion. For example, if a C++ object is thrown, or a Language Environment exception or signal is raised, objects on the stack will not have their destructors run.

If a source file has try/catch blocks or throws objects, you cannot compile it with the NOEXH option.

Effect on IPA Compile Step

The EXH option has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step issues a diagnostic message if you specify the EXH option for that step.

EXPMAC | NOEXPMAC

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOEXPMAC	NOEXPMAC	NOEXPMAC	NOEXPMAC			

CATEGORY: Listing

```
►►──┬─EXP───┬─────────────────────────────────────────────────────►◄
    └─NOEXP─┘
```

The EXPMAC option instructs the compiler to show all expanded macros in the source listing. If you want to use the EXPMAC option, you must also specify the SOURCE compiler option to generate a source listing. If you specify the EXPMAC option but omit the SOURCE option, the compiler issues a warning message, and does not produce a source listing.

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c89`, `cc` or `c++` commands.

Effect on IPA Compile Step

The `EXPMAC` option has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

The IPA Link Step accepts the `EXPMAC` option, but ignores it.

EXPORTALL | NOEXPORTALL

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOEXPORTALL`	`NOEXPORTALL`	`NOEXPORTALL`	`NOEXPORTALL`			

CATEGORY: Object Code Control

```
►►──┬─EXPO───┬──────────────────────────────────────────────────►◄
    └─NOEXPO─┘
```

The `EXPORTALL` option instructs the compiler to export all external functions and variables in the compilation unit so that a DLL application can use them. Use this option if you are creating a DLL and want to export **all** externally defined functions and variables. You may not export the `main()` function.

Notes:

1. If you only want to export some of the externally defined functions and variables, use `#pragma export`, or the `_Export` keyword for C++. For more information see the *z/OS C/C++ Language Reference*.
2. For C, you must use the `LONGNAME` and `RENT` options with the `EXPORTALL` option. If you use the `EXPORTALL` option without `RENT` and `LONGNAME`, the z/OS C compiler turns them on.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. The `EXPORTALL` option also affects the regular object module if you request one by specifying the `IPA(OBJECT)` option.

Effect on IPA Link Step

The IPA Link step accepts the `EXPORTALL` option, but ignores it.

If you use the `EXPORTALL` option during the IPA Compile step, you severely limit IPA optimization. Refer to “DLL | NODLL” on page 99 for more information about the effects of this option on IPA processing.

FASTTEMPINC | NOFASTTEMPINC

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOFASTT						

CATEGORY: File Management

```
►►──┬─FASTT───┬──────────────────────────────────────────────────────►◄
    └─NOFASTT─┘
```

The FASTTEMPINC option may improve template instantiation compilation time when large numbers of recursive templates are used in an application.

The FASTTEMPINC option defers generating object code until the final version of all template definitions have been determined. Then, a single compilation pass is made to generate the final object code. This means that time is not wasted on generating object code that will be discarded and generated again.

When NOFASTT is used, the compiler generates object code each time a tempinc source file is compiled. If recursive template definitions in a subsequent tempinc source file cause additional template definitions to be added to a previously processed file, an additional recompilation pass is required.

Use FASTT if you have large numbers of recursive templates. If your application has very few recursive template definitions, the time saved by not doing code generation may be less than the time spent in source analysis on the additional template compilation pass. In this case, it may be better to use NOFASTT.

Effect on IPA Compile Step

The FASTT option only affects the processing of source. It has no effect on code generation; therefore, it has the same effect on IPA Compile as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step issues a diagnostic message if you specify the FASTT option for that step.

FLAG | NOFLAG

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities Note that `FLAG(I)` is used if the `-V` flag is set.					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`FLAG(I)`	`FLAG(W)`	`FLAG(W)`	`FLAG(W)`	`FLAG(W)`	`FLAG(W)`	`FLAG(W)`

CATEGORY: Debug/Diagnostic

```
►►──┬─FL──(severity)─┬──────────────────────────────────────────────────►◄
    └─NOFL───────────┘
```

The `FLAG` option specifies the minimum severity level for which you want notification. You specify the minimum severity level by using the compiler option `FLAG (`*`severity`*`)`, where *severity* is one of the following:

`I`	An informational message that is generated by the compiler. This is the default.
`W`	A warning message that calls attention to a possible error, although the statement to which it refers is syntactically valid.
`E`	An error message that shows that the compiler has detected an error and cannot produce an object deck.
`S`	A severe error message that describes an error that forces the compilation to terminate.
`U`	An unrecoverable error message that describes an error that forces the compilation to terminate.

If you specified the options `SOURCE` or `LIST`, the messages generated by the compiler appear immediately following the incorrect source line, and in the message summary at the end of the compiler listing. See “Appendix G. z/OS C/C++ Compiler Return Codes and Messages” on page 591 for a list of the messages.

The `NOFLAG` option is the same as the `FLAG(S)` option.

Effect on IPA Compile Step

The `FLAG` option has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step uses the `FLAG` value that you specify for that step.

FLOAT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`FLOAT(HEX, FOLD, NOMAF, NORRM, NOAFP*)` *dependent on `ARCH()` level						

CATEGORY: Object Code Control

```
►►──FLOAT──(──┬─HEX | IEEE──────┬──)────────────────────────────────────────►◄
              ├─FOLD | NOFOLD───┤
              ├─MAF | NOMAF─────┤
              ├─RRM | NORRM─────┤
              └─AFP | NOAFP─────┘
```

The `FLOAT` option selects the format of floating-point numbers; the format can be either base 2 IEEE-754 binary format, or base 16 S/390 hexadecimal format. In the description below, the IEEE-754 binary format is referred to as the binary floating-point format, and the S/390 hexadecimal format as the hexadecimal floating-point format. `FLOAT` has the following suboptions:

`HEX | IEEE`

DEFAULT: `HEX`

Specifies the format of floating-point numbers and instructions:

- `IEEE` instructs the compiler to generate binary floating-point numbers and instructions. The unabbreviated form of this suboption is `IEEE754`.
- `HEX` instructs the compiler to generate hexadecimal formatted floating-point numbers and instructions. The unabbreviated form of this suboption is `HEXADECIMAL`. In previous releases of z/OS C/C++, the floating-point format was always hexadecimal.

`FOLD | NOFOLD`

DEFAULT: `FOLD`

Specifies that constant floating-point expressions in function scope are to be evaluated at compile time rather than at run time. This is known as *folding*.

In binary floating-point mode, the folding logic uses the rounding mode set by the `ROUND` option.

In hexadecimal floating-point mode, the rounding is always towards zero. If you specify `NOFOLD` in hexadecimal mode, the compiler issues a warning and uses `FOLD`.

`MAF | NOMAF`

DEFAULT:

- `NOMAF`
- If `NOSTRICT` and `FLOAT(IEEE)` are specified, `MAF` is the default.

Uses floating-point Multiply and Add, and Multiply and Subtract instructions where possible, instead of the separate Multiply Float, Add Float, or Multiply Float, Subtract Float instruction pairs. On the current generation of machines, the multiply-and-add and multiply-and-subtract operations are much slower than the corresponding multiply and add/subtract instructions. They should be used only when improved precision of intermediate results is desired.

Note: The suboption `MAF` does not have any effect on extended floating-point operations.

`MAF` is not available for hexadecimal floating-point mode.

`RRM | NORRM`

DEFAULT: `NORRM`

`RRM` (run-time rounding mode) tells the compiler that the run-time rounding mode may not be the default, *round-to-nearest*, and prevents compiler optimizations that rely on *round-to-nearest* rounding mode. Use this option if your program changes the rounding mode by any means. Otherwise, the program may compute incorrect results.

`RRM` is not available for hexadecimal floating-point mode.

`AFP | NOAFP`

DEFAULT:

- If the level of the `ARCH` option is lower than 3, the default is `NOAFP`
- If the level of the `ARCH` option is 3 or higher, the default is `AFP`

`AFP` instructs the compiler to generate code which makes full use of the full complement of 16 floating point registers. These include the four original floating-point registers, numbered 0, 2, 4, and 6, and the Additional Floating Point (AFP) registers, numbered 1, 3, 5, and 7 through 15.

The AFP registers are physically available only on the newer S/390 machine models, starting with the processors that are represented by the `ARCH(3)` setting. However, when the application executes under OS/390 Version 2 Release 6 on a processor that does not have the AFP registers, the operating system is able to intercept the use of an AFP register and emulate the operation such that the AFP register appears to be available to the application.

Note: This emulation has a significant performance cost to the application's execution on the non-AFP processors. This is why the default is `NOAFP` when `ARCH(2)` or lower is specified.

NOAFP limits the compiler's code generation to using only the original four floating-point registers, 0, 2, 4, and 6, which are available on all S/390 machine models.

Using IEEE Floating-Point

You should use IEEE floating-point in the following situations:

- You deal with data that are already in IEEE floating-point format
- You need the increased exponent range (see *z/OS C/C++ Language Reference* for information on exponent ranges with IEEE-754 floating-point)
- You want the changes in programming paradigm provided by infinities and NaN (not a number)

For more information about the IEEE format, refer to the IEEE 754-1985 IEEE Standard for Binary Floating-Point Arithmetic.

When you use IEEE floating-point, make sure that you are in the same rounding mode at compile time (specified by the ROUND(*mode*) option), as at run time. Entire compilation units will be compiled with the same rounding mode throughout the compilation. If you switch runtime rounding modes inside a function, your results may vary depending upon the optimization level used and other characteristics of your code: switch rounding mode inside functions with caution.

If you have existing data in hexadecimal floating-point (the original base 16 S/390 hexadecimal floating-point format), and have no need to communicate these data to platforms that do not support this format, there is no reason for you to change to IEEE floating-point format.

Applications that mix the two formats are not supported.

The binary floating-point instruction set is physically available only on processors that are part of the ARCH(3) group or higher. You can request FLOAT(IEEE) code generation for an application that will run on an ARCH(2) or earlier processor, if that processor runs on the OS/390 Version 2 Release 6 or higher operating system. This operating system level is able to intercept the use of an "illegal" binary floating-point instruction, and emulate the execution of that instruction such that the application logic is unaware of the emulation. This emulation comes at a significant cost to application performance, and should only be used under special circumstances. For example, to run exactly the same executable object module on backup processors within your organization, or because you make incidental use of binary floating-point numbers.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition. Only compatible subprograms are included in a given partition. Compatible subprograms have the same floating-point mode, and the same values for the FLOAT suboptions, and the ROUND and STRICT options:

- Floating-point mode (binary or hexadecimal)

  The floating-point mode for a partition is set to the floating-point mode (binary or hexadecimal) of the first subprogram that is placed in the partition. Subprograms that follow are placed in partitions that have the same floating-point mode; a binary floating-point mode subprogram is placed in a binary floating-point mode partition, and a hexadecimal mode subprogram is placed in a hexadecimal mode partition.

If you specify `FLOAT(HEX)` or `FLOAT(IEEE)` during the IPA Link step, the option is accepted, but ignored. This is because it is not possible to change the floating-point mode after source analysis has been performed.

The Prolog and Partition Map sections of the IPA Link step listing display the setting of the floating-point mode.

- `AFP | NOAFP`

  The value of `AFP` for a partition is set to the `AFP` value of the first subprogram that is placed in the partition. Subprograms that have the same `AFP` value are then placed in that partition.

  You can override the setting of `AFP` by specifying the suboption on the IPA Link step. If you do so, all partitions will contain that value, and the Prolog section of the IPA Link step listing will display the value.

  The Partition Map section of the IPA Link step listing and the END information in the IPA object file display the current value of the `AFP` suboption.

- `FOLD | NOFOLD`

  Hexadecimal floating-point mode partitions are always set to `FOLD`.

  For binary floating-point partitions, the value of `FOLD` for a partition is set to the `FOLD` value of the first subprogram that is placed in the partition. Subprograms that have the same `FOLD` value are then placed in that partition. During IPA inlining, subprograms with different `FOLD` settings may be combined in the same partition. When this occurs, the resulting partition is always set to `NOFOLD`.

  You can override the setting of `FOLD | NOFOLD` by specifying the suboption on the IPA Link step. If you do so, all binary floating-point mode partitions will contain that value, and the Prolog section of the IPA Link step listing will display the value.

  For binary floating-point mode partitions, the Partition Map section of the IPA Link step listing displays the current value of the `FOLD` suboption.

- `MAF | NOMAF`

  For IPA object files generated with the `FLOAT(IEEE)` option, the value of `MAF` for a partition is set to the `MAF` value of the first subprogram that is placed in the partition. Subprograms that have the same `MAF` for this suboption are then placed in that partition.

  For IPA object files generated with the `FLOAT(IEEE)` option, you can override the setting of `MAF | NOMAF` by specifying the suboption on the IPA Link step. If you do so, all binary floating-point mode partitions will contain that value, and the Prolog section of the IPA Link step listing will display the value.

  For binary floating-point mode partitions, the Partition Map section of the IPA Link step listing displays the current value of the `MAF` suboption.

  Hexadecimal mode partitions are always set to `NOMAF`. You cannot override this setting.

- `RRM | NORRM`

  For IPA object files generated with the `FLOAT(IEEE)` option, the value of `RRM` for a partition is set to the `RRM` value of the first subprogram that is placed in the partition. During IPA inlining, subprograms with different `RRM` settings may be combined in the same partition. When this occurs, the resulting partition is always set to `RRM`.

For IPA object files generated with the FLOAT(IEEE) option, you can override the setting of RRM | NORRM by specifying the suboption on the IPA Link step. If you do so, all binary floating-point mode partitions will contain that value, and the Prolog section of the IPA Link step listing will display the value.

For binary floating-point mode partitions, the Partition Map section of the IPA Link step listing displays the current value of the RRM suboption.

Hexadecimal mode partitions are always set to NORRM. You cannot override this setting.

- ROUND option

  For IPA object files generated with the FLOAT(IEEE) option, the value of the ROUND option for a partition is set to the value of the first subprogram that is placed in the partition.

  You can override the setting of ROUND by specifying the option on the IPA Link step. If you do so, all binary floating-point mode partitions will contain that value, and the Prolog section of the IPA Link step listing will display the value.

  For binary floating-point mode partitions, the Partition Map section of the IPA Link step listing displays the current value of the ROUND suboption.

  Hexadecimal mode partitions are always set to *round towards zero*. You cannot override this setting.

- STRICT option

  The value of the STRICT option for a partition is set to the value of the first subprogram that is placed in the partition. During IPA inlining, subprograms with different STRICT settings may be combined in the same partition. When this occurs, the resulting partition is always set to STRICT.

  You can override the setting of STRICT by specifying the option on the IPA Link step. If you do so, the Prolog section of the IPA Link step listing will display the value.

  If there are no Compilation Units with subprogram-specific STRICT options, all partitions will have the same STRICT value.

  If there are any Compilation Units with subprogram-specific STRICT options, separate partitions will continue to be generated for those subprograms with a STRICT option, which differs from the IPA Link option.

  The Partition Map sections of the IPA Link step listing and the object module display the value of the STRICT option.

Note: The inlining of subprograms (C functions, C++ functions and methods) is inhibited if the FLOAT suboptions (including the floating-point mode), and the ROUND and STRICT options are not all compatible between compilation units. Calls between incompatible compilation units result in reduced performance. For best performance, compile your applications with consistent options.

GENPCH | NOGENPCH

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
NOGENPCH	NOGENPCH(./)	NOGENPCH(./)	NOGENPCH(./)			

CATEGORY: File Management

```
►►──┬─GENP───┬──┬──────────────────────────────────────────┬──────────────►◄
    └─NOGENP─┘  └─(──┬─Sequential filename──────────┬──)─┘
                     ├─Partitioned data set─────────┤
                     ├─Partitioned data set (member)┤
                     ├─Hierarchical filename────────┤
                     └─Hierarchical directory───────┘
```

The GENP option creates precompiled header files. If you specify the GENP option, the compiler generates a precompiled header, even if one already exists.

If you specify the GENP and USEP options together, the compiler determines if the file exists. If it does, the compiler updates the file if necessary, and USEP takes effect. If it does not exist, the compiler creates the file, and USEP takes effect. If you consistently use both options, for example by coding them in your JCL, you can ensure that you are always using current precompiled header files.

If you specify GENP(*filename*), the compiler places the precompiled header data in the specified file. If you do not specify a file name for the GENP option, the compiler uses the SYSCPCH ddname if you allocated one. If you did not allocate SYSCPCH, the compiler constructs the file name as follows:

- If you are compiling a data set, the compiler uses the source file name to form the name of the precompiled header file data set. The high-level qualifier is replaced with the userid under which the compiler is running, and PCH (for C) or PCHPP (for C++) is appended as the low-level qualifier.
- If the source file is an HFS file, the compiler writes the precompiled header file to a file that has the name of the source file with a .pch (for C) or .pchpp (for C++) extension in the current working directory.

For more information on using GENP and USEP together, see “Using the GENP and USEP Compiler Options” on page 325.

Notes:

1. The compiler ignores GENP if you specify the options PPONLY, SHOWINC, or EXPMAC. For further information on these options, see “PPONLY | NOPPONLY” on page 160, “SHOWINC | NOSHOWINC” on page 171, and “EXPMAC | NOEXPMAC” on page 104.
2. You cannot use a C precompiled header file for C++, or a C++ precompiled header file for C.
3. If you specify different file names with the GENP and USEP options, the compiler uses the last specified file name with both options. For further information, see “USEPCH | NOUSEPCH” on page 194.

Effect on IPA Compile Step

The GENP option has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

The IPA Link Step accepts the GENP option, but ignores it.

GOFF | NOGOFF

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOGOFF						

CATEGORY: Object Code Control

```
►►──┬─GOFF───┬──────────────────────────────────────────────────►◄
    └─NOGOFF─┘
```

The GOFF option instructs the compiler to produce an object file in the Generalized Object File Format (GOFF). The GOFF format supersedes the S/370 Object Module and Extended Object Module formats. It removes various limitations of the previous format (for example, 16 Megabyte section size) and provides a number of useful extensions, including native z/OS support for long names and attributes. GOFF incorporates some aspects of industry standards such as XCOFF and ELF.

When you specify the GOFF option, the compiler uses LONGNAME and CSECT() by default. You can override these default values by explicitly specifying the NOLONGNAME or the CSECT option.

When you specify the GOFF option, you must use the binder to bind the output object. You cannot use the prelinker to process GOFF objects.

Effect on IPA Compile Step

The IPA Compile step does not support the NOLONGNAME option, nor #pragma nolongname. If this is attempted, the compilation will terminate with a diagnostic message.

The GOFF option affects the regular object module if you request one by specifying the IPA(OBJECT) option. This option affects the IPA-optimized object module generated when you specify the IPA(OBJECT) option.

The IPA information in an IPA object file is always generated using the XOBJ format.

Effect on IPA Link Step

The IPA Link Step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition. The GOFF option affects the object format of the code and data generated for each partition.

Information from non-IPA object modules is formatted based on the original format. GOFF format information remains in GOFF format, but all others are passed in XOBJ format.

GONUMBER | NOGONUMBER

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOGONUMBER	NOGONUMBER	NOGONUMBER	NOGONUMBER	NOGONUMBER	NOGONUMBER	NOGONUMBER

CATEGORY: Debug/Diagnostic

```
►►──┬─GONUM───┬──────────────────────────────────────────────────►◄
    └─NOGONUM─┘
```

The `GONUMBER` option generates line number tables that correspond to the input source file. These tables are for use by the Debug Tool and for error trace back information when an exception occurs.

The compiler turns on this option when you use the `TEST` option.

Note: When you specify the `GONUMBER` option, a comment that indicates its use is generated in your object module to aid you in diagnosing your program.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

If you specify the `GONUMBER` option on the IPA Compile step, the compiler saves information about the source file line numbers in the IPA object file. The `GONUMBER` and `LIST` options use this information during the IPA Link step.

If you do not specify the `GONUMBER` option on the IPA Compile step, the object file produced contains the line number information for source files that contain function begin, function end, function call, and function return statements. This is the minimum line number information that the IPA Compile step produces. You can then use the `TEST` option on the IPA Link step to generate corresponding test hooks

In the z/OS UNIX System Services environment, this option is turned on by specifying `-g` when using the `c89`, `cc` or `c++` commands.

Effect on IPA Link Step

If you specify the `GONUMBER` option for the IPA Link step, the IPA Link step creates GONUMBER tables during code generation. The level of detail in these tables depends on the options that you used for the IPA Compile step :

- If you specified the `GONUMBER`, `LIST`, `IPA(GONUMBER)`, or `IPA(LIST)` option on the IPA Compile step, the GONUMBER tables contain complete information.
- If you did not specify any of these options on the IPA Compile step, the source file and line number information in the IPA Link listing or GONUMBER tables consists only of the following:
  – function entry, function exit, function call, and function call return source lines. This is the minimum line number information that the IPA Compile step produces.
  – All other object code statements have the file and line number of the function entry, function exit, function call, and function call return that was last encountered. This is similar to the situation of encountering source statements within a macro.

Refer to "Interactions between Compiler Options and IPA Suboptions" on page 61 and "LIST | NOLIST" on page 133 for more information.

HALT(num)

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
HALT(16)	HALT(16)	HALT(16)	HALT(16)	HALT(16)	HALT(16)	HALT(16)

CATEGORY: Source Code Control

```
►►──HALT──(num)──────────────────────────────────────────────────►◄
```

The `HALT` option stops compilation, depending on the return code from the compiler. This option applies to the compilation of all members of a PDS or an HFS directory. If the return code from compiling a particular member is greater than or equal to the value *num* specified in the `HALT` option, no more members are compiled.

Valid codes for *num* correspond to return codes from the compiler. See "Appendix G. z/OS C/C++ Compiler Return Codes and Messages" on page 591 for a list of return codes.

Effect on IPA Compile Step

The `HALT` option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The `HALT` option affects the IPA Link step in a way similar to the way it affects the IPA Compile step, but the message severity levels may be different. Also, the severity levels for the IPA Link step and a C++ compilation include the ″unrecoverable″ level.

IGNERRNO | NOIGNERRNO

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOIGNERRNO						

CATEGORY: Object Code Control

```
►►──┬─IGNER───┬──────────────────────────────────────────────────►◄
    └─NOIGNER─┘
```

The `IGNERRNO` option informs the compiler that your application is not using `errno`. Specifying this option allows the compiler to explore additional optimization opportunities for library functions in `LIBANSI`.

ANSI library functions use `errno` to return the error condition. If your program does not use `errno`, the compiler has more freedom to explore optimization opportunities for some of these functions (for example, `sqrt()`). You can control this optimization by using the `IGNERRNO` option.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. The `IGNERRNO` option also affects the regular object module if you request one by specifying the `IPA(OBJECT)` option.

Effect on IPA Link Step

The IPA Link step accepts the `IGNERRNO` option, but ignores it. The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition. Only compatible subprograms are included in a given partition. Compatible subprograms have the same `IGNERRNO` option setting. For the purpose of this compatibility checking, objects produced by compilers prior to Version 2 Release 9, where `IGNERRNO` is not supported, are considered `NOIGNERRNO`.

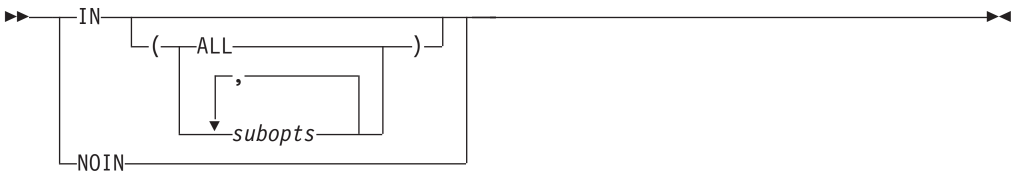
The value of the `IGNERRNO` option for a partition is set to the value of the first subprogram that is placed in the partition. The Partition Map sections of the IPA Link step listing and the object module display the value of the `IGNERRNO` option.

INFO | NOINFO

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOINFO			NOINFO			

CATEGORY: Debug/Diagnostic



The INFO option instructs the compiler to generate warning messages. Use *subopts* if you want to specify the type of warning messages.

If you specify INFO with no suboptions, it is the same as specifying INFO(ALL). The following is a list of the *subopts*:

CLS	Emits class informational warning messages.
CMP	Emits conditional expression check messages.
CND	Emits messages on redundancies or problems in conditional expressions.
CNV	Emits messages about conversions.
CNS	Emits redundant operation on constants messages.
CPY	Emits warnings about copy constructors.
EFF	Emits information about statements with no effect.
ENU	Emits information about ENUM checks.
GNR	Emits information about the generation of temporary variables.
GEN	Emits message if compiler generates temporaries.
LAN	Emits language level checks.
PAR	Emits warning messages on unused parameters.
POR	Emits warnings about nonportable constructs.
PPC	Emits messages on possible problems with using the preprocessor.
PPT	Emits trace of preprocessor actions.
REA	Emits warnings about unreached statements.

RET	Emits warnings about return statement consistency.
TRD	Emits warnings about possible truncation of data.
UND	Emits warnings about undefined classes.
USE	Emits information about usage of variables.
VFT	Indicates where `vftable` is generated.
ALL	Emits all of the above

no suboptions
Same result as `INFO(ALL)`.

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c++` command. The suboption is `ALL` when `-V` is specified.

Effect on IPA Compile Step

The `INFO` option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step issues a diagnostic message if you specify the `INFO` option.

INITAUTO | NOINITAUTO

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOINITAUTO						

CATEGORY: Object Code Control

```
►►──┬─INITA──(──nnnnnnnn──┬────────┬──)─┬──────────────────────────►◄
    │                     └─, WORD─┘    │
    └─NOINITA───────────────────────────┘
```

The `INITAUTO` option tells the compiler to generate code to initialize automatic variables. Automatic variables require storage only while the block in which they are declared is active. See the *z/OS C/C++ Language Reference* for more information on automatic variables.

Automatic variables without initializers are not implicitly initialized. The `INITAUTO` option instructs the compiler to generate code to initialize these variables with a user-defined value.

In the above syntax, the hexadecimal value you specify for *nnnnnnnn* represents the initial value for automatic storage in bytes. It can be two to eight hexadecimal digits in length. There is no default for this value.

The suboption `Word` is optional, and can be abbreviated to `W`. If you specify `Word`, *nnnnnnnnn* is a word initializer; otherwise it is a byte initializer. Only one initializer can be in effect for the compilation. If you specify `INITAUTO` more than once, the compiler uses the last setting.

If you specify a byte initializer, and specify more than 2 digits for *nnnnnnnn*, the compiler uses the last 2 digits. If you specify a word initializer, the compiler uses the last 2 digits to initialize a byte, and all digits to initialize a word.

Note: The word initializer is useful in checking uninitialized pointers.

Since extra code is generated, this option can reduce the runtime performance of the program.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. The `INITAUTO` option also affects the regular object module if you request one by specifying the `IPA(OBJECT)` option.

Effect on IPA Link Step

You can specify the `INITAUTO` option for an IPA link step, and it will override the setting in the compile step.

If you do not specify the `INITAUTO` option in the IPA link step, the setting in the IPA compile step will be used. The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition. Only compatible subprograms are included in a given partition. Compatible subprograms have the same `INITAUTO` setting.

The IPA Link steps sets the `INITAUTO` setting for a partition to the specification of the first subprogram that is placed in the partition. It places subprograms that follow in partitions that have the same `INITAUTO` setting.

You can override the setting of `INITAUTO` by specifying the option on the IPA Link step. If you do so, all partitions will use that value, and the Prolog section of the IPA Link step listing will display the value.

The Partition Map sections of the IPA Link step listing and the object module display the value of the `INITAUTO` option.

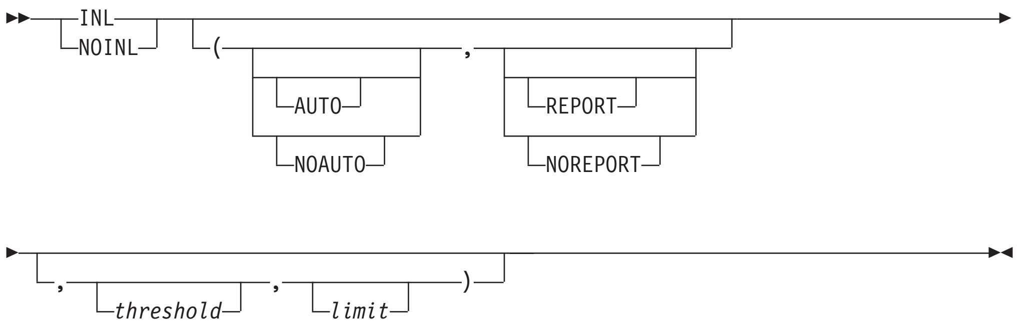
INLINE | NOINLINE

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓		✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities** **Note that these values change if the `-0` flag is set.**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**

C Compile:
 	
If `NOOPT` is in effect:
`NOINLINE (AUTO,NOREPORT, 100,1000)`
 	
`NOOPT` is the default for C compile
 	
If `OPT` is in effect:
`INLINE (AUTO,NOREPORT, 100,1000)`
 	
IPA Link:
 	
If `NOOPT` is in effect:
`NOINLINE (AUTO,NOREPORT, 1000,8000)`
 	
If `OPT` is in effect:
`INLINE (AUTO,NOREPORT, 1000,8000)`
 	
`OPT` is the default for IPA Link.
	For `NOOPT`:
	`NOINLINE (AUTO, NOREPORT, 100, 1000)`
	
	For `OPT`:
	`INLINE (AUTO, NOREPORT, 100, 1000)`
		For `NOOPT`:
		`NOINLINE (AUTO, NOREPORT, 100, 1000)`
		
		For `OPT`:
		`INLINE (AUTO, NOREPORT, 100, 1000)`
				`NOINLINE (AUTO, NOREPORT, ,)`	`NOINLINE (AUTO, NOREPORT, ,)`	`NOINLINE (AUTO, NOREPORT, ,)`

CATEGORY: Object Code Control



The `INLINE` option instructs the compiler to place the code for selected subprograms at the point of call; this is called *inlining.* It eliminates the linkage overhead and exposes the entire inlined subprogram for optimization by the global optimizer. It has the following effects:

- The compiler invokes the compilation unit inliner to perform inlining of functions within the current compilation unit.
- If the compiler inlines all invocations of a static subprogram, it removes the non-inlined instance of the subprogram.

- If the compiler inlines all invocations of an externally visible subprogram, it does not remove the non-inlined instance of the subprogram. This allows callers who are outside of the current compilation unit to invoke the non-inlined instance.
- If you specify `INLINE(,REPORT,,)` or `INLRPT`, the compiler generates the Inline Report listing section.

For more information on optimization and the `INLINE` option, refer to the section about optimizing code in the *z/OS C/C++ Programming Guide*.

You can specify `INLINE` without suboptions if you want to use the defaults. You must include a comma between each suboption even if you want to use the default for one of the suboptions. You must specify the suboptions in the following order:

`AUTO` | `NOAUTO`
: The inliner runs in automatic mode and inlines subprograms within the *threshold* and *limit*.

  If you specify `NOAUTO`, the inliner only inlines those subprograms specified with the `#pragma inline` directive. The `#pragma inline` and `#pragma noinline` directives allow you to determine which subprograms are to be inlined and which are not when the `INLINE` option is specified. These `#pragma` directives have no effect if you specify `NOINLINE`. See the *z/OS C/C++ Language Reference* for more information on `#pragma` directives.

  The default is `AUTO`

`REPORT` | `NOREPORT`
: An inline report becomes part of the listing file. The inline report consists of the following:
  - An inline summary
  - A detailed call structure

  You can obtain the same report if you use the `INLRPT` and `OPT` options. For more information on the inline report, see "Inline Report" on page 255, "Inline Report" on page 242, and "Inline Report for IPA Inliner" on page 265.

  The default is `NOREPORT`

threshold
: The maximum relative size of a subprogram to inline. For C compile, the default for *threshold* is 100 Abstract Code Units (ACU) instructions. For the IPA Link step, the default for *threshold* is 1000 ACUs. ACUs are proportional in size to the executable code in the subprogram; the z/OS C compiler translates your z/OS C code into ACUs. The maximum *threshold* is `INT_MAX`, as defined in the header file `LIMITS.H`. Specifying a threshold of `0` is the same as specifying `NOAUTO`.

limit
: The maximum relative size a subprogram can grow before auto-inlining stops. For C compile, the default for *limit* is 1000 ACUs for a subprogram. For the IPA Link step, the default for *limit* is 8000 ACUs for that subprogram. The maximum for *limit* is `INT_MAX`, as defined in the header file `LIMITS.H`. Specifying a limit of `0` is equivalent to specifying `NOAUTO`.

You can specify the `INLINE` | `NOINLINE` option on the invocation line and in the `#pragma options` preprocessor directive. When you use both methods at the same time, the compiler merges the options. If an option on the invocation line conflicts with an option in the `#pragma options` directive, the one on the invocation line takes precedence.

For example, because you typically do not want to inline your subprograms when you are developing a program, you can specify the NOINLINE option on a #pragma options preprocessor directive. When you want to inline your subprograms, you can override the NOINLINE option by specifying INLINE on the invocation line rather than by editing your source program. The following example illustrates these rules.

Source file:
```
#pragma options (NOINLINE(NOAUTO,NOREPORT,,2000))
```

Invocation line:
```
INLINE (AUTO,,,)
```

Result:
```
INLINE (AUTO,NOREPORT,100,2000)
```

Notes:

1. When you specify the INLINE compiler option, a comment, with the values of the suboptions, is generated in your object module to aid you in diagnosing your program.
2. If the compiler option OPT is specified, INLINE becomes the default.
3. Specify the INLRPT, LIST, or SOURCE compiler options to redirect the output from the INLINE(,REPORT,,) option.
4. If you specify INLINE and TEST:
   at OPT(0), INLINE is ignored.
   at OPT, inlining is done
5. C++ code is always inlined at OPT
6. If you specify NOINLINE, no subprograms will be inlined even if you have #pragma inline directives in your code.
7. If you specify INLINE, subprograms may not be inlined or inline other subprograms when COMPACT is specified (either directly or via #pragma option_override). Generate and check the inline report to determine the final status of inlining. The inlining may not occur when OPT(0) is specified via the #pragma option_override.

You can specify this option using the #pragma option directive for C.

In the z/OS UNIX System Services environment, the INLINE(,REPORT,,) option is turned on by specifying -V when using the c89 or cc commands.

Effect on IPA Compile Step

The INLINE option generates inlined code for the regular compiler object; therefore, it affects the IPA Compile step only if you specify IPA(OBJECT). If you specify IPA(NOOBJECT), INLINE has no effect, and there is no reason to use it.

Effect on IPA Link Step

If you specify the INLINE option on the IPA Link step, it has the following effects:

- The IPA Link step invokes the IPA inliner, which inlines subprograms (functions and C++ methods) in the entire program.
- The IPA Link step uses #pragma inline|noinline directive information and inline subprogram specifier information from the IPA Compile step for source program inlining control. Specifying the INLINE option on the IPA Compile step has no effect on IPA Link step inlining processing.

  You can use the IPA Link control file inline and noinline directives to explicitly control the inlining of subprograms on the IPA Link step. These directives override IPA Compile step #pragma inline | noinline directives and inline subprogram specifiers.

- If the IPA Link step inlines all invocations of a subprogram, it removes the non-inlined instance of the subprogram, unless the subprogram entry point was exported using a `#pragma export` directive or the `EXPORTALL` compiler option, or was retained using the IPA Link control file `retain` directive. IPA Link processes static subprograms and externally visible subprograms in the same manner.

The IPA inliner has the inlining capabilities of the compilation unit inliner. In addition, the IPA inliner detects complex recursion, and may inline it. If you specify the `INLRPT` option, the IPA Link listing contains the IPA Inline Report section. This section is similar to the report that the compilation unit inliner generates. If you specify `NOINLINE(,REPORT,,)` or `NOINLINE INLRPT`, IPA generates an IPA Inline Report section that specifies that nothing was inlined.

INLRPT | NOINLRPT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOINLRPT`			`NOINLRPT (/dev/fd1)`			

CATEGORY: Listing

```
►►──┬─INLR───┬──┬─────────────────────────────────────────────┬──────────────►◄
    └─NOINLR─┘  └─(──┬─Sequential filename──────────────┬─)─┘
                     ├─Partitioned data set─────────────┤
                     ├─Partitioned data set (member)────┤
                     ├─Hierarchical filename────────────┤
                     └─Hierarchical directory───────────┘
```

If you use the `OPTIMIZE` option, you can also use `INLRPT` to specify that the compiler generate a report as part of the compiler listing. The report provides the status of subprograms that were inlined, specifies whether they were inlined or not and displays the reasons for the compiler's action.

You can specify *filename* for the inline report output file. If you do not specify *filename*, the compiler uses the `SYSCPRT` ddname if you allocated one. If you did not allocate `SYSCPRT`, the compiler uses the source file name to generate a file name.

The `NOINLR` option can optionally take a *filename* suboption. This *filename* then becomes the default. If you subsequently use the `INLR` option without *filename*, the compiler uses the *filename* that you specified in the earlier specification or `NOINLR`. For example,

```
CXX HELLO (NOINLR(/hello.lis) INLR OPT
```

is the same as specifying:

```
CXX HELLO (INLR(/hello.lis) OPT
```

Note: If you specify *filename* with any of the SOURCE, LIST, or INLRPT options, all the listing sections are combined into the last *filename* specified.

If you specify this multiple times, the compiler uses the last specified option with the last specified suboption. The following two specifications have the same result:

1. `CXX HELLO (NOINLR(/hello.lis) INLR(/n1.lis)  NOINLR(/test.lis) INLR`
2. `CXX HELLO (INLR(/test.lis)`

In the z/OS UNIX System Services environment, this option is turned on by specifying -V when using the c++ command.

Effect on IPA Compile Step

The INLRPT option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

If you specify the INLRPT option on the IPA Link step, the IPA Link step listing contains an IPA Inline Report section. Refer to "INLINE | NOINLINE" on page 120 for more information about generating an IPA Inline Report section.

IPA | NOIPA

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOIPA	NOIPA	NOIPA	NOIPA	NOIPA(NOCONTROL (ipa.ctl), DUP,NOER, NOMAP, NOUPCASE, NONCAL) IPA(LINK, LEVEL(1))	NOIPA(NOCONTROL (ipa.ctl), DUP,NOER, NOMAP, NOUPCASE, NONCAL) IPA(LINK, LEVEL(1))	NOIPA(NOCONTROL (ipa.ctl), DUP,NOER, NOMAP, NOUPCASE, NONCAL) IPA(LINK, LEVEL(1))

CATEGORY: Object Code Control/IPA Link Control

```
►►──┬─IPA───┬──┬──────────────────────────────────────────────────────────┬──►◄
    └─NOIPA─┘  │   ┌─,──────────────────────────────────────────────┐     │
               │   ▼                                                │     │
               └─(─┬─NOLINK │ LINK───────────────────────────────┬─┴─)───┘
                   ├─OBJ │ NOOBJ │ OBJONLY──────────────────────┤
                   ├─ATTR │ NOATT───────────────────────────────┤
                   ├─COM│NOCOM──────────────────────────────────┤
                   ├─GONUM │ NOGONUM────────────────────────────┤
                   ├─LIS │ NOLIS────────────────────────────────┤
                   ├─OPT │ NOPT─────────────────────────────────┤
                   ├─XR │ NOXR──────────────────────────────────┤
                   ├─LEVEL──┬─(0)─┬─────────────────────────────┤
                   │        ├─(1)─┤                             │
                   │        └─(2)─┘                             │
                   ├─CONTROL │ NOCONTROL──┬──────────────────┬──┤
                   │                      └─(─fileid─)───────┘  │
                   ├─DUP │ NODUP────────────────────────────────┤
                   ├─ER │ NOER──────────────────────────────────┤
                   ├─MAP │ NOMAP────────────────────────────────┤
                   ├─NCAL │ NONCAL──────────────────────────────┤
                   └─UPCASE │ NOUPCASE──────────────────────────┘
```

The IPA option instructs the compiler to perform Interprocedural Analysis across compilation units.

The NOIPA option instructs the compiler to perform a regular compilation.

IPA Compile Step Suboptions

IPA(NOLINK) invokes the IPA Compile step. NOLINK is the default suboption of the IPA option. Only the following IPA suboptions affect the IPA Compile step. You can specify other IPA suboptions, but they do not affect the IPA Compile step.

ATTRIBUTE | NOATTRIBUTE	Indicates whether the compiler saves information about symbols in the IPA object file. The IPA Link step uses this information if you specify the ATTR or XREF option on that step. The difference between specifying IPA(ATTR) and specifying ATTR or XREF is that IPA(ATTR) does not generate a Cross Reference or Static Map listing sections after IPA Compile step source analysis is complete. It also does not generate a Storage Offset, Static Map, or External Symbol Cross Reference listing section during IPA Compile step code generation. The default is IPA(NOATTRIBUTE). The abbreviations are IPA(ATTR|NOATTR). If you specify the ATTR or XREF option, it overrides the IPA(NOATTRIBUTE) option.
COMPRESS | NOCOMPRESS	Indicates that the IPA object information is compressed to significantly reduce the size of the IPA object file. The default is IPA(COMPRESS). The abbreviations are IPA(COM|NOCOM).
GONUMBER |NOGONUMBER	Indicates whether the compiler saves information

	about source file line numbers in the IPA object file. The difference between specifying `IPA(GONUMBER)` and `GONUMBER` is that `IPA(GONUMBER)` does not cause GONUMBER tables to be built during IPA Compile step code generation. If the compiler does not build GONUMBER tables, the size of the object module is smaller. Refer to "GONUMBER | NOGONUMBER" on page 115 for information about the effect of this suboption on the IPA Link step. Refer also to "Interactions between Compiler Options and IPA Suboptions" on page 61. The default is `IPA(NOGONUMBER)`. The abbreviations are `IPA(GONUM|NOGONUM)`. If you specify the `GONUMBER` or `LIST` option, it overrides the `IPA(NOGONUMBER)` option.
`LIST | NOLIST`	Indicates whether the compiler saves information about source line numbers in the IPA object file. The difference between specifying `IPA(LIST)` and `LIST` is that `IPA(LIST)` does not cause the IPA Compile step to generate a Pseudo Assembly listing. Refer to "LIST | NOLIST" on page 133 for information about the effect of this suboption on the IPA Link step. Refer also to "Interactions between Compiler Options and IPA Suboptions" on page 61. The default is `IPA(NOLIST)`. The abbreviations are `IPA(LIS|NOLIS)`. If you specify the `GONUMBER` or `LIST` option, it overrides the `IPA(NOLIST)` option.
`OBJECT | NOOBJECT | OBJONLY`	Controls the content of the object file. • `OBJECT` The options `IPA(NOLINK,OBJECT)` result in an IPA Compile step. The compiler performs IPA compile-time optimizations and generates IPA object information for the resulting program information. In addition, the compiler generates non-IPA object code and data that is based on the original program information. Refer to the *z/OS C/C++ Programming Guide* for a list of optimizations. The object file may be used by an IPA Link step, a prelink/link, or a bind. • `NOOBJECT` The options `IPA(NOLINK,NOOBJECT)` result in an IPA Compile step. The compiler performs IPA compile-time optimizations and generates IPA object information for the resulting program information. No non-IPA object code or data is generated.

The object file may be used by an IPA Link step only.

- OBJONLY

  The IPA(OBJONLY) compilation is an intermediate level of optimization. This results in a modified regular compile, not an IPA Compile step. Unlike the IPA Compile step, no IPA information is written to the object file.

  During compilation, this step performs the same IPA-specific compile-time optimizations as the IPA Compile step, performs the requested non-IPA optimizations, and then generates optimized object code and data.

  The object file may be used by an IPA Link step, a prelink/link, or a bind. If it is used as input to an IPA Link step, no IPA link-time optimizations can be performed for this compilation unit because no IPA information is available.

  For all options in this mode, the *Effect on IPA Compile Step* and *Effect on IPA Link Step* considerations do not apply.

The default is IPA(OBJECT). The abbreviations are IPA(OBJ|NOOBJ|OBJO).

OPTIMIZE | NOOPTIMIZE	The default is IPA(OPTIMIZE). If you specify IPA(NOOPTIMIZE), the compiler issues an informational message and turns on IPA(OPTIMIZE). The abbreviations are IPA(OPT|NOOPT) . IPA(OPTIMIZE) generates information (in the IPA object file) that will be needed by the OPT compiler option during IPA Link processing. If you specify the IPA(OBJECT), the IPA(OPTIMIZE), and the NOOPTIMIZE option during the IPA Compile step, the compiler creates a non-optimized object module for debugging. If you specify the OPT(1) or OPT(2) option on a subsequent IPA Link step, you can create an optimized object module without first rerunning the IPA Compile step.
XREF | NOXREF	Indicates whether the compiler save information about symbols in the IPA object file that will be used in the IPA Link step if you specify ATTR or XREF on that step. The difference between specifying IPA(XREF) and specifying ATTR or XREF is that IPA(XREF) does not cause the compiler to generate a Cross Reference or Static Map listing sections after IPA Compile step source analysis is complete. It also does not cause the compiler to generate a Storage Offset, Static Map, or External Symbol Cross Reference listing section during IPA Compile step code generation.

Refer to “XREF | NOXREF” on page 198 for information about the effects of this suboption on the IPA Link step.

The default is IPA(NOXREF). The abbreviations are IPA(XR|NOXR). If you specify the ATTR or XREF option, it overrides the IPA(NOXREF) option.

IPA Link Step Suboptions

IPA(LINK) invokes the IPA Link step. Only the following IPA suboptions affect the IPA Link step. If you specify other IPA suboptions, they do not affect the IPA Link step.

CONTROL[(fileid)] | NOCONTROL[(fileid)]

Specifies whether a file that contains IPA directives is available for processing. You can specify an optional fileid. If you specify both IPA(NOCONTROL(*fileid*)) and IPA(CONTROL), in that order, the IPA Link step resolves the option to IPA(CONTROL(*fileid*)).

The default fileid is DD:IPACNTL if you specify the IPA(CONTROL) option. The default is IPA(NOCONTROL).

For more information about the IPA control file directives, refer to “IPA Link Step Control File” on page 339.

DUP | NODUP

Indicates whether the IPA Link step writes a message and a list of duplicate symbols to the console.

The default is IPA(DUP).

ER | NOER

Indicates whether the IPA Link step writes a message and a list of unresolved symbols to the console.

The default is IPA(NOER).

LEVEL(0|1|2)

Indicates the level of IPA optimization that the IPA Link step should perform after it links the object files into the call graph.

If you specify LEVEL(0), IPA performs subprogram pruning and program partitioning only.

If you specify LEVEL(1), IPA performs all of the optimizations that it does at LEVEL(0), as well as subprogram inlining and global variable coalescing. IPA performs more precise alias analysis for pointer dereferences and subprogram calls.

Under IPA Level 1, many optimizations such as constant propagation and pointer analysis are performed at the intraprocedural (subprogram) level. If you specify LEVEL(2), IPA performs specific optimizations across the entire program, which can lead to significant improvement in the generated code.

The compiler option OPTIMIZE that you specify on the IPA Link step controls subsequent optimization for each partition during code generation. Regardless of the optimization level you specified during the IPA Compile step, you can request IPA optimization, regular code generation optimization, both, or neither, on the IPA Link step.

The default is IPA(LEVEL(1)).

`MAP | NOMAP`
: Specifies that the IPA Link step will produce a listing. The listing contains a Prolog and the following sections:
- Object File Map
- Source File Map
- Compiler Options Map
- Global Symbols Map
- Partition Map for each partition

The default is `IPA(NOMAP)`.

See "Using the IPA Link Step Listing" on page 256 for more information.

`NCAL | NONCAL`
: Indicates whether the IPA Link step performs an automatic library search to resolve references in files that the IPA Compile step produces. Also indicates whether the IPA Link step performs library searches to locate an object file or files that satisfy unresolved symbol references within the current set of object information.

This suboption controls both explicit searches triggered by the LIBRARY IPA Link control statement, and the implicit SYSLIB search that occurs at the end of IPA Link input processing.

To help you remember the difference between `NCAL` and `NONCAL`, you may wish to think of `NCAL` as "nocall" and `NONCAL` as "no nocall", (or "call").

The default is `IPA(NONCAL)`.

`UPCASE | NOUPCASE`
: Determines whether the IPA Link step makes an additional automatic library call pass for SYSLIB if unresolved references remain at the end of standard IPA Link processing. Symbol matching is not case sensitive in this pass.

This suboption provides support for linking assembler language object routines, without forcing you to make source changes. The preferred approach is to add `#pragma map` definitions for these symbols, so that the correct symbols are found during normal IPA Link automatic library call processing.

The default is `IPA(NOUPCASE)`. The abbreviations are `IPA(UPC|NOUPC)`.

Refer to the Interprocedural Analysis chapter in the *z/OS C/C++ Programming Guide* for an overview and more details about Interprocedural Analysis.

LANGLVL

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
LANGLVL(EXTENDED)	LANGLVL(ANSI)	LANGLVL (COMMONC)	LANGLVL (ANSI)			

CATEGORY: Source Code Control

```
►►──LANG──(──┬─ANSI─────┬──)──────────────────────────────────────────────►◄
             ├─SAA──────┤
             ├─SAAL2────┤
             ├─COMPAT───┤
             ├─EXTENDED─┤
             └─COMMONC──┘
```

The LANGLVL option defines a macro that specifies a language level. You must then include this macro in your code to force conditional compilation. For example, with the use of #ifdef directives. You can write portable code if you correctly code the different parts of your program according to the language level. You use the macro in preprocessor directives in header files. The LANGLVL suboptions are:

LANGLVL(ANSI)
: Indicates language constructs that are defined by ANSI. Some non-ANSI stub routines will exist even if you specify LANGLVL(ANSI), for compatibility with previous releases. The macro __ANSI__ is defined as 1.

Notes:

1. You cannot use the compiler options LANGLVL(ANSI) and NOEXH together, because NOEXH breaks ANSI support. If you specify either of the following, the compiler issues a warning message to indicate that it ignores NOEXH:
   - NOEXH LANGLVL(ANSI)
   - LANGLVL(ANSI) NOEXH
2. When you specify LANGLVL(ANSI), the compiler can still read and analyze the _Packed keyword in z/OS C. If you want to make your code purely ANSI, you should redefine _Packed in a header file as follows:
   ```
   #ifdef __ANSI__
     #define _Packed
   #endif
   ```
   The compiler will now see the _Packed attribute as a blank when LANGLVL(ANSI) is specified at compile time, and the language level of the code will be ANSI.

LANGLVL(COMPAT)
: Indicates that code is compiled to be compatible with older levels of C++. Module initialization occurs in link order. This suboption is only available under z/OS C++. The macro __COMPAT__ is defined as 1.

LANGLVL(COMMONC)
: Indicates language constructs that are defined by XPG, many of which LANGLVL(EXTENDED) already supports. LANGLVL(ANSI) and LANGLVL(EXTENDED) do not support the following, but LANGLVL(COMMONC) does:

- Unsignedness is preserved for standard integral promotions. That is, unsigned char is promoted to unsigned int.
- Trigraphs within literals are not processed
- `sizeof` operator is permitted on bitfields
- Bitfields other than `int` are tolerated, and a warning message is issued.
- Macro parameters within quotation marks are expanded
- Macros may be redefined without first being undefined
- The empty comment in a subprogram-like macro is equivalent to the ANSI/ISO token concatenation operator

The `COMMONC` suboption is available only for z/OS C. The macro __`COMMONC`__ is defined as 1 when you specify `LANGLVL(COMMONC)`.

If you specify `LANGLVL(COMMONC)`, the `ANSIALIAS` option is automatically turned off. If you want `ANSIALIAS` turned on, you must explicitly specify it.

Note: The option `ANSIALIAS` assumes code that supports ANSI. Using `LANGLVL(COMMONC)` and `ANSIALIAS` together may have undesirable effects on your code at a high optimization level. See “ANSIALIAS | NOANSIALIAS” on page 80 for more information.

`LANGLVL(EXTENDED)`
Indicates all language constructs available with z/OS C/C++. Enables extensions to the ANSI draft. The macro __`EXTENDED`__is defined as 1.

`LANGLVL(SAA)`
Indicates language constructs that are defined by SAA. This suboption is only available under z/OS C. See the *z/OS C/C++ Language Reference* for more information.

`LANGLVL(SAAL2)`
Indicates language constructs that are defined by SAA Level 2. This suboption is only available under z/OS C. See the *z/OS C/C++ Language Reference* for more information.

Effect on IPA Compile Step

The `LANGLVL` option has the same effect on the IPA Compile step as it does on regular compilation

Effect on IPA Link Step

The IPA Link Step accepts but ignores the `LANGLVL` option.

LIBANSI | NOLIBANSI

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOLIBANSI	NOLIBANSI	NOLIBANSI	NOLIBANSI			

CATEGORY: Code Optimization

```
►►──┬─LIB───┬──────────────────────────────────────────────────────►◄
    └─NOLIB─┘
```

The `LIBANSI` option indicates whether the functions with the name of an ANSI C library function are in fact ANSI C library functions. If you specify `LIBANSI`, the compiler generates code that is based on existing knowledge concerning the behaviour of the ANSI C library function. For example, whether or not any side effects are associated with a particular system function.

A comment that indicates the use of the `LIBANSI` option will be generated in your object module to aid you in diagnosing your program.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

If you specify the `LIBANSI` option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the `IPA(OBJECT)` option.

Effect on IPA Link Step

The `LIBANSI` option will be in effect for the IPA Link step unless the `NOLIBANSI` option is specified. The value of the `LIBANSI` option from the IPA compile step is ignored, but is shown in the IPA link listing Compile Option Map for reference.

LIST | NOLIST

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOLIST	NOLIST (/dev/fd1)	NOLIST (/dev/fd1)	NOLIST (/dev/fd1)	NOLIST (/dev/fd1)	NOLIST (/dev/fd1)	NOLIST (/dev/fd1)

CATEGORY: Listing

```
►►──┬─LIS───┬──┬──────────────────────────────────────────┬────────►◄
    └─NOLIS─┘  └─(──┬─Sequential filename────────────┬─)─┘
                    ├─Partitioned data set───────────┤
                    ├─Partitioned data set (member)──┤
                    ├─Hierarchical filename──────────┤
                    └─Hierarchical directory─────────┘
```

The `LIST` option instructs the compiler to generate a listing of the machine instructions in the object module (in a format similar to assembler language instructions) in the compiler listing.

`LIST`(*filename*) places the compiler listing in the specified file. If you do not specify a file name for the `LIST` option, the compiler uses the `SYSCPRT` ddname if you allocated one. Otherwise, the compiler generates a file name as follows:

- If you are compiling a data set, the compiler uses the source file name to form name of the listing data set. The high-level qualifier is replaced with the userid under which the compiler is running, and .LIST is appended as the low-level qualifier.
- If you are compiling an HFS file, the compiler stores the listing in a file that has the name of the source file with .lst extension.

The `NOLIST` option optionally takes a *filename* suboption. This *filename* then becomes the default. If you subsequently use the `LIST` option without a *filename* suboption, the compiler uses the *filename* that you specified in the earlier `NOLIST`. For example, the following specifications have the same effect:

```
CXX HELLO (NOLIST(/hello.lis) LIST
```

```
CXX HELLO (LIST(/hello.lis)
```

If you specify data set names in a C or C++ program, with the `SOURCE`, `LIST` or `INLRPT` options, all the listing sections are combined into the last data set name specified.

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c89`, `cc` or `c++` commands.

Notes:

1. Usage of information such as registers, pointers, data areas, and control blocks that are shown in the object listing are not programming interface information.
2. If you use the following form of the command in a JES3 batch environment where xxx is an unallocated data set, you may get undefined results.

   ```
   LIST(xxx)
   ```

Effect on IPA Compile Step

If you specify the `LIST` option on the IPA Compile step, the compiler saves information about the source file and line numbers in the IPA object file. This information is available during the IPA Link step for use by the `LIST` or `GONUMBER` options.

If you do not specify the `GONUMBER` option on the IPA Compile step, the object file produced contains the line number information for source files that contain function begin, function end, function call, and function return statements. This is the minimum line number information that the IPA Compile step produces. You can then use the `TEST` option on the IPA Link step to generate corresponding test hooks

Refer to “Interactions between Compiler Options and IPA Suboptions” on page 61 and “GONUMBER | NOGONUMBER” on page 115 for more information.

Effect on IPA Link Step

If you specify the `LIST` option, the IPA Link listing contains a Pseudo Assembly section for each partition that contains executable code. Data-only partitions do not generate a Pseudo Assembly listing section.

The source file and line number shown for each object code statement depend on the amount of detail the IPA Compile step saves in the IPA object file, as follows:

- If you specified the `GONUMBER`, `LIST`, `IPA(GONUMBER)`, or `IPA(LIST)` option for the IPA Compile step, the IPA Link step accurately shows the source file and line number information.

- If you did not specify any of these options on the IPA Compile step, the source file and line number information in the IPA Link listing or GONUMBER tables consists only of the following:
  - function entry, function exit, function call, and function call return source lines. This is the minimum line number information that the IPA Compile step produces.
  - All other object code statements have the file and line number of the function entry, function exit, function call, and function call return that was last encountered. This is similar to the situation of encountering source statements within a macro.

Refer to “Interactions between Compiler Options and IPA Suboptions” on page 61 and “GONUMBER | NOGONUMBER” on page 115 for more information.

LOCALE | NOLOCALE

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities** (These Utilities pick up the locale value of the environment using `setlocale(LC_ALL,NULL)`). Because the compiler runs with the `POSIX(OFF)` option, categories that are set to `C` are changed to `POSIX`.)					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
`NOLOCALE`	`LOCALE(POSIX)`	`LOCALE(POSIX)`	`LOCALE(POSIX)`	`LOCALE(POSIX)`	`LOCALE(POSIX)`	`LOCALE(POSIX)`

CATEGORY: Preprocessor

```
►►──┬─LOC──┬────────┬─┬─────────────────────────────────────────►◄
    │      └─(name)─┘ │
    └─NOLOC───────────┘
```

The `LOCALE` option specifies the locale to be used by the compiler as the current locale throughout the compilation unit. To specify a locale, use the following format:

`LOCALE(`*name*`)`

The suboption *name* indicates the name of the locale to be used by the compiler at compile time. If you omit *name*, the compiler uses the current default locale in the environment. If *name* does not represent a valid locale name, the compiler ignores the `LOCALE`, and assumes `NOLOCALE`.

`NOLOCALE` indicates that the compiler only uses the default code page, which is IBM-1047.

You cannot use the `LOCALE | NOLOCALE` option in the z/OS C `#pragma options` directive. You can only specify it on the command line or in the PARMS list in the JCL.

If you specify the LOCALE option, the locale name and the associated code set appear in the header of the listing. A locale name is also generated in the object module.

The LC_TIME category of the current locale controls the format of the time and the date in the compiler-generated listing file. The identifiers that appear in the tables in the listing file are sorted as specified by the LC_COLLATE category of the locale specified in the option.

Note: The formats of the predefined macros __DATE__ , __TIME__, and __TIMESTAMP__ are not locale-sensitive.

For more information on locales, refer to the *z/OS C/C++ Programming Guide*.

Effect on IPA Compile Step

The LOCALE option controls processing only for the IPA step for which you specify it.

During the IPA Compile step, the compiler converts source code using the code page that is associated with the locale specified by the LOCALE compile-time option. As with non-IPA compilations, the conversion applies to identifiers, literals, and listings. The locale that you specify on the IPA Compile step is recorded in the IPA object file.

You should use the same code page for IPA Compile step processing for all of your program's source files. This code page should match the code page of the runtime environment. Otherwise, your application may not run correctly.

Effect on IPA Link Step

The locale that you specify on the IPA Compile step does not determine the locale that the IPA Link step uses. The LOCALE option that you specify on the IPA Link step is used for the following:

- The encoding of the message text and the listing text.
- Date and time formatting in the Source File Map section of the listing and in the text in the object comment string that records the date and time of IPA Link step processing.
- Sorting of identifiers in listings. The IPA Link step uses the sort order associated with the locale for the lists of symbols in the Inline Report (Summary), Global Symbols Map, and Partition Map listing sections.

If the code page you used for a compilation unit for the IPA Compile step does not match the code page you used for the IPA Link step, the IPA Link step issues an informational message.

If you specify the IPA(MAP) option, the IPA Link step displays information about the LOCALE option, as follows:

- The Prolog section of the listing displays the LOCALE or NOLOCALE option. If you specified the LOCALE option, the Prolog displays the locale and code set that are in effect.
- The Compiler Options Map listing section displays the LOCALE option active on the IPA Compile step for each IPA object. If you specified conflicting code sets between the IPA Compile and IPA Link steps, the listing includes a warning message after each Compiler Options Map entry that displays a conflict.
- The Partition Map listing section shows the current LOCALE option.

LONGNAME | NOLONGNAME

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
C: `NOLONGNAME` C++:`LONGNAME`	`LONGNAME`	`LONGNAME`	`LONGNAME`			

CATEGORY: Object Code Control

```
►►──┬─LO───┬──────────────────────────────────────────────────────────►◄
    └─NOLO─┘
```

The `LONGNAME` option generates untruncated and mixed case external names in the object module produced by the compiler for functions with non-C++ linkage. Functions with C++ linkage are always untruncated and mixed-case external names. These names may be up to 1024 characters in length. The system binder recognizes the format of long external names in object modules, but the system linkage editor does not.

For z/OS C, if you specify the `ALIAS` option with `LONGNAME`, the compiler generates a `NAME` control statement, but no `ALIAS` control statements.

If you use `#pragma map` to associate an external name with an identifier, the compiler generates the external name in the object module. That is, `#pragma map` has the same behavior for the `LONGNAME` and `NOLONGNAME` compiler options. Also, `#pragma csect` has the same behavior for the `LONGNAME` and `NOLONGNAME` compiler options.

When you specify `NOLONGNAME`, only functions that do not have C++ linkage are given truncated and uppercase names.

A comment that indicates the setting of the `LONGNAME` option will be generated in your object module to aid you in diagnosing your program.

Effect on IPA Compile Step

You must specify either the `LONGNAME` compiler option or the `#pragma longname` preprocessor directive for the IPA Compile step (unless you are using the `c89` utility). Otherwise, the compiler issues an unrecoverable error diagnostic message.

Effect on IPA Link Step

The IPA Link step ignores this option if you specify it, and uses the `LONGNAME` option for all partitions it generates.

LSEARCH | NOLSEARCH

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOLSEARCH	NOLSEARCH	NOLSEARCH	NOLSEARCH			

CATEGORY: File Management

```
►►──┬─LSE──(──path────)─┬──────────────────────────────────────────►◄
    └─NOLSE─────────────┘
```

The `LSEARCH` option directs the preprocessor to look for the user include files in the specified libraries.

The suboption *path* specifies one of the following:

- The name of a partitioned or sequential data set that contains user include files.
- An HFS path that contains user include files.
- A search path that is more complex. See "Additional Syntax" on page 140 for details.

The `#include "`*`filename`*`"` format of the `#include` C/C++ preprocessor directive indicates user include files. See "Using Include Files" on page 308 for a description of the `#include` preprocessor directive.

For further information on library search sequences, see "Search Sequences for Include Files" on page 316.

Searching for PDS or PDSE files

Example

You coded your include files as follows:

```
#include "sub/fred.h"
#include "fred.inl"
```

You specified `LSEARCH` as follows:

```
LSEARCH(USER.+,'USERID.GENERAL.+')
```

The compiler uses the following search sequence to look for your include files:

1. First, the compiler looks for `user/sub/fred.h` in this data set:

   ```
   USERID.USER.SUB.H(FRED)
   ```

2. If that PDS member does not exist, the compiler looks in the data set:

   ```
   USERID.GENERAL.SUB.H(FRED)
   ```

3. If that PDS member does not exist, the compiler looks in DD:USERLIB, and then checks the system header files.
4. Next, the compiler looks for `fred.inl` in the data set:
   ```
   USERID.USER.INL(FRED)
   ```
5. If that PDS member does not exist, the compiler will look in the data set:
   ```
   USERID.GENERAL.INL(FRED)
   ```
6. If that PDS member does not exist, the compiler looks in DD:USERLIB, and then checks the system header files.

Searching for HFS Files

The compiler forms the search path for HFS files by appending the path and name of the #include file to the path that you specified in the `LSEARCH` option.

Example 1

You code `#include "sub/fred.h"` and specify:

```
LSEARCH(/u/mike)
```

The compiler looks for the include file `/u/mike/sub/fred.h` .

Example 2

You specify your header file as `#include "fred.h"` , and your `LSEARCH` option as:

```
LSEARCH(/u/mike, ./sub)
```

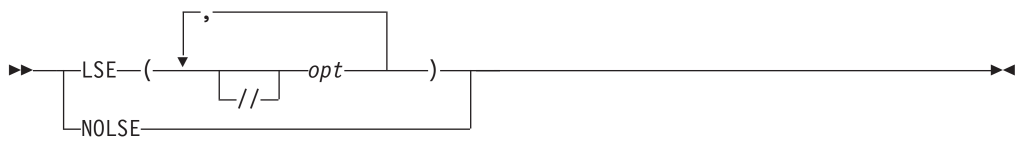
The compiler uses the following search sequence to look for your include files:

1. The compiler looks for `fred.h` in:
   ```
   /u/mike/fred.h
   ```
2. If that HFS file does not exist, the compiler looks in:
   ```
     ./sub/fred.h
   ```
3. If that HFS file does not exist, the compiler looks in the libraries specified on the `USERLIB DD` statement.
4. If `USERLIB DD` is not allocated, the compiler follows the search order for system include files.

The `NOLSEARCH` option instructs the preprocessor to search only those libraries that are specified on the `USERLIB DD` statement. A `NOLSEARCH` option cancels all previous `LSEARCH` specifications, and the compiler uses any `LSEARCH` options that follow it. When you specify more than one `LSEARCH` option, the compiler uses all the libraries in these `LSEARCH` options to find the user include files.

Note: If the *filename* in the `#include` directive is in absolute form, the compiler does not perform a search. See “Determining whether the File Name is in Absolute Form” on page 313 for more details on absolute `#include` *filename*.

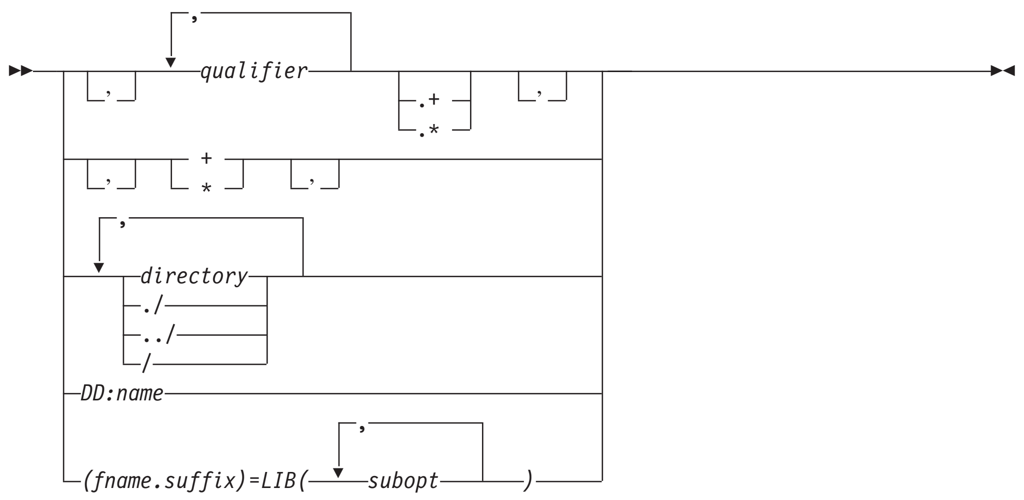
Additional Syntax



You must use the double slashes (//) to specify data set library searches when you specify the `OE` compiler option. (You may use them regardless of the `OE` option).

The `USERLIB` ddname is considered the last suboption for `LSEARCH`, so that specifying `LSEARCH (X)` is equivalent to specifying `LSEARCH (X,DD:USERLIB)`.

Parts of the `#include` *filename* are appended to each `LSEARCH` *opt* to search for the include file. *opt* has the format:



In the above syntax diagram, *opt* specifies one of the following:

- The name of a partitioned or sequential data set that contains user include files
- An HFS path name that should be searched for the include file. You can also use ./ to specify the current directory and ../ to specify the parent directory for your HFS file.
- A DD statement for a sequential data set or a partitioned data set. When you specify a ddname in the search and the include file has a member name, the member name of the include file is used as the name for the `DD:` *name* search suboption, for example:

  ```
  LSEARCH(DD:NEWLIB)
  #include "a.b(c)"
  ```

  The resulting file name is `DD:NEWLIB(C)`.
- A specification of the form `(` *`fname.suffix`* `) = (` *`subopt,subopt,...`* `)` where:
  – *fname* is the name of the include file, or *
  – *suffix* is the suffix of the include file, or *

– *subopt* indicates a subpath to be used in the search for the include files that match the pattern of *fname.suffix*. There should be at least one *subopt*. The possible values are:
  - LIB(*[pds,...]*) where each *pds* is a partitioned data set name. They are searched in the same order as they are specified.

    There is no effect on the search path if no *pds* is specified, but a warning is issued.
  - LIBs are cumulative; for example, LIB(A),LIB(B) is equivalent to LIB(A, B).
  - NOLIB specifies that all LIB(...) previously specified for this pattern should be ignored at this point.

When the #include *filename* matches the pattern of *fname.suffix*, the search continues according to the subopts in the order specified. An asterisk (*) in *fname* or *suffix* matches anything. If the compiler does not find the file, it attempts other searches according to the remaining options in LSEARCH.

Specifying Hierarchical File System Files

When specifying Hierarchical File System (HFS) library searches, do not put double slashes at the beginning of the LSEARCH *opt* . Use *pathnames* separated by slashes (/) in the LSEARCH *opt* for an HFS library. When the LSEARCH *opt* does not start with double slashes, any single slash in the name indicates an HFS library. If you do not have path separators (/), then setting the OE compile option on indicates that this is an HFS library; otherwise the library is interpreted as a data set. See "Using SEARCH and LSEARCH" on page 315 for additional information on HFS files.

The *opt* specified for LSEARCH is combined with the *filename* in #include to form the include file name, for example:

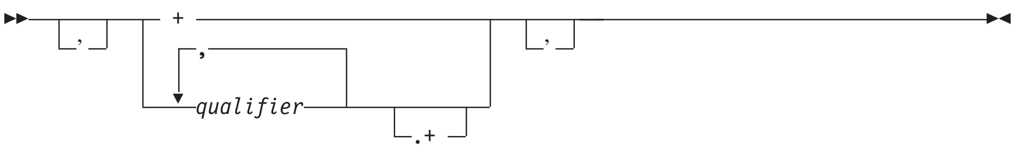
```
LSEARCH(/u/mike/myfiles)
#include "new/headers.h"
```

The resulting HFS file name is /u/mike/myfiles/new/headers.h.

Specifying Sequential Data Sets and PDSs

Use an asterisk (*) or a plus sign (+) in the LSEARCH *opt* to specify whether the library is a sequential or partitioned data set.

Partitioned Data Set (PDS): When you want to specify a set of PDSs as the search path, you add a period followed by an plus sign (.+) at the end of the last qualifier in the *opt*. If you do not have any qualifier, specify a single plus sign (+) as the *opt*. The *opt* has the following syntax for specifying partitioned data set:



where *qualifier* is a data set qualifier.

Start and end the *opt* with single quotation marks (') to indicate that this is an absolute data set specification. Single quotation marks around a single plus sign (+) indicate that the *filename* that is specified in #include is an absolute partitioned data set.

When you do not specify a member name with the `#include` directive, for example, `#include "PR1.MIKE.H"`, the PDS name for the search is formed by replacing the plus sign with the following parts of the *filename* of the `#include` directive:

- For the PDS file name:
  1. All the *path*s and slashes (slashes are replaced by periods)
  2. All the periods and *qualifier*s after the leftmost *qualifier*
- For the PDS member name, the leftmost *qualifier* is used as the member name

See the first example in Table 19.

However, if you specified a member name in the *filename* of the `#include` directive, for example, `#include "PR1.MIKE.H(M1)"`, the PDS name for the search is formed by replacing the plus sign with qualified name of the PDS. See the second example in Table 19.

See "Forming Data Set Names with LSEARCH | SEARCH Options" on page 310 for more information on forming PDS names.

Note: To specify a single PDS as the *opt*, do not specify a trailing asterisk (*) or plus sign (+). The library is then treated as a PDS but the PDS name is formed by just using the leftmost *qualifier* of the `#include` *filename* as the member name. For example:

```
LSEARCH(AAAA.BBBB)
#include "sys/ff.gg.hh"

Resulting PDS name is
userid.AAAA.BBBB(FF)
```

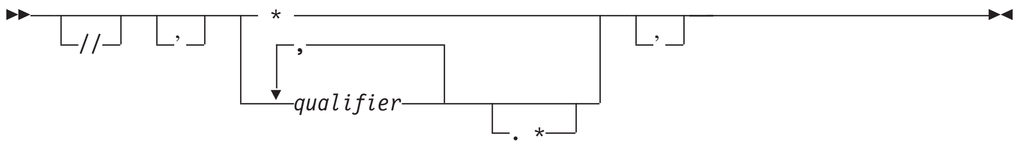
Also see the third example in Table 19.

Examples: The following example shows you how to specify a PDS search path:

Table 19. Partitioned Data Set Examples

include Directive	LSEARCH option	Result
#include ″PR1.MIKE.H″	LSEARCH('CC.+')	'CC.MIKE.H(PR1)'
#include ″PR.KE.H(M1)″	LSEARCH('CC.+')	'CC.PR.KE.H(M1)'
#include ″A.B″	LSEARCH(CC)	*userid*.CC(A)
#include ″A.B.D″	LSEARCH(CC.+)	*userid*.CC.B.D(A)
#include ″a/b/dd.h″	LSEARCH('CC.+')	'CC.A.B.H(DD)'
#include ″a/dd.ee.h″	LSEARCH('CC.+')	'CC.A.EE.H(DD)'
#include ″a/b/dd.h″	LSEARCH('+')	'A.B.H(DD)'
#include ″a/b/dd.h″	LSEARCH(+)	*userid*.A.B.H(DD)
#include ″A.B(C)″	LSEARCH('D.+')	'D.A.B(C)'

Sequential Data Set: When you want to specify a set of sequential data sets as the search path, you add a period followed by an asterisk (.*) at the end of the last qualifier in the *opt*. If you do not have any qualifiers, specify one asterisk (*) as the *opt*. The `opt` has the following syntax for specifying a sequential data set:



where `qualifier` is a data set qualifier.

Start and end the `opt` with single quotation marks (') to indicate that this is an absolute data set specification. Single quotation marks (') around a single asterisk (*) means that the file name that is specified in `#include` is an absolute sequential data set.

The asterisk is replaced by all of the qualifiers and periods in the `#include` *filename* to form the complete name for the search (as shown in the following table).

Examples: The following example shows you how to specify a search path for a sequential data set:

Table 20. Sequential Data Set Examples

include Directive	LSEARCH option	Result
#include ″A.B″	LSEARCH(CC.*)	*userid*.CC.A.B
#include ″a/b/dd.h″	LSEARCH('CC.*')	'CC.DD.H'
#include ″a/b/dd.h″	LSEARCH('*')	'DD.H'
#include ″a/b/dd.h″	LSEARCH(*)	*userid*.DD.H

Note: If the trailing asterisk is not used in the `LSEARCH` *opt*, then the specified library is a PDS:

```
#include "A.B"
LSEARCH('CC')
```

Result is `'CC(A)'` which is a PDS.

Effect on IPA Compile Step

The `LSEARCH` option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the `LSEARCH` option, but ignores it.

MARGINS | NOMARGINS

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
C++ and C(V-format): `NOMARGINS` C(F-format): `MARGINS(1,72)`	`NOMARGINS`	`NOMARGINS`	`NOMARGINS`			

CATEGORY: Source Code Control

The `MARGINS` option specifies the columns in the input record that are to be scanned for input to the compiler. The compiler ignores text in the source input that does not fall within the range that is specified on the `MARGINS` option.

You can use the `MARGINS` and `SEQUENCE` options together. The `MARGINS` option is applied first to determine which columns are to be scanned. The `SEQUENCE` option is then applied to determine which of these columns are not to be scanned. If the `SEQUENCE` settings do not fall within the `MARGINS` settings, the `SEQUENCE` option has no effect.

When a source (or include) file is opened, it initially gets the margins and sequence specified on the command line (or the defaults if none was specified). You can reset these settings by using `#pragma margins` or `#pragma sequence` at any point in the file. When an `#include` file returns, the previous file keeps the settings it had when it encountered the `#include` directive.

The `NOMARGINS` option specifies that the entire input source record is to be scanned for input to the compiler.

z/OS C++

```
►►──┬─MAR───┬──────────────────────────────────────────────────►◄
    └─NOMAR─┘
```

In a C++ program, the `MARGINS` option specifies that columns 1 through 72 in the input record are to be scanned for input to the compiler. The compiler ignores any text in the source input that does not fall within that range.

z/OS C

```
►►──┬─MAR──(m,n)─┬─────────────────────────────────────────────►◄
    └─NOMAR──────┘
```

In a C program, the `MARGINS` option has the following additional syntax:

```
MARGINS(m,n)
```

where:

m	specifies the first column of the source input that contains valid z/OS C code. The value of `m` must be greater than `0` and less than `32761`.

n	specifies the last column of the source input that contains valid z/OS C code. The value of `n` must be greater than `m` and less than `32761`. An asterisk (*) can be assigned to `n` to indicate the last column of the input record. If you specify `MARGINS (9,*)`, the compiler scans from column 9 to the end of the record for input source statements.

If the `MARGINS` option is specified along with the `SOURCE` option in a C program, only the range specified on the `MARGINS` option is shown in the compiler source listing.

Notes:

1. The `MARGINS` option does not reformat listings.
2. If your program uses the `#include` preprocessor directive to include z/OS C library header files **and** you want to use the `MARGINS` option, you must ensure that the specifications on the `MARGINS` option does not exclude columns `20` through `50`. That is, the value of `m` must be less than `20`, and the value of `n` must be greater than `50`. If your program does not include any z/OS C library header files, you can specify any setting you want on the `MARGINS` option when the setting is consistent with your own include files.

Effect on IPA Compile Step

The `MARGINS` option is used for source code analysis, and has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the `MARGINS` option, but ignores it.

MAXMEM | NOMAXMEM

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`MAXMEM(2097152)` or `MAXMEM(*)` or `MAXMEM(0)`	`MAXMEM(*)`	`MAXMEM(*)`	`MAXMEM(*)`			

CATEGORY: Object Code Control

```
►►──┬─MAXM──(──size──)─┬──────────────────────────────────────────────►◄
    └─NOMAXM───────────┘
```

When compiling with `OPT`, the `MAXMEM(size)` option limits the amount of memory used for local tables of specific, memory intensive optimizations to *size* kilobytes. The valid range for *size* is 0 to 2097152. You can use asterisk as a value for *size* , `MAXMEM(*)`, to indicate the highest possible value, which is also the default. `NOMAXMEM`, `MAXMEM(0)`, and `MAXMEM(*)` are equivalent. Use the `MAXMEM` option if you want to specify a memory size of less value than the default.

If the memory specified by the MAXMEM option is insufficient for a particular optimization, the compilation is completed in such a way that the quality of the optimization is reduced, and a warning message is issued.

When a large *size* is specified for MAXMEM, compilation may be aborted because of insufficient virtual storage, depending on the source file being compiled, the size of the subprogram in the source, and the virtual storage available for the compilation.

The advantage of using the MAXMEM option is that, for large and complex applications, the compiler produces a slightly less-optimized object module and generates a warning message, instead of terminating the compilation with an error message of “insufficient virtual storage”.

Notes:
1. The limit that is set by MAXMEM is the amount of memory for specific optimizations, and not for the compiler as a whole. Tables that are required during the entire compilation process are not affected by or included in this limit.
2. Setting a large limit has no negative effect on the compilation of source files when the compiler needs less memory.
3. Limiting the scope of optimization does not necessarily mean that the resulting program will be slower, only that the compiler may finish before finding all opportunities to increase performance.
4. Increasing the limit does not necessarily mean that the resulting program will be faster, only that the compiler may be able to find opportunities to increase performance.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

If you specify the MAXMEM option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

The option value you specify on the IPA Compile step for each IPA object file appears in the IPA Link step Compiler Options Map listing section.

Effect on IPA Link Step

If you specify the MAXMEM option on the IPA Link step, the value of the option is used. The IPA Link step Prolog and Partition Map listing sections display the value of the option.

If you do not specify the option on the IPA Link step, the value it uses for a partition is the maximum MAXMEM value you specified for the IPA Compile step for any compilation unit that provided code for that partition. The IPA Link Step Prolog listing section does not display the value of the MAXMEM option, but the Partition Map listing section does.

MEMORY | NOMEMORY

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
MEMORY	MEMORY	MEMORY	MEMORY	MEMORY	MEMORY	MEMORY

CATEGORY: File Management

```
►►──┬─MEM───┬────────────────────────────────────────────────────►◄
    └─NOMEM─┘
```

The MEMORY option specifies that the compiler is to use a MEMORY file in place of a work-file if possible. See the *z/OS C/C++ Programming Guide* for more information on memory files.

This option increases compilation speed, but you may require additional memory to use it. If you use this option and the compilation fails because of a storage error, you must increase your storage size or recompile your program using the NOMEMORY option.

Effect on IPA Compile Step

The MEMORY compiler option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The MEMORY option has the same effect on the IPA Link step as it does on a regular compilation. If the IPA Link step fails due to an out-of-memory condition, provide additional virtual storage. If additional storage is unavailable, specify the NOMEMORY option.

NESTINC | NONESTINC

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NESTINC(255)	NESTINC(255)	NESTINC(255)	NESTINC(255)			

CATEGORY: Source Code Control

```
►►──┬─NEST──(num)─┬──────────────────────────────────────────────►◄
    └─NONEST──────┘
```

The NESTINC option specifies the number of nested include files to be allowed in your source program. You can specify a limit of any integer from 0 to SHRT_MAX, which indicates the maximum limit, as defined in the header file LIMITS.H. To specify the maximum limit, use an asterisk (*). If you specify an invalid value, the compiler issues a warning message, and uses the default limit, 255.

Specifying NONESTINC is equivalent to specifying NESTINC(255).

Note: If you use heavily nested include files, your program requires more storage to compile.

Effect on IPA Compile Step

The NESTINC option is used for source code analysis, and has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the NESTINC option, but ignores it.

OBJECT | NOOBJECT

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities** **Note that this changes if the -o flag is set.**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
OBJECT	OBJECT (*file_name*.o)	OBJECT (*file_name*.o)	OBJECT (*file_name*.o)	OBJECT (//DD:SYSLIPA)	OBJECT (//DD:SYSLIPA)	OBJECT (//DD:SYSLIPA)

CATEGORY: File Management and Object Code Control

```
►►──┬─OBJ───┬───┬─────────────────────────────────────────┬────────────────────►◄
    └─NOOBJ─┘   └─(──┬─Sequential filename───────────┬─)──┘
                     ├─Partitioned data set──────────┤
                     ├─Partitioned data set (member)─┤
                     ├─Hierarchical filename─────────┤
                     └─Hierarchical directory────────┘
```

The OBJECT option specifies whether the compiler is to produce an object module.

The GOFF compiler option specifies the object format that will be used to encode the object information.

You can specify OBJECT(*filename*) to place the object module in that file. If you do not specify a file name for the OBJECT option, the compiler uses the SYSLIN ddname if you allocated it. Otherwise, the compiler generates a file name as follows:

- If you are compiling a data set, the compiler uses the source file name to form the name of the object module data set. The high-level qualifier is replaced with the userid under which the compiler is running, and .OBJ is appended as the low-level qualifier.
- If you are compiling an HFS file, the compiler stores the object module in a file that has the name of the source file with an .o extension.

The `NOOBJ` option can optionally take a *filename* suboption. This *filename* then becomes the default. If you subsequently use the `OBJ` option without a *filename* suboption, the compiler uses the *filename* that you specified in the earlier `NOOBJ`. For example, the following specifications have the same result:

```
 CXX HELLO (NOOBJ(/hello.obj) OBJ
CXX HELLO (OBJ(/hello.obj)
```

If you specify `OBJ` and `NOOBJ` multiple times, the compiler uses the last specified option with the last specified suboption. For example, the following specifications have the same result:

```
CXX HELLO (NOOBJ(/hello.obj) OBJ(/n1.obj)  NOOBJ(/test.obj) OBJ
CXX HELLO (OBJ(/test.obj)
```

If you request a listing by using the `SOURCE`, `INLRPT`, or `LIST` option, and you also specify `OBJECT`, the name of the object module is printed in the listing prolog.

You can specify this option using the `#pragma option` directive for C.

In the z/OS UNIX System Services environment, you can specify the object location by using the `-c -o objectname` options when using the `c89`, `cc`, or `c++` commands.

z/OS C: For z/OS C programs, see Table 21 on page 200 for a description of the relationship between the `OBJECT` and `DECK` compiler options.

Note: If you use the following form of the command in a JES3 batch environment where xxx is an unallocated data set, you may get undefined results.

```
OBJECT(xxx)
```

Effect on IPA Compile Step

IPA Compile uses the same rules as the regular compile to determine the file name or data set of the object module it generates. If you specify `NOOBJECT` and `NODECK`, the IPA Compile step suppresses object output, but performs all analysis and code generation processing (other than writing object records).

Note: You should not confuse the `OBJECT` compiler option with the `IPA(OBJECT)` suboption. The `OBJECT` option controls file destination. The `IPA(OBJECT)` suboption controls file content. Refer to “IPA | NOIPA” on page 125 for information about the `IPA(OBJECT)` suboption.

Effect on IPA Link Step

This option also affects the IPA Link step. If you specify both `OBJECT` and `DECK` on the IPA Link step, IPA issues a warning message and stores the object module in the data set you specified on the SYSLIN DD name.

`c89` does not normally keep the object file output from the IPA Link step, as the output is an intermediate file in the link-edit phase processing. To find out how to make the object file permanent, refer to the `{ _TMPS}` environment variable information in the `c89` section of *Z/OS UNIX System Services Command Reference*.

Note: The `OBJECT` compiler option is not the same as the `OBJECT` suboption of the `IPA` option. Refer to "IPA | NOIPA" on page 125 for information about the `IPA(OBJECT)` option.

OE | NOOE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOOE	OE	OE	OE	OE	OE	OE

CATEGORY: Source Code Control

```
►►──┬─OE───┬──┬──────────────────┬──────────────────────────────────►◄
    └─NOOE─┘  └─(──filename──)───┘
```

Notes:

1. To compile new applications, you should use this option instead of `OMVS | NOOMVS`.
2. Diagnostics and listing information will refer to the file name that is specified for the `OE` option (in addition to the search information).

The `OE` option specifies that the compiler use the POSIX.2 standard rules for searching for files specified with #include directives. These rules state that the current path of the file presently being processed is the path used as the starting point for searches of include files contained in that file.

The `NOOE` option can optionally take a *filename* suboption. This *filename* then becomes the default. If you subsequently use the `OE` option without a *filename* suboption, the compiler uses the *filename* that you specified in the earlier `NOOE`. For example, the following specifications have the same result:

```
CXX HELLO (NOOE(/hello.c) OE
CXX HELLO (OE(/hello.c)
```

If you specify `OE` and `NOOE` multiple times, the compiler uses the last specified option with the last specified suboption. For example, the following specifications have the same result:

```
CXX HELLO (NOOE(/hello.c) OE(/n1.c)  NOOE(/test.c) OE
CXX HELLO (OE(/test.c)
```

When the `OE` option is in effect and the main input file is an HFS file, the path of *filename* is used instead of the path of the main input file name. If the file names indicated in other options appear ambiguous between z/OS and HFS, the presence of the `OE` option tells the compiler to interpret the ambiguous names as HFS file names. User include files that are specified in the main input file are searched starting from the path of *filename* . If the main input file is not an HFS file, *filename* is ignored.

For example, if the compiler is invoked to compile HFS file `/a/b/hello.c` it searches directory `/a/b/` for include files specified in `/a/b/hello.c`, in accordance with POSIX.2 rules . If the compiler is invoked with the `OE(/c/d/hello.c)` option for the same source file, the directory specified as the suboption for the `OE` option, `/c/d/`, is used to locate include files specified in `/a/b/hello.c`.

Effect on IPA Compile Step

The `OE` compiler option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

On the IPA Link step, the `OE` option controls the display of file names.

OFFSET | NOOFFSET

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOOFFSET	NOOFFSET	NOOFFSET	NOOFFSET	NOOFFSET	NOOFFSET	NOOFFSET

CATEGORY: Listing

```
►►──┬─OF───┬──────────────────────────────────────────────────────────►◄
    └─NOOF─┘
```

The `OFFSET` option instructs the compiler to display, in the pseudo-assembly listing generated by the `LIST` option, the offset addresses relative to the entry point or start of each function.

If you use the `OFFSET` option, you must also specify the `LIST` option to generate the pseudo-assembly listing. If you specify the `OFFSET` option but omit the `LIST` option, the compiler generates a warning message, and does not produce a pseudo-assembly listing.

The `NOOFFSET` option specifies that the compiler is to display, in the pseudo-assembly listing generated by the `LIST` option, the offset addresses relative to the beginning of the generated code and not the entry point.

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c89`, `cc` or `c++` commands.

Effect on IPA Compile Step

If you specify the IPA(OBJECT) option (that is, if you request code generation), the OFFSET option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

If you specify the LIST option during IPA Link, the IPA Link listing will be affected (in the same way as a regular compilation) by the OFFSET option setting in effect at that time.

The OFFSET option that you specified on the IPA Compile step has no effect on the IPA Link step.

OPTFILE | NOOPTFILE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOOPTFILE						

CATEGORY: File Management

```
►►──┬─OPTF───┬──┬────────────┬──────────────────────────────────►◄
    └─NOOPTF─┘  └─(filename)─┘
```

The OPTFILE option directs the compiler to look for compiler options in the file specified by *filename*.

You can specify any valid filename, including a DD name such as (DD:MYOPTS). The DD name may refer to instream data in your JCL. If you do not specify *filename*, the compiler uses DD:SYSOPTF.

The NOOPTF option can optionally take a *filename* suboption. This *filename* then becomes the default. If you subsequently use the OPTF option without a *filename* suboption, the compiler uses the *filename* that you specified in the earlier NOOPTF. For example, the following specifications have the same result:

```
CXX HELLO (NOOPTF(/hello.opt) OPTF
CXX HELLO (OPTF(/hello.opt)
```

The options are specified in a free format with the same syntax as they would have on the command line or in JCL. The code points for the special characters *\f*, *\v*, and *\t* are whitespace characters. Everything that is specified in the file is taken to be part of a compiler option (except for the continuation character), and unrecognized entries are flagged. Nothing on a line is ignored.

If the record format of the options file is fixed and the record length is greater than 72, columns 73 to the end-of-line are treated as sequence numbers and are ignored.

Notes:

1. You cannot nest the `OPTFILE` option. If the `OPTFILE` option is also used in the file that is specified by another `OPTFILE` option, it is ignored.
2. If you specify `NOOPTFILE` after a valid `OPTFILE`, it does not undo the effect of the previous `OPTFILE`. This is because the compiler has already processed the options in the options file that you specified with `OPTFILE`. The only reason to use `NOOPTFILE` is to specify an option file name that a later specification of `OPTFILE` can use.
3. If the file cannot be opened or cannot be read, a warning message is issued and the `OPTFILE` option is ignored.
4. The options file can be an empty file.
5. You can use an option file only once in a compilation. For example, if you use the following options:

   ```
   OPTFILE(DD:OF)     OPTFILE
   ```

   the compiler processes the option `OPTFILE(DD:OF)`, but the second option `OPTFILE` is not processed. A diagnostic message is produced, because the second specification of `OPTFILE` uses the same option file as the first.

   You can specify `OPTFILE` more than once in a compilation, if you use a different options file with each specification. For example:

   ```
   OPTFILE(DD:OF)   OPTFILE(DD:OF1)
   ```

Examples

1. Suppose that you use the following JCL:

   ```
   //   CPARM='SO OPTFILE(PROJ1OPT) EXPORTALL'
   ```

   If the file `PROJ1OPT` contains `OBJECT LONGNAME`, the effect on the compiler is the same as if you specified the following:

   ```
   //   CPARM='SO OBJECT LONGNAME EXPORTALL'
   ```

2. Suppose that you include the following in the JCL:

   ```
   //   CPARM='OBJECT OPTFILE(PROJ1OPT) LONGNAME OPTFILE(PROJ2OPT) LIST'
   ```

   If the file `PROJ1OPT` contains `SO LIST` and the file `PROJ2OPT` contains `GONUM`, the net effect to the compiler is the same as if you specified the following:

   ```
   //   CPARM='OBJECT SO LIST LONGNAME GONUM LIST'
   ```

3. If a F80 format options file looks like this:

   ```
      | ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
                                 LIST                                           00000010
                                                                  INLRPT        00000020
   MARGINS                                                                      00000030
      OPT                                                                       00000040
      XREF                                                                      00000050
   ```

   The compile has the same effect as if you specified the following options on the command line or in a `PARMS=` statement in your JCL:

   ```
     LIST INLRPT MARGINS OPT XREF
   ```

4. The following example shows how to use the options file as an instream file in JCL:

```
//COMP  EXEC CBCC,
//      INFILE='<userid>.USER.CXX(LNKLST)',
//      OUTFILE='<userid>.USER.OBJ(LNKLST),DISP=SHR ',
//      CPARM='OPTFILE(DD:OPTION)'
//OPTION DD DATA,DLM=@@
                            LIST
                                                               INLRPT
MARGINS
   OPT
   XREF
@@
```

Effect on IPA Compile Step

The `OPTFILE` option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The `OPTFILE` option has the same effect on the IPA Link step as it does on a regular compilation.

OPTIMIZE | NOOPTIMIZE

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities** **Note that the default is set to `OPTIMIZE(1)` if the `-WI` flag is set.**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
C and C++ compile: `NOOPTIMIZE` IPA Link: `OPTIMIZE(2)`	`OPTIMIZE(0)` for no IPA, `OPTIMIZE (1)` for IPA Compile	`OPTIMIZE(0)` for no IPA, `OPTIMIZE (1)` for IPA Compile	`OPTIMIZE(0)` for no IPA, `OPTIMIZE (1)` for IPA Compile	`OPTIMIZE(1)`	`OPTIMIZE(1)`	`OPTIMIZE(1)`

CATEGORY: Object Code Control

```
►►──┬─OPT──(level)─┬──────────────────────────────────────────────►◄
    └─NOOPT────────┘
```

The `OPTIMIZE` option instructs the compiler to optimize the generated machine instructions to produce a faster running object module. This type of optimization can also reduce the amount of main storage that is required for the generated object module. Using `OPTIMIZE` will increase compile time over `NOOPTIMIZE` and may have greater storage requirements. During optimization, the compiler may move code to increase run-time efficiency; as a result, statement numbers in the program listing may not correspond to the statement numbers used in runtime messages.

A list of the valid suboptions for `OPT` and their descriptions follow: *level* can have the following values:

0 Indicates that no optimization is to be done; this is equivalent to `NOOPTIMIZE`. You should use this option in the early stages of your application development since the compilation is efficient but the execution is not. This option also allows you to take full advantage of the debugger.

1 `OPTIMIZE(1)` is an obsolete artifact of the Version 2 Release 4 compiler. We suggest that you use `OPTIMIZE(2)`, which ensures that you will have compatibility with future compilers.

2 Indicates that global optimizations are to be performed. You should be aware that the size of your functions, the complexity of your code, the coding style, and support of the ANSI standard may affect the global optimization of your program. You may need significant additional memory to compile at this optimization level.

no level
`OPTIMIZE` specified with no level defaults , depending on the compilation environment and IPA mode. See the Option Default table above for details.

In the z/OS UNIX System Services environment, this option is turned on by specifying -O (the letter) or -0 (the number), -1, or -2 when using the c89, cc, or c++ commands.

You can specify this option using the #pragma option directive for C.

You can specify this option for a specific subprogram using the #pragma option_override(subprogram_name, OPT(LEVEL,n)) directive.

The OPTIMIZE option will control the overall optimization value. Any subprogram-specific optimization levels specified at compile time by #pragma option_override(subprogram_name, OPT(LEVEL,n)) directives will be retained. Subprograms with an OPT(LEVEL,0)) value will receive minimal code generation optimization. Subprograms may not be inlined or inline other subprograms. Generate and check the inline report to determine the final status of inlining.

Inlining of functions in conjunction with other optimizations provides optimal run time performance. See "INLINE | NOINLINE" on page 120 for more information about the `INLINE` option and the *z/OS C/C++ Programming Guide* for more information about optimization.

If you specify `OPTIMIZE` with `TEST`, you can only set breakpoints at function call, function entry, function exit, and function return points.

The option `INLINE` is automatically turned on when you specify `OPTIMIZE`, unless you have explicitly specified the `NOINLINE` option.

A comment that notes the level of optimization will be generated in your object module to aid you in diagnosing your program.

Effect of ANSIALIAS: When the `ANSIALIAS` option is specified, the optimizer assumes that pointers can point only to objects of the same type, and performs more aggressive optimization. However, if this assumption is not true and `ANSIALIAS` is specified, wrong program code could be generated. If you are not sure, use `NOANSIALIAS`. For more information, see "ANSIALIAS | NOANSIALIAS" on page 80.

Effect on IPA(OBJONLY) Compilation

During a compilation with IPA compile-time optimizations active, any subprogram-specific optimization levels specified by #pragma option_override(subprogram_name, OPT(LEVEL,n)) directives will be retained. Subprograms with an OPT(LEVEL,0)) value will receive minimal IPA and code generation optimization. Subprograms may not be inlined or inline other subprograms. Generate and check the inline report to determine the final status of inlining.

Effect on IPA Compile Step

On the IPA Compile step, all values (except for (0)) of the `OPTIMIZE` compiler option and the `OPT` suboption of the `IPA` option have an equivalent effect.

Refer to the descriptions of the `OPTIMIZE` and `LEVEL` suboptions of the `IPA` option in "IPA | NOIPA" on page 125 for information about using the `OPTIMIZE` option under IPA.

Effect on IPA Link Step

`OPTIMIZE(2)` is the default for the IPA Link step, but you can specify any level of optimization. The IPA Link step Prolog listing section will display the value of this option.

This optimization level will control the overall optimization value. Any subprogram-specific optimization levels specified at IPA Compile time by #pragma option_override(subprogram_name, OPT(LEVEL,n)) directives will be retained. Subprograms with an OPT(LEVEL,0)) value will receive minimal IPA and code generation optimization, and will not participate in IPA Inlining.

The IPA Link step merges and optimizes your application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition. Only compatible subprograms are included in a given partition. Compatible subprograms have the same `OPTIMIZE` setting.

The `OPTIMIZE` setting for a partition is set to that of the first subprogram that is placed in the partition. Subprograms that follow are placed in partitions that have the same `OPTIMIZE` setting. A `OPTIMIZE(0)` mode is placed in a `OPTIMIZE(0)`partition, and a `OPTIMIZE(2)` is placed in a `OPTIMIZE(2)` partition.

The option value that you specified for each IPA object file on the IPA Compile step appears in the IPA Link step Compiler Options Map listing section.

The Partition Map sections of the IPA Link step listing and the object module `END` information section display the value of the `OPTIMIZE` option. The Partition Map also displays any subprogram-specific `OPTIMIZE` values.

If you specify `OPTIMIZE(1)` or `OPTIMIZE(2)` for the IPA Link step, but only `OPTIMIZE(0)` for the IPA Compile step, your program may be slower or larger than if you specified `OPTIMIZE(1)` or `OPTIMIZE(2)` for the IPA Compile step. This situation occurs because the IPA Compile step does not perform as many optimizations if you specify `OPTIMIZE(0)`.

Refer to the descriptions for the `OPTIMIZE` and `LEVEL` suboptions of the `IPA` option in "IPA | NOIPA" on page 125 for information about using the `OPTIMIZE` option under IPA.

PHASEID | NOPHASEID

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOPHASEID						

CATEGORY: Debug/Diagnostic

```
►►──┬─PHASEID───┬──────────────────────────────────────────────────►◄
    └─NOPHASEID─┘
```

If you specify the PHASEID option, it causes each compiler component (phase) to issue an informational message as the phase begins execution. This message identifies compiler phase module name, product identification, and build level. Use the PHASEID option to assist you with determining the maintenance level of each compiler component (phase).

The compiler issues a separate CBC0000(I) message each time compiler execution causes a given compiler component (phase) to be entered. This may be many times for a given compilation.

The FLAG option has no effect on the PHASEID informational message.

Effect on IPA Compile Step

The PHASEID option has the same effect on the IPA Compile Step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step uses the PHASEID option that you specify for that step.

PLIST

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
PLIST(HOST)	PLIST(HOST)	PLIST(HOST)	PLIST(HOST)			

CATEGORY: Program Execution

```
►►──PLIST──(─┬─HOST─┬─)──────────────────────────────────────────────►◄
             └─OS───┘
```

When compiling `main()` programs, use the `PLIST` option to direct how the parameters from the caller are passed to `main()`.

If you specify `PLIST(HOST)`, the parameters are presented to `main()` as an argument list (`argv, argc`).

If you specify `PLIST(OS)`, the parameters are passed without restructuring, and the standard calling conventions of the operating system are used. See the *z/OS Language Environment Programming Guide* for details on how to access these parameters.

If you are compiling a `main()` program to run under IMS, you must specify the `PLIST(OS)` and `TARGET(IMS)` options together.

Effect on IPA Compile Step

If you specified `PLIST` for any compilation unit in the IPA Compile step, it generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the `IPA(OBJECT)` option.

Effect on IPA Link Step

If you specify `PLIST` for the IPA Compile step, you do not need to specify it again on the IPA Link step. The IPA Link step uses the information generated for the compilation unit that contains the `main()` function, or for the first compilation unit it finds if it cannot find a compilation unit containing `main()`.

If you specify this option on both the IPA Compile and the IPA Link steps, the setting on the IPA Link step overrides the setting on the IPA Compile step. This situation occurs whether you use `PLIST` as a compiler option or specify it using the `#pragma runopts` directive (on the IPA Compile step).

PORT | NOPORT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOPORT(NOPPS)`						

CATEGORY: Portability

```
►►──┬─PORT───┬──┬──────────────────┬─────────────────────────────────────►◄
    └─NOPORT─┘  └─(──┬─PPS───┬──)──┘
                     └─NOPPS─┘
```

The `PORT` option allows you to adjust the error recovery action that the compiler takes when it encounters an ill-formed `#pragma pack` directive. When you specify `PORT(PPS)`, the compiler uses the strict error recovery mode. When you specify any other value for either `PORT` or `NOPORT`, the compiler uses the default error recovery mode. When you specify `PORT` without a suboption, the suboption setting is inherited from the default setting or from previous `PORT` specifications.

Default Error Recovery

When the default error recovery mode is active, the compiler recovers from errors in the `#pragma pack` directive as follows:

- `#pragma pack(`*first_value*
  - If *first_value* is a valid S/390 value for `#pragma pack`, packing is done as specified by *first_value*. The compiler detects the missing closing parentheses and issues a warning message
  - If *first_value* is not a valid S/390 value for `#pragma pack`, no packing changes are made. The compiler ignores the `#pragma pack` directive and issues a warning message
- `#pragma pack(`*first_value bad_tokens*
  - If *first_value* is a valid S/390 value for `#pragma pack`, packing is done as specified by *first_value*. If *bad_tokens* is invalid, the compiler detects it and issues a warning message.
  - If *first_value* is not a valid S/390 value for `#pragma pack`, no packing changes will be performed. The compiler will ignore the `#pragma pack` directive and issue a warning message
- `#pragma pack(`*valid_value*`)` *extra_trailing_tokens*

  The compiler ignores the extra text and does not issue a message

Strict Error Recovery

To use the compiler's strict error recovery mode, you must explicitly request it by specifying `PORT(PPS)`.

When the strict error recovery mode is active, if the compiler detects errors in the `#pragma pack` directive, it ignores the pragma and does not make any packing changes. For example, for any of the following specifications of the `#pragma pack` directive:

```
#pragma pack(first_value

#pragma pack(first_value bad_tokens

#pragma pack(valid_value) extra_trailing_tokens
```

Effect on IPA Compile Step

The `PORT` option is used for source code analysis, and has the same effect on the IPA compile step as it does on a regular compile.

Effect on IPA Link Step

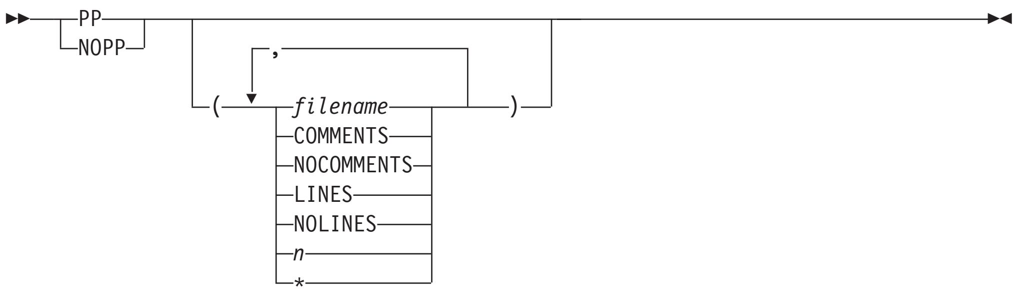
The IPA link step issues a diagnostic message if you specify the `PORT` option for that step.

PPONLY | NOPPONLY

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOPPONLY	NOPPONLY (NOCOMMENTS, NOLINES, /dev/fd1, 2048)	NOPPONLY (NOCOMMENTS, NOLINES, /dev/fd1, 2048)	NOPPONLY (NOCOMMENTS, NOLINES, /dev/fd1, 2048)			

CATEGORY: Preprocessor



The PPONLY option specifies that only the preprocessor is to be run against the source file. This output of the preprocessor consists of the original source file with all the macros expanded and all the include files inserted. It is in a format that can be compiled.

The suboptions are:

COMMENTS | NOCOMMENTS	The COMMENTS suboption preserves comments in the preprocessed output. The default is NOCOMMENTS.
LINES | NOLINES	The LINES suboption issues #line directives at include file boundaries, block boundaries and where there are more than 3 blank lines. The default is NOLINES.
filename	The name for the pre-processed output file. The *filename* may be a data set or an HFS file. If you do not specify a file name for the PPONLY option, the SYSUT10 ddname is used if it has been allocated. If SYSUT10 has not been allocated, the file name is generated as follows: • If a data set is being compiled, the name of the pre-processed output data set is formed using the source file name. The high-level qualifier is

	replaced with the userid under which the compiler is running, and .EXPAND is appended as the low-level qualifier. • If the source file is an HFS file, the pre-processed output is written to an HFS file that has the source file name with .i extension.
n	If a parameter *n*, which is an integer between 2 and 32760 inclusive, is specified, all lines are folded at column *n*.
*	If an asterisk (*) is specified, the lines are folded at the maximum record length of 32760. Otherwise, all lines are folded to fit into the output file, based on the record length of the output file.

The PPONLY suboptions are cumulative. If you specify suboptions in multiple instances of PPONLY and NOPPONLY , all the suboptions are combined and used for the last occurrence of the option. For example, the following three specifications have the same result:

```
CXX HELLO (NOPPONLY(/aa.exp) PPONLY(LINES) PPONLY(NOLINES)
CXX HELLO (PPONLY(/aa.exp,LINES,NOLINES)
CXX HELLO (PPONLY(/aa.exp,NOLINES)
```

All #line and #pragma preprocessor directives (except for margins and sequence directives) remain. When you specify PPONLY(*), #line directives are generated to keep the line numbers generated for the output file from the preprocessor similar to the line numbers generated for the source file. All consecutive blank lines are suppressed.

If you specify the PPONLY option, the compiler turns on the TERMINAL option. If you specify the SHOWINC, XREF, AGGREGATE, or EXPMAC options with the PPONLY option, the compiler issues a warning, and ignores the options.

If you specify the PPONLY and LOCALE options, all the #pragma filetag directives in the source file are suppressed. The compiler generates its #pragma filetag directive at the first line in the preprocessed output file in the following format:

```
??=pragma filetag ("locale code page")
```

In the above, ??= is a trigraph representation of the # character.

The code page in the pragma is the code set that is specified in the LOCALE option. For more information on locales, refer to the *z/OS C/C++ Programming Guide*.

The NOPPONLY option specifies that both the preprocessor and the compiler are to be run against the source file.

If you specify both PPONLY and NOPPONLY, the last one that is specified is used.

In the z/OS UNIX System Services environment, this option is turned on by specifying -E when using the c89, cc or c++ commands. To turn on the COMMENTS suboption, specify -C. The user cannot specify PPONLY, they must use -E and -C. The {_ELINES} envar is also relevant (for further information on {_ELINES}, refer to “Environment Variables” on page 567).The output always goes to stdout.

Effect on IPA Compile Step

The `PPONLY` has the same effect on the IPA Compile step as it does on a regular compilation. It processes source code, then causes the compiler to stop processing before it begins the IPA Compile step. You should not use this option for the IPA Compile step.

Effect on IPA Link Step

The IPA Link step accepts the `PPONLY` option, but ignores it.

REDIR | NOREDIR

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`REDIR`	`REDIR`	`REDIR`	`REDIR`			

CATEGORY: Program Execution

```
►►──┬─RED───┬──────────────────────────────────────────────►◄
    └─NORED─┘
```

The `REDIR` option directs the compiler to create an object module that, when linked and run, allows you to redirect `stdin` , `stdout` and `stderr` for your program from the command line.

Effect on IPA Compile Step

If you specify the `REDIR` option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the `IPA(OBJECT)` option.

Effect on IPA Link Step

If you specify the `REDIR` option for the IPA Compile step, you do not need to specify it again on the IPA Link step. The IPA Link step uses the information generated for the compilation unit that contains the `main()` function, or for the first compilation unit it finds if it cannot find a compilation unit containing `main()`.

If you specify this option on both the IPA Compile and the IPA Link steps, the setting on the IPA Link step overrides the setting on the IPA Compile step. This situation occurs whether you use `REDIR` and `NOREDIR` as compiler options or specify them using the `#pragma runopts` directive (on the IPA Compile step).

RENT | NORENT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NORENT	RENT	RENT				

CATEGORY: Object Code Control

```
►►──┬─RENT───┬──────────────────────────────────────────────────────►◄
    └─NORENT─┘
```

The RENT option specifies that the compiler is to take code that is not naturally reentrant and make it reentrant. Refer to the *z/OS Language Environment Programming Guide* for a detailed description of reentrancy.

If you use the RENT option, the Linkage Editor cannot directly process the object module that is produced. You must use either the binder, which is described in "Chapter 12. Binding z/OS C/C++ Programs" on page 353, or the prelinker, which is described in "Appendix A. Prelinking and Linking z/OS C/C++ Programs" on page 461.

Notes:

1. Whenever you specify the RENT compiler option, a comment that indicates its use is generated in your object module to aid you in diagnosing your program.
2. z/OS C++ code always uses constructed reentrancy.
3. RENT variables reside in the modifiable writable static area for both z/OS C and z/OS C++ programs.
4. NORENT variables reside in the code area (which may be write protected) for both z/OS C and z/OS C++ programs.

The NORENT option specifies that the compiler is not to specifically generate reentrant code from non-reentrant code. Any naturally reentrant code remains reentrant.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

If you specify RENT or use `#pragma strings(readonly)` or `#pragma variable(RENT|NORENT)` during the IPA Compile step, the information in the IPA object file reflects the state of each symbol.

Effect on IPA Link Step

If you specify the `RENT` option on the IPA Link step, it ignores the option. The reentrant/nonreentrant state of each symbol is maintained during IPA optimization and code generation.

If you generate an IPA Link listing by using the `LIST` or `MAP` compiler option, the IPA Link step generates a Partition Map listing section for each partition. If any symbols within a partition are reentrant, the options section of the Partition Map displays the `RENT` compiler option.

ROCONST | NOROCONST

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOROCONST`						

CATEGORY: Object Code Control

```
►►──┬─ROC───┬──────────────────────────────────────────────────►◄
    └─NOROC─┘
```

The `ROCONST` option informs the compiler that the `const` qualifier is respected by the program. Variables defined with the `const` keyword will not be overridden by a casting operation.

Use `ROCONST` with the `RENT` option so that a `const` variable is not placed into the Writeable Static Area. This reduces the memory requirement for DLLs. This option has the same effect for all `const` variables as the `#pragma variable(`*`var_name`*`, NORENT)` directive. See the *z/OS C/C++ Language Reference* for more information on `pragma` directives.

Note that such `const` variables cannot be exported.

Interaction with #pragma variable

If the specification for a `const` variable in a `#pragma variable` directive is in conflict with the option, the `pragma variable` takes precedence. The compiler issues an an informational message.

Interaction with #pragma export

If you set the `ROCONST` option, and if there is a `#pragma export` for a `const` variable, the `pragma` directive takes precedence. The compiler issues an informational message. The variable will still be exported and the variable will be reentrant.

Effect on IPA Compile Step

If you specify the ROCONST option during the IPA Compile step, the information in the IPA object file reflects the state of each symbol.

Effect on IPA Link Step

If you specify the ROCONST option on the IPA Link step, it ignores the option. The reentrant/nonreentrant and const/nonconst state of each symbol are maintained during IPA optimization and code generation.

The IPA Link step merges and optimizes your application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition. Only compatible subprograms are included in a given partition. Compatible subprograms have the same ROCONST setting.

The ROCONST setting for a partition is set to the specification of the first subprogram that is placed in the partition. Subprograms that follow are placed in partitions that have the same ROCONST setting. A NOROCONST mode is placed in a NOROCONST partition, and a ROCONST is placed in a ROCONST partition.

The option value that you specified for each IPA object file on the IPA Compile step appears in the IPA Link step Compiler Options Map listing section.

The RENT, ROCONST, and ROSTRING options all contribute to the reentrant/nonreentrant state for each symbol. If any symbols within a partition are reentrant, the option section of the Partition Map displays the RENT compiler option.

The Partition Map sections of the IPA Link step listing and the object module END information section display the value of the ROCONST option.

ROSTRING | NOROSTRING

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOROSTRING						

CATEGORY: Object Code Control

```
►►──┬─ROS───┬──────────────────────────────────────────────────►◄
    └─NOROS─┘
```

The ROSTRING option informs the compiler that string literals are read-only. This option has the same effect as the #pragma strings(readonly) directive. See the *z/OS C/C++ Language Reference* for more information on pragma directives.

Specifying the `ROSTRING` option allows the compiler to place string literals into read-only memory. When you compile the program with the `RENT` option, such string literals are not placed into the Writeable Static Area (WSA). This reduces the memory requirement for DLLs.

Effect on IPA Compile Step

If you specify the `ROSTRING` option during the IPA Compile step, the information in the IPA object file reflects the state of each symbol.

Effect on IPA Link Step

If you specify the `ROSTRING` option on the IPA Link step, it ignores the option. The reentrant or nonreentrant state of each symbol is maintained during IPA optimization and code generation.

The Partition Map section of the IPA Link step listing and the object module do not display information about the `ROSTRING` option for that partition. The `RENT`, `ROCONST`, and `ROSTRING` options all contribute to the reentrant or nonreentrant state for each symbol. If any symbols within a partition are reentrant, the option section of the Partition Map displays the `RENT` compiler option.

ROUND

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
ROUND(N)						

CATEGORY: Object Code Control

```
►►──ROUND──(──┬─N─┬──)──────────────────────────────────────────────►◄
              ├─M─┤
              ├─P─┤
              └─Z─┘
```

The `ROUND(`*`mode`*`)` option sets the rounding mode for floating-point compilations at compile time where *mode* can be one of the following:

`N`	round to the nearest representable number
`M`	round towards minus infinity
`P`	round towards plus infinity
`Z`	round towards zero
ROUND()	is the same as `ROUND(N)`

The ROUND(mode) option only applies to IEEE floating-point mode. In hexadecimal mode, the rounding is always towards zero. If you specify ROUND(mode) in hexadecimal floating-point mode, where mode is not Z, the compiler ignores ROUND(mode) and issues a warning.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. The ROUND option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

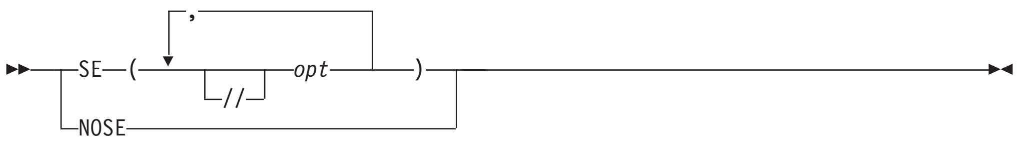
The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these section is a partition. The IPA Link step uses information from the IPA Compile step to ensure that an object is included in a compatible partition. Refer to the "FLOAT" on page 108 for further information.

SEARCH | NOSEARCH

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities** The c89, cc, and c++ utilities explicitly specify this option in the z/OS UNIX System Services shell. The suboptions are determined by the following: • Additional include search directories identified by the c89 -I options. Refer to Appendix F for more information. • z/OS UNIX environment variable settings: {_INCDIRS}, {_INCLIBS}, and {_CSYSLIB}. They are normally set during compiler installation to reflect the compiler and runtime include libraries. Refer to "Environment Variables" on page 567 for more information. This option is specified as NOSEARCH, SEARCH by these utilities, so it resets the SEARCH parameters you specify.					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
For C++, SE(//'CEE.SCEEH.+', //'CBC>SCLBH.+') For C, SE(//'CEE.SCEEH.+')						

CATEGORY: File Management



The SEARCH option directs the preprocessor to look for system include files in the specified libraries. System include files are those files that are associated with the

#include <filename> form of the #include preprocessor directive. See “Using Include Files” on page 308 for a description of the #include preprocessor directive.

For further information on library search sequences, see “Search Sequences for Include Files” on page 316.

The suboptions for the SEARCH option are identical to those for the LSEARCH option, as described on page “LSEARCH | NOLSEARCH” on page 138.

The SYSLIB ddname is considered the last suboption for SEARCH, so that specifying SEARCH (X) is equivalent to specifying SEARCH (X,DD:SYSLIB).

Any NOSEARCH option cancels all previous SEARCH specifications, and any SEARCH options that follow it are used. When more than one SEARCH compile option is specified, all libraries in the SEARCH options are used to find the system include files.

The NOSEARCH option instructs the preprocessor to search only those libraries that are specified on the SYSLIB DD statement.

Notes:
1. SEARCH allows the compiler to distinguish between header files that have the same name but reside in different data sets. If NOSEARCH is in effect, the compiler searches for header files only in the data sets concatenated under the SYSLIB DD statement. As the compiler includes the header files, it uses the first file it finds, which may not be the correct one. Thus the build may encounter unpredictable errors in the subsequent link-edit or bind, or may result in a malfunctioning application.
2. If the *filename* in the #include directive is in absolute form, searching is not performed. See “Determining whether the File Name is in Absolute Form” on page 313 for more details on absolute #include *filename*.

Effect on IPA Compile Step
The SEARCH option is used for source code searching, and has the same effect on an IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step
The IPA Link step accepts the SEARCH option, but ignores it.

SERVICE | NOSERVICE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOSERVICE						

CATEGORY: Debug/Diagnostic

```
►►──┬─SERV──────────┬──────────────────────────────────────────────►◄
    ├─(──string──)──┤
    └─NOSERV────────┘
```

The SERVICE option places a string in the object module. The string is loaded into memory when the program is executing. If the application fails abnormally, the string is displayed in the traceback.

For z/OS C, you can also specify this option in the source file by using the `#pragma options` directive. If the `SERVICE` option is specified both on a `#pragma` options directive and on the command line, the option that is specified on the command line will be used.

You must enclose your string within opening and closing parentheses. You do not need to include the string in quotes.

The following restrictions apply to the string specified:

- The string cannot exceed 64 characters in length. If it does, excess characters are removed, and the string is truncated to 64 characters. Leading and trailing blanks are also truncated.

  Note: Leading and trailing spaces are removed first and then the excess characters are truncated.
- All quotes that are specified in the string are removed.
- All characters, including DBCS characters, are valid as part of the string provided they are within the opening and closing parentheses.
- Parentheses that are specified as part of the string must be balanced. That is, for each opening parentheses, there must be a closing one. The parentheses must match after truncation.
- When using the `#pragma options` directive (C only), the text is converted according to the locale in effect.
- Only characters which belong to the invariant character set should be used, to ensure that the signature within the object module remains readable across locales.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

The `SERVICE` option has the same effect on the IPA Compile step (if you request code generation by specifying the `OBJECT` suboption of the `IPA` option) as it does on a regular compilation.

Effect on IPA Link Step

If you specify the `SERVICE` option on the IPA Compile step, or specify `#pragma options(SERVICE)` in your code, it has no effect on the IPA Link step. Only the `SERVICE` option you specify on the IPA Link step affects the generation of the service string for that step.

SEQUENCE | NOSEQUENCE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
C++ and C(V-format and HFS): `NOSEQUENCE` C(F-format): `SEQUENCE(73,80)`	`NOSEQUENCE`	`NOSEQUENCE`	`NOSEQUENCE`			

CATEGORY: Source Code Control

The `SEQUENCE` option defines the section of the input record that is to contain sequence numbers. No attempt is made to sort the input lines or records into the specified sequence or to report records out of sequence.

z/OS C++

```
►►──┬─SEQ───┬──────────────────────────────────────────────────►◄
    └─NOSEQ─┘
```

For z/OS C++ programs of variable file length, the `SEQUENCE` option defines columns 73 through 80 of the input record to contain sequence numbers. No attempt is made to sort the input lines or records into those columns or to report records out of sequence.

z/OS C

```
►►──┬─SEQ──(m,n)─┬─────────────────────────────────────────────►◄
    └─NOSEQ──────┘
```

Under z/OS C the `SEQUENCE` option has the additional syntax:

`SEQUENCE(`*m*`,`*n*`)`

where:

m Specifies the column number of the left-hand margin. The value of *m* must be greater than `0` and less than `32767`.

n Specifies the column number of the right-hand margin. The value of *n* must be greater than *m* and less than `32767`. An asterisk (*) can be assigned to *n* to indicate the last column of the input record. Thus, `SEQUENCE (74,*)` shows that sequence numbers are between column `74` and the end of the input record.

Note: If your program uses the #include preprocessor directive to include z/OS C library header files *and* you want to use the SEQUENCE option, you must ensure that the specifications on the SEQUENCE option do not include any columns from 20 through 50. That is, both *m* and *n* must be less than 20, or both must be greater than 50. If your program does not include any z/OS C library header files, you can specify any setting you want on the SEQUENCE option when the setting is consistent with your own include files.

Effect on IPA Compile Step

The SEQUENCE option is used for source code analysis, and has the same effect on an IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the SEQUENCE option, but ignores it.

SHOWINC | NOSHOWINC

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOSHOWINC	NOSHOWINC	NOSHOWINC	NOSHOWINC			

CATEGORY: Listing

```
►►──┬─SHOW───┬──────────────────────────────────────────────►◄
    └─NOSHOW─┘
```

The SHOWINC option instructs the compiler to show, in both the compiler listing and the pseudo-assembler listing, all include files processed. In the listing, the compiler replaces all #include preprocessor directives with the source that is contained in the include file. This option only applies if you also specify the SOURCE option.

In the z/OS UNIX System Services environment, this option is turned on by specifying -V when using the c89, cc or c++ commands.

Effect on IPA Compile Step

The SHOWINC option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the SHOWINC option, but ignores it.

SOURCE | NOSOURCE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOSOURCE	NOSOURCE (/dev/fd1)	NOSOURCE (/dev/fd1)	NOSOURCE (/dev/fd1)			

CATEGORY: Listing

```
►►──┬─SO───┬──┬──────────────────────────────────────────────┬──────────────────►◄
    └─NOSO─┘  └─(──┬─Sequential filename──────────────┬──)─┘
                   ├─Partitioned data set─────────────┤
                   ├─Partitioned data set (member)────┤
                   ├─Hierarchical filename────────────┤
                   └─Hierarchical directory───────────┘
```

The SOURCE option generates a listing that shows the original source input statements plus any diagnostic messages.

If you specify SOURCE(*filename*), the compiler places the listing in the file that you specified. If you do not specify a file name for the SOURCE option, the compiler uses the SYSCPRT ddname if you allocated one. Otherwise, the compiler constructs the file name as follows:

- If you are compiling a data set, the compiler uses the source file name to form the name of the listing data set. The high-level qualifier is replaced with the userid under which the compiler is running, and .LIST is appended as the low-level qualifier.
- if the source file is an HFS file, the listing is written to a file that has the name of the source file with a .lst extension in the current working directory.

The NOSOURCE option can optionally take a *filename* suboption. This *filename* then becomes the default. If you subsequently use the SOURCE option without a *filename* suboption, the compiler uses the *filename* that you specified in the earlier NOSOURCE. For example, the following specifications have the same result:

```
CXX HELLO (NOSO(/hello.lis) SO
CXX HELLO (SO(/hello.lis)
```

If you specify SOURCE and NOSOURCE multiple times, the compiler uses the last specified option with the last specified suboption. For example, the following specifications have the same result:

```
CXX HELLO (NOSO(/hello.lis) SO(/n1.lis)  NOSO(/test.lis) SO

CXX HELLO (SO(/test.lis)
```

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c89`, `cc` or `c++` commands.

Notes:

1. If you specify data set names with the `SOURCE`, `LIST`, or `INLRPT` option, the compiler combines all the listing sections into the last data set name specified.
2. If you use the following form of the command in a JES3 batch environment where xxx is an unallocated data set, you may get undefined results.

   `SOURCE(xxx)`

Effect on IPA Compile Step

The `SOURCE` option has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the `SOURCE` option, but ignores it.

SPILL | NOSPILL

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
`SPILL(128)`	`SPILL(128)`	`SPILL(128)`	`SPILL(128)`			

CATEGORY: Object Code Control

```
►►──┬─SP──(──size──)──────┬──────────────────────────────────────────►◄
    └─NOSP──┬──────────────┤
            └─(──size──)───┘
```

The `SPILL` option specifies the size of the spill area to be used for the compilation. When too many registers are in use at once, the compiler dumps some of the registers into temporary storage, called the spill area.

If you have to expand the spill area, you will receive a compiler message telling you the size to which you should increase it. Once you know the spill area that your source program requires, you can specify the required *size* (in bytes) as shown in the syntax diagram above. Alternatively for C code, you can add a `#pragma options(SPILL(`*size*`))` directive to your source. The maximum spill area size is `1073741823` bytes or $2^{30}–1$ bytes. Typically, you will only need to specify this option when compiling very large programs with `OPTIMIZE`.

Notes:

1. There is an upper limit for the combined area for your spill area, local variables and arguments passed to called functions at `OPT`. For best use of the stack, do not pass large arguments, such as structures, by value.

2. If you specify NOSPILL, the compiler defaults to SPILL(128).

You can specify the SPILL option using the #pragma option directive for C.

You can specify this option for a specific subprogram using the #pragma option_override(subprogram_name, OPT(SPILL,size)) directive.

Effect on IPA Compile Step

If you specify the SPILL option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

If you specify the SPILL option for the IPA Link step, the compiler sets the Compilation Unit values of the SPILL option that you specify. The IPA Link step Prolog listing section will display the value of this option.

If you do not specify the SPILL option in the IPA Link step, the setting from the IPA Compile step for each Compilation Unit will be used.

In either case, subprogram-specific SPILL options will be retained.

The IPA Link step merges and optimizes your application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to determine if a subprogram can be placed in a particular partition.

The initial overall SPILL value for a compilation unit is set to the IPA Link SPILL option value, if specified. Otherwise, it is the SPILL option that you specified during the IPA Compile step for the compilation unit.

The SPILL value for each subprogram in a partition is determined as follows:

- The SPILL value is set to the compilation unit SPILL value, unless a subprogram-specific SPILL option is present.
- During inlining, the caller subprogram SPILL value will be set to the maximum of the caller and callee SPILL values.

The overall SPILL value for a partition is set to the maximum SPILL value of any subprogram contained within that partition.

The option value that you specified for each IPA object file on the IPA Compile step appears in the IPA Link step Compiler Options Map listing section.

The Partition Map sections of the IPA Link step listing and the object module END information section display the value of the SPILL option. The Partition Map also displays any subprogram-specific SPILL values.

SRCMSG | NOSRCMSG

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOSRCMSG			NOSRCMSG			

CATEGORY: Debug/Diagnostic

```
►►──┬─SRCM───┬──────────────────────────────────────────────────►◄
    └─NOSRCM─┘
```

The `SRCMSG` option adds the corresponding source code lines to the diagnostic messages that are written to `stderr`. A finger line with a pointer to the column position may also be shown.

`NOSCRCMSG` indicates that the source lines are not added to the diagnostic messages.

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c++` command.

Effect on IPA Compile Step

The `SRCMSG` option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step issues a diagnostic message if you specify the `SRCMSG` option.

SSCOMM | NOSSCOMM

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOSSCOMM	NOSSCOMM	NOSSCOMM				

CATEGORY: Source Code Control

```
►►──┬─SS───┬────────────────────────────────────────────────────►◄
    └─NOSS─┘
```

The `SSCOMM` option instructs the C compiler to recognize two slashes (//) as the beginning of a comment terminates at the end of the line. It will continue to recognize /* */ as comments.

If you include your z/OS C program in your JCL stream DLM, be sure to change the delimiters so that your comments are recognized as z/OS C comments and not as JCL statements. For example:

```
//COMPILE.SYSIN DD DATA,DLM=@@
#include <stdio.h>
void main(){
// z/OS C comment
printf("hello world\n");
// A nested z/OS C /*  */ comment
}
@@
//* JCL comment
```

NOSSCOMM indicates that /* */ is the only valid comment format.

C++ Note: You can include the same delimiter in your JCL for C++ source code, however you do not need to use the SSCOMM option.

Effect on IPA Compile Step

The SSCOMM option is used for source code analysis, and has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the SSCOMM option, but ignores it.

START | NOSTART

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
START	START	START	START	START	START	START

CATEGORY: Object Code Control

```
►►──┬─STA───┬──────────────────────────────────────────────────────►◄
    └─NOSTA─┘
```

The START option specifies that CEESTART is to be generated whenever necessary.

NOSTART indicates that CEESTART is never to be generated.

Whenever you specify the START compiler option, a comment that indicates its use will be generated in your object module to aid you in diagnosing your program.

You can specify this option using the #pragma option directive for C.

Effect on IPA Compile Step

If you specify the START option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

The IPA Link step uses the value of the START option that you specify for that step. It does not use the value that you specify for the IPA Compile step.

STRICT | NOSTRICT

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
STRICT						

CATEGORY: Object Code Control

```
►►──┬─STRICT───┬───────────────────────────────────────────────►◄
    └─NOSTRICT─┘
```

The STRICT option instructs the compiler to perform computational operations in a rigidly-defined order such that the results are always determinable and recreatable.

NOSTRICT allows the compiler to reorder certain computations for better performance. However, the end result may differ from the result obtained when STRICT is specified.

In IEEE floating-point mode, NOSTRICT may turn on FLOAT(MAF). To avoid this behavior, explicitly specify FLOAT(NOMAF).

You can specify this option for a specific subprogram using the #pragma option_override(subprogram_name, OPT(STRICT)) directive.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to ensure that an object is included in a compatible partition. See "FLOAT" on page 108 for more information on the effect of the STRICT option on the IPA Link step.

STRICT_INDUCTION | NOSTRICT_INDUCTION

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOSTRICT_INDUCTION						

CATEGORY: Object Code Control

```
►►──┬─STRICT_INDUC───┬──────────────────────────────────────────────►◄
    └─NOSTRICT_INDUC─┘
```

The STRICT_INDUCTION option instructs the compiler to disable loop induction variable optimizations. These optimizations have the potential to alter the semantics of your program. Such optimizations can change the result of a program if truncation or sign extension of a loop induction variable occurs as a result of variable overflow or wrap-around.

The STRICT_INDUCTION option only affects loops which have an induction (loop counter) variable declared as a different size than a register. Unless you intend such variables to overflow or wrap-around, use NOSTRICT_INDUCTION.

You can use the ST_IND alias for STRICT_INDUCTION, and the NOST_IND alias for NOSTRICT_INDUCTION.

Effect on IPA Compile Step

If you specify the STRICT_INDUCTION option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step

The IPA Link step merges and optimizes your application’s code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to ensure that an object is included in a compatible partition.

The compiler sets the value of the STRICT_INDUCTION option for a partition to the value of the first subprogram that is placed in the partition. During IPA inlining, subprograms with different STRICT_INDUCTION settings may be combined in the same partition. When this occurs, the resulting partition is always set to STRICT_INDUCTION.

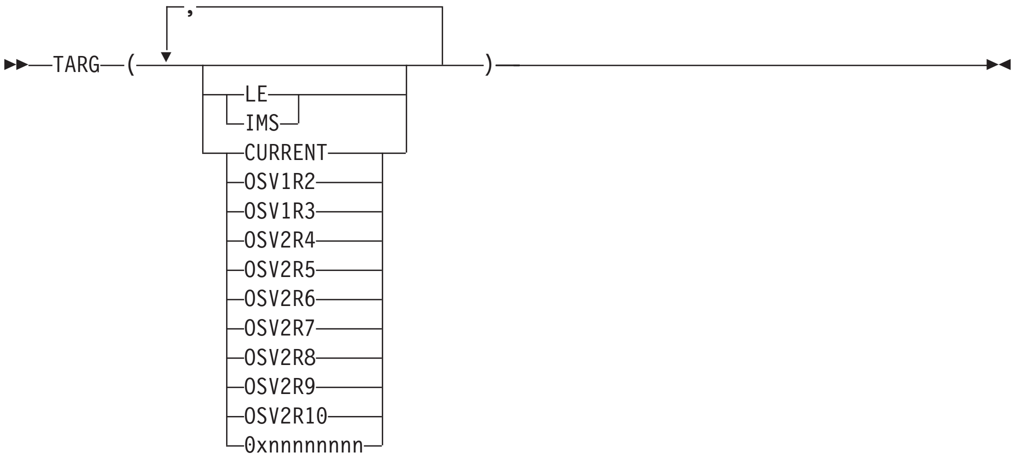
You can override the setting of STRICT_INDUCTION by specifying the option on the IPA Link step. If you do so, all partitions will contain that value, and the prolog section of the IPA Link step listing will display the value.

TARGET

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
TARGET(LE, CURRENT)	TARGET(LE)	TARGET(LE)	TARGET(LE)			

CATEGORY: Program Execution and Object Code Control



With the TARGET option, you can specify the runtime environment and OS/390 release for your program's object module that z/OS C/C++ generates. This enables you to generate code that is downward compatible with earlier levels of the operating system while at the same time disallowing you from using library functions not available on the targetted release. With the TARGET option, you can compile and link an application on a higher level system, and run the application on a lower level system.

To use the TARGET option, select a runtime environment of either LE or IMS. Then select the desired OS/390 release, for example, CURRENT or OSV1R2. If you do not select a runtime environment or OS/390 release, the compiler uses a default of TARGET(LE, CURRENT).

TARGET()	Generates object code to run under z/OS Language Environment. It is the same as TARGET (LE, CURRENT).

The following suboptions target the runtime environment:

TARGET(LE)	Generates object code to run under z/OS Language Environment. This is the default.

`TARGET(IMS)`	Generates object code to run under the Information Management System (IMS) subsystem. If you are compiling the main program, you must also specify the `PLIST(OS)` option.

For more information about these suboptions refer to “TARGET Runtime Environment Suboptions (LE,IMS)” on page 184.

The following suboptions target the OS/390 release at program run time:

`TARGET(CURRENT)`	Generates object code to run under the same version of OS/390 with which the compiler ships. As the compiler ships with R10 of OS/390, `TARGET(CURRENT)` is the same as `TARGET(OSV2R10)`. This is the default. [2]
`TARGET(OSV1R2)`	Generates object code to run under OS/390 Version 1 Release 2 and subsequent releases.
`TARGET(OSV1R3)`	Generates object code to run under OS/390 Version 1 Release 3 and subsequent releases.
`TARGET(OSV2R4)`	Generates object code to run under OS/390 Version 2 Release 4 and subsequent releases.
`TARGET(OSV2R5)`	Generates object code to run under OS/390 Version 2 Release 5 and subsequent releases.
`TARGET(OSV2R6)`	Generates object code to run under OS/390 Version 2 Release 6 and subsequent releases.
`TARGET(OSV2R7)`	Generates object code to run under OS/390 Version 2 Release 7 and subsequent releases.
`TARGET(OSV2R8)`	Generates object code to run under OS/390 Version 2 Release 8 and subsequent releases.
`TARGET(OSV2R9)`	Generates object code to run under OS/390 Version 2 Release 9 and subsequent releases.
`TARGET(OSV2R10)`	Generates object code to run under OS/390 Version 2 Release 10 and subsequent releases.
`TARGET(0x`*nnnnnnnn*`)`	An eight digit hexadecimal literal string that specifies an operating system level. This string is intended for library providers and vendors to test header files on future releases and is an advanced feature. Most applications should use the other release suboptions. The layout of this literal is the same as the `__TARGET_LIB__` macro. For more information on using this literal, please see “Using the Hexadecimal String Literal Suboption” on page 182.

For more information about these suboptions refer to “TARGET OS/390 Release Suboptions” on page 181.

2. Note that for some releases of OS/390, z/OS C/C++ might not ship a new version of the compiler. The same version of the compiler is then shipped with more than one OS/390 release. The compiler is designed to run on all these OS/390 releases. In this case, the compiler sets `CURRENT` to the OS/390 release on which it is running. (It does so by querying the Language Environment Library version of the system.) You can specify an `OSV`*x*`R`*y* suboption that corresponds to a release that is earlier or the same as `CURRENT`. You cannot specify an `OSV`*x*`R`*y* suboption that corresponds to a release later than `CURRENT`.

The compiler generates a comment that indicates the value of `TARGET` in your object module to aid you in diagnosing problems in your program.

If you specify more than one suboption from each group of suboptions (that is, the runtime environment, or the release) the compiler uses the last specified suboption for each group.

The compiler applies and resolves defaults after it views all the entered suboptions. For example, `TARGET(LE, 0x22040000, IMS, OSV1R2, LE)` resolves to `TARGET(LE, OSV1R2)`. `TARGET(LE, 0x22040000, IMS, OSV1R2)` resolves to `TARGET(IMS, OSV1R2)`. And `TARGET(LE, 0x22040000, IMS)` resolves to `TARGET(IMS, 0x22040000)`.

The default value of the `ARCHITECTURE` compiler option depends on the value of the TARGET OS/390 release suboption. For TARGET(OSV2R10 and above), the default is ARCH(2). For TARGET(OSV2R9 and below), the default is ARCH(0)

TARGET OS/390 Release Suboptions

The `TARGET` release suboptions (`CURRENT`, `OSV1R2`, `OSV1R3`, `OSV2R4`, `OSV2R5`, `OSV2R6`, `OSV2R7`, `OSV2R8`, `OSV2R9`, and `OSV2R10`) allow you to generate code that can be executed on a particular release of an OS/390 system, and on subsequent releases.

As of OS/390 V2R10, you can use the current Language Environment data sets during the assembly, compilation, pre-link, link-edit and bind phases. Prior to OS/390 V2R10, you had to use the Language Environment data sets for the targetted release. With the downward compatibility support provided by Language Environment if you target:

- OSV2R7 and above: The C/C++ compiler will issue a syntax error if the program that you are trying to compile contains functions that are not supported in your target release. The C/C++ headers provided by Language Environment, as of OS/390 V2R10, differentiate function provided in OS/390 V2R7 and higher. This support allows application developers to ″target″ a specific release, in order to ensure the application has not taken advantage of any new C/C++ library functions.
- OSV2R4 through OSV2R6: Although the system header files are not updated to differentiate functions provided in this range, toleration PTFs are available so that if an application attempts to exploit a Language Environment function that is unavailable on the release of OS/390 the application is executed on, Language Environment will raise a new condition. However, with an unhandled condition, the application is terminated. In this case, instead of seeing an error during application development time, a condition will be produced at execution time when new functions are used.
- OSV1R2 and OSV1R3: The programmer is responsible for ensuring that they are not exploiting a Language Environment function that is unavailable on the release of OS/390 that the application is executed on, and apart from the unpredictable execution-time failure resulting from trying to call a non-existent function, no error or condition will be provided at application development or execution time.

For more information on the support available with Language Environment see the section on Downward Compatibility Considerations in the *z/OS Language Environment Programming Guide*.

In order to use these suboptions, you must:

- Use the `OSV2R10` class library header files (found in the CBC.SCLBH.* data sets) during compilation

- Use the class library data sets for the targetted release during pre-link, link-edit, and bind. For information on using previous releases of the class libraries refer to Informational APAR II11576 (available at http://www.ibm.com/software/ad/c390/service/ii11576.html)

For example, to generate code to execute on an OS/390 V1R3 system, using an OS/390 V2R10 application development system:

- Use the V1R1 Language Environment data sets (CEE.SCEE*) during the assembly, compilation, pre-link, link-edit, and bind phases.
- Use the V1R3 class library data sets (SCLBCPP, SCLBOBC, SCLBOXL, SCLBSID, SCLBXL) during pre-link, link-edit, and bind.
- Specify the compiler option TARGET(OSV1R3) on the C/C++ compiles. Note: The programmer is responsible for ensuring that they are not exploiting any Language Environment functions that are unavailable on OS/390 V1R3.

See "Appendix A. Prelinking and Linking z/OS C/C++ Programs" on page 461 for details on prelinking and linking applications.

These compiler suboptions will not allow you to exploit new functions provided on the newer OS/390 system. Rather, they allow you to build an application on a newer OS/390 system and run it on a older system.

When you invoke the `TARGET(OSV`*x*`R`*y*`)` release suboptions, the compiler sets the `__TARGET_LIB__` and `__LIBREL__` macros. See the *z/OS C/C++ Language Reference* for more information about these macros.

Using the Hexadecimal String Literal Suboption: This hexadecimal literal string enables you to specify an operating system level. It is an advanced feature that is intended for library providers and vendors to test header files on future releases. Most applications should use the other release suboptions instead of this string literal. The layout of this literal is the same as the `__TARGET_LIB__` macro.

The compiler checks to ensure that there are exactly 8 hexadecimal digits. The compiler performs no further validation checks.

The compiler uses a two step process to specify the operating system level:

- The hexadecimal value will be used, as specified, to set the `__TARGET_LIB__` macro.
- The compiler determines the operating system level implied by this literal.

If the level corresponds to a valid suboption name, the compiler behaves as though that suboption is specified. Otherwise, the compiler uses the next lower operating system suboption name. If there is no lower suboption name, the compiler uses `TARGET(OSV1R2)`. Note that the compiler sets the `__TARGET_LIB__` macro to the value that you specify, even if it does not correspond to a valid operating system level. Following are some examples:

`TARGET(0x22060000)`
: Equivalent to `TARGET(OSV2R6)`.

`TARGET(0x23120000)`
: This does not match any existing operating system release suboption name. The next lower operating system level implied by this literal, which the compiler considers valid, is `CURRENT`. Thus, the compiler sets the `__TARGET_LIB__` macro to `0x23120000`, and behaves as though you have specified `TARGET(CURRENT)`.

`TARGET(0x22040001)`
: This does not match any existing operating system release suboption name because of the `1` in the last digit. The next lower operating system level implied by this literal which the compiler considers valid is `OSV2R4`. Thus, the compiler sets the `__TARGET_LIB__` macro to `0x22040001`, and behaves as though you have specified `TARGET(OSV2R4)`.

`TARGET(0x21010000)`
: This does not match any existing operating system release suboption name, and specifies a release earlier than the earliest supported release, `OSV1R2`. In this instance, the compiler sets the `__TARGET_LIB__` macro to `0x21010000`, and behaves as though you have specified `TARGET(OSV1R2)`.

Restrictions for C/C++: All input libraries used during the application build process must be the appropriate level for the target OS/390 system.

- The OS/390 V2R10 level of the Language Environment data sets can be used to target to previous releases. Use these Language Environment data sets during the assembly, compilation, pre-link, link-edit, and bind phases.
- For C++ class libraries, use the `OSV2R10` class library header files during compilation; use the class library data sets for the targetted release during pre-link, link-edit, and bind.
- Ensure that any other libraries incorporated in the application, are compatible with the target OS/390 system.

While there are no restrictions on the use of `ARCH` and `TUNE` with `TARGET`, ensure that the level specified is consistent with the target hardware.

TARGET Release Suboption	Restrictions
• `CURRENT` • `OSV2R10`	None. All options and features are allowed.
• `CURRENT` • `OSV2R10` • `OSV2R9` • `OSV2R8` • `OSV2R7` • `OSV2R6`	All options and features except XPLINK and GOFF are allowed.
• `OSV2R5` • `OSV2R4`	• The `FLOAT` option behaves as `FLOAT(HEX)`. • The `ROUND` option is ignored by the compiler. • The `long long` data type is flagged as an error.
• `OSV1R3` • `OSV1R2`	• The `DLL(CBA)` option behaves as `DLL(NOCBA)`.

Only options or features that cannot be supported on that operating system level are disabled. For example, `STRICT_INDUCTION` is allowed on all operating system levels. An option or feature that is disabled by one operating system level is also disabled by all earlier operating system levels.

Restrictions for C: TARGET(OSV*x*R*y*) is not permitted in a #pragma target() directive.

If you specify TARGET(OSV*x*R*y*) on the command line, and one or more of the disallowed options is specified, the compiler issues a warning message and disables the option.

Effect on IPA Compile Step: If you specify the TARGET option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

When you are performing the IPA compile to generate IPA Object files, ensure that you are using the appropriate header library files.

Effect on IPA Link Step: If you specify TARGET on the IPA Link step, it overrides the TARGET value that you specified for the IPA Compile step.

The IPA Link step accepts the OS/390 release suboptions, for example, CURRENT or OSV1R2. However, when using TARGET suboptions ensure that:
- All IPA Object files are compiled with the appropriate TARGET suboption and header files
- All non-IPA object files are compiled with the appropriate TARGET suboption and header files
- All other input libraries are compatible with the specified OS/390 runtime release

TARGET Runtime Environment Suboptions (LE,IMS)

The TARGET Runtime Environment suboption allows you to select a runtime environment of either Language Environment or IMS.

Effect on IPA Compile Step: If you specify the TARGET option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the IPA(OBJECT) option.

Effect on IPA Link Step: If you specify TARGET on the IPA Link step, it has the following effects:
- It overrides the TARGET value that you specified for the IPA Compile step.
- It overrides the value that you specified for #pragma runopts(ENV). If you specify TARGET(LE) or TARGET(), the IPA Link step sets the value of #pragma runopts(ENV) to OS/390. If you specify TARGET(IMS), the IPA Link step sets the value of #pragma runopts(ENV) to IMS.
- It may override the value that you specified for #pragma runopts(PLIST) or the PLIST compiler option. If you specify TARGET(LE) or TARGET(), and you set the value set for the PLIST option to something other than HOST, the IPA Link step sets the values of #pragma runopts(PLIST) and the PLIST compiler option to IMS. If you specify TARGET(IMS), the IPA Link step unconditionally sets the values of the PLIST compiler option and #pragma runopts(PLIST) to IMS.

TEMPINC | NOTEMPINC

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
PDS: `TEMPINC(TEMPINC)` HFS Directory: `TEMPINC(./tempinc)`			`TEMPINC (tempinc)`			

CATEGORY: File Management

```
►►──┬─TEMPINC───┬──┬────────────┬──────────────────────────────────►◄
    └─NOTEMPINC─┘  └─(location)─┘
```

`TEMPINC`(*location*) places all template instantiation files into *location*, which may be a PDS or an HFS directory. If you do not specify a *location*, the compiler places all template instantiation files in a default location. If the source resides in a data set, the default location is a PDS with a low-level qualifier of TEMPINC. The high-level qualifier is the userid under which the compiler is running. If the source resides in an HFS file, the default location is the HFS directory `./tempinc`.

The `NOTEMPINC` option can optionally take a *filename* suboption. This *filename* then becomes the default. If you subsequently use the `TEMPINC` option without a *filename* suboption, then the compiler uses the *filename* that you specified in the earlier `NOTEMPINC`. For example, the following specifications have the same result:

```
CXX HELLO (NOTEMPINC(/hello) TEMPINC

CXX HELLO (TEMPINC(/hello)
```

If you specify `TEMPINC` and `NOTEMPINC` multiple times, the compiler uses the last specified option with the last specified suboption. For example, the following specifications have the same result:

```
CXX HELLO (NOTEMPINC(/hello) TEMPINC(/n1)  NOTEMPINC(/test) TEMPINC

CXX HELLO (TEMPINC(/test)
```

If you have large numbers of recursive templates, consider using `FASTT`. See "FASTTEMPINC | NOFASTTEMPINC" on page 106 for details.

Note: If you use the following form of the command in a JES3 batch environment where xxx is an unallocated data set, you may get undefined results.

```
TEMPINC(xxx)
```

Effect on IPA Compile Step
The TEMPINC option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step
The IPA Link step issues a diagnostic message if you specify the TEMPINC option for that step.

TERMINAL | NOTERMINAL

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
TERMINAL	TERMINAL	TERMINAL	TERMINAL	TERMINAL	TERMINAL	TERMINAL

CATEGORY: Debug/Diagnostic

```
►►──┬─TERM───┬──────────────────────────────────────────────────────►◄
    └─NOTERM─┘
```

The TERMINAL option directs all of the compiler's diagnostic messages to stderr.

If you specify NOTERMINAL, then no diagnostic messages are sent to stderr. Under z/OS batch, the default for stderr is SYSPRINT.

If you specify the PPONLY option, the compiler turns on TERM.

Effect on IPA
The TERMINAL option affects both the IPA Compile and the IPA Link steps in the same way that it affects a regular compilation.

TEST | NOTEST

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
Default for C++ compile: NOTEST(HOOK) Default for C compile: NOTEST(HOOK,SYM, BLOCK,LINE, PATH)	NOTEST-g sets TEST -s sets NOTEST	NOTEST-g sets TEST -s sets NOTEST	NOTEST-g sets TEST -s sets NOTEST	NOTEST	NOTEST	NOTEST

z/OS C/C++

```
►►──┬─TEST───┬──(──┬────────┬──)──────────────────────────────►◄
    └─NOTEST─┘     ├─HOOK───┤
                   └─NOHOOK─┘
```

The TEST suboptions that are common to C compile, C++ compile, and IPA Link steps are:

HOOK | NOHOOK	When NOOPT is in effect	When OPT is in effect
HOOK	• For C++ compile, generates all possible hooks For C compile, generates all possible hooks based on current settings of BLOCK, LINE, and PATH suboptions. For IPA Link, generates Function Entry, Function Exit, Function Call, and Function Return hooks • For C++ compile, generates symbol information For C compile, generates symbol information unless NOSYM is specified For IPA Link, does not generate symbol information	• Generates Function Entry, Function Exit, Function Call and Function Return hooks • Does not generate symbol information
NOHOOK	• Does not generate any hooks • For C++ compile, generates symbol information. For C compile, generates symbol information based on the current settings of SYM and BLOCK For IPA Link, does not generate any symbol information	• Does not generate any hooks • Does not generate symbol information

The TEST suboptions generate symbol tables and program hooks. The Debug Tool uses these tables and hooks to debug your program. The Performance Analyzer uses these hooks to trace your program. The choices you make when compiling your program affect the amount of Debug Tool function available during your debugging session. These choices also impact the ability of the Performance Analyzer to trace your program.

To look at the flow of your code with the Debug Tool, or to trace the flow of your code with the Performance Analyzer, use the `HOOK` suboption with `OPT` in effect. These suboptions generate function entry, function exit, function call, and function return hooks. They do not generate symbol information.

When `NOOPT` is in effect, and you use the `HOOK` suboption, the debugger runs slower, but all the Debug Tool commands such as `AT ENTRY *` are available. You must specify the `HOOK` suboption in order to trace your program with the Performance Analyzer.

If you specify the `NOTEST` option, debugging information is not generated and you cannot trace your program with the Performance Analyzer.

You can use the `CSECT` option with the `TEST` option to place your debug information in a named CSECT. This enables the compiler and linker to collect the debug information in your module together which may improve the runtime performance of your program.

If you specify the `INLINE` and `TEST` compiler options when `NOOPTIMIZE` is in effect, `INLINE` is ignored.

If you specify the `TEST` option, the compiler turns on `GONUMBER`.

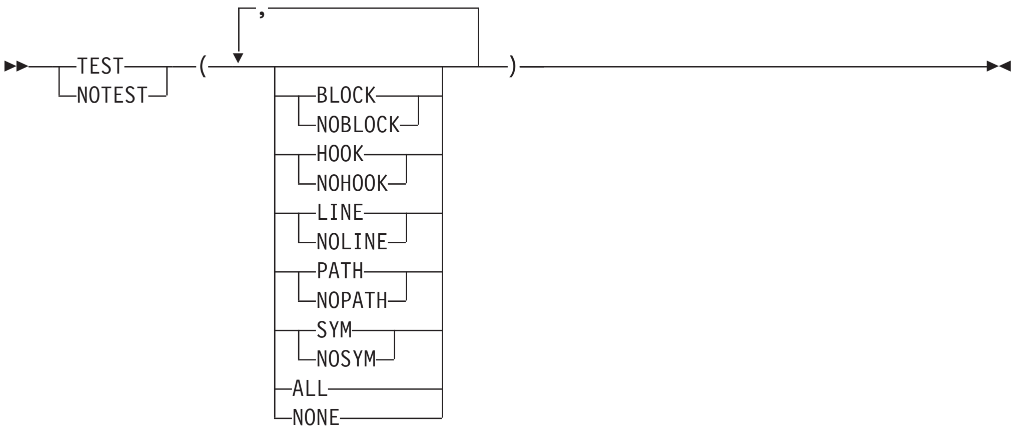
You can specify this option using the `#pragma option` directive for C.

Note: If your code uses any of the following, you cannot debug it with the MFI Debug Tool:

- IEEE code
- Code that uses the long long data type
- Code that runs in a POSIX environment

You must use either the C/C++ Productivity Tools for z/OS or dbx. For further information on the MFI Debug Tool, refer to the *Debug Tool User's Guide and Reference*, SC09-2137.

Additional z/OS C Compile Suboptions



The TEST suboptions BLOCK, LINE, and PATH regulate the points where the compiler inserts program hooks. When you set breakpoints, they are associated with the hooks which are used to instruct the Debug Tool where to gain control of your program.

The symbol table suboption SYM regulates the inclusion of symbol tables into the object output of the compiler. The Debug Tool uses the symbol tables to obtain information about the variables in the program.

SYM
: Generates symbol tables in the program's object output that gives you access to variables and other symbol information.
- You can reference all program variables by name, allowing you to examine them or use them in expressions.
- You can use the Debug Tool command GOTO to branch to a label (paragraph or section name).
- The Performance Analyzer does not use symbol information. Specify NOSYM if you want to trace the program with the Performance Analyzer.

BLOCK
: Inserts only block entry and exit hooks into your program's object output. A block is any number of data definitions, declarations, or statements that are enclosed within a single set of braces. BLOCK also creates entry hooks and exit hooks for nested blocks. If SYM is enabled, symbol tables are generated for variables local to these nested blocks.
- You can only gain control at entry and exit of blocks.
- Issuing a command such as STEP causes your program to run, until it reaches the exit point.
- The Performance Analyzer does not use block entry and exit hooks. Specify NOBLOCK if you want to trace the program with the Performance Analyzer.

LINE
: Generates hooks at most executable statements. Hooks are not generated for the following:
- Lines that identify blocks (lines that contain braces)
- Null statements
- Labels
- Statements that begin in an #include file.
- The Performance Analyzer does not use statement hooks. Specify NOLINE if you want to trace the program with the Performance Analyzer.

PATH
: Generates hooks at all path points; for example, hooks are inserted at if-then-else points before a function call and after a function call.
- This option does not influence the generation of entry and exit hooks for nested blocks. You must specify the BLOCK suboption if you desire such hooks.
- The Debug Tool can gain control only at path points and block entry and exit points. If you attempt to STEP through your program, the Debug Tool gains control only at statements that coincide with path points, giving the appearance that not all statements are executed.
- The Debug Tool command GOTO is valid only for statements and labels that coincide with path points.
- The Performance Analyzer uses function call and function return hooks. Specify PATH if you want to trace the program with the Performance Analyzer.

`ALL` Inserts block and line hooks, and generates symbol table. Hooks are generated at all statements, all path points (if-then-else, calls, and so on), and all function entry and exit points.

`ALL` is equivalent to `TEST(HOOK, BLOCK, LINE, PATH, SYM)`.

`NONE` Generates all compiled-in hooks only at function entry and exit points. Block hooks and line hooks are not inserted, and the symbol tables are suppressed.

`TEST(NONE)` is equivalent to `TEST(HOOK, NOBLOCK, NOLINE, NOPATH, NOSYM)`.

Note: When the `OPTIMIZE` and `TEST` options are both specified, the `TEST` suboptions are set by the compiler to `TEST(HOOK, NOBLOCK, NOLINE, NOPATH, NOSYM)` regardless of what the user has specified. The behavior of the `TEST` option in this case is as described in the table in the z/OS C/C++ section of the `TEST | NOTEST` option for the `HOOK` suboption.

For more information on debugging your program, see the *Debug Tool User's Guide and Reference*.

For z/OS C compile, you can specify the `TEST | NOTEST` option on the command line and in the `#pragma options` preprocessor directive. When you use both methods, the option on the command line takes precedence. For example, if you usually do not want to generate debugging information when you compile a program, you can specify the `NOTEST` option on a `#pragma options` preprocessor directive. When you do want to generate debugging information, you can then override the `NOTEST` option by specifying `TEST` on the command line rather than editing your source program. Suboptions that you specified in a `#pragma options(NOTEST)` directive, or with the `NOTEST` compiler option, are used if `TEST` is subsequently specified on the command line.

Effect on IPA Compile Step

On the IPA Compile step, you can specify all of the `TEST` suboptions that are appropriate for the language of the code that you are compiling. However, they affect processing only if you requested code generation, and only the conventional object file is affected. If you specify the `NOOBJECT` suboption of the `IPA` compiler option on the IPA Compile step, the IPA Compile step ignores the `TEST` option.

Effect on IPA Link Step

The IPA Link step supports only the `TEST`, `TEST(HOOK)`, `TEST(NOHOOK)`, and `NOTEST` options. If you specify `TEST(HOOK)` or `TEST`, the IPA Link step generates function call, entry, exit, and return hooks. It does not generate symbol table information. If you specify `NOTEST`, the IPA Link step does not generate any debugging information. If you specify `TEST(NOHOOK)`, the IPA Link step generates limited debug information without any hooks. If you specify any other `TEST` suboptions for the IPA Link step, it turns them off and issues a warning message.

TUNE

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
TUNE(3) If the TUNE level is lower than the specified ARCH level, the compiler forces TUNE to match the ARCH level.						

CATEGORY: Object Code Control

```
▶▶──TUN(n)──────────────────────────────────────────────────────────▶◀
```

The TUNE option specifies the architecture for which the executable program will be optimized. The TUNE level controls how the compiler selects and orders the available machine instructions, while staying within the restrictions of the ARCH level in effect. It does so in order to provide the highest performance possible on the given TUNE architecture from those that are allowed in the generated code. It also controls instruction scheduling (the order in which instructions are generated to perform a particular operation). Note that TUNE impacts performance only; it does not impact the processor model on which you will be able to run your application.

Select TUNE to match the architecture of the machine where your application will run most often. Use TUNE in cooperation with ARCH. TUNE must always be greater or equal to ARCH because you will want to tune an application for a machine on which it can run. The compiler enforces this by adjusting TUNE up rather than ARCH down. TUNE does not specify where an application can run. It is primarily an optimization option. For many models, the best TUNE level is not the best ARCH level. For example, the correct choices for model 9672-Rx5 (G4) are ARCH(2) and TUNE(3). For more information on the interaction between TUNE and ARCH see "ARCHITECTURE" on page 81.

Specify the group to which a model number belongs as a sub-parameter. If you specify a model which does not exist or is not supported, a warning message is issued stating that the suboption is invalid and that the default will be used.

Current models that are supported:

0 This option generates code that is executable on all models, but it will not be able to take advantage of architectural differences on the models specified below.

1 This option generates code that is executable on all models but is optimized for the following models:
- 9021-520, 9021-640, 9021-660, 9021-740, 9021-820, 9021-860, and 9021-900
- 9021-xx1, 9021-xx2, and 9672-Rx2 (G1)

2 This option generates code that is executable on all models but that is optimized for the following models:
- 9672-Rx3 (G2), 9672-Rx4 (G3), and 2003
- 9672-Rx1, 9672-Exx, and 9672-Pxx

3 This option is the default. This option generates code that is executable on all models but that is optimized for the following and follow-on models: 9672-Rx5 (G4), 9672-xx6 (G5), and 9672-xx7 (G6).

Note: For the above system machine models, x indicates any value. For example, 9672-Rx4 means 9672-RA4 through to 9672-RX4, not just 9672-RX4.

A comment that indicates the level of the TUNE option will be generated in your object module to aid you in diagnosing your program.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

If you specify the `TUNE` option for any compilation unit in the IPA Compile step, the compiler saves information for the IPA Link step. This option also affects the regular object module if you request one by specifying the `IPA(OBJECT)` option.

Effect on IPA Link Step

The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition.

If you specify the `TUNE` option for the IPA Link step, it uses the value of the option you specify. The value you specify appears in the IPA Link step Prolog listing section and all Partition Map listing sections.

If you do not specify the option on the IPA Link step, the value it uses for a partition depends upon the `TUNE` option you specified during the IPA Compile step for any compilation unit that provided code for that partition. If you specified the same `TUNE` value for all compilation units, the IPA Link step uses that value. If you specified different `TUNE` values, the IPA Link step uses the highest value of `TUNE`.

If the resulting level of `TUNE` is lower than the level of `ARCH`, `TUNE` is set to the level of `ARCH`.

The Partition Map section of the IPA Link step listing, and the object module display the final option value for each partition. If you override this option on the IPA Link step, the Prolog section of the IPA Link step listing displays the value of the option.

The Compiler Options Map section of the IPA Link step listing displays the value of the `TUNE` option that you specified on the IPA Compile step for each object file.

UNDEFINE

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
no action						

CATEGORY: Preprocessor

```
          ┌─,────┐
          ↓      │
►►──UNDEF──(──name──┴──)──────────────────────────────────►◄
```

UNDEFINE(name) removes any value that name may have and makes its value undefined. For example, if you set OS2 to 1 with DEF(OS2=1), you can use UNDEF(OS2) option to remove that value.

In the z/OS UNIX System Services environment, you can unset variables by specifying -U when using the c89, cc, or c++ commands.

Effect on IPA Compile Step

The UNDEFINE option is used for source code analysis, and has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the UNDEFINE option, but ignores it.

UPCONV | NOUPCONV

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOUPCONV	UPCONV	NOUPCONV				

CATEGORY: Source Code Control

```
►►──┬─UPC───┬──────────────────────────────────────────────►◄
    └─NOUPC─┘
```

The UPCONV option causes the z/OS C compiler to follow unsignedness preserving rules when doing z/OS C/C++ type conversions; that is, when widening all integral types (char, short, int, long). Use this option when compiling older z/OS C/C++ programs that depend on the older conversion rules.

Whenever you specify the UPCONV compiler option, a comment noting its use will be generated in your object module to aid you in diagnosing your program.

Effect on IPA Compile Step

The UPCONV option is used for source code analysis, and has the same effect on the IPA Compile step that it does on a regular compilation.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Link Step

The IPA Link step accepts `UPCONV` option, but ignores it.

USEPCH | NOUSEPCH

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOUSEPCH`	`NOUSEPCH(./)`	`NOUSEPCH(./)`	`NOUSEPCH(./)`			

CATEGORY: File Management

```
►►──┬─USEP───┬───┬────────────────────────────────────────────┬──────────────►◄
    └─NOUSEP─┘   └─(──┬─Sequential filename──────────────┬─)─┘
                      ├─Partitioned data set─────────────┤
                      ├─Partitioned data set (member)────┤
                      ├─Hierarchical filename────────────┤
                      └─Hierarchical directory───────────┘
```

The `USEP` option instructs the compiler to use precompiled header files. If you use `USEP` alone, the compiler searches for the specified file and uses it. If the file you specified does not exist, the compiler continues with a normal compilation.

If you specify the `GENP` and `USEP` options together, the compiler determines if the last file that is specified exists. If it does, the compiler updates the file if necessary, and `USEP` takes effect. If it does not exist, the compiler creates the file, and `USEP` takes effect. If you consistently use both options, for example by coding them in your JCL, you can ensure that you are always using current precompiled header files.

If you specify `USEP(filename)` or `GENP(filename) USEP`, the compiler uses the specified name for the precompiled header file. If you do not specify a filename for either option, the compiler uses the `SYSCPCH` ddname if you allocated one. Otherwise, the compiler constructs the file name as follows:

- If you are compiling a data set, the compiler uses the source file name to form the name of the precompiled header file data set. The high-level qualifier is replaced with the userid under which the compiler is running, and `PCH` (for C) or `PCHPP` (for C++) is appended as the low-level qualifier.
- If the source file is an HFS file, the precompiled header file name is formed using the name of the source file with a `.pch` (for C) or `.pchpp`(for C++) extension in the current working directory.

For more information on using `GENP` and `USEP` together, see "Using the GENP and USEP Compiler Options" on page 325.

Notes:

1. The compiler ignores this option if you specify the options PPONLY, SHOWINC, or EXPMAC.
2. You cannot use a C precompiled header file for C++, or a C++ precompiled header file for C.
3. If you specify different file names with the GENP and USEP options, the file name that is specified last is used with both options.

Effect on IPA Compile Step

The USEP option is used for source code analysis, and has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

The IPA Link step accepts the USEP option, but ignores it.

WSIZEOF | NOWSIZEOF

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOWSIZEOF						

CATEGORY: Source Code Control

```
►►──┬─WSIZEOF───┬──────────────────────────────────────────────────►◄
    └─NOWSIZEOF─┘
```

When you use the WSIZEOF option, sizeof returns the size of the widened type for function return types instead of the size of the original return type. For example, if you compile the following program with the WSIZEOF option, the value of i is 4.

```
char foo();
i = sizeof foo();
```

C/C++ compilers prior to and including C/C++ MVS/ESA Version 3 Release 1 returned the size of the widened type instead of the original type for function return types.

The z/OS C/C++ compiler now gives i the value 1, which is the size of the original type *char*.

If your source code depends on the behavior of the old compilers, use the WSIZEOF option to return the size of widened type for function return types.

The `WSIZEOF` option has exactly the same effect as putting a `#pragma wsizeof(on)` at the beginning of your source file. For more information on `#pragma wsizeof(on)`, see *z/OS C/C++ Language Reference*.

You cannot specify the `WSIZEOF` option in a `#pragma options` directive.

Effect on IPA Compile Step

The `WSIZEOF` option has the same effect on the IPA Compile step that it does on a regular compilation.

Effect on IPA Link Step

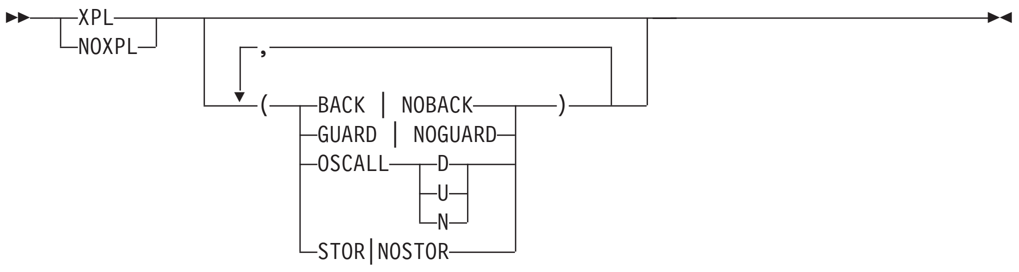
The IPA Link step accepts the `WSIZEOF` option, but ignores it.

XPLINK | NOXPLINK

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
NOXPLINK						

CATEGORY: Object Code Control



The `XPLINK` (Extra Performance Linkage) option instructs the compiler to generate extra performance linkage for subroutine calls. Use this option to increase the performance of your applications.

Using the `XPLINK` option increases the performance of C/C++ routines by reducing linkage overhead and by passing function call parameters in registers. It supports both reentrant and non-reentrant code, as well as calls to functions exported from DLLs.

The extra performance linkage resulting from `XPLINK` is a common linkage convention for C and C++. Therefore, it is possible for a C function pointer to reference a non-″extern C″ C++ function. It is also possible for a non-″extern C″ C++ function to reference a C function pointer. With this linkage, casting integers to function pointers is the same as on other platforms such as AIX, making it easier to port applications to z/OS using the C/C++ compiler.

You can not bind XPLINK object decks together with non-XPLINK object decks, with the exception of object decks using OS_UPSTACK or OS_NOSTACK". XPLINK parts of an application can work with non-XPLINK parts across DLL and fetch() boundaries.

When compiling using the XPLINK option, the compiler uses the following options as defaults:
- CSECT()
- GOFF
- LONGNAME
- RENT

You may override these options. However, the XPLINK option requires the GOFF option. If you specify the NOGOFF option, the compiler issues an informational message and promotes the option to GOFF.

In addition, the XPLINK option requires that the value of ARCH must be 2 or greater. The compiler issues an error message if you specify ARCH(0) or ARCH(1) with XPLINK.

The XPLINK option accepts the following suboptions:

BACKCHAIN | NOBACKCHAIN

DEFAULT: NOBACKCHAIN

If you specify BACKCHAIN, the compiler generates a prolog that saves information about the calling function in the called function's stack frame. This facilitates debugging using storage dumps. Use this suboption in conjunction with STOREARGS to make storage dumps more useful.

GUARD | NOGUARD

DEFAULT: GUARD

If you specify NOGUARD, the compiler generates an explicit check for stack overflow, which enables the storage runtime option. Using this suboption causes a performance degradation at runtime, even if you do not use the Language Environment runtime STORAGE option.

OSCALL(UPSTACK| DOWNSTACK| NOSTACK)

DEFAULT: NOSTACK

This suboption directs the compiler to use the linkage (UPSTACK, DOWNSTACK, or NOSTACK) as specified in this suboption for any #pragma linkage(identifier, OS) calls in your application.

This value causes the compiler to use the following linkage wherever linkage OS is specified by `#pragma linkage` in C, or `extern "linkage"` in C++:

Linkage Used	Linkage Specified
UPSTACK	OS_UPSTACK
NOSTACK	OS_NOSTACK or OS31_NOSTACK (equivalent)
DOWNSTACK	OS_DOWNSTACK

For example, since the default of this option is NOSTACK, any `#pragma linkage(identifier,OS)` in C code, works just as if `#pragma linkage(identifier,OS31_NOSTACK)` had been specified.

The abbreviated form of this suboption is OSCALL(U | D | N).

This suboption only applies to routines that are references but not defined in the compilation unit.

`STOREARGS`| `NOSTOREARGS`

DEFAULT: `NOSTOREARGS`

If you specify the STOREARGS suboption, the compiler generates code to store arguments that are normally only passed in registers, into the caller's argument area. This facilitates debugging using storage dumps. Use this suboption in conjunction with the BACKCHAIN suboption to make storage dumps more useful.

Note that the values in the argument area may be modified by the called function.

The abbreviated form of this suboption is STOR.

Effect on IPA Compile Step

The IPA Compile step generates information for the IPA Link step. The `XPLINK` option also affects the regular object module if you request one by specifying the IPA(OBJECT) option. This option affects the IPA optimized object module that is generated by specifying the IPA(OBJONLY) option.

Effect on IPA Link Step

The IPA Link step does not support IPA object files compiled with the `XPLINK` option.

XREF | NOXREF

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓	✓	✓	✓	✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
NOXREF	NOXREF	NOXREF	NOXREF	NOXREF	NOXREF	NOXREF

CATEGORY: Listing

```
►►──┬─XR───┬──────────────────────────────────────────────────────►◄
    └─NOXR─┘
```

The `XREF` option generates a cross-reference listing that shows file definition, line definition, reference, and modification information for each symbol. It also generates the External Symbol Cross Reference and Static Map.

For C, a separate offset listing of the variables will appear after the cross reference table.

You can specify this option using the `#pragma option` directive for C.

In the z/OS UNIX System Services environment, this option is turned on by specifying `-V` when using the `c89`, `cc`, or `c++` commands.

Effect on IPA Compile Step

During the IPA Compile step, the compiler saves symbol storage offset information in the IPA object file as follows:

- For C, if you specify the `XREF`, `IPA(ATTRIBUTE)`, or `IPA(XREF)` options or the `#pragma option (XREF)`
- For C++, if you specify the `ATTR`, `XREF`, `IPA(ATTRIBUTE)`, or `IPA(XREF)` options

If regular object code/data is produced using the `IPA(OBJECT)` option, the cross reference sections of the compile listing will be controlled by the `ATTR` and `XREF` options.

Effect on IPA Link Step

If you specify the `ATTR` or `XREF` options for the IPA Link step, it generates External Symbol Cross Reference and Static Map listing sections for each partition.

The IPA Link step creates a Storage Offset listing section if during the IPA Compile step you requested the additional symbol storage offset information for your IPA objects.

Description of Compatible Compiler Options

The following section describes compiler options which are compatible with previous versions of the compiler. Use these options only if they are already coded in such things as your JCL, make files, scripts, and so on. For new programs, you should use the replacement option that is listed in Table 5 on page 69. Compiler options are listed alphabetically. The syntax diagrams show the abbreviated forms of the compiler options.

Note: Some parameters such as the output data set may differ between the option that is described and its replacement option. Read the description of the replacement option before you use it.

DECK | NODECK

Option Scope				
C Compile	**C++ Compile**	**IPA Link**	**Special IPA Processing**	
			IPA Compile	**IPA Link**
✓		✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	**Set by z/OS UNIX System Services Utilities**					
	Regular Compile			**IPA Link**		
	c89	**cc**	**c++**	**c89**	**cc**	**c++**
NODECK	NODECK	NODECK				

CATEGORY: File Management

```
►►──┬─DECK───┬──────────────────────────────────────────────────►◄
    └─NODECK─┘
```

The DECK option specifies whether the compiler is to produce an object module and store it in the data set defined by the SYSPUNCH DD statement. For new z/OS C/C++ programs, use the OBJECT option. Table 21 details the relationship between the DECK and OBJECT options. For a description of OBJECT see "OBJECT | NOOBJECT" on page 148.

Table 21. Relationship between DECK and OBJECT

DECK | NODECK Option	OBJECT | NOOBJECT Option	Result
NODECK	OBJECT	Object module is generated and stored in data set defined by SYSLIN DD
DECK	OBJECT	Object module is generated. It is stored in data set defined by SYSLIN DD. A warning will be issued.
NODECK	NOOBJECT	No object module is generated
DECK	NOOBJECT	Object module is generated and stored in data set defined by SYSPUNCH DD
Note: The defaults are OBJECT and NODECK		

You can specify this option using the #pragma option directive for C.

Effect on IPA Compile Step

The DECK option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

The DECK option has the same effect on the IPA Compile step as it does on a regular compilation. The DECK option only affects the step on which it is specified, so if you specify it for the IPA Compile step, it has no effect on the IPA Link step.

HWOPTS | NOHWOPTS

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
NOHWOPTS						

CATEGORY: Preprocessor

```
►►──┬─HWO──┬─(STR)───┬─┬──────────────────────────────────────────►◄
    │      └─(NOSTR)─┘ │
    └─NOHWO────────────┘
```

Note: The `ARCH` option has replaced the `HWOPTS` option. `HWOPTS(STRING)` maps to `ARCH(1)`, and `HWOPTS(NOSTRING)` maps to `ARCH(0)`. If you specify both `HWOPTS` and `ARCH`, `ARCH` takes effect. Use the `ARCH` option instead of `HWOPTS` when compiling new z/OS C programs.

See "ARCHITECTURE" on page 81.

The `HWOPTS` option specifies whether the compiler is to generate code to take advantage of different hardware. Suboptions are:

`STRING`	Creates code for hardware that has Logical String Assist (LSA). On such hardware, built-in functions will have better performance if you select this option.
`NOSTRING`	Creates code for hardware that does not have LSA.

You can specify this option using the `#pragma option` directive for C.

Effect on IPA Compile Step

If you specify the `HWOPTS` option for any compilation unit in the IPA Compile step, the compiler generates information for the IPA Link step. This option also affects the regular object module if you request one by specifying the `IPA(OBJECT)` option.

Effect on IPA Link Step

The `HWOPTS` option has the same effect on the IPA Link step as the `ARCH(1)` option. Refer to "ARCHITECTURE" on page 81 for more information.

OMVS | NOOMVS

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		✓

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOOMVS`						

CATEGORY: Source Code Control

```
►►──┬─OMVS──┬──────────────┬─┬────────────────────────────────────►◄
    │       └─(─filename─)─┘ │
    └─NOOMVS─────────────────┘
```

In the z/OS C environment, `OMVS` is a synonym for the `OE` option. Use the `OE` option, because it provides greater flexibility and you can use it for both z/OS C and z/OS C++.

You can specify *filename* which is the name of a partitioned or sequential data set that contains user include files. For more information on `OE`, refer to “OE | NOOE” on page 150.

Effect on IPA Compile Step

The `OMVS` compiler option has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

On the IPA Link step, the `OMVS` option controls the display of file names.

SYSLIB

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOSYSLIB`	`NOSYSLIB`	`NOSYSLIB`	`NOSYSLIB`			

CATEGORY: File Management

```
►►──┬─SYSL──(──pdsnames-list──)─┬──────────────────────────────────►◄
    └─NOSYSL────────────────────┘
```

Note: When compiling new z/OS C/C++ applications, use `SEARCH` instead of `SYSLIB`.

The `SYSLIB` option specifies a list of PDSs that contain system header files. The PDSs in the list are dynamically allocated to the SYSLIB DD name. If you already have a SYSLIB ddname specified, the compiler uses that ddname instead of the list that you specified, and issues a warning message.

If you want to override the default SYSLIB that the `CC` exec allocates, you must allocate the ddname SYSLIB **before** you invoke `CC`. If the ddname SYSLIB is not already allocated before you invoke the `CC` exec, `CC` will allocate the default SYSLIB. If you invoke `CC` with the `SYSLIB` compiler option, the compiler ignores the option specification, and `CC` will allocate the default SYSLIB `CEE.SCEEH.H` and `CEE.SCEEH.SYS.H`.

Effect on IPA Compile Step

The `SYSLIB` option is used for source code analysis, and has the same effect on the IPA Compile step as it does on a regular compilation.

Effect on IPA Link Step

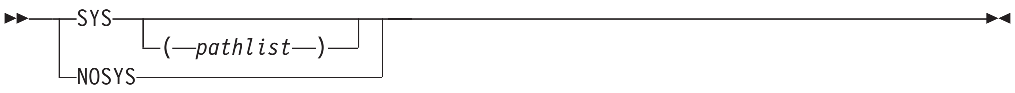
The IPA Link step accepts the `SYSLIB` option, but ignores it.

SYSPATH | NOSYSPATH

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOSYSPATH`			`NOSYSPATH`			

CATEGORY: File Management



Note: When compiling new z/OS C++ applications or for HFS searching, use `SEARCH` instead of `SYSPATH`. If you use both `SEARCH` and `SYSPATH`, the compiler uses the option that you specified last, and ignores the other.

The `SYSPATH` option specifies pathnames. The compiler uses these pathnames to construct names of PDSs that it will search for system header files. In addition, z/OS C++ supports `SYSLIB` ddnames so that header files can be searched in exactly the same way they are in z/OS C. The compiler uses `SYSLIB` to search for include file **after** it performs a search using the z/OS C++ `SYSPATH` directive.

You must specify each path in the `SYSPATH` compiler option as a sequence of directory names that are separated by a slash (/). The directory name must be one of the following:

- A valid PDS qualifier that does not contain a dot (.)
- The current directory (.)
- The parent directory (..)

The following are all valid path names:
- /cbc/cxxproto
- /Usr/Include
- /tcpip/include
- /dept120/../usr/include/.
- local /include

Note: `/dept120/../usr/include/.` resolves to the same path as `/usr/include`

If an include file begins with a slash (/), it is considered **absolute**; otherwise, it is considered **relative**. A relative file uses the `SYSPATH` information, in addition to itself, to build a PDS member name, whereas an absolute file does not require the SYSPATH information. An absolute file is considered explicit, and the compiler does not perform a search.

A SYSPATH of /CBC/SCBCH tells the compiler to search for the following:
- *.h files in 'CBC.SCBCH.H'
- *.c files in 'CBC.SCBCH.C'
- *.inl files in 'CBC.SCBCH.INL'

For additional information see "Using Include Files" on page 308.

To reset the current syspath information, you must specify NOSYSPATH followed by SYSPATH. You must specify NOSYSPATH to reset your installation-defined SYSPATH.

Effect on IPA Compile Step

The SYSPATH option has the same effect on the IPA Compile step as it has on a regular compilation.

Effect on IPA Link Step

The IPA Link step issues a diagnostic message if you specify the SYSPATH option for that step.

USERLIB

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
no action	NOUSERLIB	NOUSERLIB	NOUSERLIB			

CATEGORY: File Management

```
►►──┬─USERL──(──pdsnames-list──)─┬──────────────────────────────────►◄
    └─NOUSERL────────────────────┘
```

Note: When compiling new z/OS C/C++ applications, use the LSEARCH option instead of USERLIB. If you use both LSEARCH and USERLIB, the compiler uses the option that you specified last, and ignores the other.

The USERLIB option specifies a list of PDSs that contain user header files. The PDSs in the list are dynamically allocated to the USERLIB ddname. If you already have a USERLIB ddname specified, the compiler uses that ddname instead of the list that you specified, and issues a warning.

Effect on IPA Compile Step

The USERLIB option is used for source code analysis, and has the same effect on the IPA Compile step as it has on a regular compilation.

Effect on IPA Link Step

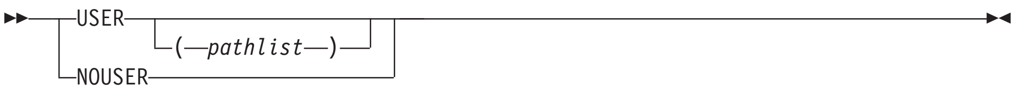
The IPA Link step accepts the `USERLIB` option, but ignores it.

USERPATH | NOUSERPATH

Option Scope				
C Compile	C++ Compile	IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

Option Default						
z/OS C/C++ Compiler (Batch and TSO Environments)	Set by z/OS UNIX System Services Utilities					
	Regular Compile			IPA Link		
	c89	cc	c++	c89	cc	c++
`NOUSERPATH`			`NOUSERPATH`			

CATEGORY: File Management



Note: When compiling new z/OS C++ applications or for HFS searching, use `LSEARCH` instead of `USERPATH`. If you use both `LSEARCH` and `USERPATH`, the compiler uses the option that you specified last, and ignores the other.

The `USERPATH` option specifies paths to search for user-defined header files. The compiler uses these pathnames to construct names of PDSs that it searches for your header files. The `USERPATH` option only applies to searches for PDSs.

You must specify each file in the `USERPATH` compiler option as a sequence of directory names that are separated by a slash (/). The directory name must be one of the following:
- A valid PDS qualifier that does not contain a dot (.)
- The current directory (.)
- The parent directory (..)

The following are all valid path names:
- /Usr/Include
- /tcpip/include
- /dept120/../usr/include/.
- local /include

Note: `/dept120/../usr/include/.` resolves to the same path as `/usr/include`

If an include file begins with a slash (/), it is **absolute**; otherwise, it is **relative**. A relative file uses the `USERPATH` information, in addition to itself, to build a PDS member name, whereas an absolute path does not require the `USERPATH` information. If z/OS format is used, the file is explicit and no additional searching is performed.

For example, a `USERPATH` of `/USER/HEADERS` instructs the compiler to search for the following:

`*.h` files in `'USER.HEADERS.H'`
`*.c` files in `'USER.HEADERS.C'`
`*.inl` files in `'USER.HEADERS.INL'`

You can also use the `LSEARCH` option to control the search for include files. For additional information, see "Using Include Files" on page 308.

Effect on IPA Compile Step

The `USERPATH` option has the same effect on the IPA Compile step as it has on a regular compilation.

Effect on IPA Link Step

The IPA Link step issues a diagnostic message if you specify the `USERPATH` option for that step.

Using the z/OS C Compiler Listing

If you select the `SOURCE` or `LIST` option, the compiler creates a listing that contains information about the source program and the compilation. If the compilation terminates before reaching a particular stage of processing, the compiler does not generate corresponding parts of the listing. The listing contains standard information that always appears, together with optional information that is supplied by default or specified through compiler options.

In an interactive environment you can also use the `TERMINAL` option to direct all compiler diagnostic messages to your terminal. The `TERMINAL` option directs only the diagnostic messages part of the compiler listing to your terminal.

Note: Although the compiler listing is for your use, it is not a programming interface and is subject to change.

IPA Considerations

The listings that the IPA Compile step produces are basically the same as those that a regular compilation produces. Any differences are noted throughout this section.

The IPA Link step listing has a separate format from the listings mentioned above. Many listing sections are similar to those that are produced by a regular compilation or the IPA Compile step with the `IPA(OBJECT)` option specified. Refer to "Using the IPA Link Step Listing" on page 256 for information about IPA Link step listings.

Example of a C Compiler Listing

Figure 17 on page 207 shows an example of a C compiler listing.

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                    04/14/2000 14:42:32   Page     1

                                      * * * * *   P R O L O G   * * * * *

 Compile Time Library . . . . . . : 220A0000
 Command options:
    Program name. . . . . . . . . : 'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'
    Compiler options. . . . . . . : *NOGONUMBER *NOALIAS    *NODECK     *NORENT     *TERMINAL   *NOUPCONV   *SOURCE     *LIST
                                  : *XREF       *AGGR       *NOPPONLY   *NOEXPMAC   *NOSHOWINC  *NOOFFSET   *MEMORY     *NOSSCOMM
                                  : *NOLONGNAME *START      *EXECOPS    *ARGPARSE   *NOEXPORTAL *NODLL(NOCALLBACKANY)
                                  : *NOLIBANSI  *NOWSIZEOF  *REDIR      *ANSIALIAS  *NODIGRAPH  *NOROCONST  *NOROSTRING
                                  : *TUNE(3)    *ARCH(2)    *SPILL(128) *MAXMEM(2097152)        *NOCOMPACT
                                  : *TARGET(LE,CURRENT)     *FLAG(I)    *NOTEST(SYM,BLOCK,LINE,PATH,HOOK)   *NOOPTIMIZE
                                  : *INLINE(AUTO,REPORT,100,1000)       *NESTINC(255)
                                  : *NOCHECKOUT(NOPPTRACE,PPCHECK,GOTO,ACCURACY,PARM,NOENUM,
                                  :             NOEXTERN,TRUNC,INIT,NOPORT,GENERAL,CAST)
                                  : *FLOAT(HEX,FOLD,NOAFP)  *STRICT     *NOIGNERRNO *NOINITAUTO
                                  : *NOCOMPRESS *NOSTRICT_INDUCTION     *AGGRCOPY(NOOVERLAP)
                                  : *NOCSECT
                                  : *NOEVENTS
                                  : *OBJECT
                                  : *NOGENPCH
                                  : *NOUSEPCH
                                  : *NOOPTFILE
                                  : *NOSERVICE
                                  : *NOOE
                                  : *NOIPA
                                  : *SEARCH(//'CEE.SCEEH.+')
                                  : *NOLSEARCH
                                  : *NOLOCALE   *HALT(16)   *PLIST(HOST)
                                  : *NOCONVLIT
                                  : *NOGOFF
                                  : *NOXPLINK(NOBACKCHAIN,NOSTOREARGS,GUARD,OSCALL(NOSTACK))
    Version Macros. . . . . . . . : __COMPILER_VER__=0x220A0000 __LIBREL__=0x220A0000 __TARGET_LIB__=0x220A0000
    Language level. . . . . . . . : *EXTENDED
    Source margins. . . . . . . . :
      Varying length. . . . . . . : 1 - 32767
      Fixed length. . . . . . . . : 1 - 72
    Sequence columns. . . . . . . :
      Varying length. . . . . . . : none
      Fixed length. . . . . . . . : 73 - 80

                               * * * * *   E N D   O F   P R O L O G   * * * * *
```

Figure 17. Example of a C listing (Part 1 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page     2

                                          * * * * *   S O U R C E   * * * * *

 LINE  STMT                                                                                                          SEQNBR INCNO
            *...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+..*
    1       |#include <stdio.h>                                                                                 |      1
    2       |                                                                                                   |      2
    3       |#include "cbc3uaan.h"                                                                              |      3
    4       |                                                                                                   |      4
    5       |void convert(double);                                                                              |      5
    6       |                                                                                                   |      6
    7       |int main(int argc, char **argv)                                                                    |      7
    8       |{                                                                                                  |      8
    9       |    double c_temp;                                                                                 |      9
   10       |                                                                                                   |     10
   11     1 |    if (argc == 1) {  /* get Celsius value from stdin */                                           |     11
   12       |       int ch;                                                                                     |     12
   13       |                                                                                                   |     13
   14     2 |       printf("Enter Celsius temperature: \n");                                                    |     14
   15       |                                                                                                   |     15
   16     3 |       if (scanf("%f", :c_temp) != 1) {                                                            |     16
   17     4 |          printf("You must enter a valid temperature\n");                                          |     17
   18       |       }                                                                                           |     18
   19       |       else {                                                                                      |     19
   20     5 |          convert(c_temp);                                                                         |     20
   21       |       }                                                                                           |     21
   22       |    }                                                                                              |     22
   23       |    else {  /* convert the command-line arguments to Fahrenheit */                                 |     23
   24       |       int i;                                                                                      |     24
   25       |                                                                                                   |     25
   26     6 |       for (i = 1; i < argc; ++i) {                                                                |     26
   27     7 |           if (sscanf(argv[i], "%f", :c_temp) != 1)                                                |     27
   28     8 |              printf("%s is not a valid temperature\n",argv[i]);                                   |     28
   29       |           else                                                                                    |     29
   30     9 |              convert(c_temp);                                                                     |     30
   31       |       }                                                                                           |     31
   32       |    }                                                                                              |     32
   33       |}                                                                                                  |     33
   34       |                                                                                                   |     34
   35       |void convert(double c_temp) {                                                                      |     35
   36    10 |   double f_temp = (c_temp * CONV + OFFSET);                                                       |     36
   37    11 |   printf("%5.2f Celsius is %5.2f Fahrenheit\n",c_temp, f_temp);                                   |     37
   38       |}                                                                                                  |     38
                                    * * * * *   E N D   O F   S O U R C E   * * * * *
```

Figure 17. Example of a C listing (Part 2 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'              04/14/2000 14:42:32   Page     3

                        * * * * *   I N C L U D E S   * * * * *

INCLUDE FILES  ---  FILE#   NAME

                       1   TSCTEST.CEE210.SCEEH.H(STDIO)
                       2   TSCTEST.CEE210.SCEEH.H(FEATURES)
                       3   TSCTEST.CEE210.SCEEH.SYS.H(TYPES)
                       4   TSCTEST.OSV2R10.SCBCSAM(CBC3UAAN)

                        * * * * *   E N D   O F   I N C L U D E S   * * * * *
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'              04/14/2000 14:42:32   Page     4

                                * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>

___valist           1-1:121         Class = typedef, Length = 8
                                    Type = array[2] of pointer to unsigned char
                                    1-1:124, 1-1:308, 1-1:309, 1-1:310

__abend             1-1:580         Type = struct with no tag in union at offset 0

__alloc             1-1:590         Type = struct with no tag in union at offset 0

__amrc_ptr          1-1:618         Class = typedef, Length = 4
                                    Type = pointer to struct __amrctype

__amrc_type         1-1:614         Class = typedef, Length = 224
                                    Type = struct __amrctype
                                    1-1:618

__amrctype          1-1:562         Class = struct tag

__amrc2_ptr         1-1:631         Class = typedef, Length = 4
                                    Type = pointer to struct __amrc2type

__amrc2_type        1-1:627         Class = typedef, Length = 32
                                    Type = struct __amrc2type
                                    1-1:631

__amrc2type         1-1:623         Class = struct tag

__blksize           1-1:479         Type = unsigned long in struct __fileData at offset 8

__bufPtr            1-1:59          Type = pointer to unsigned char in struct __file at offset 0

__cntlinterpret     1-1:64          Type = unsigned int:1 in struct __file at offset 20(0)

__code              1-1:591         Type = union with no tag in struct __amrctype at offset 0

__countIn           1-1:60          Type = long in struct __file at offset 4

__countOut          1-1:61          Type = long in struct __file at offset 8

__cusp              1-1:172         Class = typedef, Length = 4
                                    Type = pointer to const unsigned short

__device            1-1:478         Type = enum with no tag in struct __fileData at offset 4

__device_t          1-1:424         Class = typedef, Length = 1
                                    Type = enum with no tag
                                    1-1:478

__disk              1-1:409         Class = enumeration constant: 265389688, Length = 4
                                    Type = int

__dsname            1-1:484         Type = pointer to unsigned char in struct __fileData at offset 28
```

Figure 17. Example of a C listing (Part 3 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page     5

                                * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>
__dsorgConcat       1-1:455         Type = unsigned int:1 in struct __fileData at offset 1(3)

__dsorgHiper        1-1:457         Type = unsigned int:1 in struct __fileData at offset 1(5)

__dsorgHFS          1-1:462         Type = unsigned int:1 in struct __fileData at offset 2(0)

__dsorgMem          1-1:456         Type = unsigned int:1 in struct __fileData at offset 1(4)

__dsorgPDSdir       1-1:453         Type = unsigned int:1 in struct __fileData at offset 1(1)

__dsorgPDSmem       1-1:452         Type = unsigned int:1 in struct __fileData at offset 1(0)

__dsorgPDSE         1-1:469         Type = unsigned int:1 in struct __fileData at offset 2(7)

__dsorgPO           1-1:451         Type = unsigned int:1 in struct __fileData at offset 0(7)

__dsorgPS           1-1:454         Type = unsigned int:1 in struct __fileData at offset 1(2)

__dsorgTemp         1-1:458         Type = unsigned int:1 in struct __fileData at offset 1(6)

__dsorgVSAM         1-1:459         Type = unsigned int:1 in struct __fileData at offset 1(7)

__dummy             1-1:414         Class = enumeration constant: 26035193464, Length = 4
                                    Type = int

__error             1-1:574         Type = long in union at offset 0

__error2            1-1:624         Type = long in struct __amrc2type at offset 0

__fcbgetc           1-1:62          Type = pointer to function returning int in struct __file at offset 12

__fcbputc           1-1:63          Type = pointer to function returning int in struct __file at offset 16

__fdbk              1-1:585         Type = unsigned char in struct at offset 3

__fdbk_fill         1-1:582         Type = unsigned char in struct at offset 0

__feedback          1-1:586         Type = struct with no tag in union at offset 0

__ffile             1-1:67          Class = struct tag
                                    1-1:72, 1-1:78

__file              1-1:54          Class = struct tag
                                    1-1:55, 1-1:56, 1-1:69

__fileptr           1-1:625         Type = pointer to struct __ffile in struct __amrc2type at offset 4

__fileData          1-1:443         Class = struct tag
                                    1-1:488

__fill              1-1:553         Type = unsigned long in struct at offset 0

__filler1           1-1:396         Type = unsigned char in struct __S99emparms at offset 3
```

Figure 17. Example of a C listing (Part 4 of 35)

```
5647A01 V2 R10 OS/390 C              'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'              04/14/2000 14:42:32   Page     6

                                * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>

__fill2             1-1:605         Type = array[2] of unsigned long in struct at offset 136

__fp                1-1:69          Type = pointer to struct __file in struct __ffile at offset 0

__fpos_elem         1-1:83          Type = array[8] of long in struct __fpos_t at offset 0

__fpos_t            1-1:82          Class = struct tag
                                    1-1:86

__ftncd             1-1:584         Type = unsigned char in struct at offset 2

__hfs               1-1:421         Class = enumeration constant: 38920095352, Length = 4
                                    Type = int

__hiperspace        1-1:422         Class = enumeration constant: 43215062648, Length = 4
                                    Type = int

__last_op           1-1:598         Type = unsigned int in struct __amrctype at offset 8

__len               1-1:601         Type = unsigned long in struct at offset 4

__len_fill          1-1:600         Type = unsigned long in struct at offset 0

__maxreclen         1-1:480         Type = unsigned long in struct __fileData at offset 12

__memory            1-1:420         Class = enumeration constant: 34625128056, Length = 4
                                    Type = int

__modeflag          1-1:468         Type = unsigned int:4 in struct __fileData at offset 2(3)

__msg               1-1:608         Type = struct with no tag in struct __amrctype at offset 12

__msgfile           1-1:417         Class = enumeration constant: 30330160760, Length = 4
                                    Type = int

__openmode          1-1:467         Type = unsigned int:2 in struct __fileData at offset 2(1)

__other             1-1:423         Class = enumeration constant: 1095482050168, Length = 4
                                    Type = int

__parmr0            1-1:603         Type = unsigned long in struct at offset 128

__parmr1            1-1:604         Type = unsigned long in struct at offset 132

__printer           1-1:411         Class = enumeration constant: 8855324280, Length = 4
                                    Type = int

__rc                1-1:579         Type = unsigned short in struct at offset 2

__rc                1-1:583         Type = unsigned char in struct at offset 1

__recfmASA          1-1:449         Type = unsigned int:1 in struct __fileData at offset 0(5)
```

Figure 17. Example of a C listing (Part 5 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page     7

                              * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>

__recfmBlk          1-1:448         Type = unsigned int:1 in struct __fileData at offset 0(4)

__recfmF            1-1:444         Type = unsigned int:1 in struct __fileData at offset 0(0)

__recfmM            1-1:450         Type = unsigned int:1 in struct __fileData at offset 0(6)

__recfmS            1-1:447         Type = unsigned int:1 in struct __fileData at offset 0(3)

__recfmU            1-1:446         Type = unsigned int:1 in struct __fileData at offset 0(2)

__recfmV            1-1:445         Type = unsigned int:1 in struct __fileData at offset 0(1)

__recnum            1-1:554         Type = unsigned long in struct at offset 4

__reserved          1-1:379         Type = unsigned char in struct __S99rbx at offset 16

__reserved          1-1:626         Type = array[6] of long in struct __amrc2type at offset 8

__reserve2          1-1:473         Type = unsigned int:5 in struct __fileData at offset 3(3)

__reserve4          1-1:485         Type = unsigned int in struct __fileData at offset 32

__reserv1           1-1:401         Type = int in struct __S99emparms at offset 20

__reserv2           1-1:387         Type = int in struct __S99rbx at offset 32

__reserv2           1-1:402         Type = int in struct __S99emparms at offset 24

__rplfdbwd          1-1:611         Type = array[4] of unsigned char in struct __amrctype at offset 220

__rrds_key_type     1-1:555         Class = typedef, Length = 8
                                    Type = struct with no tag

__str               1-1:602         Type = array[120] of unsigned char in struct at offset 8

__str2              1-1:606         Type = array[64] of unsigned char in struct at offset 144

__svc99_error       1-1:589         Type = unsigned short in struct at offset 2

__svc99_info        1-1:588         Type = unsigned short in struct at offset 0

__syscode           1-1:578         Type = unsigned short in struct at offset 0

__tape              1-1:412         Class = enumeration constant: 13150291576, Length = 4
                                    Type = int

__tdq               1-1:413         Class = enumeration constant: 21740226168, Length = 4
                                    Type = int

__terminal          1-1:410         Class = enumeration constant: 4560356984, Length = 4
                                    Type = int
```

Figure 17. Example of a C listing (Part 6 of 35)

```
5647A01 V2 R10 OS/390 C              'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'              04/14/2000 14:42:32   Page     8

                                * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>
__vsamkeylen        1-1:482         Type = unsigned long in struct __fileData at offset 20

__vsamtype          1-1:481         Type = unsigned short in struct __fileData at offset 16

__vsamRKP           1-1:483         Type = unsigned long in struct __fileData at offset 24

__vsamRLS           1-1:472         Type = unsigned int:3 in struct __fileData at offset 3(0)

__EMBUFP            1-1:400         Type = pointer to void in struct __S99emparms at offset 16

__EMCPPLP           1-1:399         Type = pointer to void in struct __S99emparms at offset 12

__EMFUNCT           1-1:393         Type = unsigned char in struct __S99emparms at offset 0

__EMIDNUM           1-1:394         Type = unsigned char in struct __S99emparms at offset 1

__EMNMSGBK          1-1:395         Type = unsigned char in struct __S99emparms at offset 2

__EMRETCOD          1-1:398         Type = int in struct __S99emparms at offset 8

__EMS99RBP          1-1:397         Type = pointer to void in struct __S99emparms at offset 4

__FILEP             1-1:72          Class = typedef, Length = 4
                                    Type = pointer to struct __ffile

__RBA               1-1:592         Type = unsigned long in struct __amrctype at offset 4

__S99emparms        1-1:392         Class = struct tag
                                    1-1:405

__S99emparms_t      1-1:405         Class = typedef, Length = 28
                                    Type = struct __S99emparms

__S99parms          1-1:365         Class = typedef, Length = 20
                                    Type = struct __S99struc
                                    1-1:528

__S99rbx            1-1:369         Class = struct tag
                                    1-1:390

__S99rbx_t          1-1:390         Class = typedef, Length = 36
                                    Type = struct __S99rbx

__S99struc          1-1:348         Class = struct tag
                                    1-1:365

__S99ECPPL          1-1:378         Type = pointer to void in struct __S99rbx at offset 12

__S99EERR           1-1:385         Type = unsigned short in struct __S99rbx at offset 28

__S99EID            1-1:371         Type = array[6] of unsigned char in struct __S99rbx at offset 0

__S99EINFO          1-1:386         Type = unsigned short in struct __S99rbx at offset 30
```

Figure 17. Example of a C listing (Part 7 of 35)

```
5647A01 V2 R10 OS/390 C                 'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page     9

                                * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>

__S99EKEY           1-1:375         Type = unsigned char in struct __S99rbx at offset 9

__S99EMGSV          1-1:376         Type = unsigned char in struct __S99rbx at offset 10

__S99EMSGP          1-1:384         Type = pointer to void in struct __S99rbx at offset 24

__S99ENMSG          1-1:377         Type = unsigned char in struct __S99rbx at offset 11

__S99EOPTS          1-1:373         Type = unsigned char in struct __S99rbx at offset 7

__S99ERCF           1-1:382         Type = unsigned char in struct __S99rbx at offset 19

__S99ERCO           1-1:381         Type = unsigned char in struct __S99rbx at offset 18

__S99ERES           1-1:380         Type = unsigned char in struct __S99rbx at offset 17

__S99ERROR          1-1:354         Type = unsigned short in struct __S99struc at offset 4

__S99ESUBP          1-1:374         Type = unsigned char in struct __S99rbx at offset 8

__S99EVER           1-1:372         Type = unsigned char in struct __S99rbx at offset 6

__S99EWRC           1-1:383         Type = int in struct __S99rbx at offset 20

__S99FLAG1          1-1:352         Type = unsigned short in struct __S99struc at offset 2

__S99FLAG2          1-1:360         Type = unsigned int in struct __S99struc at offset 16

__S99INFO           1-1:355         Type = unsigned short in struct __S99struc at offset 6

__S99RBLN           1-1:350         Type = unsigned char in struct __S99struc at offset 0

__S99S99X           1-1:358         Type = pointer to void in struct __S99struc at offset 12

__S99TXTPP          1-1:356         Type = pointer to void in struct __S99struc at offset 8

__S99VERB           1-1:351         Type = unsigned char in struct __S99struc at offset 1

_gtca                               Class = extern
                                    Type = function returning pointer to const void
                                    1-1:145

_Gtab                               Class = extern
                                    Type = function returning pointer to pointer to void
                                    1-1:135

_GETCFUNC           1-1:55          Class = typedef
                                    Type = function returning int
                                    1-1:62

_PUTCFUNC           1-1:56          Class = typedef
                                    Type = function returning int
```

Figure 17. Example of a C listing (Part 8 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                 04/14/2000 14:42:32   Page    10

                                * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>
                                    1-1:63

argc                7-0:7           Class = parameter, Length = 4
                                    Type = int in function main
                                    11-0:11, 26-0:26, 26-0:26

argv                7-0:7           Class = parameter, Length = 4
                                    Type = pointer to pointer to unsigned char in function main
                                    27-0:27, 28-0:28

c_temp              35-0:35         Class = parameter, Length = 8
                                    Type = double in function convert
                                    36-0:36, 37-0:37

c_temp              9-0:9           Class = auto, Length = 8
                                    Type = double in function main
                                    16-0:16, 20-0:20, 27-0:27, 30-0:30

ch                  12-0:12         Class = auto, Length = 4
                                    Type = int in function main

clearerr                            Class = extern
                                    Type = function returning void
                                    1-1:262

clrmemf                             Class = extern
                                    Type = function returning int
                                    1-1:532

convert             35-0:35         Class = extern
                                    Type = function returning void
                                    5-0:5, 20-0:20, 30-0:30

f_temp              36-0:36         Class = auto, Length = 8
                                    Type = double in function convert
                                    36-0:36, 37-0:37

fclose                              Class = extern
                                    Type = function returning int
                                    1-1:263

fdelrec                             Class = extern
                                    Type = function returning int
                                    1-1:530

feof                                Class = extern
                                    Type = function returning int
                                    1-1:264

ferror                              Class = extern
                                    Type = function returning int
                                    1-1:265
```

Figure 17. Example of a C listing (Part 9 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'               04/14/2000 14:42:32   Page    11

                                ***** C R O S S   R E F E R E N C E   L I S T I N G *****

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>
fflush                              Class = extern
                                    Type = function returning int
                                    1-1:266

fgetc                               Class = extern
                                    Type = function returning int
                                    1-1:267

fgetpos                             Class = extern
                                    Type = function returning int
                                    1-1:268

fgets                               Class = extern
                                    Type = function returning pointer to unsigned char
                                    1-1:269

fldata                              Class = extern
                                    Type = function returning int
                                    1-1:533

fldata_t            1-1:488         Class = typedef, Length = 36
                                    Type = struct __fileData
                                    1-1:533

flocate                             Class = extern
                                    Type = function returning int
                                    1-1:529

fopen                               Class = extern
                                    Type = function returning pointer to struct __ffile
                                    1-1:270

fpos_t              1-1:86          Class = typedef, Length = 32
                                    Type = struct __fpos_t
                                    1-1:268, 1-1:281

fprintf                             Class = extern
                                    Type = function returning int
                                    1-1:272

fputc                               Class = extern
                                    Type = function returning int
                                    1-1:273

fputs                               Class = extern
                                    Type = function returning int
                                    1-1:274

fread                               Class = extern
                                    Type = function returning unsigned int
                                    1-1:275

freopen                             Class = extern
```

Figure 17. Example of a C listing (Part 10 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page    12

                                * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>
                                    Type = function returning pointer to struct __ffile
                                    1-1:277

fscanf                              Class = extern
                                    Type = function returning int
                                    1-1:279

fseek                               Class = extern
                                    Type = function returning int
                                    1-1:280

fsetpos                             Class = extern
                                    Type = function returning int
                                    1-1:281

ftell                               Class = extern
                                    Type = function returning long
                                    1-1:282

fupdate                             Class = extern
                                    Type = function returning unsigned int
                                    1-1:531

fwrite                              Class = extern
                                    Type = function returning unsigned int
                                    1-1:283

getc                                Class = extern
                                    Type = function returning int
                                    1-1:285

getchar                             Class = extern
                                    Type = function returning int
                                    1-1:286

gets                                Class = extern
                                    Type = function returning pointer to unsigned char
                                    1-1:287

i                   24-0:24         Class = auto, Length = 4
                                    Type = int in function main
                                    26-0:26, 26-0:26, 27-0:27, 28-0:28, 26-0:26, 26-0:26, 26-0:26

main                7-0:7           Class = extern
                                    Type = function returning int

perror                              Class = extern
                                    Type = function returning void
                                    1-1:288

printf                              Class = extern
                                    Type = function returning int
                                    1-1:289, 14-0:14, 17-0:17, 28-0:28, 37-0:37
```

Figure 17. Example of a C listing (Part 11 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page    13

                                * * * * *   C R O S S   R E F E R E N C E   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>

putc                                Class = extern
                                    Type = function returning int
                                    1-1:290

putchar                             Class = extern
                                    Type = function returning int
                                    1-1:291

puts                                Class = extern
                                    Type = function returning int
                                    1-1:292

remove                              Class = extern
                                    Type = function returning int
                                    1-1:293

rename                              Class = extern
                                    Type = function returning int
                                    1-1:294

rewind                              Class = extern
                                    Type = function returning void
                                    1-1:295

scanf                               Class = extern
                                    Type = function returning int
                                    1-1:296, 16-0:16

setbuf                              Class = extern
                                    Type = function returning void
                                    1-1:297

setvbuf                             Class = extern
                                    Type = function returning int
                                    1-1:298

size_t              1-1:49          Class = typedef, Length = 4
                                    Type = unsigned int
                                    1-1:275, 1-1:275, 1-1:276, 1-1:283, 1-1:283, 1-1:283, 1-1:299, 1-1:529, 1-1:531, 1-1:531

sprintf                             Class = extern
                                    Type = function returning int
                                    1-1:301

sscanf                              Class = extern
                                    Type = function returning int
                                    1-1:303, 27-0:27

svc99                               Class = extern
                                    Type = function returning int
                                    1-1:528
```

Figure 17. Example of a C listing (Part 12 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page    14

                                ***** C R O S S   R E F E R E N C E   L I S T I N G *****

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>
tmpfile                             Class = extern
                                    Type = function returning pointer to struct __ffile
                                    1-1:305

tmpnam                              Class = extern
                                    Type = function returning pointer to unsigned char
                                    1-1:306

ungetc                              Class = extern
                                    Type = function returning int
                                    1-1:307

va_list             1-1:124         Class = typedef, Length = 8
                                    Type = array[2] of pointer to unsigned char

vfprintf                            Class = extern
                                    Type = function returning int
                                    1-1:308

vprintf                             Class = extern
                                    Type = function returning int
                                    1-1:309

vsprintf                            Class = extern
                                    Type = function returning int
                                    1-1:310

FILE                1-1:78          Class = typedef, Length = 4
                                    Type = struct __ffile
                                    1-1:262, 1-1:263, 1-1:264, 1-1:265, 1-1:266, 1-1:267, 1-1:268, 1-1:269, 1-1:270, 1-1:272,
                                    1-1:273, 1-1:274, 1-1:276, 1-1:277, 1-1:278, 1-1:279, 1-1:280, 1-1:281, 1-1:282, 1-1:284,
                                    1-1:285, 1-1:290, 1-1:295, 1-1:297, 1-1:298, 1-1:305, 1-1:307, 1-1:308, 1-1:529, 1-1:530,
                                    1-1:531, 1-1:533, 1-1:625

                         ***** E N D   O F   C R O S S   R E F E R E N C E   L I S T I N G *****
```

Figure 17. Example of a C listing (Part 13 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'               04/14/2000 14:42:32   Page    15

                                  * * * * *   S T R U C T U R E   M A P S   * * * * *

===================================================================================================================================
| Aggregate map for:                                                                               Total size: 8 bytes            |
|.................................................................................................................................|
|__rrds_key_type                                                                                                                  |
|=================================================================================================================================|
|     Offset        |     Length        | Member Name                                                                             |
|   Bytes(Bits)     |   Bytes(Bits)     |                                                                                         |
|===================|===================|=========================================================================================|
|       0           |       4           |  __fill                                                                                 |
|       4           |       4           |  __recnum                                                                               |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for:                                                                               Total size: 4 bytes            |
|.................................................................................................................................|
|__code                                                                                                                           |
|=================================================================================================================================|
|     Offset        |     Length        | Member Name                                                                             |
|   Bytes(Bits)     |   Bytes(Bits)     |                                                                                         |
|===================|===================|=========================================================================================|
|       0           |       4           |  __error                                                                                |
|       0           |       4           |  __abend                                                                                |
|       0           |       2           |    __syscode                                                                            |
|       2           |       2           |    __rc                                                                                 |
|       0           |       4           |  __feedback                                                                             |
|       0           |       1           |    __fdbk_fill                                                                          |
|       1           |       1           |    __rc                                                                                 |
|       2           |       1           |    __ftncd                                                                              |
|       3           |       1           |    __fdbk                                                                               |
|       0           |       4           |  __alloc                                                                                |
|       0           |       2           |    __svc99_info                                                                         |
|       2           |       2           |    __svc99_error                                                                        |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for:                                                                               Total size: 4 bytes            |
|.................................................................................................................................|
|__abend                                                                                                                          |
|=================================================================================================================================|
|     Offset        |     Length        | Member Name                                                                             |
|   Bytes(Bits)     |   Bytes(Bits)     |                                                                                         |
|===================|===================|=========================================================================================|
|       0           |       2           |  __syscode                                                                              |
|       2           |       2           |  __rc                                                                                   |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for:                                                                               Total size: 4 bytes            |
|.................................................................................................................................|
|__feedback                                                                                                                       |
|=================================================================================================================================|
|     Offset        |     Length        | Member Name                                                                             |
|   Bytes(Bits)     |   Bytes(Bits)     |                                                                                         |
|===================|===================|=========================================================================================|
|       0           |       1           |  __fdbk_fill                                                                            |
```

Figure 17. Example of a C listing (Part 14 of 35)

```
                                  * * * * *   S T R U C T U R E   M A P S   * * * * *
|       1           |        1          |  __rc                                                                                      |
|       2           |        1          |  __ftncd                                                                                   |
|       3           |        1          |  __fdbk                                                                                    |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for:                                                                            Total size: 4 bytes              |
|.................................................................................................................................|
|__alloc                                                                                                                          |
|=================================================================================================================================|
|     Offset        |      Length       | Member Name                                                                              |
|   Bytes(Bits)     |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|       0           |        2          |  __svc99_info                                                                            |
|       2           |        2          |  __svc99_error                                                                           |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for:                                                                            Total size: 208 bytes            |
|.................................................................................................................................|
|__msg                                                                                                                            |
|=================================================================================================================================|
|     Offset        |      Length       | Member Name                                                                              |
|   Bytes(Bits)     |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|       0           |        4          |  __len_fill                                                                              |
|       4           |        4          |  __len                                                                                   |
|       8           |      120          |  __str[120]                                                                              |
|     128           |        4          |  __parmr0                                                                                |
|     132           |        4          |  __parmr1                                                                                |
|     136           |        8          |  __fill2[2]                                                                              |
|     144           |       64          |  __str2[64]                                                                              |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: struct __amrc_type                                                         Total size: 224 bytes            |
|                                                                                                                                 |
|=================================================================================================================================|
|     Offset        |      Length       | Member Name                                                                              |
|   Bytes(Bits)     |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|       0           |        4          |  __code                                                                                  |
|       0           |        4          |    __error                                                                               |
|       0           |        4          |    __abend                                                                               |
|       0           |        2          |      __syscode                                                                           |
|       2           |        2          |      __rc                                                                                |
|       0           |        4          |    __feedback                                                                            |
|       0           |        1          |      __fdbk_fill                                                                         |
|       1           |        1          |      __rc                                                                                |
|       2           |        1          |      __ftncd                                                                             |
|       3           |        1          |      __fdbk                                                                              |
|       0           |        4          |    __alloc                                                                               |
|       0           |        2          |      __svc99_info                                                                        |
|       2           |        2          |      __svc99_error                                                                       |
|       4           |        4          |  __RBA                                                                                   |
|       8           |        4          |  __last_op                                                                               |
```

Figure 17. Example of a C listing (Part 15 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'               04/14/2000 14:42:32   Page    17

                                  * * * * *   S T R U C T U R E   M A P S   * * * * *
|       12           |       208          |  __msg                                                                           |
|       12           |         4          |    __len_fill                                                                    |
|       16           |         4          |    __len                                                                         |
|       20           |       120          |    __str[120]                                                                    |
|      140           |         4          |    __parmr0                                                                      |
|      144           |         4          |    __parmr1                                                                      |
|      148           |         8          |    __fill2[2]                                                                    |
|      156           |        64          |    __str2[64]                                                                    |
|      220           |         4          |  __rplfdbwd[4]                                                                   |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: _Packed struct __amrc_type                                                  Total size: 224 bytes          |
|                                                                                                                                |
|================================================================================================================================|
|      Offset        |      Length        | Member Name                                                                     |
|    Bytes(Bits)     |    Bytes(Bits)     |                                                                                 |
|================== | ================== | ================================================================================|
|        0           |         4          |  __code                                                                          |
|        0           |         4          |    __error                                                                       |
|        0           |         4          |    __abend                                                                       |
|        0           |         2          |      __syscode                                                                   |
|        2           |         2          |      __rc                                                                        |
|        0           |         4          |    __feedback                                                                    |
|        0           |         1          |      __fdbk_fill                                                                 |
|        1           |         1          |      __rc                                                                        |
|        2           |         1          |      __ftncd                                                                     |
|        3           |         1          |      __fdbk                                                                      |
|        0           |         4          |    __alloc                                                                       |
|        0           |         2          |      __svc99_info                                                                |
|        2           |         2          |      __svc99_error                                                               |
|        4           |         4          |  __RBA                                                                           |
|        8           |         4          |  __last_op                                                                       |
|       12           |       208          |  __msg                                                                           |
|       12           |         4          |    __len_fill                                                                    |
|       16           |         4          |    __len                                                                         |
|       20           |       120          |    __str[120]                                                                    |
|      140           |         4          |    __parmr0                                                                      |
|      144           |         4          |    __parmr1                                                                      |
|      148           |         8          |    __fill2[2]                                                                    |
|      156           |        64          |    __str2[64]                                                                    |
|      220           |         4          |  __rplfdbwd[4]                                                                   |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: struct __amrctype                                                           Total size: 224 bytes          |
|                                                                                                                                |
|================================================================================================================================|
|      Offset        |      Length        | Member Name                                                                     |
|    Bytes(Bits)     |    Bytes(Bits)     |                                                                                 |
|================== | ================== | ================================================================================|
|        0           |         4          |  __code                                                                          |
|        0           |         4          |    __error                                                                       |
|        0           |         4          |    __abend                                                                       |
|        0           |         2          |      __syscode                                                                   |
|        2           |         2          |      __rc                                                                        |
```

Figure 17. Example of a C listing (Part 16 of 35)

```
                                 * * * * *   S T R U C T U R E   M A P S   * * * * *
|       0          |        4          |     __feedback                                                                          |
|       0          |        1          |       __fdbk_fill                                                                       |
|       1          |        1          |       __rc                                                                              |
|       2          |        1          |       __ftncd                                                                           |
|       3          |        1          |       __fdbk                                                                            |
|       0          |        4          |     __alloc                                                                             |
|       0          |        2          |       __svc99_info                                                                      |
|       2          |        2          |       __svc99_error                                                                     |
|       4          |        4          |   __RBA                                                                                 |
|       8          |        4          |   __last_op                                                                             |
|      12          |      208          |   __msg                                                                                 |
|      12          |        4          |     __len_fill                                                                          |
|      16          |        4          |     __len                                                                               |
|      20          |      120          |     __str[120]                                                                          |
|     140          |        4          |     __parmr0                                                                            |
|     144          |        4          |     __parmr1                                                                            |
|     148          |        8          |     __fill2[2]                                                                          |
|     156          |       64          |     __str2[64]                                                                          |
|     220          |        4          |   __rplfdbwd[4]                                                                         |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: _Packed struct __amrctype                                                     Total size: 224 bytes           |
|===================================================================================================================================|
|      Offset      |      Length       | Member Name                                                                            |
|    Bytes(Bits)   |    Bytes(Bits)    |                                                                                        |
|==================|===================|========================================================================================|
|       0          |        4          |   __code                                                                                |
|       0          |        4          |     __error                                                                             |
|       0          |        4          |     __abend                                                                             |
|       0          |        2          |       __syscode                                                                         |
|       2          |        2          |       __rc                                                                              |
|       0          |        4          |     __feedback                                                                          |
|       0          |        1          |       __fdbk_fill                                                                       |
|       1          |        1          |       __rc                                                                              |
|       2          |        1          |       __ftncd                                                                           |
|       3          |        1          |       __fdbk                                                                            |
|       0          |        4          |     __alloc                                                                             |
|       0          |        2          |       __svc99_info                                                                      |
|       2          |        2          |       __svc99_error                                                                     |
|       4          |        4          |   __RBA                                                                                 |
|       8          |        4          |   __last_op                                                                             |
|      12          |      208          |   __msg                                                                                 |
|      12          |        4          |     __len_fill                                                                          |
|      16          |        4          |     __len                                                                               |
|      20          |      120          |     __str[120]                                                                          |
|     140          |        4          |     __parmr0                                                                            |
|     144          |        4          |     __parmr1                                                                            |
|     148          |        8          |     __fill2[2]                                                                          |
|     156          |       64          |     __str2[64]                                                                          |
|     220          |        4          |   __rplfdbwd[4]                                                                         |
=====================================================================================================================================
```

Figure 17. Example of a C listing (Part 17 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page    19

                                  * * * * *   S T R U C T U R E   M A P S   * * * * *

===================================================================================================================================
| Aggregate map for: struct __amrc2_type                                                            Total size: 32 bytes           |
|                                                                                                                                  |
|==================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                              |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|       0           |       4           |  __error2                                                                                |
|       4           |       4           |  __fileptr                                                                               |
|       8           |      24           |  __reserved[6]                                                                           |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for: _Packed struct __amrc2_type                                                    Total size: 32 bytes           |
|                                                                                                                                  |
|==================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                              |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|       0           |       4           |  __error2                                                                                |
|       4           |       4           |  __fileptr                                                                               |
|       8           |      24           |  __reserved[6]                                                                           |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for: struct __amrc2type                                                             Total size: 32 bytes           |
|                                                                                                                                  |
|==================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                              |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|       0           |       4           |  __error2                                                                                |
|       4           |       4           |  __fileptr                                                                               |
|       8           |      24           |  __reserved[6]                                                                           |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for: _Packed struct __amrc2type                                                     Total size: 32 bytes           |
|                                                                                                                                  |
|==================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                              |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|       0           |       4           |  __error2                                                                                |
|       4           |       4           |  __fileptr                                                                               |
|       8           |      24           |  __reserved[6]                                                                           |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for: struct __ffile                                                                 Total size: 4 bytes            |
|                                                                                                                                  |
|==================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                              |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
```

Figure 17. Example of a C listing (Part 18 of 35)

```
                                  * * * * *   S T R U C T U R E   M A P S   * * * * *
|       0           |       4           |  __fp                                                                                  |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: _Packed struct __ffile                                                      Total size: 4 bytes            |
|                                                                                                                                |
|================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                            |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                        |
|===================|===================|========================================================================================|
|       0           |       4           |  __fp                                                                                  |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: struct __file                                                               Total size: 24 bytes           |
|                                                                                                                                |
|================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                            |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                        |
|===================|===================|========================================================================================|
|       0           |       4           |  __bufPtr                                                                              |
|       4           |       4           |  __countIn                                                                             |
|       8           |       4           |  __countOut                                                                            |
|      12           |       4           |  __fcbgetc                                                                             |
|      16           |       4           |  __fcbputc                                                                             |
|      20           |       0(1)        |  __cntlinterpret                                                                       |
|      20(1)        |       3(7)        |  ***PADDING***                                                                         |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: _Packed struct __file                                                       Total size: 21 bytes           |
|                                                                                                                                |
|================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                            |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                        |
|===================|===================|========================================================================================|
|       0           |       4           |  __bufPtr                                                                              |
|       4           |       4           |  __countIn                                                                             |
|       8           |       4           |  __countOut                                                                            |
|      12           |       4           |  __fcbgetc                                                                             |
|      16           |       4           |  __fcbputc                                                                             |
|      20           |       0(1)        |  __cntlinterpret                                                                       |
|      20(1)        |       0(7)        |  ***PADDING***                                                                         |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: struct __fileData                                                           Total size: 36 bytes           |
|                                                                                                                                |
|================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                            |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                        |
|===================|===================|========================================================================================|
|       0           |       0(1)        |  __recfmF                                                                              |
|       0(1)        |       0(1)        |  __recfmV                                                                              |
|       0(2)        |       0(1)        |  __recfmU                                                                              |
|       0(3)        |       0(1)        |  __recfmS                                                                              |
```

Figure 17. Example of a C listing (Part 19 of 35)

```
                                 * * * * *   S T R U C T U R E   M A P S   * * * * *
|       0(4)        |       0(1)        |  __recfmBlk                                                                                |
|       0(5)        |       0(1)        |  __recfmASA                                                                                |
|       0(6)        |       0(1)        |  __recfmM                                                                                  |
|       0(7)        |       0(1)        |  __dsorgPO                                                                                 |
|       1           |       0(1)        |  __dsorgPDSmem                                                                             |
|       1(1)        |       0(1)        |  __dsorgPDSdir                                                                             |
|       1(2)        |       0(1)        |  __dsorgPS                                                                                 |
|       1(3)        |       0(1)        |  __dsorgConcat                                                                             |
|       1(4)        |       0(1)        |  __dsorgMem                                                                                |
|       1(5)        |       0(1)        |  __dsorgHiper                                                                              |
|       1(6)        |       0(1)        |  __dsorgTemp                                                                               |
|       1(7)        |       0(1)        |  __dsorgVSAM                                                                               |
|       2           |       0(1)        |  __dsorgHFS                                                                                |
|       2(1)        |       0(2)        |  __openmode                                                                                |
|       2(3)        |       0(4)        |  __modeflag                                                                                |
|       2(7)        |       0(1)        |  __dsorgPDSE                                                                               |
|       3           |       0(3)        |  __vsamRLS                                                                                 |
|       3(3)        |       0(5)        |  __reserve2                                                                                |
|       4           |       1           |  __device                                                                                  |
|       5           |       3           |  ***PADDING***                                                                             |
|       8           |       4           |  __blksize                                                                                 |
|      12           |       4           |  __maxreclen                                                                               |
|      16           |       2           |  __vsamtype                                                                                |
|      18           |       2           |  ***PADDING***                                                                             |
|      20           |       4           |  __vsamkeylen                                                                              |
|      24           |       4           |  __vsamRKP                                                                                 |
|      28           |       4           |  __dsname                                                                                  |
|      32           |       4           |  __reserve4                                                                                |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: _Packed struct __fileData                                                      Total size: 31 bytes         |
|                                                                                                                                |
|================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                               |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                           |
|===================|===================|===========================================================================================|
|       0           |       0(1)        |  __recfmF                                                                                  |
|       0(1)        |       0(1)        |  __recfmV                                                                                  |
|       0(2)        |       0(1)        |  __recfmU                                                                                  |
|       0(3)        |       0(1)        |  __recfmS                                                                                  |
|       0(4)        |       0(1)        |  __recfmBlk                                                                                |
|       0(5)        |       0(1)        |  __recfmASA                                                                                |
|       0(6)        |       0(1)        |  __recfmM                                                                                  |
|       0(7)        |       0(1)        |  __dsorgPO                                                                                 |
|       1           |       0(1)        |  __dsorgPDSmem                                                                             |
|       1(1)        |       0(1)        |  __dsorgPDSdir                                                                             |
|       1(2)        |       0(1)        |  __dsorgPS                                                                                 |
|       1(3)        |       0(1)        |  __dsorgConcat                                                                             |
|       1(4)        |       0(1)        |  __dsorgMem                                                                                |
|       1(5)        |       0(1)        |  __dsorgHiper                                                                              |
|       1(6)        |       0(1)        |  __dsorgTemp                                                                               |
|       1(7)        |       0(1)        |  __dsorgVSAM                                                                               |
|       2           |       0(1)        |  __dsorgHFS                                                                                |
|       2(1)        |       0(2)        |  __openmode                                                                                |
|       2(3)        |       0(4)        |  __modeflag                                                                                |
```

Figure 17. Example of a C listing (Part 20 of 35)

```
                                  * * * * *   S T R U C T U R E   M A P S   * * * * *
|       2(7)        |       0(1)        |  __dsorgPDSE                                                                            |
|       3           |       0(3)        |  __vsamRLS                                                                              |
|       3(3)        |       0(5)        |  __reserve2                                                                             |
|       4           |       1           |  __device                                                                               |
|       5           |       4           |  __blksize                                                                              |
|       9           |       4           |  __maxreclen                                                                            |
|      13           |       2           |  __vsamtype                                                                             |
|      15           |       4           |  __vsamkeylen                                                                           |
|      19           |       4           |  __vsamRKP                                                                              |
|      23           |       4           |  __dsname                                                                               |
|      27           |       4           |  __reserve4                                                                             |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: struct __fpos_t                                                         Total size: 32 bytes             |
|                                                                                                                               |
|===============================================================================================================================|
|      Offset       |      Length       | Member Name                                                                            |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                        |
|===================|===================|========================================================================================|
|       0           |      32           |  __fpos_elem[8]                                                                         |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: _Packed struct __fpos_t                                                 Total size: 32 bytes             |
|                                                                                                                               |
|===============================================================================================================================|
|      Offset       |      Length       | Member Name                                                                            |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                        |
|===================|===================|========================================================================================|
|       0           |      32           |  __fpos_elem[8]                                                                         |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: struct __S99emparms                                                     Total size: 28 bytes             |
|                                                                                                                               |
|===============================================================================================================================|
|      Offset       |      Length       | Member Name                                                                            |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                        |
|===================|===================|========================================================================================|
|       0           |       1           |  __EMFUNCT                                                                              |
|       1           |       1           |  __EMIDNUM                                                                              |
|       2           |       1           |  __EMNMSGBK                                                                             |
|       3           |       1           |  __filler1                                                                              |
|       4           |       4           |  __EMS99RBP                                                                             |
|       8           |       4           |  __EMRETCOD                                                                             |
|      12           |       4           |  __EMCPPLP                                                                              |
|      16           |       4           |  __EMBUFP                                                                               |
|      20           |       4           |  __reserv1                                                                              |
|      24           |       4           |  __reserv2                                                                              |
=====================================================================================================================================
```

Figure 17. Example of a C listing (Part 21 of 35)

```
5647A01 V2 R10 OS/390 C              'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'              04/14/2000 14:42:32   Page    23

                                  * * * * *   S T R U C T U R E   M A P S   * * * * *

===================================================================================================================================
| Aggregate map for: _Packed struct __S99emparms                                                     Total size: 28 bytes           |
|                                                                                                                                   |
|===================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                                 |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                             |
|===================|===================|=============================================================================================|
|        0          |        1          |  __EMFUNCT                                                                                  |
|        1          |        1          |  __EMIDNUM                                                                                  |
|        2          |        1          |  __EMNMSGBK                                                                                 |
|        3          |        1          |  __filler1                                                                                  |
|        4          |        4          |  __EMS99RBP                                                                                 |
|        8          |        4          |  __EMRETCOD                                                                                 |
|       12          |        4          |  __EMCPPLP                                                                                  |
|       16          |        4          |  __EMBUFP                                                                                   |
|       20          |        4          |  __reserv1                                                                                  |
|       24          |        4          |  __reserv2                                                                                  |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for: struct __S99parms                                                               Total size: 20 bytes           |
|                                                                                                                                   |
|===================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                                 |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                             |
|===================|===================|=============================================================================================|
|        0          |        1          |  __S99RBLN                                                                                  |
|        1          |        1          |  __S99VERB                                                                                  |
|        2          |        2          |  __S99FLAG1                                                                                 |
|        4          |        2          |  __S99ERROR                                                                                 |
|        6          |        2          |  __S99INFO                                                                                  |
|        8          |        4          |  __S99TXTPP                                                                                 |
|       12          |        4          |  __S99S99X                                                                                  |
|       16          |        4          |  __S99FLAG2                                                                                 |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for: _Packed struct __S99parms                                                       Total size: 20 bytes           |
|                                                                                                                                   |
|===================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                                 |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                             |
|===================|===================|=============================================================================================|
|        0          |        1          |  __S99RBLN                                                                                  |
|        1          |        1          |  __S99VERB                                                                                  |
|        2          |        2          |  __S99FLAG1                                                                                 |
|        4          |        2          |  __S99ERROR                                                                                 |
|        6          |        2          |  __S99INFO                                                                                  |
|        8          |        4          |  __S99TXTPP                                                                                 |
|       12          |        4          |  __S99S99X                                                                                  |
|       16          |        4          |  __S99FLAG2                                                                                 |
===================================================================================================================================
```

Figure 17. Example of a C listing (Part 22 of 35)

```
5647A01 V2 R10 OS/390 C             'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'             04/14/2000 14:42:32   Page    24

                                  * * * * *   S T R U C T U R E   M A P S   * * * * *

==================================================================================================================================
| Aggregate map for: struct __S99rbx                                                              Total size: 36 bytes           |
|                                                                                                                                |
|================================================================================================================================|
|      Offset        |      Length        | Member Name                                                                           |
|    Bytes(Bits)     |    Bytes(Bits)     |                                                                                       |
|====================|====================|=======================================================================================|
|        0           |        6           |  __S99EID[6]                                                                          |
|        6           |        1           |  __S99EVER                                                                            |
|        7           |        1           |  __S99EOPTS                                                                           |
|        8           |        1           |  __S99ESUBP                                                                           |
|        9           |        1           |  __S99EKEY                                                                            |
|       10           |        1           |  __S99EMGSV                                                                           |
|       11           |        1           |  __S99ENMSG                                                                           |
|       12           |        4           |  __S99ECPPL                                                                           |
|       16           |        1           |  __reserved                                                                           |
|       17           |        1           |  __S99ERES                                                                            |
|       18           |        1           |  __S99ERCO                                                                            |
|       19           |        1           |  __S99ERCF                                                                            |
|       20           |        4           |  __S99EWRC                                                                            |
|       24           |        4           |  __S99EMSGP                                                                           |
|       28           |        2           |  __S99EERR                                                                            |
|       30           |        2           |  __S99EINFO                                                                           |
|       32           |        4           |  __reserv2                                                                            |
==================================================================================================================================

==================================================================================================================================
| Aggregate map for: _Packed struct __S99rbx                                                      Total size: 36 bytes           |
|                                                                                                                                |
|================================================================================================================================|
|      Offset        |      Length        | Member Name                                                                           |
|    Bytes(Bits)     |    Bytes(Bits)     |                                                                                       |
|====================|====================|=======================================================================================|
|        0           |        6           |  __S99EID[6]                                                                          |
|        6           |        1           |  __S99EVER                                                                            |
|        7           |        1           |  __S99EOPTS                                                                           |
|        8           |        1           |  __S99ESUBP                                                                           |
|        9           |        1           |  __S99EKEY                                                                            |
|       10           |        1           |  __S99EMGSV                                                                           |
|       11           |        1           |  __S99ENMSG                                                                           |
|       12           |        4           |  __S99ECPPL                                                                           |
|       16           |        1           |  __reserved                                                                           |
|       17           |        1           |  __S99ERES                                                                            |
|       18           |        1           |  __S99ERCO                                                                            |
|       19           |        1           |  __S99ERCF                                                                            |
|       20           |        4           |  __S99EWRC                                                                            |
|       24           |        4           |  __S99EMSGP                                                                           |
|       28           |        2           |  __S99EERR                                                                            |
|       30           |        2           |  __S99EINFO                                                                           |
|       32           |        4           |  __reserv2                                                                            |
==================================================================================================================================
```

Figure 17. Example of a C listing (Part 23 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'              04/14/2000 14:42:32   Page    25

                                  * * * * *   S T R U C T U R E   M A P S   * * * * *

===================================================================================================================================
| Aggregate map for: struct __S99struc                                                              Total size: 20 bytes           |
|                                                                                                                                  |
|==================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                              |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|        0          |        1          |  __S99RBLN                                                                               |
|        1          |        1          |  __S99VERB                                                                               |
|        2          |        2          |  __S99FLAG1                                                                              |
|        4          |        2          |  __S99ERROR                                                                              |
|        6          |        2          |  __S99INFO                                                                               |
|        8          |        4          |  __S99TXTPP                                                                              |
|       12          |        4          |  __S99S99X                                                                               |
|       16          |        4          |  __S99FLAG2                                                                              |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for: _Packed struct __S99struc                                                      Total size: 20 bytes           |
|                                                                                                                                  |
|==================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                              |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|        0          |        1          |  __S99RBLN                                                                               |
|        1          |        1          |  __S99VERB                                                                               |
|        2          |        2          |  __S99FLAG1                                                                              |
|        4          |        2          |  __S99ERROR                                                                              |
|        6          |        2          |  __S99INFO                                                                               |
|        8          |        4          |  __S99TXTPP                                                                              |
|       12          |        4          |  __S99S99X                                                                               |
|       16          |        4          |  __S99FLAG2                                                                              |
===================================================================================================================================

===================================================================================================================================
| Aggregate map for: struct fldata_t                                                                Total size: 36 bytes           |
|                                                                                                                                  |
|==================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                              |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                          |
|===================|===================|==========================================================================================|
|        0          |         0(1)      |  __recfmF                                                                                |
|        0(1)       |         0(1)      |  __recfmV                                                                                |
|        0(2)       |         0(1)      |  __recfmU                                                                                |
|        0(3)       |         0(1)      |  __recfmS                                                                                |
|        0(4)       |         0(1)      |  __recfmBlk                                                                              |
|        0(5)       |         0(1)      |  __recfmASA                                                                              |
|        0(6)       |         0(1)      |  __recfmM                                                                                |
|        0(7)       |         0(1)      |  __dsorgPO                                                                               |
|        1          |         0(1)      |  __dsorgPDSmem                                                                           |
|        1(1)       |         0(1)      |  __dsorgPDSdir                                                                           |
|        1(2)       |         0(1)      |  __dsorgPS                                                                               |
|        1(3)       |         0(1)      |  __dsorgConcat                                                                           |
|        1(4)       |         0(1)      |  __dsorgMem                                                                              |
|        1(5)       |         0(1)      |  __dsorgHiper                                                                            |
```

Figure 17. Example of a C listing (Part 24 of 35)

```
                                  * * * * *   S T R U C T U R E   M A P S   * * * * *
|       1(6)         |       0(1)         |  __dsorgTemp                                                                        |
|       1(7)         |       0(1)         |  __dsorgVSAM                                                                        |
|       2            |       0(1)         |  __dsorgHFS                                                                         |
|       2(1)         |       0(2)         |  __openmode                                                                         |
|       2(3)         |       0(4)         |  __modeflag                                                                         |
|       2(7)         |       0(1)         |  __dsorgPDSE                                                                        |
|       3            |       0(3)         |  __vsamRLS                                                                          |
|       3(3)         |       0(5)         |  __reserve2                                                                         |
|       4            |       1            |  __device                                                                           |
|       5            |       3            |  ***PADDING***                                                                      |
|       8            |       4            |  __blksize                                                                          |
|      12            |       4            |  __maxreclen                                                                        |
|      16            |       2            |  __vsamtype                                                                         |
|      18            |       2            |  ***PADDING***                                                                      |
|      20            |       4            |  __vsamkeylen                                                                       |
|      24            |       4            |  __vsamRKP                                                                          |
|      28            |       4            |  __dsname                                                                           |
|      32            |       4            |  __reserve4                                                                         |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: _Packed struct fldata_t                                                      Total size: 31 bytes             |
|                                                                                                                                   |
|===================================================================================================================================|
|      Offset        |      Length        | Member Name                                                                         |
|    Bytes(Bits)     |    Bytes(Bits)     |                                                                                     |
|====================|====================|=====================================================================================|
|       0            |       0(1)         |  __recfmF                                                                           |
|       0(1)         |       0(1)         |  __recfmV                                                                           |
|       0(2)         |       0(1)         |  __recfmU                                                                           |
|       0(3)         |       0(1)         |  __recfmS                                                                           |
|       0(4)         |       0(1)         |  __recfmBlk                                                                         |
|       0(5)         |       0(1)         |  __recfmASA                                                                         |
|       0(6)         |       0(1)         |  __recfmM                                                                           |
|       0(7)         |       0(1)         |  __dsorgPO                                                                          |
|       1            |       0(1)         |  __dsorgPDSmem                                                                      |
|       1(1)         |       0(1)         |  __dsorgPDSdir                                                                      |
|       1(2)         |       0(1)         |  __dsorgPS                                                                          |
|       1(3)         |       0(1)         |  __dsorgConcat                                                                      |
|       1(4)         |       0(1)         |  __dsorgMem                                                                         |
|       1(5)         |       0(1)         |  __dsorgHiper                                                                       |
|       1(6)         |       0(1)         |  __dsorgTemp                                                                        |
|       1(7)         |       0(1)         |  __dsorgVSAM                                                                        |
|       2            |       0(1)         |  __dsorgHFS                                                                         |
|       2(1)         |       0(2)         |  __openmode                                                                         |
|       2(3)         |       0(4)         |  __modeflag                                                                         |
|       2(7)         |       0(1)         |  __dsorgPDSE                                                                        |
|       3            |       0(3)         |  __vsamRLS                                                                          |
|       3(3)         |       0(5)         |  __reserve2                                                                         |
|       4            |       1            |  __device                                                                           |
|       5            |       4            |  __blksize                                                                          |
|       9            |       4            |  __maxreclen                                                                        |
|      13            |       2            |  __vsamtype                                                                         |
|      15            |       4            |  __vsamkeylen                                                                       |
|      19            |       4            |  __vsamRKP                                                                          |
|      23            |       4            |  __dsname                                                                           |
```

Figure 17. Example of a C listing (Part 25 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'               04/14/2000 14:42:32   Page    27

                                  * * * * *   S T R U C T U R E   M A P S   * * * * *
|      27           |       4           |  __reserve4                                                                         |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: struct fpos_t                                                               Total size: 32 bytes           |
|                                                                                                                                   |
|===================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                         |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                     |
|===================|===================|=====================================================================================|
|       0           |       32          |  __fpos_elem[8]                                                                     |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: _Packed struct fpos_t                                                       Total size: 32 bytes           |
|                                                                                                                                   |
|===================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                         |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                     |
|===================|===================|=====================================================================================|
|       0           |       32          |  __fpos_elem[8]                                                                     |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: struct FILE                                                                 Total size: 4 bytes            |
|                                                                                                                                   |
|===================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                         |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                     |
|===================|===================|=====================================================================================|
|       0           |       4           |  __fp                                                                               |
=====================================================================================================================================

=====================================================================================================================================
| Aggregate map for: _Packed struct FILE                                                         Total size: 4 bytes            |
|                                                                                                                                   |
|===================================================================================================================================|
|      Offset       |      Length       | Member Name                                                                         |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                                                     |
|===================|===================|=====================================================================================|
|       0           |       4           |  __fp                                                                               |
=====================================================================================================================================
                           * * * * *   E N D   O F   S T R U C T U R E   M A P S   * * * * *
```

Figure 17. Example of a C listing (Part 26 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page    28

                             * * * * *   M E S S A G E   S U M M A R Y   * * * * *

        Total            Informational(00)          Warning(10)         Error(30)           Severe Error(40)

          0                    0                        0                   0                        0
                        * * * * *   E N D   O F   M E S S A G E   S U M M A R Y   * * * * *
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page    29

                    Inline Report (Summary)

 Reason:    P : #pragma noinline was specified for this routine
            F : #pragma inline was specified for this routine
            A : Automatic inlining
            - : No reason
 Action:    I : Routine is inlined at least once
            L : Routine is initially too large to be inlined
            T : Routine expands too large to be inlined
            C : Candidate for inlining but not inlined
            N : No direct calls to routine are found in file (no action)
            U : Some calls not inlined due to recursion or parameter mismatch
            - : No action
 Status:    D : Internal routine is discarded
            R : A direct call remains to internal routine (cannot discard)
            A : Routine has its address taken (cannot discard)
            E : External routine (cannot discard)
            - : Status unchanged
 Calls/I      : Number of calls to defined routines / Number inline
 Called/I     : Number of times called / Number of times inlined

 Reason  Action  Status   Size (init)    Calls/I   Called/I     Name

    A      I       E      15              0        2/2         convert
    A      T,N     E      111   (73)      2/2      0           main

 Mode = AUTO    Inlining Threshold = 100    Expansion Limit = 1000

5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page    30

                    Inline Report (Call Structure)

 Defined Function      : convert
     Calls To          : 0
  Called From(2,2)     : main(2,2)

 Defined Function      : main
     Calls To(2,2)     : convert(2,2)
  Called From          : 0

5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                04/14/2000 14:42:32   Page    31

OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

000000  41F0  F060                                LA    r15,96(,r15)
000004  07FF                                      BR    r15
000006  0700                                      NOPR  0
000008  00000000                                        =F'0'

                          Timestamp and Version Information
00000C  F2F0  F0F0                                      =C'2000'          Compiled Year
000010  F0F4  F1F4                                      =C'0414'          Compiled Date MMDD
000014  F1F4  F4F2  F3F2                                =C'144232'        Compiled Time HHMMSS
00001A  F0F2  F0C1  F0F0                                =C'020A00'        Compiler Version
                          Timestamp and Version End
```

Figure 17. Example of a C listing (Part 27 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)': main           04/14/2000 14:42:32   Page    32

OFFSET OBJECT CODE       LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                         PPA1: Entry Point Constants
000020  1CCEA106                                       =F'483303686'      Flags
000024  000002F0                                       =A(PPA2-main)
000028  00000000                                       =F'0'              No PPA3
00002C  00000000                                       =F'0'              No EPD
000030  FF800000                                       =F'-8388608'       Register save mask
000034  00000000                                       =F'0'              Member flags
000038  90                                             =AL1(144)          Flags
000039  000000                                         =AL3(0)            Callee's DSA use/8
00003C  0040                                           =H'64'             Flags
00003E  0012                                           =H'18'             Offset/2 to CDL
000040  00000000                                       =F'0'              Reserved
000044  500000E9                                       =F'1342177513'     CDL function length/2
000048  00000040                                       =F'64'             CDL function EP offset
00004C  38260000                                       =F'942014464'      CDL prolog
000050  00000000                                       =F'0'              CDL epilog
000054  00000000                                       =F'0'              CDL end
000058  0004  ****                                     AL2(4),C'main'
                         PPA1 End

                         00001 |       *  #include <stdio.h>
                         00002 |       *
                         00003 |       *  #include "cbc3uaan.h"
                         00004 |       *
                         00005 |       *  void convert(double);
                         00006 |       *
                         00007 |       *  int main(int argc, char **argv)
000060                   00007 |        main     DS    0D
000060  47F0  F022       00007 |                 B     34(,r15)
000064  01C3C5C5                                       CEE eyecatcher
000068  000000D8                                       DSA size
00006C  FFFFFFC0                                       =A(PPA1-main)
000070  47F0  F001       00007 |                 B     1(,r15)
000074  58F0  C31C       00007 |                 L     r15,796(,r12)
000078  184E             00007 |                 LR    r4,r14
00007A  05EF             00007 |                 BALR  r14,r15
00007C  00000000                                       =F'0'
000080  07F3             00007 |                 BR    r3
000082  90E6  D00C       00007 |                 STM   r14,r6,12(r13)
000086  58E0  D04C       00007 |                 L     r14,76(,r13)
00008A  4100  E0D8       00007 |                 LA    r0,216(,r14)
00008E  5500  C314       00007 |                 CL    r0,788(,r12)
000092  4130  F03A       00007 |                 LA    r3,58(,r15)
000096  4720  F014       00007 |                 BH    20(,r15)
00009A  58F0  C280       00007 |                 L     r15,640(,r12)
00009E  90F0  E048       00007 |                 STM   r15,r0,72(r14)
0000A2  9210  E000       00007 |                 MVI   0(r14),16
0000A6  50D0  E004       00007 |                 ST    r13,4(,r14)
0000AA  18DE             00007 |                 LR    r13,r14
0000AC                   End of Prolog

0000AC  5850  319A       00007 |                 L     r5,=A(@STATIC)(,r3,410)
0000B0  5860  319E       00007 |                 L     r6,=A(@CONSTANT_AREA)(,r3,414)
0000B4  5010  D0D0       00007 |                 ST    r1,#SR_PARM_1(,r13,208)
                         00008 |       *  {
                         00009 |       *      double c_temp;
```

Figure 17. Example of a C listing (Part 28 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)': main            04/14/2000 14:42:32   Page    33

OFFSET OBJECT CODE       LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                         00010 |       *
                         00011 |       *      if (argc == 1) {  /* get Celsius value from stdin */
0000B8  5810  D0D0       00011 |                 L     r1,#SR_PARM_1(,r13,208)
0000BC  5800  1000       00011 |                 L     r0,argc(,r1,0)
0000C0  A70E  0001       00011 |                 CHI   r0,H'1'
0000C4  4770  30B2       00011 |                 BNE   @1L1
                         00012 |       *         int ch;
                         00013 |       *
                         00014 |       *         printf("Enter Celsius temperature: \n");
0000C8  4100  5008       00014 |                 LA    r0,""2(,r5,8)
0000CC  58F0  31A2       00014 |                 L     r15,=V(PRINTF)(,r3,418)
0000D0  4110  D098       00014 |                 LA    r1,#MX_TEMP1(,r13,152)
0000D4  5000  D098       00014 |                 ST    r0,#MX_TEMP1(,r13,152)
0000D8  05EF             00014 |                 BALR  r14,r15
                         00015 |       *
                         00016 |       *         if (scanf("%f", :c_temp) != 1) {
0000DA  4100  D0B0       00016 |                 LA    r0,c_temp(,r13,176)
0000DE  1825             00016 |                 LR    r2,r5
0000E0  58F0  31A6       00016 |                 L     r15,=V(SCANF)(,r3,422)
0000E4  4110  D098       00016 |                 LA    r1,#MX_TEMP1(,r13,152)
0000E8  5020  D098       00016 |                 ST    r2,#MX_TEMP1(,r13,152)
0000EC  5000  D09C       00016 |                 ST    r0,#MX_TEMP1(,r13,156)
0000F0  05EF             00016 |                 BALR  r14,r15
0000F2  180F             00016 |                 LR    r0,r15
0000F4  A70E  0001       00016 |                 CHI   r0,H'1'
0000F8  4780  3078       00016 |                 BE    @1L2
                         00017 |       *            printf("You must enter a valid temperature\n");
0000FC  4100  506C       00017 |                 LA    r0,""4(,r5,108)
000100  58F0  31A2       00017 |                 L     r15,=V(PRINTF)(,r3,418)
000104  4110  D098       00017 |                 LA    r1,#MX_TEMP1(,r13,152)
000108  5000  D098       00017 |                 ST    r0,#MX_TEMP1(,r13,152)
00010C  05EF             00017 |                 BALR  r14,r15
00010E  47F0  30AE       00017 |                 B     @1L3
000112                   00017 |        @1L2     DS    0H
                         00018 |       *         }
                         00019 |       *         else {
                         00020 |       *            convert(c_temp);
000112  6800  D0B0       00020 |                 LD    f0,c_temp(,r13,176)
000116  6000  D0C0       00020 |                 STD   f0,c_temp:convert(,r13,192)
00011A  6820  6000       00036 |       +         LD    f2,+CONSTANT_AREA(,r6,0)
00011E  2C02             00036 |       +         MDR   f0,f2
000120  6820  6008       00036 |       +         LD    f2,+CONSTANT_AREA(,r6,8)
000124  2A02             00036 |       +         ADR   f0,f2
000126  6000  D0C8       00036 |       +         STD   f0,f_temp:convert(,r13,200)
00012A  6820  D0C0       00037 |       +         LD    f2,c_temp:convert(,r13,192)
00012E  4100  5048       00037 |       +         LA    r0,""12(,r5,72)
000132  6020  D09C       00037 |       +         STD   f2,#MX_TEMP1(,r13,156)
000136  6000  D0A4       00037 |       +         STD   f0,#MX_TEMP1(,r13,164)
00013A  58F0  31A2       00037 |       +         L     r15,=V(PRINTF)(,r3,418)
00013E  4110  D098       00037 |       +         LA    r1,#MX_TEMP1(,r13,152)
000142  5000  D098       00037 |       +         ST    r0,#MX_TEMP1(,r13,152)
000146  05EF             00037 |       +         BALR  r14,r15
000148                   00038 |       +@1L10    DS    0H
000148                   00038 |       +@1L3     DS    0H
000148  47F0  3182       00038 |       +         B     @1L4
00014C                   00038 |       +@1L1     DS    0H
```

Figure 17. Example of a C listing (Part 29 of 35)

```
OFFSET OBJECT CODE       LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                         00021 |       *         }
                         00022 |       *      }
                         00023 |       *      else {  /* convert the command-line arguments to Fahrenheit */
                         00024 |       *         int i;
                         00025 |       *
                         00026 |       *         for (i = 1; i < argc; ++i) {
00014C  4100  0001       00026 |                 LA    r0,1
000150  5000  D0B8       00026 |                 ST    r0,i(,r13,184)
000154  5810  D0D0       00026 |                 L     r1,#SR_PARM_1(,r13,208)
000158  5810  1000       00026 |                 L     r1,argc(,r1,0)
00015C  1901             00026 |                 CR    r0,r1
00015E  47B0  3182       00026 |                 BNL   @1L6
000162                   00026 |        @1L5     DS    0H
                         00027 |       *             if (sscanf(argv[i], "%f", :c_temp) != 1)
000162  5810  D0D0       00027 |                 L     r1,#SR_PARM_1(,r13,208)
000166  5810  1004       00027 |                 L     r1,argv(,r1,4)
00016A  5820  D0B8       00027 |                 L     r2,i(,r13,184)
00016E  8920  0002       00027 |                 SLL   r2,2
000172  5842  1000       00027 |                 L     r4,(*)uchar*(r2,r1,0)
000176  4100  D0B0       00027 |                 LA    r0,c_temp(,r13,176)
00017A  4120  5004       00027 |                 LA    r2,""6(,r5,4)
00017E  58F0  31AA       00027 |                 L     r15,=V(SSCANF)(,r3,426)
000182  4110  D098       00027 |                 LA    r1,#MX_TEMP1(,r13,152)
000186  5040  D098       00027 |                 ST    r4,#MX_TEMP1(,r13,152)
00018A  5020  D09C       00027 |                 ST    r2,#MX_TEMP1(,r13,156)
00018E  5000  D0A0       00027 |                 ST    r0,#MX_TEMP1(,r13,160)
000192  05EF             00027 |                 BALR  r14,r15
000194  180F             00027 |                 LR    r0,r15
000196  A70E  0001       00027 |                 CHI   r0,H'1'
00019A  4780  3132       00027 |                 BE    @1L7
                         00028 |       *                printf("%s is not a valid temperature\n",argv[i]);
00019E  5810  D0D0       00028 |                 L     r1,#SR_PARM_1(,r13,208)
0001A2  5810  1004       00028 |                 L     r1,argv(,r1,4)
0001A6  5820  D0B8       00028 |                 L     r2,i(,r13,184)
0001AA  8920  0002       00028 |                 SLL   r2,2
0001AE  5802  1000       00028 |                 L     r0,(*)uchar*(r2,r1,0)
0001B2  4120  5028       00028 |                 LA    r2,""7(,r5,40)
0001B6  58F0  31A2       00028 |                 L     r15,=V(PRINTF)(,r3,418)
0001BA  4110  D098       00028 |                 LA    r1,#MX_TEMP1(,r13,152)
0001BE  5020  D098       00028 |                 ST    r2,#MX_TEMP1(,r13,152)
0001C2  5000  D09C       00028 |                 ST    r0,#MX_TEMP1(,r13,156)
0001C6  05EF             00028 |                 BALR  r14,r15
0001C8  47F0  3168       00028 |                 B     @1L8
0001CC                   00028 |        @1L7     DS    0H
                         00029 |       *             else
                         00030 |       *                convert(c_temp);
0001CC  6800  D0B0       00030 |                 LD    f0,c_temp(,r13,176)
0001D0  6000  D0C0       00030 |                 STD   f0,c_temp:convert(,r13,192)
0001D4  6820  6000       00036 |       +         LD    f2,+CONSTANT_AREA(,r6,0)
0001D8  2C02             00036 |       +         MDR   f0,f2
0001DA  6820  6008       00036 |       +         LD    f2,+CONSTANT_AREA(,r6,8)
0001DE  2A02             00036 |       +         ADR   f0,f2
0001E0  6000  D0C8       00036 |       +         STD   f0,f_temp:convert(,r13,200)
0001E4  6820  D0C0       00037 |       +         LD    f2,c_temp:convert(,r13,192)
0001E8  4100  5048       00037 |       +         LA    r0,""12(,r5,72)
0001EC  6020  D09C       00037 |       +         STD   f2,#MX_TEMP1(,r13,156)
```

Figure 17. Example of a C listing (Part 30 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)': main           04/14/2000 14:42:32   Page    35

OFFSET OBJECT CODE       LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

0001F0  6000  D0A4       00037 |       +         STD   f0,#MX_TEMP1(,r13,164)
0001F4  58F0  31A2       00037 |       +         L     r15,=V(PRINTF)(,r3,418)
0001F8  4110  D098       00037 |       +         LA    r1,#MX_TEMP1(,r13,152)
0001FC  5000  D098       00037 |       +         ST    r0,#MX_TEMP1(,r13,152)
000200  05EF             00037 |       +         BALR  r14,r15
000202                   00038 |       +@1L11    DS    0H
000202                   00038 |       +@1L8     DS    0H
000202  5800  D0B8       00038 |       +         L     r0,i(,r13,184)
000206  A70A  0001       00038 |       +         AHI   r0,H'1'
00020A  5000  D0B8       00038 |       +         ST    r0,i(,r13,184)
00020E  5810  D0D0       00038 |       +         L     r1,#SR_PARM_1(,r13,208)
000212  5810  1000       00038 |       +         L     r1,argc(,r1,0)
000216  1901             00038 |       +         CR    r0,r1
000218  4740  30C8       00038 |       +         BL    @1L5
00021C                   00038 |       +@1L9     DS    0H
00021C                   00038 |       +@1L6     DS    0H
00021C                   00038 |       +@1L4     DS    0H
                         00031 |       *         }
                         00032 |       *      }
                         00033 |       *  }
00021C  41F0  0000       00033 |                 LA    r15,0
000220                   00033 |        @1L13    DS    0H

000220                   Start of Epilog
000220  180D             00033 |                 LR    r0,r13
000222  58D0  D004       00033 |                 L     r13,4(,r13)
000226  58E0  D00C       00033 |                 L     r14,12(,r13)
00022A  9826  D01C       00033 |                 LM    r2,r6,28(r13)
00022E  051E             00033 |                 BALR  r1,r14
000230  0707             00033 |                 NOPR  7
000232  0000

000234                   Start of Literals
000234  00000000                                       =A(@STATIC)
000238  00000340                                       =A(@CONSTANT_AREA)
00023C  00000000                                       =V(PRINTF)
000240  00000000                                       =V(SCANF)
000244  00000000                                       =V(SSCANF)
000248                   End of Literals

                         ***   General purpose registers used: 1111111000001111
                         ***   Floating point  registers used: 1010101000000000
                         ***   Size of register spill area: 128(max) 0(used)
                         ***   Size of dynamic storage: 216
                         ***   Size of executable code: 466
```

Figure 17. Example of a C listing (Part 31 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)': convert          04/14/2000 14:42:32   Page    36

OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                          PPA1: Entry Point Constants
000248  1CCEA106                             =F'483303686'      Flags
00024C  000000C0                             =A(PPA2-convert)
000250  00000000                             =F'0'              No PPA3
000254  00000000                             =F'0'              No EPD
000258  FF800000                             =F'-8388608'       Register save mask
00025C  00000000                             =F'0'              Member flags
000260  90                                   =AL1(144)          Flags
000261  000000                               =AL3(0)            Callee's DSA use/8
000264  0040                                 =H'64'             Flags
000266  0012                                 =H'18'             Offset/2 to CDL
000268  00000000                             =F'0'              Reserved
00026C  50000051                             =F'1342177361'     CDL function length/2
000270  00000048                             =F'72'             CDL function EP offset
000274  38260000                             =F'942014464'      CDL prolog
000278  40080049                             =F'1074266185'     CDL epilog
00027C  00000000                             =F'0'              CDL end
000280  0007  ****                           AL2(7),C'convert'
                          PPA1 End

                          00001 |       *  #include <stdio.h>
                          00002 |       *
                          00003 |       *  #include "cbc3uaan.h"
                          00004 |       *
                          00005 |       *  void convert(double);
                          00006 |       *
                          00007 |       *  int main(int argc, char **argv)
                          00008 |       *  {
                          00009 |       *      double c_temp;
                          00010 |       *
                          00011 |       *      if (argc == 1) {  /* get Celsius value from stdin */
                          00012 |       *         int ch;
                          00013 |       *
                          00014 |       *         printf("Enter Celsius temperature: \n");
                          00015 |       *
                          00016 |       *         if (scanf("%f", :c_temp) != 1) {
                          00017 |       *            printf("You must enter a valid temperature\n");
                          00018 |       *         }
                          00019 |       *         else {
                          00020 |       *            convert(c_temp);
                          00021 |       *         }
                          00022 |       *      }
                          00023 |       *      else {  /* convert the command-line arguments to Fahrenheit */
                          00024 |       *         int i;
                          00025 |       *
                          00026 |       *         for (i = 1; i < argc; ++i) {
                          00027 |       *             if (sscanf(argv[i], "%f", :c_temp) != 1)
                          00028 |       *                printf("%s is not a valid temperature\n",argv[i]);
                          00029 |       *             else
                          00030 |       *                convert(c_temp);
                          00031 |       *         }
                          00032 |       *      }
                          00033 |       *  }
                          00034 |       *
                          00035 |       *  void convert(double c_temp) {
000290                    00035 |        convert  DS    0D
```

Figure 17. Example of a C listing (Part 32 of 35)

```
5647A01 V2 R10 OS/390 C                'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)': convert          04/14/2000 14:42:32   Page    37

OFFSET OBJECT CODE       LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

000290  47F0  F022       00035 |                 B     34(,r15)
000294  01C3C5C5                                       CEE eyecatcher
000298  000000C0                                       DSA size
00029C  FFFFFFB8                                       =A(PPA1-convert)
0002A0  47F0  F001       00035 |                 B     1(,r15)
0002A4  58F0  C31C       00035 |                 L     r15,796(,r12)
0002A8  184E             00035 |                 LR    r4,r14
0002AA  05EF             00035 |                 BALR  r14,r15
0002AC  00000000                                       =F'0'
0002B0  07F3             00035 |                 BR    r3
0002B2  90E6  D00C       00035 |                 STM   r14,r6,12(r13)
0002B6  58E0  D04C       00035 |                 L     r14,76(,r13)
0002BA  4100  E0C0       00035 |                 LA    r0,192(,r14)
0002BE  5500  C314       00035 |                 CL    r0,788(,r12)
0002C2  4130  F03A       00035 |                 LA    r3,58(,r15)
0002C6  4720  F014       00035 |                 BH    20(,r15)
0002CA  58F0  C280       00035 |                 L     r15,640(,r12)
0002CE  90F0  E048       00035 |                 STM   r15,r0,72(r14)
0002D2  9210  E000       00035 |                 MVI   0(r14),16
0002D6  50D0  E004       00035 |                 ST    r13,4(,r14)
0002DA  18DE             00035 |                 LR    r13,r14
0002DC                   End of Prolog

0002DC  5850  306A       00035 |                 L     r5,=A(@STATIC)(,r3,106)
0002E0  5860  306E       00035 |                 L     r6,=A(@CONSTANT_AREA)(,r3,110)
0002E4  5010  D0B8       00035 |                 ST    r1,#SR_PARM_2(,r13,184)
                         00036 |        *     double f_temp = (c_temp * CONV + OFFSET);
0002E8  5810  D0B8       00036 |                 L     r1,#SR_PARM_2(,r13,184)
0002EC  6800  1000       00036 |                 LD    f0,c_temp(,r1,0)
0002F0  6820  6000       00036 |                 LD    f2,+CONSTANT_AREA(,r6,0)
0002F4  2C02             00036 |                 MDR   f0,f2
0002F6  6820  6008       00036 |                 LD    f2,+CONSTANT_AREA(,r6,8)
0002FA  2A02             00036 |                 ADR   f0,f2
0002FC  6000  D0B0       00036 |                 STD   f0,f_temp(,r13,176)
                         00037 |        *     printf("%5.2f Celsius is %5.2f Fahrenheit\n",c_temp, f_temp);
000300  5810  D0B8       00037 |                 L     r1,#SR_PARM_2(,r13,184)
000304  6820  1000       00037 |                 LD    f2,c_temp(,r1,0)
000308  4100  5048       00037 |                 LA    r0,""12(,r5,72)
00030C  6020  D09C       00037 |                 STD   f2,#MX_TEMP2(,r13,156)
000310  6000  D0A4       00037 |                 STD   f0,#MX_TEMP2(,r13,164)
000314  58F0  3072       00037 |                 L     r15,=V(PRINTF)(,r3,114)
000318  4110  D098       00037 |                 LA    r1,#MX_TEMP2(,r13,152)
00031C  5000  D098       00037 |                 ST    r0,#MX_TEMP2(,r13,152)
000320  05EF             00037 |                 BALR  r14,r15
                         00038 |        *  }
000322                   00038 |         @2L14    DS    0H

000322                   Start of Epilog
000322  58D0  D004       00038 |                 L     r13,4(,r13)
000326  58E0  D00C       00038 |                 L     r14,12(,r13)
00032A  9826  D01C       00038 |                 LM    r2,r6,28(r13)
00032E  051E             00038 |                 BALR  r1,r14
000330  0707             00038 |                 NOPR  7
000332  0000

000334                   Start of Literals
```

Figure 17. Example of a C listing (Part 33 of 35)

```
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)': convert          04/14/2000 14:42:32   Page    38

OFFSET OBJECT CODE       LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

000334  00000000                                    =A(@STATIC)
000338  00000340                                    =A(@CONSTANT_AREA)
00033C  00000000                                    =V(PRINTF)
000340                   End of Literals

                         ***   General purpose registers used: 1101111000001111
                         ***   Floating point  registers used: 1010101000000000
                         ***   Size of register spill area: 128(max) 0(used)
                         ***   Size of dynamic storage: 192
                         ***   Size of executable code: 162

                         Constant Area
000340  411CCCCC CCCCCCCC 42200000 00000000    |................|

                         PPA2: Compile Unit Block
000350  0300  2202                                  =F'50340354'       Flags
000354  FFFF  FCB0                                  =A(CEESTART-PPA2)
000358  0000  0000                                  =F'0'              No PPA4
00035C  FFFF  FCBC                                  =A(TIMESTMP-PPA2)
000360  0000  0000                                  =F'0'              No primary
000364  0000  0000                                  =F'0'              Flags
                         PPA2 End
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                   04/14/2000 14:42:32   Page    39

                                  E X T E R N A L   S Y M B O L   D I C T I O N A R Y

          NAME        TYPE  ID  ADDR    LENGTH           NAME        TYPE  ID  ADDR    LENGTH

                      PC    1 000000   000368                        PC    2 000000   000090
          MAIN        LD    0 000060   000001           CONVERT      LD    0 000290   000001
          CEESG003    ER    3 000000                    PRINTF       ER    4 000000
          SCANF       ER    5 000000                    SSCANF       ER    6 000000
          CEESTART    ER    7 000000                    CEEMAIN      SD    8 000000   00000C
          EDCINPL     ER    9 000000                    MAIN         ER   10 000000
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                   04/14/2000 14:42:32   Page    40

                            E X T E R N A L   S Y M B O L   C R O S S   R E F E R E N C E

          ORIGINAL NAME                                 EXTERNAL SYMBOL NAME

          main                                          MAIN
          convert                                       CONVERT
          CEESG003                                      CEESG003
          printf                                        PRINTF
          scanf                                         SCANF
          sscanf                                        SSCANF
          CEESTART                                      CEESTART
          CEEMAIN                                       CEEMAIN
          EDCINPL                                       EDCINPL
5647A01 V2 R10 OS/390 C               'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'                   04/14/2000 14:42:32   Page    41

                               * * * * *   S T O R A G E   O F F S E T   L I S T I N G   * * * * *

IDENTIFIER          DEFINITION      ATTRIBUTES
                                    <SEQNBR>-<FILE NO>:<FILE LINE NO>

argc                7-0:7           Class = parameter,          Location = 0(r1),                             Length = 4

argv                7-0:7           Class = parameter,          Location = 4(r1),                             Length = 4

c_temp              9-0:9           Class = automatic,          Location = 176(r13),                          Length = 8

i                   24-0:24         Class = automatic,          Location = 184(r13),                          Length = 4

c_temp              35-0:35         Class = parameter,          Location = 0(r1),                             Length = 8

f_temp              36-0:36         Class = automatic,          Location = 176(r13),                          Length = 8
                         * * * * *   E N D   O F   S T O R A G E   O F F S E T   L I S T I N G   * * * * *
```

Figure 17. Example of a C listing (Part 34 of 35)

```
5647A01 V2 R10 OS/390 C              'TSCTEST.OSV2R10.SCBCSAM(CBC3UAAM)'              04/14/2000 14:42:32   Page    42
                                           * * * * *  S T A T I C    M A P  * * * * *
OFFSET (HEX)   LENGTH (HEX)   NAME

           0              3    ""3
           4              3    ""6
           8             1D    ""2
          28             1F    ""7
          48             23    ""12
          6C             24    ""4
                           * * * * *  E N D    O F    S T A T I C    M A P  * * * * *
                             * * * * *  E N D   O F   C O M P I L A T I O N  * * * * *
```

Figure 17. Example of a C listing (Part 35 of 35)

z/OS C Compiler Listing Components

The following sections describe the components of a C compiler listing. These are available for regular and IPA compilations. Differences in the IPA versions of the listings are noted. "Using the IPA Link Step Listing" on page 256 describes IPA-specific listings.

Heading Information

The first page of the listing is identified by the product number, the compiler version and release numbers, the name of the data set or HFS file containing the source code, the date and time compilation began (formatted according to the current locale), and the page number.

Note: If the name of the data set or HFS file that contains the source code is greater than 32 characters, it is truncated. Only the rightmost 32 characters appear in the listing.

Prolog Section

The Prolog section provides information about the compile-time library, file identifiers, compiler options, and other items in effect when the compiler was invoked.

All options except those with no default (for example, `DEFINE`) are shown in the listing. Any problems with the compiler options appear after the body of the Prolog section.

IPA Considerations: If you specify IPA suboptions that are irrelevant to the IPA Compile step, the Prolog does not display them. If IPA processing is not active, IPA suboptions do not appear in the Prolog.

The following sections describe the optional parts of the listing and the compiler options that generate them.

Source Program

If you specify the `SOURCE` option, the listing file includes input to the compiler.

Note: If you specify the `SHOWINC` option, the source listing shows the included text after the `#include` directives.

Includes Section

The compiler generates the Includes Section when you use *include* files, and specify the options `SOURCE`, `LIST`, or `INLRPT`.

Cross-Reference Listing

The XREF option generates a cross-reference table that contains a list of the identifiers from the source program and the line numbers in which they appear.

Structure and Union Maps

You obtain structure and union maps by using the AGGREGATE option. The table shows how each structure and union in the program is mapped. It contains the following:

- Name of the structure or union and the elements within the structure or union
- Byte offset of each element from the beginning of the structure or union, and the bit offset for unaligned bit data.
- Length of each element
- Total length of each structure, union, and substructure.

Messages

If the preprocessor or the compiler detects an error, or the possibility of an error, it generates messages. If you specify the SOURCE compiler option, preprocessor error messages appear immediately after the source statement in error. You can generate your own messages in the preprocessing stage by using the #error preprocessor directive. For information on #error, see the *z/OS C/C++ Language Reference*.

If you specify the compiler options CHECKOUT or INFO(), the compiler will generate informational diagnostic messages.

For more information on the compiler messages, see "FLAG | NOFLAG" on page 107 , and "Appendix G. z/OS C/C++ Compiler Return Codes and Messages" on page 591.

Message Summary

This listing section displays the total number of messages and the number of messages for each severity level.

Inline Report

If you specify the OPTIMIZE and INLINE(,REPORT,,) options, or the OPTIMIZE and INLRPT options, an Inline Report is included in the listing. This report contains an inline summary and a detailed call structure.

Note: No report is produced when your source file contains only one defined subprogram.

The summary contains information such as:

- Name of each defined subprogram. Subprogram names appear in alphabetical order.
- Reason for action on a subprogram:
  - A #pragma noinline was specified for the subprogram.
  - A #pragmainline was specified for the subprogram.
  - Auto-inlining acted on the subprogram.
  - There was no reason to inline the subprogram.
- Action on a subprogram:
  - Subprogram was inlined at least once.
  - Subprogram was not inlined because of initial size constraints.
  - Subprogram was not inlined because of expansion beyond size constraint.
  - Subprogram was a candidate for inlining, but was not inlined.
  - Subprogram was a candidate for inlining, but was not referenced.
  - The subprogram is directly recursive, or some calls have mismatching parameters.

- Status of original subprogram after inlining:
  – Subprogram is discarded because it is no longer referenced and is defined as `static` internal.
  – Subprogram was not discarded for various reasons :
    - Subprogram is external. (It can be called from outside the compilation unit.)
    - Some call to this subprogram remains.
    - Subprogram has its address taken.
- Initial relative size of subprogram (in Abstract Code Units (ACU)).
- Final relative size of subprogram (in ACUs) after inlining.
- Number of calls within the subprogram and the number of these calls that were inlined into subprogram.
- Number of times the subprogram is called by others in the compile unit and the number of times the subprogram was inlined.
- Mode that is selected and the value of *threshold* and *limit* specified for the compilation.

The detailed call structure contains specific information of each subprogram such as:
- Subprograms that it calls
- Subprograms that call it
- Subprograms in which it is inlined.

The information can help you to better analyze your program if you want to use the inliner in selective mode.

Inlining may result in additional messages. For example, if inlining a subprogram with automatic storage increases the automatic storage of the subprogram it is being inlined into by more than 4K, a message is generated.

Pseudo Assembly Listing

The option `LIST` generates a listing of the machine instructions in the object module in a form similar to assembler language.

This Pseudo Assembly listing displays the source statement line numbers and the line number of inlined code to aid you in debugging inlined code.

External Symbol Dictionary

The `LIST` compiler option generates the External Symbol Dictionary. The External Symbol Dictionary lists the names that the compiler generates for the output object module. It includes address information and size information about each symbol.

External Symbol Cross Reference Listing

The `XREF` compiler option generates the External Symbol Cross Reference section. It shows the original name and corresponding mangled name for each symbol.

Storage Offset Listing

If you specify the `XREF` option, the listing file includes offset information on identifiers.

Static Map

Static Map displays the contents of the @STATIC data area, which holds the file scope read/write static variables. It displays the offset (as a hexadecimal number), the length (as a hexadecimal number), and the names of the objects mapped to @STATIC. Under certain circumstances, the compiler might decide to map other objects to @STATIC. In the example of the listing, the unnamed string ″Enter

Celsius temperature: \n″ is stored in the @STATIC area at offset 48 and its length is 23 (both numbers are in hexadecimal notation), under the name ″″12.

If you specify the XREF, IPA (ATTRIBUTE) or IPA (XREF) options, the listing file includes offset information for file scope read/write static variables.

Using the z/OS C++ Compiler Listing

If you select the SOURCE, INLRPT, or LIST option, the compiler creates a listing that contains information about the source program and the compilation. If the compilation terminates before reaching a particular stage of processing, the compiler does not generate corresponding parts of the listing. The listing contains standard information that always appears, together with optional information that is supplied by default or specified through compiler options.

In an interactive environment you can also use the TERMINAL option to direct all compiler diagnostic messages to your terminal. The TERMINAL option directs only the diagnostic messages part of the compiler listing to your terminal.

Note: Although the compiler listing is for your use, it is not a programming interface and is subject to change.

IPA Considerations

The listings that the IPA Compile step produces are basically the same as those that a regular compilation produces. Any differences are noted throughout this section.

The IPA Link step listing has a separate format from the listings mentioned above. Many listing sections are similar to those that are produced by a regular compilation or the IPA Compile step with the IPA(OBJECT) option specified. Refer to "Using the IPA Link Step Listing" on page 256 for information about IPA Link step listings.

Example of a C++ Compiler Listing

Figure 18 on page 245 shows an example of a z/OS C++ compiler listing. Vertical ellipses indicate sections that have been truncated.

```
5647A01 V2 R10 OS/390 C++               'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                 03/14/2000 11:09:52
                                       * * * * *   P R O L O G   * * * * *
    Compiler options. . . . . . . :  ANSIALIAS       ARGPARSE        NOCOMPACT       NOCOMPRESS      CVFT            NODIGRAPH
                                  :  NOEVENTS        EXECOPS         EXH             NOEXPMAC        NOEXPORTALL     NOFASTTEMPINC
                                  :  NOGOFF          NOGONUMBER      NOIGNERRNO      INLRPT          NOLIBANSI       LIST
                                  :  LONGNAME        NOMARGINS       MEMORY          OBJECT          NOOFFSET        REDIR
                                  :  NOROCONST       NOROSTRING      NOSEQUENCE      NOSHOWINC       SOURCE          NOSRCMSG
                                  :  START           TERMINAL        NOWSIZEOF
                                  :  XREF            NOATTRIBUTE     NOSTRICT_INDUCTION
                                  :  ARCH(2)         FLAG(I)         HALT(16)        MAXMEM(2097152) NESTINC(255)    OPTIMIZE(1)
                                  :  PLIST(HOST)     SPILL(128)      TUNE(3)
                                  :  AGGRCOPY(NOOVERLAP)
                                  :  NOCSECT
                                  :  DLL(NOCALLBACKANY)
                                  :  FLOAT(HEX,FOLD,NOAFP) STRICT
                                  :  NOGENPCH
                                  :  NOINFO
                                  :  INITAUTO(00)
                                  :  NOIPA
                                  :  LANGLVL(EXTENDED) NOTEST(HOOK)
                                  :  NOLOCALE
                                  :  NOOE
                                  :  NOPORT
                                  :  NOPPONLY
                                  :  NOSERVICE
                                  :  TARGET(LE,CURRENT)
                                  :  TEMPINC
                                  :  OPTFILE(DD:OPTS)
                                  :  SEARCH(//'CEE.SCEEH.+',//'CBC.SCLBH.+')
                                  :  NOUSEPCH
                                  :  NOXPLINK(NOBACKCHAIN,NOSTOREARGS,GUARD,OSCALL(UPSTACK))
    Version Macros. . . . . . . . :  __COMPILER_VER__=0x220A0000 __LIBREL__=0x220A0000 __TARGET_LIB__=0x220A0000
5647A01 V2 R10 OS/390 C++               'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                 03/14/2000 11:09:52
                                         * * * * *   S O U R C E   * * * * *
         1 |//
         2 |// Sample Program:  Biorhythm
         3 |// Description   :  Calculates biorhythm based on the current
         4 |//                  system date and birth date entered
         5 |//
         6 |// File 2 of 2-other file is CBC3UBRH
         7 |
         8 |#include <stdio.h>
         9 |#include <string.h>
        10 |#include <math.h>
        11 |#include <time.h>
        12 |#include <iostream.h>
        13 |#include <iomanip.h>
        14 |
        15 |#include "cbc3ubrh.h"  //BioRhythm class and Date class
        16 |
        17 |int main(void) {
        18 |  BioRhythm bio;
        19 |  int code;
        20 |
        21 |  if (!bio.ok()) {
        22 |    cerr << "Error in birthdate specification - format is yyyy/mm/dd";
        23 |    code = 8;
        24 |  }
        25 |  else {
        26 |    cout << bio;  // write out birthdate for bio
        27 |    code = 0;
        28 |  }
        29 |  return(code);
        30 |}
        31 |
        32 |static ostream& operator<<(ostream& os, BioRhythm& bio) {
        33 |  os << "Total Days  :  " << bio.AgeInDays() << "\n";
        34 |  os << "Physical    :  " << bio.Physical() << "\n";
        35 |  os << "Emotional   :  " << bio.Emotional() << "\n";
        36 |  os << "Intellectual:  " << bio.Intellectual() << "\n";
        37 |
        38 |return(os);
        39 |}
```

Figure 18. Example of a C++ Compiler Listing (Part 1 of 9)

```

Date::Date() {
  time_t lTime;
  struct tm *newTime;

  time(&lTime);
  newTime = localtime(&lTime);
  cout << "local time is %s \n",asctime(newTime);

  curYear = newTime->tm_year + 1900;
  curDay  = newTime->tm_yday + 1;
}

BirthDate::BirthDate(const char *birthText) {
  strcpy(text, birthText);
}

BirthDate::BirthDate() {
  cout << "Please enter your birthdate in the form yyyy/mm/dd\n";
  cin << setw(dateLen+1) << text;
}

Date::DaysSince(const char *text) {

  int year, month, day, totDays, delim;
  int daysInYear = 0;
  int i;
  int leap = 0;

  int rc = sscanf(text, "%4d%c%2d%c%2d",
                &year, &delim, &month, &delim, &day);
  --month;
  if (rc != 5 || year  < 0 || year  > 9999 ||
               month < 0 || month >   11 ||
               day   < 1 || day   >   31 ||
              (day   > numDays[month]&& month != 1)) {
    return(-1);
  }
  if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
     leap = 1;

  if (month == 1 && day < numDays[month]) {
    if (day < 29)
      return(-1);
    else if (!leap)
      return (-1);
  }

  for (i=0;i<month;++i) {
    daysInYear += numDays[i];
  }
  daysInYear += day;

  // correct for leap year
  if (leap == 1 &&
     (month < 1 || (month == 1 && day == 29)))
     ++daysInYear;

  totDays = (curDay - daysInYear) + (curYear - year)*365;

  // now, correct for leap year
  for (i=year+1; i < curYear; ++i) {
    if ((i % 4 == 0 && i % 100 != 0) || i % 400 == 0) {
      ++totDays;
    }
  }
  return(totDays);
}                                   * * * * *   E N D   O F   S O U R C E   * * * * *
```

Figure 18. Example of a C++ Compiler Listing (Part 2 of 9)

```
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                   03/14/2000 11:09:52
                                * * * * *  C R O S S   R E F E R E N C E   L I S T I N G  * * * * *
   ___valist :
            1:121 (D)    1:124 (R)    1:308 (R)    1:309 (R)    1:310 (R)
   __amrc_type :
            1:614 (D)    1:618 (R)
   __amrc2_type :
            1:627 (D)    1:631 (R)
   __device_t :
            1:424 (D)    1:478 (R)
   __fabs :
            5:177 (R)

:
                         * * * * *  E N D   O F   C R O S S   R E F E R E N C E   L I S T I N G  * * * * *
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                   03/14/2000 11:09:52
                        * * * * *  I N C L U D E S  * * * * *
    1 = TSCTEST.CEE210.SCEEH.H(STDIO)
    2 = TSCTEST.CEE210.SCEEH.H(FEATURES)
    3 = TSCTEST.CEE210.SCEEH.SYS.H(TYPES)
    4 = TSCTEST.CEE210.SCEEH.H(STRING)
    5 = TSCTEST.CEE210.SCEEH.H(MATH)
    6 = TSCTEST.CEE210.SCEEH.H(TIME)
    7 = TSCTEST.OSV2R10.SCLBH.H(IOSTREAM)
    8 = TSCTEST.CEE210.SCEEH.H(MEMORY)
    9 = TSCTEST.CEE210.SCEEH.H(WCHAR)
   10 = TSCTEST.OSV2R10.SCLBH.H(IOMANIP)
   11 = TSCTEST.OSV2R10.SCLBH.H(IOSTREAM)
   12 = TSCTEST.OSV2R10.SCLBH.H(GENERIC)
   13 = TSCTEST.OSV2R10.SCBCSAM(CBC3UBRH)
                        * * * * *  E N D   O F   I N C L U D E S  * * * * *

5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                   03/14/2000 11:09:52
                            * * * * *  M E S S A G E   S U M M A R Y  * * * * *

 TOTAL   UNRECOVERABLE  SEVERE       ERROR     WARNING    INFORMATIONAL
               (U)       (S)         (E)        (W)          (I)
    0           0         0           0          0            0

                        * * * * *  E N D   O F   M E S S A G E   S U M M A R Y  * * * * *
```

Figure 18. Example of a C++ Compiler Listing (Part 3 of 9)

```
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                 03/14/2000 11:09:52

                     Inline Report (Summary)

  Reason:    P : #pragma noinline was specified for this routine
             F : #pragma inline was specified for this routine
             A : Automatic inlining
             - : No reason
  Action:    I : Routine is inlined at least once
             L : Routine is initially too large to be inlined
             T : Routine expands too large to be inlined
             C : Candidate for inlining but not inlined
             N : No direct calls to routine are found in file (no action)
             U : Some calls not inlined due to recursion or parameter mismatch
             - : No action
  Status:    D : Internal routine is discarded
             R : A direct call remains to internal routine (cannot discard)
             A : Routine has its address taken (cannot discard)
             E : External routine (cannot discard)
             - : Status unchanged
  Calls/I      : Number of calls to defined routines / Number inline
  Called/I     : Number of times called / Number of times inlined

  Reason  Action  Status   Size (init)    Calls/I   Called/I     Name

     A      I       E      34              0         2/2        Date::Date()
     F      I       D      34    (17)      2/1       1/1        BioRhythm::BioRhythm()
     A      T       -      107   (31)      2/2       1/0        BirthDate::BirthDate()
     A      N       E      55    (16)      1/1       0          BirthDate::BirthDate(const char*)
     F      N       A      37    (9)       1/1       0          BioRhythm::__dftdt()
     F      I       D      21              0         2/2        BioRhythm::~BioRhythm()
     A      T       -      182   (62)      3/3       1/0        operator<<(ostream&,BioRhythm&)
     F      I       E      30              0         1/1        operator>>(istream&,const smanip_int&)
     P      -       -      111   (44)      3/2       0          main
     F      I       D      18              0         3/3        BioRhythm::Cycle(int)
     F      I       D      12              1/0       1/1        BirthDate::DaysOld()
     A      L       -      274             0         1/0        Date::DaysSince(const char*)
     F      I       D      35    (10)      1/1       1/1        BioRhythm::Emotional()
     F      I       D      35    (10)      1/1       1/1        BioRhythm::Intellectual()
     F      I       D      35    (10)      1/1       1/1        BioRhythm::Physical()

  Mode = AUTO    Inlining Threshold = 100    Expansion Limit = 2000

5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                 03/14/2000 11:09:52

                     Inline Report (Call Structure)

  Defined Function      : Date::Date()
      Calls To          : 0
   Called From(2,2)     : BirthDate::BirthDate()(1,1) BirthDate::BirthDate(const char*)(1,1)

  Defined Function      : BioRhythm::BioRhythm()
      Calls To(2,1)     : BirthDate::BirthDate()(1,0) BirthDate::DaysOld()(1,1)
   Called From(1,1)     : main(1,1)

  Defined Function      : BirthDate::BirthDate()
      Calls To(2,2)     : Date::Date()(1,1) operator>>(istream&,const smanip_int&)(1,1)
   Called From(1,0)     : BioRhythm::BioRhythm()(1,0)

  Defined Function      : BirthDate::BirthDate(const char*)
      Calls To(1,1)     : Date::Date()(1,1)
   Called From          : 0
⋮
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                 03/14/2000 11:09:52

                     Inline Report (Additional Information)

 INFORMATIONAL CBC5052: Function  is (or grows) too large to be inlined. BirthDate::BirthDate()

 INFORMATIONAL CBC5052: Function  is (or grows) too large to be inlined. operator<<(ostream&,BioRhythm&

 INFORMATIONAL CBC5052: Function  is (or grows) too large to be inlined. Date::DaysSince(const char*)
```

Figure 18. Example of a C++ Compiler Listing (Part 4 of 9)

```
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                  03/14/2000 11:09:52           6

 OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

 000000  41F0  F468                                LA    r15,1128(,r15)
 000004  07FF                                      BR    r15
 000006  0700                                      NOPR  0
 000008  00000000                                        =F'0'

                           Timestamp and Version Information
 00000C  F2F0  F0F0                                      =C'2000'           Compiled Year
 000010  F0F3  F1F4                                      =C'0314'           Compiled Date MMDD
 000014  F1F1  F0F9  F5F2                                =C'110952'         Compiled Time HHMMSS
 00001A  F0F2  F0C1  F0F0                                =C'020A00'         Compiler Version
                           Timestamp and Version End


5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)': operator>>(is...) 03/14/2000 11:09:52           7

 OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                           PPA1: Entry Point Constants
 000020  1CCEA109                                        =F'483303689'      Flags
 000024  00000F08                                        =A(PPA2-operator>>(istream&,const smanip_int&))
 000028  00000000                                        =F'0'              No PPA3
 00002C  00000000                                        =F'0'              No EPD
 000030  FE000000                                        =F'-33554432'      Register save mask
 000034  00000001                                        =F'1'              Member flags
 000038  E0                                              =AL1(224)          Flags
 000039  000001                                          =AL3(1)            Callee's DSA use/8
 00003C  0040                                            =H'64'             Flags
 00003E  0012                                            =H'18'             Offset/2 to CDL
 000040  00000000                                        =F'0'              State variable location
 000044  00000000                                        =F'0'              CDL function length/2
 000048  00000000                                        =F'0'              CDL function EP offset
 00004C  18260000                                        =F'405143552'      CDL prolog
 000050  00000000                                        =F'0'              CDL epilog
 000054  00000000                                        =F'0'              CDL end
 000058  0026  ****                                      AL2(38),C'operator>>(istream&,const smanip_int&)'
                           PPA1 End

                                          operator>>(istream&,const smanip_int
                                          &)
 000080                    00138 |   10            DS    0D
 000080  47F0  F001        00138 |   10            B     1(,r15)
 000084  01C3C5C5                                        CEE eyecatcher
 000088  000000C8                                        DSA size
 00008C  FFFFFFA0                                        =A(PPA1-operator>>(istream&,const smanip_int&))
 000090  90E4  D00C        00138 |   10            STM   r14,r4,12(r13)
 000094  58E0  D04C        00138 |   10            L     r14,76(,r13)
 000098  4100  E0C8        00138 |   10            LA    r0,200(,r14)
 00009C  5500  C314        00138 |   10            CL    r0,788(,r12)
 0000A0  4140  F04C        00138 |   10            LA    r4,76(,r15)
 0000A4  47D0  F03A        00138 |   10            BNH   58(,r15)
 0000A8  58F0  C31C        00138 |   10            L     r15,796(,r12)
 0000AC  184E              00138 |   10            LR    r4,r14
 0000AE  05EF              00138 |   10            BALR  r14,r15
 0000B0  00000008                                        =F'8'
 0000B4  0540              00138 |   10            BALR  r4,0
 0000B6  4140  4016        00138 |   10            LA    r4,22(,r4)
 0000BA  5000  E04C        00138 |   10            ST    r0,76(,r14)
 0000BE  9210  E000        00138 |   10            MVI   0(r14),16
 0000C2  50D0  E004        00138 |   10            ST    r13,4(,r14)
 0000C6  5800  D014        00138 |   10            L     r0,20(,r13)
 0000CA  18DE              00138 |   10            LR    r13,r14
 0000CC                    End of Prolog
⋮
                           ***   General purpose registers used: 1111100000001111
                           ***   Floating point  registers used: 1010101000000000
                           ***   Size of register spill area: 128(max) 0(used)
                           ***   Size of dynamic storage: 200
                           ***   Size of executable code: 128
⋮
```

Figure 18. Example of a C++ Compiler Listing (Part 5 of 9)

```
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'              03/14/2000 11:09:52          37
                                   E X T E R N A L   S Y M B O L   D I C T I O N A R Y

                       TYPE  ID  ADDR    LENGTH            NAME

                        SD    1 000000   000FA0            @STATICP
                        PR    3 000000   000004            dateLen__4Date
                        PR    4 000000   000004            numMonths__4Date
                        PR    5 000000   000030            numDays__4Date
                        PR    6 000000   000004            pCycle__9BioRhythm
                        PR    7 000000   000004            eCycle__9BioRhythm
                        PR    8 000000   000004            iCycle__9BioRhythm
                        SD    9 000000   000008            @@DLLI
                        LD    0 000080   000001            __rs__FR7istreamRC10smanip_int
                        LD    0 000468   000001            main
                        LD    0 0005B0   000001            __ct__4DateFv
                        LD    0 0006F0   000001            DaysSince__4DateFPCc
                        LD    0 0009A0   000001            __ct__9BirthDateFv
                        LD    0 000B60   000001            __ct__9BirthDateFPCc
                        ER   10 000000                     CEESG003
                        ER   11 000000                     CBCSG003
                        UR   14 000000                     time
                        UR   16 000000                     sscanf
                        UR   17 000000                     __rs__7istreamFPc
                        UR   18 000000                     setw__Fi
                        UR   19 000000                     asctime
                        UR   20 000000                     cin
                        ER   21 000000                     @@TRGLOR
                        UR   22 000000                     localtime
                        UR   23 000000                     cout
                        UR   24 000000                     __ls__7ostreamFd
                        UR   25 000000                     CEETDSIN
                        UR   26 000000                     fmod
                        UR   27 000000                     __ls__7ostreamFi
                        UR   28 000000                     __dl__FPv
                        UR   29 000000                     __ls__7ostreamFPCc
                        UR   30 000000                     cerr
                        ER   31 000000                     CEESTART
                        SD   32 000000   000008            @@PPA2
                        SD   33 000000   00000C            CEEMAIN
                        ER   34 000000                     EDCINPL
```

Figure 18. Example of a C++ Compiler Listing (Part 6 of 9)

```
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'              03/14/2000 11:09:52          38
                             E X T E R N A L   S Y M B O L   C R O S S   R E F E R E N C E

           ORIGINAL NAME                                EXTERNAL SYMBOL NAME

           @STATICP                                     @STATICP
           Date::dateLen                                dateLen__4Date
           Date::numMonths                              numMonths__4Date
           Date::numDays                                numDays__4Date
           BioRhythm::pCycle                            pCycle__9BioRhythm
           BioRhythm::eCycle                            eCycle__9BioRhythm
           BioRhythm::iCycle                            iCycle__9BioRhythm
           @@DLLI                                       @@DLLI
           operator>>(istream&                          __rs__FR7istreamRC10smanip_int
           ,const smanip_int&)
           main                                         main
           Date::Date()                                 __ct__4DateFv
           Date::DaysSince(const char*)                 DaysSince__4DateFPCc
           BirthDate::BirthDate()                       __ct__9BirthDateFv
           BirthDate::BirthDate(const char*)            __ct__9BirthDateFPCc
           CEESG003                                     CEESG003
           CBCSG003                                     CBCSG003
           time                                         time
           sscanf                                       sscanf
           istream::operator>>(char*)                   __rs__7istreamFPc
           setw(int)                                    setw__Fi
           asctime                                      asctime
           cin                                          cin
           @@TRGLOR                                     @@TRGLOR
           localtime                                    localtime
           cout                                         cout
           ostream::operator<<(double)                  __ls__7ostreamFd
           __sin                                        CEETDSIN
           fmod                                         fmod
           ostream::operator<<(int)                     __ls__7ostreamFi
           operator delete(void*)                       __dl__FPv
           ostream::operator<<(const char*)             __ls__7ostreamFPCc
           cerr                                         cerr
           CEESTART                                     CEESTART
           @@PPA2                                       @@PPA2
           CEEMAIN                                      CEEMAIN
           EDCINPL                                      EDCINPL
```

Figure 18. Example of a C++ Compiler Listing (Part 7 of 9)

```
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                 03/14/2000 11:09:52          39
                                * * * * *   S T O R A G E   O F F S E T   L I S T I N G   * * * * *
 IDENTIFIER          DEFINITION      ATTRIBUTES
                                     <SEQNBR>-<FILE NO>:<FILE LINE NO>

 bio                 18-0:18         Class = automatic,           Location = 176(r13),                        Length = 24

 code                19-0:19         Class = automatic,           Location = ,                                Length = 4

 os                  32-0:32         Class = parameter,           Location = 188(r1),                         Length = 4

 bio                 32-0:32         Class = parameter,           Location = 192(r1),                         Length = 4

 lTime               42-0:42         Class = automatic,           Location = 204(r13),                        Length = 4

 newTime             43-0:43         Class = automatic,           Location = ,                                Length = 4

 birthText           53-0:53         Class = parameter,           Location = 192(r1),                         Length = 4

 text                62-0:62         Class = parameter,           Location = 192(r1),                         Length = 4

 year                64-0:64         Class = automatic,           Location = 244(r13),                        Length = 4

 month               64-0:64         Class = automatic,           Location = 240(r13),                        Length = 4

 day                 64-0:64         Class = automatic,           Location = 232(r13),                        Length = 4

 totDays             64-0:64         Class = automatic,           Location = ,                                Length = 4

 delim               64-0:64         Class = automatic,           Location = 236(r13),                        Length = 4

 daysInYear          65-0:65         Class = automatic,           Location = ,                                Length = 4

 i                   66-0:66         Class = automatic,           Location = ,                                Length = 4

 leap                67-0:67         Class = automatic,           Location = ,                                Length = 4

 rc                  69-0:69         Class = automatic,           Location = ,                                Length = 4

 i                   138-10:138      Class = parameter,           Location = 188(r1),                         Length = 4

 m                   138-10:138      Class = parameter,           Location = 192(r1),                         Length = 4

 s                   138-10:138      Class = automatic,           Location = ,                                Length = 4

 __dtorFlags         43-13:43        Class = parameter,           Location = 0(r1),                           Length = 4

 phase               63-13:63        Class = parameter,           Location = 0(r1),                           Length = 4

 cin                 1020-7:1020     Class = external reference,  Location = WSA + Q(<cin>),                  Length = 76

 cout                1021-7:1021     Class = external reference,  Location = WSA + Q(<cout>),                 Length = 72

 cerr                1022-7:1022     Class = external reference,  Location = WSA + Q(<corr>),                 Length = 72

 dateLen__4Date      18-13:18        Class = external definition, Location = WSA + Q(dateLen__4Date),         Length = 4

 numMonths__4Date    19-13:19        Class = external definition, Location = WSA + Q(numMonths__4Date),       Length = 4
5647A01 V2 R10 OS/390 C++              'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'                 03/14/2000 11:09:52          40
 IDENTIFIER          DEFINITION      ATTRIBUTES
                                     <SEQNBR>-<FILE NO>:<FILE LINE NO>


 numDays__4Date      20-13:20        Class = external definition, Location = WSA + Q(numDays__4Date),         Length = 48

 pCycle__9BioRhythm  72-13:72        Class = external definition, Location = WSA + Q(pCycle__9BioRhythm),     Length = 4

 eCycle__9BioRhythm  73-13:73        Class = external definition, Location = WSA + Q(eCycle__9BioRhythm),     Length = 4

 iCycle__9BioRhythm  74-13:74        Class = external definition, Location = WSA + Q(iCycle__9BioRhythm),     Length = 4
                          * * * * *   E N D   O F   S T O R A G E   O F F S E T   L I S T I N G   * * * * *
```

Figure 18. Example of a C++ Compiler Listing (Part 8 of 9)

```
5647A01 V2 R10 OS/390 C++             'TSCTEST.OSV2R10.SCBCSAM(CBC3UBRC)'              03/14/2000 11:09:52          41
                                                * * * * *   S T A T I C     M A P   * * * * *
 OFFSET (HEX)   LENGTH (HEX)   NAME

         0             2    ""7
         4             2    ""9
         8             2    ""11
         C             2    ""13
        10             4    cerr
        14             4    __ls__7ostreamFPCc
        18             4    cout
        1C             4    __dl__FPv
        20             4    __ls__7ostreamFi
        24             4    fmod
        28             4    __sin
        2C             4    __ls__7ostreamFd
        30             4    time
        34             4    localtime
        38             4    asctime
        3C             4    cin
        40             4    setw__Fi
        44             4    __rs__7istreamFPc
        48             4    sscanf
        4C             4    numDays__4Date
        50             E    ""18
        60             F    ""6
        70             F    ""8
        80             F    ""10
        90             F    ""12
        A0            13    ""14
        B8            1C    __fsm_tab
        D4            34    ""16
       108            38    ""3

                             * * * * *   E N D     O F     S T A T I C     M A P   * * * * *

                                * * * * *   E N D   O F   C O M P I L A T I O N   * * * * *
```

Figure 18. Example of a C++ Compiler Listing (Part 9 of 9)

z/OS C++ Compiler Listing Components

The following sections describe the components of a C++ compiler listing.These are available for regular and IPA compilations. Differences in the IPA versions of the listings are noted. “Using the IPA Link Step Listing” on page 256 describes IPA-specific listings.

Heading Information

The first page of the listing is identified by the product number, the compiler version and release numbers, the name of the data set or HFS file containing the source code, the date and time compilation began (formatted according to the current locale), and the page number.

Note: If the name of the data set or HFS file that contains the source code is greater than 32 characters, it is truncated. Only the rightmost 32 characters appear in the listing.

Prolog Section

The Prolog section provides information about the compile-time library, file identifiers, compiler options, and other items in effect when the compiler was invoked.

All options except those with no default (for example, `DEFINE`) are shown in the listing. Any problems with the compiler options appear after the body of the Prolog section.

IPA Considerations: If you specify IPA suboptions that are irrelevant to the IPA Compile step, the Prolog does not display them. If IPA processing is not active, IPA suboptions do not appear in the Prolog.

The following sections describe the optional parts of the listing and the compiler options that generate them.

Source Program

If you specify the SOURCE option, the listing file includes input to the compiler.

Note: If you specify the SHOWINC option, the source listing shows the included text after the #include directives.

Cross-Reference Listing

The option XREF generates a cross-reference table that contains a list of the identifiers from the source program. The table also displays a list of reference, modification, and definition information for each identifier.

The option ATTR generates a cross-reference table that contains a list of the identifiers from the source program, with a list of attributes for each identifier.

If you specify both ATTR and XREF, the cross-reference listing is a composite of the two forms. It contains the list of identifiers, as well as the attribute and reference, modification, and definition information for each identifier. The list is in the form:

```
identifier : attribute
       n:m (x)
```

where:

n	corresponds to the file number from the INCLUDE LIST. If the identifier is from the main program, n is 0.
m	corresponds to the line number in the file *n*.
x	is the cross reference code. It takes one of the following values: R - referenced D - defined M - modified

together with the line numbers in which they appear.

Includes Section

The compiler generates the Includes Section when you use *include* files, and specify the options SOURCE, LIST, or INLRPT.

Messages

If the preprocessor or the compiler detects an error, or the possibility of an error, it generates messages. If you specify the SOURCE compiler option, preprocessor error messages appear immediately after the source statement in error. You can generate your own messages in the preprocessing stage by using #error. For information on #error, see the *z/OS C/C++ Language Reference*.

If you specify the compiler options FLAG(I), CHECKOUT or INFO(), the compiler will generate informational diagnostic messages.

For a description of compiler messages, see “Appendix G. z/OS C/C++ Compiler Return Codes and Messages” on page 591.

Message Summary

This listing section displays the total number of messages and the number of messages for each severity level.

Inline Report

If the `OPTIMIZE` and `INLRPT` options are specified, an Inline Report will be included in the listing. This report contains an inline summary and a detailed call structure.

Note: No report is produced when your source file contains only one defined subprogram.

The summary contains information such as:

- Name of each defined subprogram. Subprogram names appear in alphabetical order.
- Reason for action on a subprogram:
  - A `#pragma noinline` was specified for that subprogram. The `P` indicates that inlining could not be performed.
  - `inline` was specified for that subprogram. For z/OS C++, this is a result of the inline specifier. For C, this a result of the `#pragma inline`. The `F` indicates that the subprogram was declared inline.
  - Auto-inlining acted on that subprogram.
  - There was no reason to inline the subprogram.
- Action on a subprogram:
  - Subprogram was inlined at least once.
  - Subprogram was not inlined because of initial size constraints.
  - Subprogram was not inlined because of expansion beyond size constraint.
  - Subprogram was a candidate for inlining, but was not inlined.
  - Subprogram was a candidate for inlining, but was not referenced.
  - This subprogram is directly recursive, or some calls have mismatching parameters.
- Status of original subprogram after inlining:
  - Subprogram is discarded because it is no longer referenced and is defined as `static` internal.
  - Subprogram was not discarded for various reasons :
    - Subprogram is external. (It can be called from outside the compilation unit.)
    - Some call to this subprogram remains.
    - Subprogram has its address taken.
- Initial relative size of subprogram (in Abstract Code Units (ACU)).
- Final relative size of subprogram (in ACUs) after inlining.
- Number of calls within the subprogram and the number of these calls that were inlined into the subprogram.
- Number of times the subprogram is called by others in the compile unit and the number of times this subprogram was inlined.
- Mode that is selected and the value of *threshold* and *limit* specified for this compilation.

The detailed call structure contains specific information of each subprogram such as:

- What subprograms it calls
- What subprograms call it
- In which subprograms it is inlined.

The information can help you to better analyze your program if you want to use the inliner in selective mode.

There may be additional messages as a result of the inlining. For example, if inlining a subprogram with automatic storage would increases the automatic storage of the subprogram it is being inlined into by more than 4K, a message is emitted.

Pseudo Assembly Listing

The option `LIST` generates a listing of the machine instructions in the object module in a form similar to assembler language.

This Pseudo Assembly listing displays the source statement line numbers and the line number of any inlined code to aid you in debugging inlined code.

External Symbol Dictionary

The `LIST` compiler option generates the External Symbol Dictionary. The External Symbol Dictionary lists the names that the compiler generates for the output object module. It includes address information and size information about each symbol.

External Symbol Cross Reference Listing

The `ATTR` or `XREF` compiler options generate the External Symbol Cross Reference section. It shows the original name and corresponding mangled name for each symbol. For additional information on mangled names, see "Chapter 17. Filter Utility" on page 425.

Storage Offset Listing

If you specify the `ATTR` or `XREF` option, the listing file includes offset information on identifiers.

Static Map

Static Map displays the contents of the @STATIC data area, which holds the file scope read/write static variables. It displays the offset (as a hexadecimal number), the length (as a hexadecimal number), and the names of the objects mapped to @STATIC. Under certain circumstances, the compiler might decide to map other objects to @STATIC.

If you specify the `ATTR` or `XREF` option, the listing file includes offset information for file scope read/write static variables.

Using the IPA Link Step Listing

The IPA Link step generates a listing file if you specify any of the following options:

- `ATTR`
- `INLINE(,REPORT,,)`
- `INLRPT`
- `IPA(MAP)`
- `LIST`
- `XREF`

Note: IPA does not support source listings or source annotations within Pseudo Assembly listings. The Pseudo Assembly listings do display the file and line number of the source code that contributed to a segment of pseudo assembly code.

Example of an IPA Link Step Listing

Figure 19 on page 257 shows an example of an IPA Link step listing.

```
15647A01 V2 R10 OS/390 C/C++ IPA            'TSIPA.TEST.LINKCNTL(INCLCNTL)'                04/19/2000 18:06:57   Page     1

                                          * * * * *   P R O L O G   * * * * *

  Compile Time Library . . . . . . : 220A0000
  Command options:
     Primary input name. . . . . . : 'TSIPA.TEST.LINKCNTL(INCLCNTL)'
     Compiler options. . . . . . . : *IPA(LINK,MAP,NOREFMAP,LEVEL(1),DUP,ER,NONCAL,NOUPCASE,NOCONTROL)             *NOGONUMBER
                                   : *NOALIAS     *NODECK      *TERMINAL    *LIST        *XREF        *NOATTR      *NOOFFSET
                                   : *MEMORY      *NOCSECT     *LIBANSI     *FLAG(I)
                                   : *NOTEST(NOSYM,NOBLOCK,NOLINE,NOPATH,HOOK)           *OPTIMIZE(1)
                                   : *INLINE(AUTO,REPORT,1000,8000)         *OBJECT      *OPTFILE(DD:OPTION)       *NOSERVICE
                                   : *NOOE        *NOLOCALE    *HALT(16)    *NOGOFF

                                   * * * * *   E N D   O F   P R O L O G   * * * * *
15647A01 V2 R10 OS/390 C/C++ IPA            'TSIPA.TEST.LINKCNTL(INCLCNTL)'                04/19/2000 18:06:57   Page     2

                             * * * * *   O B J E C T   F I L E   M A P   * * * * *

 *ORIGIN  IPA  FILE ID  FILE NAME
    P               1   TSIPA.TEST.LINKCNTL(INCLCNTL)
    PI     Y        2   TSIPA.TEST.PASS1.OBJ(INCLMAIN)
    PI     Y        3   TSIPA.TEST.PASS1.OBJ(INCLRTN1)
    PI     Y        4   TSIPA.TEST.PASS1.OBJ(INCLRTN2)

 ORIGIN:  P=primary input     PI=primary INCLUDE       SI=secondary INCLUDE   IN=internal
          A=automatic call     U=UPCASE automatic call   R=RENAME card          L=C Library

                      * * * * *   E N D   O F   O B J E C T   F I L E   M A P   * * * * *
15647A01 V2 R10 OS/390 C/C++ IPA            'TSIPA.TEST.LINKCNTL(INCLCNTL)'                04/19/2000 18:06:57   Page     3

                        * * * * *   C O M P I L E R   O P T I O N S   M A P   * * * * *

  SOURCE FILE ID      COMPILE OPTIONS
              1        *AGGRCOPY(NOOVERLAP)      *NOALIAS     *ANSIALIAS   *ARCH(2)     *ARGPARSE
                       *CHARSET(BIAS=EBCDIC,LIB=EBCDIC)       *NOCOMPACT   *NOCOMPRESS  *NODLL(NOCALLBACKANY)     *ENV(MVS)
                       *EXECOPS     *FLOAT(HEX,FOLD,NOAFP)    *NOGONUMBER  *NOIGNERRNO  *NOINITAUTO
                       *IPA(NOLINK,NOOBJECT,COMPRESS)         *NOLIBANSI   *NOLIST      *NOLOCALE    *LONGNAME
                       *MAXMEM(2097152)          *OPTIMIZE(1) *PLIST(HOST) *REDIR       *NORENT      *NOROCONST   *SPILL(128)
                       *NOSTART     *STRICT      *NOSTRICT_INDUCTION       *NOTEST      *TUNE(3)     *NOXPLINK    *NOXREF

              2        *AGGRCOPY(NOOVERLAP)      *NOALIAS     *ANSIALIAS   *ARCH(2)     *ARGPARSE
                       *CHARSET(BIAS=EBCDIC,LIB=EBCDIC)       *NOCOMPACT   *NOCOMPRESS  *NODLL(NOCALLBACKANY)     *ENV(MVS)
                       *EXECOPS     *FLOAT(HEX,FOLD,NOAFP)    *NOGONUMBER  *NOIGNERRNO  *NOINITAUTO
                       *IPA(NOLINK,NOOBJECT,COMPRESS)         *NOLIBANSI   *NOLIST      *NOLOCALE    *LONGNAME
                       *MAXMEM(2097152)          *OPTIMIZE(1) *PLIST(HOST) *REDIR       *NORENT      *NOROCONST   *SPILL(128)
                       *NOSTART     *STRICT      *NOSTRICT_INDUCTION       *NOTEST      *TUNE(3)     *NOXPLINK    *NOXREF

              3        *AGGRCOPY(NOOVERLAP)      *NOALIAS     *ANSIALIAS   *ARCH(2)     *ARGPARSE
                       *CHARSET(BIAS=EBCDIC,LIB=EBCDIC)       *NOCOMPACT   *NOCOMPRESS  *NODLL(NOCALLBACKANY)     *ENV(MVS)
                       *EXECOPS     *FLOAT(HEX,FOLD,NOAFP)    *NOGONUMBER  *NOIGNERRNO  *NOINITAUTO
                       *IPA(NOLINK,NOOBJECT,COMPRESS)         *NOLIBANSI   *NOLIST      *NOLOCALE    *LONGNAME
                       *MAXMEM(2097152)          *OPTIMIZE(1) *PLIST(HOST) *REDIR       *NORENT      *NOROCONST   *SPILL(128)
                       *NOSTART     *STRICT      *NOSTRICT_INDUCTION       *NOTEST      *TUNE(3)     *NOXPLINK    *NOXREF


                 * * * * *   E N D   O F   C O M P I L E R   O P T I O N S   M A P   * * * * *
```

Figure 19. Example of an IPA Link Step Listing (Part 1 of 7)

```
15647A01 V2 R10 OS/390 C/C++ IPA            'TSIPA.TEST.LINKCNTL(INCLCNTL)'             04/19/2000 18:06:57   Page     4
                              * * * * *   I N L I N E   R E P O R T   * * * * *

                    IPA Inline Report (Summary)

  Reason:    P : #pragma noinline was specified for this routine
             F : #pragma inline was specified for this routine
             A : Automatic inlining
             C : Partition conflict
             N : Not IPA Object
             - : No reason
  Action:    I : Routine is inlined at least once
             L : Routine is initially too large to be inlined
             T : Routine expands too large to be inlined
             C : Candidate for inlining but not inlined
             N : No direct calls to routine are found in file (no action)
             U : Some calls not inlined due to recursion or parameter mismatch
             - : No action
  Status:    D : Internal routine is discarded
             R : A direct call remains to internal routine (cannot discard)
             A : Routine has its address taken (cannot discard)
             E : External routine (cannot discard)
             - : Status unchanged
  Calls/I      : Number of calls to defined routines / Number inline
  Called/I     : Number of times called / Number of times inlined

  Reason  Action  Status   Size (init)    Calls/I   Called/I     Name

     A      N       -      200 (102)        11/11     0          main
     A      I       D      0 (18)           0         1/1        Incl_Rtn1
     A      I       D      0 (8)            0         10/10      Incl_Rtn2
  Mode = AUTO     Inlining Threshold =    1000    Expansion Limit =    8000

15647A01 V2 R10 OS/390 C/C++ IPA            'TSIPA.TEST.LINKCNTL(INCLCNTL)'             04/19/2000 18:06:57   Page     5

                    IPA Inline Report (Call Structure)

  Defined Subprogram    : main
      Calls To(11,11)   : Incl_Rtn2(10,10)
                          Incl_Rtn1(1,1)
   Called From          : 0

  Defined Subprogram    : Incl_Rtn2
      Calls To          : 0
   Called From(10,10)   : main(10,10)

  Defined Subprogram    : Incl_Rtn1
      Calls To          : 0
   Called From(1,1)     : main(1,1)


                       * * * * *   E N D   O F   I N L I N E   R E P O R T   * * * * *
```

Figure 19. Example of an IPA Link Step Listing (Part 2 of 7)

```
15647A01 V2 R10 OS/390 C/C++ IPA                                 Partition 0                 04/19/2000 18:06:57   Page     6
                               * * * * *   P A R T I T I O N   M A P   * * * * *

 PARTITION 0

 PARTITION CSECT NAMES:
      Code: none
    Static: none
      Test: none

 PARTITION DESCRIPTION:
    Initialization data partition

 COMPILER OPTIONS FOR PARTITION 0:
   *AGGRCOPY(NOOVERLAP)      *NOALIAS     *ARCH(2)     *ARGPARSE    *CHARSET(BIAS=EBCDIC,LIB=EBCDIC)       *NOCOMPACT   *NOCOMPRESS
   *NOCSECT     *NODLL       *ENV(MVS)    *EXECOPS     *FLOAT(HEX,FOLD,NOAFP)    *NOGOFF      *NOGONUMBER  *NOIGNERRNO  *NOINITAUTO
   *IPA(LINK)   *LIBANSI     *LIST        *NOLOCALE    *LONGNAME    *MAXMEM(2097152)          *OPTIMIZE(1) *PLIST(HOST) *REDIR
   *NORENT      *NOROCONST   *SPILL(128)  *START       *STRICT      *NOSTRICT_INDUCTION       *NOTEST      *TUNE(3)     *NOXPLINK
   *XREF

 SYMBOLS IN PARTITION 0:

  *TYPE     FILE ID    SYMBOL
     D           1     gbl

   TYPE:  F=function     D=data

 SOURCE FILES FOR PARTITION 0:

  *ORIGIN     FILE ID     SOURCE FILE NAME
      P            1      TSIPA.TEST.C(INCLMAIN)

  ORIGIN:  P=primary input     PI=primary INCLUDE

                        * * * * *   E N D   O F   P A R T I T I O N   M A P   * * * * *
```

Figure 19. Example of an IPA Link Step Listing (Part 3 of 7)

```
15647A01 V2 R10 OS/390 C/C++ IPA                                 Partition 0                  04/19/2000 18:06:57   Page     7

 OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                           Timestamp and Version Information
 000000  F2F0  F0F0                                      =C'2000'           Compiled Year
 000004  F0F4  F1F9                                      =C'0419'           Compiled Date MMDD
 000008  F1F8  F0F6  F2F5                                =C'180625'         Compiled Time HHMMSS
 00000E  F0F2  F0C1  F0F0                                =C'020A00'         Compiler Version
                           Timestamp and Version End

15647A01 V2 R10 OS/390 C/C++ IPA                                 Partition 0                  04/19/2000 18:06:57   Page     8

                                   E X T E R N A L   S Y M B O L   D I C T I O N A R Y

                       TYPE  ID  ADDR    LENGTH            NAME

                        SD    1 000000   000018            @STATICP
                        SD    2 000000   000004            gbl
                        SD    3 000000   000008            @@PPA2
15647A01 V2 R10 OS/390 C/C++ IPA                                 Partition 0                  04/19/2000 18:06:57   Page     9

                             E X T E R N A L   S Y M B O L   C R O S S   R E F E R E N C E

           ORIGINAL NAME                                   EXTERNAL SYMBOL NAME

           @STATICP                                        @STATICP
           gbl                                             gbl
           @@PPA2                                          @@PPA2
15647A01 V2 R10 OS/390 C/C++ IPA                                 Partition 1                  04/19/2000 18:06:57   Page    10

                               * * * * *   P A R T I T I O N   M A P   * * * * *

 PARTITION 1 OF 1

 PARTITION SIZE:
    Actual: 3624
     Limit: 102400

 PARTITION CSECT NAMES:
      Code: none
    Static: none
      Test: none

 PARTITION DESCRIPTION:
    Primary partition

 COMPILER OPTIONS FOR PARTITION 1:
   *AGGRCOPY(NOOVERLAP)      *NOALIAS     *ARCH(2)     *ARGPARSE    *CHARSET(BIAS=EBCDIC,LIB=EBCDIC)       *NOCOMPACT   *NOCOMPRESS
   *NOCSECT     *NODLL       *ENV(MVS)    *EXECOPS     *FLOAT(HEX,FOLD,NOAFP)    *NOGOFF      *NOGONUMBER  *NOIGNERRNO  *NOINITAUTO
   *IPA(LINK)   *LIBANSI     *LIST        *NOLOCALE    *LONGNAME    *MAXMEM(2097152)          *OPTIMIZE(1) *PLIST(HOST) *REDIR
   *NORENT      *NOROCONST   *SPILL(128)  *START       *STRICT      *NOSTRICT_INDUCTION       *NOTEST      *TUNE(3)     *NOXPLINK
   *XREF

 SYMBOLS IN PARTITION 1:

  *TYPE     FILE ID    SYMBOL
     F           1     main

   TYPE:  F=function     D=data

 SOURCE FILES FOR PARTITION 1:

  *ORIGIN     FILE ID     SOURCE FILE NAME
      P            1      TSIPA.TEST.C(INCLMAIN)
      P            2      TSIPA.TEST.C(INCLRTN1)
      P            3      TSIPA.TEST.C(INCLRTN2)

  ORIGIN:  P=primary input     PI=primary INCLUDE

                        * * * * *   E N D   O F   P A R T I T I O N   M A P   * * * * *
```

Figure 19. Example of an IPA Link Step Listing (Part 4 of 7)

```
15647A01 V2 R10 OS/390 C/C++ IPA                                 Partition 1               04/19/2000 18:06:57   Page    11

 OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

 000000  41F0  F060                                   LA    r15,96(,r15)
 000004  07FF                                         BR    r15
 000006  0700                                         NOPR  0
 000008  00000000                                           =F'0'

                           Timestamp and Version Information
 00000C  F2F0  F0F0                                         =C'2000'          Compiled Year
 000010  F0F4  F1F9                                         =C'0419'          Compiled Date MMDD
 000014  F1F8  F0F6  F2F5                                   =C'180625'        Compiled Time HHMMSS
 00001A  F0F2  F0C1  F0F0                                   =C'020A00'        Compiler Version
                           Timestamp and Version End

15647A01 V2 R10 OS/390 C/C++ IPA                                 Partition 1: main         04/19/2000 18:06:57   Page    12

 OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                           PPA1: Entry Point Constants
 000020  1CCEA106                                           =F'483303686'     Flags
 000024  00000098                                           =A(PPA2-main)
 000028  00000000                                           =F'0'             No PPA3
 00002C  00000000                                           =F'0'             No EPD
 000030  FE000000                                           =F'-33554432'     Register save mask
 000034  00000000                                           =F'0'             Member flags
 000038  90                                                 =AL1(144)         Flags
 000039  000000                                             =AL3(0)           Callee's DSA use/8
 00003C  0040                                               =H'64'            Flags
 00003E  0012                                               =H'18'            Offset/2 to CDL
 000040  00000000                                           =F'0'             Reserved
 000044  5000004A                                           =F'1342177354'    CDL function length/2
 000048  00000040                                           =F'64'            CDL function EP offset
 00004C  38240000                                           =F'941883392'     CDL prolog
 000050  40090041                                           =F'1074331713'    CDL epilog
 000054  00000000                                           =F'0'             CDL end
 000058  0004  ****                                         AL2(4),C'main'
                           PPA1 End

 000060                    00010 |     1   main     DS    0D
 000060  47F0  F022        00010 |     1            B     34(,r15)
 000064  01C3C5C5                                           CEE eyecatcher
 000068  00000098                                           DSA size
 00006C  FFFFFFC0                                           =A(PPA1-main)
 000070  47F0  F001        00010 |     1            B     1(,r15)
 000074  58F0  C31C        00010 |     1            L     r15,796(,r12)
 000078  184E              00010 |     1            LR    r4,r14
 00007A  05EF              00010 |     1            BALR  r14,r15
 00007C  00000000                                           =F'0'
 000080  07F3              00010 |     1            BR    r3
 000082  90E4  D00C        00010 |     1            STM   r14,r4,12(r13)
 000086  58E0  D04C        00010 |     1            L     r14,76(,r13)
 00008A  4100  E098        00010 |     1            LA    r0,152(,r14)
 00008E  5500  C314        00010 |     1            CL    r0,788(,r12)
 000092  4130  F03A        00010 |     1            LA    r3,58(,r15)
 000096  4720  F014        00010 |     1            BH    20(,r15)
 00009A  5000  E04C        00010 |     1            ST    r0,76(,r14)
 00009E  9210  E000        00010 |     1            MVI   0(r14),16
 0000A2  50D0  E004        00010 |     1            ST    r13,4(,r14)
 0000A6  18DE              00010 |     1            LR    r13,r14
 0000A8                    End of Prolog

 0000A8  180F              00008 |     3  +         LR    r0,r15
 0000AA  8900  0001        00008 |     3  +         SLL   r0,1
 0000AE  181F              00008 |     3  +         LR    r1,r15
 0000B0  B252  0010        00008 |     3  +         MSR   r1,r0
 0000B4  180F              00008 |     3  +         LR    r0,r15
 0000B6  B252  0001        00008 |     3  +         MSR   r0,r1
 0000BA  181F              00008 |     3  +         LR    r1,r15
```

Figure 19. Example of an IPA Link Step Listing (Part 5 of 7)

```
0000BC  B252  0010        00008 |    3  +         MSR   r1,r0
 0000C0  180F              00008 |    3  +         LR    r0,r15
 0000C2  B252  0001        00008 |    3  +         MSR   r0,r1
 0000C6  181F              00008 |    3  +         LR    r1,r15
 0000C8  B252  0010        00008 |    3  +         MSR   r1,r0
 0000CC  180F              00008 |    3  +         LR    r0,r15
15647A01 V2 R10 OS/390 C/C++ IPA                              Partition 1: main              04/19/2000 18:06:57   Page    13

 OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

 0000CE  B252  0001        00008 |    3  +         MSR   r0,r1
 0000D2  181F              00008 |    3  +         LR    r1,r15
 0000D4  B252  0010        00008 |    3  +         MSR   r1,r0
 0000D8  180F              00008 |    3  +         LR    r0,r15
 0000DA  B252  0001        00008 |    3  +         MSR   r0,r1
 0000DE  B252  00F0        00008 |    3  +         MSR   r15,r0
 0000E2                    00022 |    1   @1L18    DS    0H

 0000E2                    Start of Epilog
 0000E2  180D              00023 |    1            LR    r0,r13
 0000E4  58D0  D004        00023 |    1            L     r13,4(,r13)
 0000E8  58E0  D00C        00023 |    1            L     r14,12(,r13)
 0000EC  9824  D01C        00023 |    1            LM    r2,r4,28(r13)
 0000F0  051E              00023 |    1            BALR  r1,r14
 0000F2  0707              00023 |    1            NOPR  7

                           ***   General purpose registers used: 1101100000001111
                           ***   Floating point  registers used: 0000000000000000
                           ***   Size of register spill area: 128(max) 0(used)
                           ***   Size of dynamic storage: 152
                           ***   Size of executable code: 148

 0000F4  0000  0000

                           PPA2: Compile Unit Block
 0000F8  0300  2202                                      =F'50340354'       Flags
 0000FC  FFFF  FF08                                      =A(CEESTART-PPA2)
 000100  0000  0000                                      =F'0'              No PPA4
 000104  FFFF  FF14                                      =A(TIMESTMP-PPA2)
 000108  0000  0000                                      =F'0'              No primary
 00010C  0000  0000                                      =F'0'              Flags
                           PPA2 End
```

Figure 19. Example of an IPA Link Step Listing (Part 6 of 7)

```
15647A01 V2 R10 OS/390 C/C++ IPA                                Partition 1                04/19/2000 18:06:57   Page    14
                                   E X T E R N A L   S Y M B O L   D I C T I O N A R Y
                       TYPE  ID  ADDR    LENGTH           NAME

                        SD    1 000000   000110           @STATICP
                        LD    0 000060   000001           main
                        ER    2 000000                    CEESG003
                        ER    3 000000                    CEESTART
                        SD    4 000000   000008           @@PPA2
                        SD    5 000000   00000C           CEEMAIN
                        ER    6 000000                    EDCINPL
15647A01 V2 R10 OS/390 C/C++ IPA                                Partition 1                04/19/2000 18:06:57   Page    15
                             E X T E R N A L   S Y M B O L   C R O S S   R E F E R E N C E
           ORIGINAL NAME                                  EXTERNAL SYMBOL NAME

           @STATICP                                       @STATICP
           main                                           main
           CEESG003                                       CEESG003
           CEESTART                                       CEESTART
           @@PPA2                                         @@PPA2
           CEEMAIN                                        CEEMAIN
           EDCINPL                                        EDCINPL
15647A01 V2 R10 OS/390 C/C++ IPA                                Partition 1                04/19/2000 18:06:57   Page    16
                             * * * * *   S O U R C E   F I L E   M A P   * * * * *
             OBJECT      SOURCE
  *ORIGIN   FILE ID     FILE ID     SOURCE FILE NAME
      P         2           1      TSIPA.TEST.C(INCLMAIN)
                                     - Compiled by 5647A01 V2 R10 OS/390 C
                                                on 04/19/2000 18:06:25
      P         3           2      TSIPA.TEST.C(INCLRTN1)
                                     - Compiled by 5647A01 V2 R10 OS/390 C
                                                on 04/19/2000 18:06:30
      P         4           3      TSIPA.TEST.C(INCLRTN2)
                                     - Compiled by 5647A01 V2 R10 OS/390 C
                                                on 04/19/2000 18:06:34

  ORIGIN:  P=primary input     PI=primary INCLUDE

                      * * * * *   E N D   O F   S O U R C E   F I L E   M A P   * * * * *
15647A01 V2 R10 OS/390 C/C++ IPA                                Partition 1                04/19/2000 18:06:57   Page    17
                             * * * * *   M E S S A G E   S U M M A R Y   * * * * *
 TOTAL   UNRECOVERABLE  SEVERE       ERROR     WARNING    INFORMATIONAL
                (U)       (S)         (E)        (W)          (I)
     0          0         0           0          0            0

                      * * * * *   E N D   O F   M E S S A G E   S U M M A R Y   * * * * *

                          * * * * *   E N D   O F   C O M P I L A T I O N   * * * * *
```

Figure 19. Example of an IPA Link Step Listing (Part 7 of 7)

IPA Link Step Listing Components

The following sections describe the components of an IPA Link step listing.

Heading Information

The first page of the listing is identified by the product number, the compiler version and release numbers, the central title area, the date and time compilation began (formatted according to the current locale), and the page number.

In the following listing sections, the central title area will contain the primary input file identifier:

- Prolog
- Object File Map
- Source File Map
- Compiler Options Map
- Global Symbols Map
- Inline Report
- Messages
- Message Summary

In the following listing sections, the central title area will contain the phrase Partition `nnnn`, where `nnnn` specifies the partition number:
- Partition Map

In the following listing sections, the title contains the phrase Partition `nnnn:name`. `nnnn` specifies the partition number, and `name` specifies the name of the first function in the partition:
- Pseudo Assembly Listing
- External Symbol Cross Reference
- Storage Offset Listing

Prolog Section

The Prolog section of the listing provides information about the compile-time library, file identifiers, compiler options, and other items in effect when the IPA Link step was invoked.

The listing displays all compiler options except those with no default (for example, `ARCHITECTURE`). If you specify `IPA` suboptions that are irrelevant to the IPA Link step, the Prolog does not display them. Any problems with compiler options appear after the body of the Prolog section and before the End of Prolog section.

Object File Map

The Object File Map displays the names of the object files that were used as input to the IPA Link step. Specify any of the following options to generate the Object File Map:
- `IPA(MAP)`
- `LIST`

Other listing sections, such as the Source File Map, use the File ID numbers that appear in this listing section.

HFS file names that are too long to fit into a single listing record continue on subsequent listing records.

Source File Map

The Source File Map listing section identifies the source files that are included in the object files. The IPA Link step generates this section if you specify any of the following options:
- `IPA(MAP)`
- `LIST`

The IPA Link step formats the compilation date and time according to the locale you specify with the `LOCALE` option in the IPA Link step. If you do not specify the `LOCALE` option, it uses the default locale.

This section appears near the end of the IPA Link step listing. If the IPA Link step terminates early due to errors, it does not generate this section.

Compiler Options Map

The Compiler Options Map listing section identifies the compiler options that were specified during the IPA Compile step for each compilation unit that is encountered when the object file is processed. For each compilation unit, it displays the final options that are relevant to IPA Link step processing. You may have specified these options through a compiler option or `#pragma` directive, or you may have picked them up as defaults.

The IPA Link step generates this listing section if you specify the `IPA(MAP)` option.

Global Symbols Map

The Global Symbols Map listing section shows how global symbols are mapped into members of global data structures by the global variable coalescing optimization process.

Each global data structure is limited to 16 MB by the z/OS object architecture. If an application has more than 16 MB of data, IPA Link must generate multiple global data structures for the application. Each global data structure is assigned a unique name.

The Global Symbols Map includes symbol information and file name information (file name information may be approximate). In addition, line number information is available for C compilations if you specified any of the following options during the IPA Compile step:

- `XREF`
- `IPA(XREF)`
- `XREF(ATTRIBUTE)`

The IPA Link step generates this listing section if you specify the `IPA(MAP)` option.

Inline Report for IPA Inliner

The Inline Report describes the actions that are performed by the IPA Inliner. The IPA Link step generates this listing section if you specify the `INLINE(,REPORT,,)`, `NOINLINE(,REPORT,,)`, or `INLRPT` option.

This report is similar to the one that is generated by the non-IPA inliner. In the IPA version of this report, the term 'subprogram' is equivalent to a C/C++ function or a C++ method. The summary contains information such as:

- Name of each defined subprogram. IPA sorts subprogram names in alphabetical order.
- Reason for action on a subprogram:
  - A `#pragma noinline` was specified for the subprogram. The `P` indicates that inlining could not be performed.
  - `inline` was specified for the subprogram. For z/OS C++, this is a result of the inline specifier. For C, this a result of the `#pragma inline`. The `F` indicates that the subprogram was declared inline.
  - The IPA Link step performed auto-inlining on the subprogram.
  - There was no reason to inline the subprogram.
  - There was a partition conflict.
  - The IPA Link step could not inline the object module because it was a non-IPA object module.
- Action on a subprogram:
  - IPA inlined subprogram at least once.
  - IPA did not inline subprogram because of initial size constraints.
  - IPA did not inline subprogram because of expansion beyond size constraint.
  - Subprogram was a candidate for inlining, but IPA did not inline it.
  - Subprogram was a candidate for inlining, but was not referenced.
  - The subprogram is directly recursive, or some calls have mismatched parameters.
- Status of original subprogram after inlining:
  - IPA discarded the subprogram because it is no longer referenced and is defined as `static` internal.
  - IPA did not discard the subprogram, for various reasons :

- Subprogram is external. (It can be called from outside the compilation unit.)
    - Subprogram call to this subprogram remains.
    - Subprogram has its address taken.
- Initial relative size of subprogram (in Abstract Code Units (ACUs)).
- Final relative size of subprogram (in ACUs) after inlining.
- Number of calls within the subprogram and the number of these calls that IPA inlined into the subprogram.
- Number of times the subprogram is called by others in the compile unit and the number of times IPA inlined the subprogram.
- Mode that is selected and the value of *threshold* and *limit* you specified for the compilation.

Static functions whose names are not unique within the application as a whole will have names prefixed with `nnnn:`, where `nnnn` is the source file number.

The detailed call structure contains specific information of each subprogram such as:
- Subprograms that it calls
- Subprograms that call it
- Subprograms in which it is inlined.

The information can help you to better analyze your program if you want to use the inliner in selective mode.

Inlining may result in additional messages. For example, if inlining a subprogram with automatic storage increases the automatic storage of the subprogram it is being inlined into by more than 4K, the IPA Link step issues a message.

This report may display information about inlining specific subprograms, at the point at which IPA determines that inlining is impossible.

The counts in this report do not include calls from non-IPA to IPA programs.

Note: Even if the IPA Link step did not perform any inlining, it generates the IPA Inline Report if you request it.

Partition Map

The Partition Map listing section describes each of the object code partitions the IPA Link step creates. It provides the following information:
- The reason for generating each partition
- How the code is packaged (the CSECTs)
- The options used to generate the object code
- The function and global data included in the partition
- The source files that were used to create the partition

The IPA Link step generates this listing section if you specify either of the following options :
- `IPA(MAP)`
- `LIST`

The Pseudo Assembly, External Symbol Dictionary, External Symbol Cross Reference, and Storage Offset listing sections follow the Partition Map listing section for the partition, if you have specified the appropriate compiler options.

Pseudo Assembly Listing

The option `LIST` generates a listing of the machine instructions in the current partition of the object module, in a form similar to assembler language.

This pseudo assembly listing displays the source statement line numbers and the line number of inlined code to aid you in debugging inlined code. Refer to "GONUMBER | NOGONUMBER" on page 115, "IPA | NOIPA" on page 125, and "LIST | NOLIST" on page 133 for information about source and line numbers in the listing section.

External Symbol Dictionary

The External Symbol Dictionary lists the names that the IPA Link step generates for the current partition of the object module. It includes address information and size information about each symbol.

External Symbol Cross Reference Listing

The IPA Link step generates this section if you specify the `ATTR` or `XREF` compiler option. It shows how the IPA Link step maps internal and ESD names for external symbols that are defined or referenced in the current partition of the object module.

Storage Offset Listing

The Storage Offset listing section displays the offsets for the data in the current partition of the object module.

During the IPA Compile step, the compiler saves symbol storage offset information in the IPA object file as follows:

- For C, if you specify the `XREF`, `IPA(ATTRIBUTE)`, `IPA(XREF)` options, or the `#pragma option (XREF)`
- For C++, if you specify the `ATTR`, `XREF`, `IPA(ATTRIBUTE)`, or `IPA(XREF)` options

If this is done and the compilation unit includes variables, the IPA Link step may generate a Storage Offset listing.

If you specify the `ATTR` or `XREF` option on the IPA Link step, and any of the compilation units that contributed variables to a particular partition had storage offset information encoded in the IPA object file, the IPA Link step generates a Storage Offset listing section for that partition.

The Storage Offset listing displays the variables that IPA did not coalesce. The symbol definition information appears as `file#:line#`.

Static Map

If you specify the `ATTR` or `XREF` option, the listing file includes offset information for file scope read/write static variables.

Messages

If the IPA Link step detects an error, or the possibility of an error, it issues one or more diagnostic messages, and generates the Messages listing section. This listing section contains a summary of the messages that are issued during IPA Link step processing.

The IPA Link step listing sorts the messages by severity. The Messages listing section displays the listing page number where each message was originally shown. It also displays the message text, and optionally, information relating the error to a file name, line (if known), and column (if known).

For more information on compiler messages, see “FLAG | NOFLAG” on page 107, and “Appendix G. z/OS C/C++ Compiler Return Codes and Messages” on page 591.

Message Summary

This listing section displays the total number of messages and the number of messages for each severity level.

Chapter 7. Binder Options and Control Statements

This chapter describes only the binder options, suboptions, and control statements that are considered important for a C or C++ programmer. For a detailed description of all the binder options and control statements, see *z/OS DFSMS Program Management*.

Binder Options

The binder processes options from left to right. If you specify a binder option more than once, the binder uses the last, or rightmost option. The default options used by the z/OS C/C++ supplied cataloged procedures, the `CXXBIND` REXX exec, and the `c89`, `cc`, and `c++` utilities are shown only where they differ from the binder default.

There are two option formats supported by the binder: `KEYWORD=`*`OPTION`* and `KEYWORD(`*`OPTION`*`)`. Both formats are supported for Batch and TSO. `c89` only supports the `KEYWORD=`*`OPTION`* format.

ALIASES(ALL | NO)

Table 22. ALIASES Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
`ALIASES=NO`			

The `ALIASES(ALL)` option instructs the binder to create hidden aliases for all externally defined symbols (functions and variables). Hidden aliases are marked as ″not executable″, to prevent an unintentional load and execution. These aliases might not be visible to some system utilities. Also, if the target of an `ALIAS` control statement is a symbol, the binder does not mark the alias as hidden.

The binder does not create hidden aliases if `ALIASES(NO)` is in effect, or if the module is saved in a PM2 or earlier format. PM2 format is available only for C NORENT NOLONGNAME compilations and programs that are first processed by the prelinker. See “COMPAT(PM1 | PM2 | PM3 | CURRENT | CURR)” on page 270 for information on setting the compatibility format.

Hidden aliases are for autocall purposes only. See “Generating Aliases for Automatic Library Call (Library Search)” on page 379.

AMODE

Table 23. AMODE Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
`AMODE=24`	`AMODE=31`	`AMODE(31)`	`AMODE=31`

To assign the addressing mode for all the entry points into a program module (the main entry point, its true aliases, and all the alternate entry points), you should code the `AMODE` parameter as follows: `AMODE={24/31/ANY/MIN}`

CALL(YES | NO)

Table 24. CALL Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
`CALL=YES`			`CALL=NO` if the `-r` flag is set. `CALL=YES` otherwise.

The `CALL(YES)` option specifies that the binder should search the libraries that are defined by the `DD SYSLIB` to find symbol definitions (see "Final Autocall Processing (SYSLIB)" on page 378).

The `CALL(NO)` option instructs the binder not to perform final autocall processing of the libraries that are defined by `DD SYSLIB` to resolve unresolved references.

CASE(UPPER | MIXED)

Table 25. CASE Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
`CASE=UPPER`	`CASE=MIXED`	`CASE(MIXED)`	`CASE=MIXED`

The `CASE` option controls the binder's sensitivity to case. When you specify `CASE(MIXED)`:

- The binder distinguishes between uppercase characters and lowercase characters, and treats two strings as different if their cases do not match exactly.
- The binder does not convert lowercase characters to uppercase in names that are encountered in input modules, control statements, and call parameters.

When you specify `CASE(UPPER)`, the binder converts all lowercase characters to uppercase during processing.

Note: z/OS C++ does not support the `CASE(UPPER)` option. Use `CASE(MIXED)` for C++ code.

COMPAT(PM1 | PM2 | PM3 | CURRENT | CURR)

Table 26. COMPAT Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
`COMPAT=CURRENT`			`COMPAT=CURRENT`

The `COMPAT` option specifies the compatibility level of the binder.

`COMPAT=CURRENT` is the current level of the binder and allows you to use all features.

`COMPAT=PM2` allows you to link-edit into a load module. The prelinker is required if any of the object files in the application use constructed reentrancy, use long names, are DLL or are C++. This option is required for modules which will be executed on OS/390 releases prior to OS/390 Version 2 Release 4.

Higher levels than PM2 do not require but can use the prelinker. Higher levels than PM2 normally require that you link into a program object or HFS file. You can link-edit into a load module if you ensure that none of the object files in the application use constructed reentrancy, use long names, are DLL or are C++.

c89/cc/c++ will default to `COMPAT=PM2` when:

- The prelinker is used during the link step.
- Link-editing without the prelinker, and the target library (according to {_PVERSION}) is lower than OS/390 Version 2 Release 4. You may encounter problems at link-edit or run-time if any of the object files in the application use constructed reentrancy, use long names, are DLL or are C++.

Otherwise, the default is `COMPAT=CURRENT`.

DYNAM(DLL | NO)

Table 27. DYNAM Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
`DYNAM=NO`	`DYNAM=DLL`	`DYNAM(DLL)`	`DYNAM=DLL` `DYNAM=NO` if prelinker is active during link-edit

The `DYNAM` option specifies whether the binder should enable the resultant module for DLL-type dynamic binding. You must specify `DYNAM(DLL)` if the program object is to be a DLL or will need to load DLLs. If you specify `DYNAM(DLL)`, the binder does the following:

- Creates the Import/Export Table in section IEWBCIE of class B_IMPEXP. This element contains information about imported and exported symbols that is necessary to support run-time library dynamic linking and loading.
- Performs DLL-specific bind processing: that is, generates linkage areas (descriptors) in class C_WSA for run-time library fixup.

Import/export tables and the definition side-deck are not created if you specify `DYNAM(NO)`, or if it is in effect by default. If you specify `DYNAM(DLL)`, the binder `RES` option is disabled. DLL-enabled modules require PM3 program objects. If you attempt to save them in down-level program objects or load modules using `COMPAT`, the binder issues a severity 12 error, and does not save the module.

LET(0 | 4 | 8 | 12)

Table 28. LET Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
`LET=4`			If `LET` is specified without a value, then you get `LET=8`.

The `LET` option specifies that a generated program object should be marked as executable even if the return code is not zero: for example, if symbols are unresolved. For example, `LET(4)` marks the generated program object as executable even if there are errors of severity 4 or less. `LET` is the equivalent of `LET(8)`.

LIST(OFF | STMT | SUMMARY | NOIMP | ALL)

Table 29. LIST Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
LIST=SUMMARY			If -V is not specified, the default is LIST=NO. If -V is specified, then LIST=NOIMP. If LIST is specified without a value, then you get LIST=SUMMARY.

The LIST option specifies the type of information that is written to the binder map. Use one of the following suboptions:

ALL	produces a listing of individual function calls, save summary, control statements, and messages
SUMMARY	produces a listing of the summary information which includes processing options, module attributes, save summary, and the entry point summary, and echoes IMPORT control statements.
NOIMP	produces the same output as SUMMARY, but does not echo IMPORT control statements.
STMT	produces a listing of control statements and binder messages
OFF	produces a listing that contains only binder messages.

Note: The binder map contains a summary of the modules only if you specify the suboptions SUMMARY, ALL, or NOIMP.

NOLIST is equivalent to LIST(OFF).

MAP(YES | NO)

Table 30. MAP Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
MAP=NO	MAP	MAP	MAP=YES when the -V flag is set. MAP=NO otherwise.

The option MAP(YES) instructs the binder to write a printed map of the program object to DD SYSPRINT. The option MAP(NO) specifies that the binder does not generate a map.

OPTIONS

This option specifies the DDname of a file that contains other options. For example, OPTIONS=OPT1 specifies that further options should be read from the DDname OPT1. This option is useful if the length of the PARM keyword in your JCL is longer that 100 characters.

REUS(NONE | SERIAL | RENT)

Table 31. REUS Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
REUS=NONE	REUS=RENT	REUS(RENT)	REUS=SERIAL when the -g flag is set. REUS=RENT otherwise.

The REUS option specifies the reusability of the output program object. For C/C++ code these are the suboptions that you are most likely to use:

RENT	Specifies that other users or programs can share a read-only copy of the code.
NONE	Specifies that the code cannot be shared. Use this option if you have NORENT variables which are modified during program execution. Such a program object cannot be in the LPA or ELPA. If you built a DLL with REUS(NONE), any program that links to the DLL will get a new load of both the code and data (C_WSA). This may be a problem if other DLLs in the same program share this DLL. See “Non-reentrant DLL Problems” on page 398.
SERIAL	Specifies that the code is loaded into modifiable storage and that it can be reused without being reloaded. Simultaneous use of the code, for example by more than one user or more than one thread, is not supported unless explicitly allowed for by the programmer. Use this option if you have NORENT variables that are modified during program execution. Such a program object cannot be in the LPA or ELPA. If you built a DLL with REUS(SERIAL), any program within a single Language Environment enclave that links to that DLL will share the same code and data (C_WSA).

RMODE

Table 32. RMODE Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
RMODE=24			RMODE=ANY

To assign the residence mode for all the entry points into a program module, you can code the RMODE parameter as follows: RMODE={24/ANY/SPLIT}

UPCASE(YES | NO)

Table 33. UPCASE Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
UPCASE=NO			

The UPCASE option specifies that some additional rename processing is to be done. You should not confuse this option with the CASE(UPPER) option.

UPCASE by itself is equivalent to UPCASE(YES). The UPCASE(YES) option enforces the uppercase mapping of some symbol names. See "Rename Processing" on page 379 for its effect.

If you use the UPCASE option, external symbols in C programs are no longer case-sensitive. The binder does not support the use of the UPCASE option with C++ code. Therefore, you should use the RENAME control statement rather than the UPCASE option.

XREF(YES | NO)

Table 34. XREF Default

Binder Default	Batch	TSO (CXXBIND)	z/OS UNIX System Services Utilities - c89/cc/c++
XREF=NO			XREF=YES when the -V flag is set. XREF=NO otherwise.

In the z/OS UNIX System Services environment, the c89, cc, and c++ utilities specify XREF=YES when you use the -V flag.

The XREF(YES) option instructs the binder to generate a cross-reference list of data variables. If the XREF(NO) option is in effect, the binder does not generate a cross-reference list of data variables.

Binder Control Statements

Binder control statements specify how the binder processes its input.

The important binder control statements for a C/C++ programmer are the following (this is not a complete list):
- AUTOCALL
- ENTRY
- INCLUDE
- IMPORT
- LIBRARY
- NAME
- RENAME

You can place the control statements in a permanent data set that has the attributes RECFM=F or RECFM=FB, and LRECL=80.

If all of the information does not fit on one control statement, you can use one or more continuations. You must put a non-blank character in column 72 if you need to continue a control statement on the next record. The first column of the continued card that follows must be blank, and the statement must continue in column 2. The binder ignores leading blanks unless they are in a quoted string. You may optionally enclose a named token in single quotes.

You can specify input files on the INCLUDE, LIBRARY, and AUTOCALL statements as HFS pathnames rather than DD names. Pathnames can be distinguished from DD names by the preceding ″/″, which indicates an absolute pathname, or ″./″, which indicates a relative pathname.

AUTOCALL Control Statement

The AUTOCALL control statement causes the binder to perform an immediate (incremental) library search on the named library. Incremental autocall attempts to resolve any unresolved symbols at this point in the processing, using a single library or library concatenation. The binder searches the library before it processes more primary or secondary input.

The AUTOCALL control statement has the following syntax:

```
►►──AUTOCALL──library─────────────────────────────────────────────────►◄
```

library If *library* identifies the DD name of the library or library concatenation, it cannot exceed 8 bytes in length.

If *library* identifies an HFS filename, it cannot exceed 1024 bytes. The binder assumes that the file is an archive file. If it is an HFS directory file, then for purposes of symbol resolution, the binder uses the filenames of the files in the directory in the same way as it uses PDSE aliases and member names.

During incremental autocall, the binder ignores LIBRARY control statements and the CALL option.

ENTRY Control Statements

The ENTRY control statement specifies the entry point for program execution.

The ENTRY control statement has the following syntax:

```
►►──ENTRY──name───────────────────────────────────────────────────────►◄
```

name The name of the entry point for execution when the program is loaded.

By default, the program entry point for a C or C++ application is CEESTART. The program entry point is nominated in one of three ways (listed from weakest to strongest nomination).
1. The name of the first section that is processed by the binder
2. The name that is nominated in the object module (CEESTART for C/C++ main())
3. The name explicitly specified on an ENTRY control statement

IMPORT Control Statements

The IMPORT control statement describes an external function or variable to be imported, and the name of the DLL that contains its definition. The DLL name can be a PDS or PDSE member, or an HFS filename. The function or variable should be one that is being exported by a DLL.

If you do not specify DYNAM(DLL), the binder ignores the IMPORT control statement.

The IMPORT control statement has the following syntax:

```
►►──IMPORT──┬─CODE─┬──,dll-name──,identifier─────────────────────────────►◄
            └─DATA─┘
```

`CODE` | `DATA`	Specifies the type of contents of the module that the imported symbol represents. A function and a variable cannot have the same name.
dll-name	The directory name (primary member or alias) or HFS filename of the load module or program object that contains the imported function or variable. The maximum length of a dll-name is 1024 characters. The maximum length of an HFS filename is 255 bytes.
identifier	The name of the symbol (function or variable) that is to be imported. The name cannot be longer than 1024 characters. If the symbol has a C++ mangled name, then you must use the mangled name on the `IMPORT` statement. If the identifier contains lowercase letters, you must specify the binder option `CASE(MIXED)`.

Typically, a DLL has an associated definition side-deck of `IMPORT` control statements, which you include when you import functions or variables from that library. You can edit the records in the side file, or substitute your own `IMPORT` control statements so that some symbols are imported from DLLs in a different library.

If your program exports symbols, the binder may also generate an output file of corresponding `IMPORT` control statements. See “Output IMPORT Statements” on page 382.

INCLUDE Control Statements

You typically place `INCLUDE` control statements in DD `SYSLIN` to include multiple program objects, load modules, or object modules in primary input.

The `INCLUDE` control statement has the following syntax:

```
►►──INCLUDE──┬─ddname───┬──┬──────────────────────┬──────────────────►◄
             └─filename─┘  │    ┌─,──────────┐    │
                           │    ▼            │    │
                           └─(──────member───┴──)─┘
```

filename	is the name of the file to be included.
ddname	is a `DD` name associated with a file to be included.
member	is the member of the `DD` to be included.

The binder attempts to read the file that is specified.

LIBRARY Control Statement

You can use the `LIBRARY` control statement to resolve conflicts that you cannot resolve by changing the order of libraries in the `SYSLIB` concatenation.

To specify that the binder should never search for an unresolved reference *neversrch*, use the following syntax for the `LIBRARY` control statement:

```
              ┌─,─────────┐
              ▼           │
►►──LIBRARY──*──(───neversrch─┴──)─────────────────────────────►◄
```

neversrch The binder never searches for the reference that you marked as *neversrch*, on this bind step or on future rebinds.

To specify that the binder should not search for an unresolved reference *nosrch*, use the following syntax for the LIBRARY control statement:

```
           ┌─,───────┐
           ▼         │
►►──LIBRARY──(───nosrch───┴──)─────────────────────────────────►◄
```

nosrch An external reference which may be unresolved at the end of SYSLIN processing. Automatic library call in SYSLIB does not search for such references on this bind step. Case-sensitivity is maintained you enclose *nosrch* in single quotes.

To direct the binder to search for an unresolved reference *srch* in a particular library, use the following syntax of the LIBRARY control statement:

```
                     ┌─,──────┐
                     ▼        │
►►──LIBRARY──ddname──(───srch───┴──)───────────────────────────►◄
```

ddname The name of a DD that defines a library (PDS or PDSE), or a concatenation of one or more PDS or PDSEs.

srch An external reference which may be a variable or a function. Should this symbol be unresolved after SYSLIN is processed, and library search is requested, the libraries pointed to by SYSLIB are not searched. Rather, the library (PDS or PDSE) that is defined by ddname will be searched for an alias or a member of name *srch*. See "Generating Aliases for Automatic Library Call (Library Search)" on page 379. If the binder finds the member, it reads it as input to the bind step. If you enclose *srch* in single quotes, the search is case-sensitive.

For example, if you have a program that has both Fortran and C code, both libraries define the member ABS and COS. You want the member COS from the Fortran library, and the member ABS from the C library. Your LIBRARY control statement would be similar to the following:

```
LIBRARY DDFORT(COS)
LIBRARY DDCLIB(ABS)
```

If you do not use the LIBRARY control statement, you will get both members from the C library or both members from the Fortran library.

NAME control statement

The NAME control statement specifies the name of the program object that is output to SYSLMOD. The NAME control statement has the following syntax:

```
►►──NAME──member_name──┬─────┬──────────────────────────────────────────►◄
                       └─(R)─┘
```

member_name	A PDS or PDSE library member name, or an HFS file name.
R	If you use the option R and the name that you specify already exists, the binder will replace the existing member with the output program object.

The output from the binder can be a single program object, or multiple program objects generated by using multiple NAME control statements.

RENAME Control Statement

The RENAME control statement requests the binder to rename the references to a symbol that remains unresolved at the end of the first pass of final autocall processing of SYSLIB. See "Final Autocall Processing (SYSLIB)" on page 378.

You can use the RENAME control statement to resolve case differences in function names.

The RENAME control statement has the following syntax:

```
►►──RENAME──old-name──,new-name──────────────────────────────────────────►◄
```

old-name	The function to be renamed. Maximum length is 1024.
new-name	The name to which old-name may be changed. Maximum length is 1024.

When the binder reads a RENAME control statement, it adds the request to the list of such requests. Nothing else is done until rename processing. See "Rename Processing" on page 379.

Chapter 8. Runtime Options

This chapter describes how to specify runtime options and `#pragma runopts` preprocessor directives available to you with z/OS C/C++ and z/OS Language Environment. For a detailed description of the z/OS Language Environment runtime options and information about how to apply them in different environments, refer to the *z/OS Language Environment Programming Reference*.

Specifying Runtime Options

To allow your application to recognize runtime options, either the `EXECOPS` compiler option, or the `#pragma runopts(execops)` directive must be in effect. The default compiler option is `EXECOPS`.

You can specify runtime options as follows:

- At execution time in one of the following ways:
  - On the `GPARM` option of the IBM-supplied cataloged procedures
  - On the option list of the TSO `CALL` command
  - On the `PARM` parameter of the `EXEC PGM=`*your-program-name* JCL statement
  - On the exported `_CEE_RUNOPTS` environment variable under the z/OS shell
- At compile time, on a `#pragma runopts` directive in your `main` program

If `EXECOPS` is in effect, use a slash '/' to separate runtime options from arguments that you pass to the application. For example:

```
GPARM='STORAGE(FE,FE,FE)/PARM1,PARM2,PARM3'
```

If `EXECOPS` is in effect, Language Environment interprets the character string that precedes the slash as runtime options. It passes the character string that follows the slash to your application as arguments. If no slash separates the arguments, Language Environment interprets the entire string as an argument.

If `EXECOPS` is not in effect, Language Environment passes the entire string to your application.

If you specify two or more contradictory options (for example in a `#pragma runopts` statement), the last option that is encountered is accepted. Runtime options that you specify at execution time have higher precedence than those specified at compile time.

For more information on the precedence and specification of runtime options for applications that are compiled with the z/OS Language Environment, refer to the *z/OS Language Environment Programming Reference*.

Using the #pragma runopts Preprocessor Directive

You can use the `#pragma runopts` preprocessor directive to specify z/OS Language Environment runtime options. You can also use `#pragma runopts` to specify the compiler options `ARGPARSE`, `ENV`, `PLIST`, `REDIR`, and `EXECOPS`. If you specify the compiler option, it has precedence over the `#pragma runopts` directive.

When the runtime option `EXECOPS` is in effect, you can specify runtime options at execution time, as previously described. These options override runtime options that you compiled into the program by using the `#pragma runopts` directive.

The #pragma runopts directive can appear in any file: main, include, or source. You can specify multiple runtime options per directive or multiple directives per compilation unit. If you want to specify the ARGPARSE or REDIR options, the #pragma runopts directive must be in the same compilation unit as main(). Neither runtime option has an effect on programs invoked under the z/OS shell. This is because the shell program handles the parsing and redirection of command line arguments within that environment.

When you specify multiple instances of #pragma runopts in separate compilation units, the compiler generates a CSECT for each compilation unit that contains a #pragma runopts directive. When you link multiple compilation units that specify #pragma runopts, the linkage editor takes only the first CSECT, thereby ignoring your other option statements. Therefore, you should always specify your #pragma runopts directive in the same source file that contains the function main().

For more information on the #pragma runopts preprocessor directive, see the *z/OS C/C++ Language Reference*.

Part 3. Compiling, Binding, and Running z/OS C/C++ Programs

This part describes how to compile, bind, and run z/OS C/C++ programs using z/OS Language Environment in the following sections:

Chapter 9. Compiling

This chapter describes how to compile your program with the z/OS C/C++ compiler and z/OS Language Environment. For specific information about compiler options see “Chapter 6. Compiler Options” on page 59.

The z/OS C/C++ compiler analyzes the source program and translates the source code into machine instructions that are known as *object code*.

You can perform regular compilations under z/OS batch, TSO, or the z/OS shell.

Input to the z/OS C/C++ Compiler

The following sections describe how to specify input to the z/OS C/C++ compiler for a regular compilation, or the IPA Compile step. For information about input for the IPA Link step, refer to “Chapter 11. Using the IPA Link Step with z/OS C/C++ Programs” on page 329.

If you are compiling a C or C++ program, input for the compiler consists of the following:
- Your z/OS C/C++ source program
- The z/OS C/C++ standard header files including IBM-supplied Class Library header files
- Your header files

When you invoke the z/OS C/C++ compiler, the operating system locates and runs the compiler. To run the compiler, you need these default data sets supplied by IBM:
- `CBC.SCBCCMP`
- `CEE.SCEERUN`

The locations of the compiler and the runtime library were determined by the system programmer who installed the product. The compiler and library should be in the `STEPLIB`, `JOBLIB`, `LPA`, or `LNKLST` concatenations. `LPA` can be from either specific modules (`IEALPAxx`) or a list (`LPALSTxx`). See the cataloged procedures shipped with the product in “Appendix D. Cataloged Procedures and REXX EXECs” on page 529.

HFS file names: Unless they appear in JCL, file names which contain the special characters blank, backslash, and double quote must escape these characters. The escape character is backslash (\).

Primary Input

For a C or C++ program (except for the IPA Link step), the primary input to the compiler is the data set that contains your C/C++ source program. If you are running the compiler in batch, identify the input source program with the `SYSIN DD` statement. You can do this by either defining the data set that contains the source code or by placing your source code directly in the JCL stream. In TSO or in z/OS UNIX System Services, identify the input source program by name as a command line argument. The primary input source file can be any one of the following:
- A sequential data set
- A member of a partitioned data set
- All members of a partitioned data set
- A hierarchical file system (HFS) file
- All HFS files in a directory

Secondary Input

For a C or C++ program (except for the IPA Link step), secondary input to the compiler consists of data sets that contain `#include` files. If you are compiling a new z/OS C/C++ program, use the `LSEARCH` compiler option instead of `USERPATH` and `USERLIB`, and `SEARCH` instead of `SYSPATH` and `SYSLIB`. `SEARCH` and `LSEARCH` provide greater flexibility in names and locations of `#include` files.

For more information on the use of these compiler options, see “LSEARCH | NOLSEARCH” on page 138 and “SEARCH | NOSEARCH” on page 167. For more information on naming `#include` files, see “Specifying Include File Names” on page 309. For information on how the compiler searches for `#include` files, see “Search Sequences for Include Files” on page 316. For more information on include files, refer to “Using Include Files” on page 308.

Output from the Compiler

You can specify compiler output files as one or more of the following:
1. A sequential data set
2. A member of a partitioned data set
3. A partitioned data set
4. A hierarchical file system (HFS) file
5. An HFS directory

For valid combinations of input file types and output file types, refer to Table 37 on page 287 .

Specifying Output Files

You can use compile options to specify compilation output files as follows:

Table 35. Compile Options That Provide Output File Names

Output File Type	**Compiler Option**
Object Module	OBJECT(filename)
Listing File	SOURCE (filename), LIST(filename), INLRPT(filename)
Preprocessor Output	PPONLY(filename)
Template Output	TEMPINC(location)
Precompiled Header Output	GENPCH(location)

When compiler options that generate output files are specified without suboptions to identify the output files, and the ddnames are not allocated, the output file names are generated based on the name of the source file. For data sets, the compiler generates a low-level qualifier by appending a suffix to the data set name of the source, as Table 36 on page 285 shows.

For example, under TSO, the compiler generates the object file `'userid.TEST.SRC.OBJ'` if you compile the following:

```
cc TEST.SRC (OBJ
```

The compiler generates the object file `'userid.TEST.SRC.OBJ(HELLO)'` if you compile the following:

```
cc 'hlqual.TEST.SRC(HELLO)' (OBJ
```

If you compile source from HFS files without specifying output filenames in the compiler options, the compiler writes the output files are to the current working directory. The compiler does the following to generate the output file names:
- appends a suffix, if it does not exist
- replaces the suffix, if it exists

The following default suffixes are used:

Table 36. Default Suffixes for Output File Types

Output File Type.	z/OS File	HFS File
Object Module	OBJ	o
Listing File	LIST	lst
Preprocessor Output	EXPAND	i
Template Output	TEMPINC	./tempinc
Precompiled Header Output	PCH, PCHPP	pch, pchpp

Notes:

1. Output files default to the HFS directory if the source resides in the HFS, or to the z/OS data set if the source resides in a data set.
2. If you have specified the `OE` option, see “OE | NOOE” on page 150 for a description of the default naming convention.
3. If you supply inline source in your JCL, you must provide a file name for the output, or route it to the job log. The compiler will not generate an output file name automatically. You can specify a file name either as a suboption for a compiler option, or on a ddname in your JCL.
4. If you are using `#pragma` options to specify a compile-time option that generates an output file, you must use a ddname to specify the output file name. The compiler will not automatically generate file names for output that is created by `#pragma` options.

Listing Output

Note: Although the compiler listing is for your use, it is not a programming interface and is subject to change.

To create a listing file that contains source, object or inline reports use the `SOURCE`, `LIST`, or `INLRPT` compile options. The listing includes the results of the default or specified options of the `CPARM` parameter (that is, the diagnostic messages and the object code listing). If you specify *filename* with two or more of these compile options, the compiler combines the listings and writes them to the last file specified in the compile options. If you did not specify *filename*, the listing will go to the `SYSCPRT` DD name, if you allocated it. Otherwise, the compiler generates a default file name as described in “LIST | NOLIST” on page 133.

Object Module Output

To create an object module and store it on disk or tape, you can use either the `OBJECT` or `DECK` (`C only`) compiler options.

If you do not specify *filename* with the `OBJECT` option, the compiler stores the object code in the file that you define in the `SYSLIN DD` statement. With the `DECK` compiler option, the compiler uses the file that you define in the `SYSPUNCH DD`. If you did not specify a suboptions, and did not allocate SYSLIN, the compiler generates a default file name, as described in “OBJECT | NOOBJECT” on page 148.

Differences in Object Modules under IPA: The object module that a regular compilation generates is different from the object module that the IPA Compile step generates. The IPA Compile step and regular compilation both produce an object module for each source file successfully processed. For the IPA Compile step, however, the output is an IPA-optimized object file, or a combined IPA/conventional object file (if you do not specify the `NOOBJECT` suboption of the `IPA` compiler option). You can use the object file that the `IPA(NOLINK,NOOBJECT)` compiler option creates as input to the IPA Link step only. It contains an external reference to `@@DOIPA`, which remains unresolved until IPA Link step processes the file. If you attempt to bind an IPA object file that was created by using the `IPA(NOLINK,NOOBJECT)` option, the binder issues an error message.

Refer to "Valid Input/Output File Types" for information about valid input/output file types.

Preprocessor Output

If you specify *filename* with the `PPONLY` compile option, the compiler writes the preprocessor output to that file. If you do not specify *filename* with the `PPONLY` option, the compiler stores the preprocessor output in the file that you define in the `SYSUT10 DD` statement. If you did not allocate `SYSUT10`, the compiler generates a default file name, as described in "PPONLY | NOPPONLY" on page 160.

Template Instantiation Output

If you specify *location*, which is either an HFS directory or a PDS, with the `TEMPINC` compile option, the compiler writes the template instantiation output to that location. If you do not specify *location* with the `TEMPINC` option, the compiler stores the `TEMPINC` output in the file that is associated with the `TEMPINC` DD name. If you did not allocate `DD:TEMPINC`, the compiler determines a default destination for the template instantiation files. See "TEMPINC | NOTEMPINC" on page 185 for more information on this default.

Valid Input/Output File Types

Depending on the type of file that is used as primary input, certain output file types are allowed. The following table describes these combinations of input and output files:

Table 37. Valid Combinations of Source and Output File Types

Input Source File	Output Data Set Specified Without (member) Name, for example `A.B.C`	Output Data Set Specified as filename(member), for example `A.B.C(D)`	Output Specified as HFS File, for example `a/b/c.o`	Output Specified as HFS Directory, for example `a/b`
Sequential Data Set, for example `A.B`	1. If file exists as a sequential data set, overwrites it 2. If file does not exist, creates sequential data set 3. Otherwise compilation fails	1. If PDS does not exist, creates the PDS and adds a member into the data set 2. If PDS exists and member does not exist, adds member in the PDS 3. If PDS and member both exist, then overwrites the member.	1. If the directory does not exist, compilation fails 2. If the directory exists but the file does not exist, creates file 3. If the file exists, overwrites the file.	Not supported
A member of a PDS using (member), for example `A.B(C)`	1. If the file exists as a sequential data set, overwrites it 2. If the file exists as a PDS, creates or overwrites member 3. If file does not exist, creates PDS and member	1. If PDS does not exist, creates PDS and member 2. If PDS exists and member does not exist, adds member 3. If PDS exists and member also exists, overwrites it	1. If directory does not exist, compilation fails 2. If directory exists and the file with the specified filename does not exist, creates file 3. If the directory exists and the file exists, overwrites file	1. If directory does not exist, compilation fails 2. If directory exists and the file with the filename *MEMBER.ext* does not exist, creates file 3. If directory exists and the file with the filename *MEMBER.ext* also exists, overwrite file
All members of a PDS, for example `A.B`	1. If file exists as a PDS, creates or overwrites members 2. If file does not exist, creates PDS and members 3. Otherwise compilation fails	Not Supported	Not Supported	1. If directory does not exist, compilation fails 2. If directory exists and the files with the filenames *`MEMBER.ext`* do not exist, creates files 3. If directory exists and the files with the filenames *`MEMBER.ext`* exist, overwrites them

Table 37. Valid Combinations of Source and Output File Types (continued)

Input Source File	Output Data Set Specified Without (member) Name, for example `A.B.C`	Output Data Set Specified as filename(member), for example `A.B.C(D)`	Output Specified as HFS File, for example `a/b/c.o`	Output Specified as HFS Directory, for example `a/b`
HFS file, for example `/a/b/d.c`	1. If file exists as a sequential data set, overwrites it 2. If file does not exist, creates sequential data set 3. Otherwise compilation fails	1. If PDS does not exist, creates the PDS and stores a member into the data set 2. If PDS exists and member does not exist, then add the member in the PDS 3. If PDS and member both exist, then overwrites the member.	1. If the directory does not exist, compilation fails 2. If the directory exists but the file does not exist, creates file 3. If the file exists, overwrites the file.	1. If the directory does not exist, compilation fails 2. If the directory exists and the file does not exist, creates it 3. If the directory exists and the file exists, overwrites it
HFS Directory, for example `a/b/`	Not supported	Not supported	Not supported	1. If the directory does not exist, compilation fails 2. If the directory exists and the files to be written do not exist, creates them 3. If the directory exists and the files to be written already exist, overwrites them

Compiling Under z/OS Batch

To compile your z/OS C/C++ source program under z/OS batch, you can either use cataloged procedures that IBM supplies, or write your own JCL statements.

Using Cataloged Procedures for z/OS C

You can use one of the following IBM-supplied cataloged procedures. Each procedure includes a compilation step to compile your program.

`EDCC`	Compile
`EDCCB`	Compile and bind
`EDCCBG`	Compile, bind, and run
`EDCI`	Run the IPA Link step
`EDCCLIB`	Compile and maintain an object library
`EDCCL`	Compile and link-edit
`EDCCPLG`	Compile, prelink, link-edit, and run

`EDCCLG`	Compile, link-edit, and run
`EDCXCB`	Compile and bind an `XPLINK` C program
`EDCXCBG`	Compile, bind, and run an `XPLINK` C program

IPA Considerations

Only the `EDCC` procedure applies to the IPA Compile step. Only the `EDCI` procedure applies to the IPA Link step.

To run the IPA Compile step, use the `EDCC` procedure, and ensure that you specify the `IPA(NOLINK)` or `IPA` compiler option. Note that you must also specify the `LONGNAME` compiler option or the `#pragma longname` directive.

To create an IPA-optimized object module, you must run the IPA Compile step for each source file in your program, and the IPA Link step once for the entire program. Once you have successfully created an IPA-optimized object module, you must bind it to create the final executable.

Using Cataloged Procedures for z/OS C++

You can use one of the following cataloged procedures that IBM supplies. Each procedure includes a compilation step to compile your program.

`CBCC`	Compile
`CBCCB`	Compile and bind
`CBCCBG`	Compile, bind, and run
`CBCI`	Run the IPA Link step
`CBCCL`	Compile, prelink, and link
`CBCCLG`	Compile, prelink, link, and run
`CBCXCB`	Compile and bind an `XPLINK` C++ program
`CBCXCBG`	Compile, bind, and run an `XPLINK` C++ program

See “Appendix D. Cataloged Procedures and REXX EXECs” on page 529 for more information on cataloged procedures.

IPA Considerations

Only the `CBCC` procedure applies to the IPA Compile step. Only the `CBCI` procedure applies to the IPA Link step.

To run the IPA Compile step, use the `CBCC` procedure, and ensure that you specify the `IPA(NOLINK)` or `IPA` compiler option. Note that you must also specify the `LONGNAME` compiler option or the `#pragma longname` directive.

To create an IPA-optimized object module, you must run the IPA Compile step for each source file in your program, and the IPA Link step once for the entire program. Once you have successfully created an IPA-optimized object module, you must bind it to create the final executable.

Using Special Characters

When invoking the compiler directly, if a string contains a single quote (') it should be written as two single quotes ('') as in:

```
//COMPILE EXEC PGM=CBCDRVR,PARM='OPTFILE(''USERID.OPTS'')'
```

If you are using the same string to pass a parameter to a JCL PROC, use four single quotes (''''), as follows:

```
//COMPILE EXEC CBCC,CPARM='OPTFILE(''''USERID.OPTS'''')'
```

A backslash need not precede special characters in HFS file names that you use in DD cards. For example:

```
//SYSLIN DD PATH='/u/user1/obj 1.o'
```

A backslash must precede special characters in HFS file names that you use in the PARM statement. For example:

```
//STEP1 EXEC PGM=CBCDRVR,PARM='/u/user1/obj\ 1.o'
```

Using Your Own JCL

The following example shows sample JCL for compiling a C program:

```
//jobname    JOB   acctno,name...
//COMPILE    EXEC  PGM=CBCDRVR,
// PARM='/SEARCH(''CEE.SCEEH.+'') NOOPT SO OBJ OPTFILE(DD:CPATH)'
//STEPLIB    DD    DSNAME=CEE.SCEERUN,DISP=SHR
//           DD    DSNAME=CBC.SCBCCMP,DISP=SHR
//SYSLIN     DD    DSNAME=MYID.MYPROG.OBJ(MEMBER),DISP=SHR
//SYSPRINT   DD    SYSOUT=*
//SYSIN      DD    DATA,DLM=@@
#include <stdio.h>
   ⋮
int main(void)
{
/*  comment   */
   ⋮
}
@@
//SYSUT1     DD    DSN=...
//SYSUT4     DD    DSN=...
   ⋮
//*
```

Figure 20. JCL for Compiling a C Program (for NOOPT, SOURCE, and OBJ)

The following example shows sample JCL for compiling a C++ program:

```
 //jobname    JOB   acctno,name...
 //COMPILE EXEC PGM=CBCDRVR,
 // PARM='/CXX SEARCH(''CEE.SCEEH.+'',''CBC.SCLBH.+''),NOOPT,SO,OBJ'
 //STEPLIB  DD  DSN=CEE.SCEERUN,DISP=SHR
 //         DD  DSN=CBC.SCBCCMP,DISP=SHR
 //SYSLIN   DD  DSN=MYID.MYPROJ.OBJ,DISP=SHR
 //SYSPRINT DD  SYSOUT=*
 //SYSIN    DD  DATA,DLM=@@
#include <stdio.h>
#include <iostream.h>
   ⋮
int main(void)
{
//  comment
   ⋮
}
@@
 //SYSUT1   DD    DSN=...
 //SYSUT4   DD    DSN=...
   ⋮
//*
```

Figure 21. JCL for Compiling a C++ Program (for NOOPT, SOURCE, and OBJ)

Use JCL to define your jobs and job steps to the operating system. Describe the steps you want the operating system to perform, and specify the resources that are required by the job. The JCL statements that are essential for running a job are:

- A `JOB` statement that identifies the start of the job
- An `EXEC` statement that identifies a job step and the program to be executed either directly or by a cataloged procedure
- `DD` (data definition) statements that identify the input/output facilities that the program that is executed in the job step requires
- JES control statements that provide information to the Job Entry Subsystem

For more information about JCL, refer to the publications that are listed in the *z/OS Information Roadmap*.

Specifying Source Files

For non-HFS files, use this format of the SYSIN JCL:

```
//SYSIN DD DSNAME=dsname,DISP=SHR
```

If you specify a PDS without a member name, all members of that PDS are compiled.

Note: If you specify a PDS as your primary input, you must specify either a PDS or an HFS directory for your output files.

For HFS files, use this format of the SYSIN JCL:

```
//SYSIN DD PATH='pathname'
```

You can specify compilation for a single file or all source files in an HFS directory, for example:

```
//SYSIN DD PATH='/u/david'
//* All files in the directory /u/david are compiled
```

Note: If you specify an HFS directory as your primary input, you must specify an HFS directory for your output files.

When you place your source code directly in the input stream, use the `SYSIN DD` statement as follows:

```
//SYSIN DD DATA,DLM=@@
```

rather than:

```
//SYSIN DD *
```

When you use the `DD *` convention, the first C/C++ comment statement that starts in column 1 will terminate the input to the compiler. This is because /*, the beginning of a C or C++ comment, is also the default delimiter.

Note: To treat columns 73 through 80 as sequence numbers, use the `SEQUENCE` compiler option.

For more information about the `DD *` convention, refer to the publications that are listed in the *z/OS Information Roadmap*.

Specifying Include Files

Use the `SEARCH` option to specify system include files, and the `LSEARCH` option to specify your include files. For example:

```
//C EXEC PGM=CBCDRVR,PARM='/CXX SEARCH(''CEE.SCEEH.+'',''CBC.SCLBH.+'')'
```

You can also use the `SYSLIB` and `USERLIB DD` statements (note that the `SYSLIB` DD statement has a different use if you are running the IPA Link step). To specify more than one library, concatenate multiple `DD` statements as follows:

```
//SYSLIB     DD     DSNAME=USERLIB,DISP=SHR
//           DD     DSNAME=DUPX,DISP=SHR
```

Note: If the concatenated data sets have different block sizes, either specify the data set with the largest block size first, or use the `DCB=`*dsname* subparameter on the first `DD` statement. For example:

```
//USERLIB DD DSNAME=TINYLIB,DISP=SHR,DCB=BIGLIB
//        DD DSNAME=BIGLIB,DISP=SHR
```

where `BIGLIB` has the largest block size. For rules regarding concatenation of data sets in JCL, refer to the *z/OS C/C++ Programming Guide*.

Specifying Output Files

You can specify output file names as suboptions to the compiler. You can direct the output to a PDS member as follows:

```
//    CPARM='LIST(MY.LISTINGS(MEMBER1))'
```

You can direct the output to an HFS file as follows:

```
//    CPARM='LIST(./listings/member1.lst)'
```

You can also use DD statements to specify output file names.

To specify non-HFS files, use DD statements with the DSNAME parameter. For example:

```
//SYSLIN DD DSN=USERID.TEST.OBJ(HELLO),DISP=SHR
```

To specify HFS directories or files, use DD statements with the PATH parameter.

```
//SYSLIN DD PATH='/u/david/test.o',PATHOPTS=(OWRONLY,OCREAT,OTRUNC)
```

on `PATH` and `PATHOPTS` parameters.

Note: Use the `PATH` and `PATHOPTs` parameters when specifying HFS files in the DD statements. For additional information on these parameters, refer to the list of publications in *z/OS Information Roadmap*.

If you do not specify the output filename as a suboption, and do not allocate the associated ddname, the compiler generates a default output file name. These are the two situations in which the compiler will not generate a default file name:
- You supply instream source in your JCL.
- You are using #pragma options to specify a compile-time option that generates an output file.

Compiling Under TSO

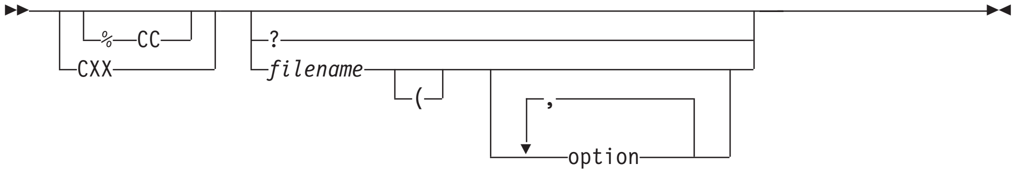
You can invoke the z/OS C/C++ compiler under TSO in any of the following ways:
- Foreground execution from TSO READY
- Foreground execution from the ISPF command line or the ISPF menu option 6
- Foreground execution from ISPF menu option 4
- Foreground execution from an ISPF edit session
- Background execution (batch) from ISPF menu option 5

All methods of foreground execution call the `CC` or `CXX` REXX EXECs supplied by IBM.

Note: To run the compiler under TSO, you must have access to the runtime libraries. To ensure that you have access to the runtime library and compiler, do one of the following:
- If you are running under ISPF in the foreground, concatenate the libraries to `ISPLLIB`.
- Have your system programmer add the libraries to the LPALST or LPA.
- Have your system programmer add the libraries to the LNKLST.
- Have your system programmer change the `LOGON PROC` so the libraries are added to the `STEPLIB` for the TSO session.
- Have your system programmer customize the REXX EXEC CBC3C00E, which is called by the `CC`, `CXX`, and other EXECs to set up the environment.

Using the CC and CXX REXX EXECs

You can use the `CC` REXX EXEC to invoke the z/OS C compiler, and the `CXX` REXX EXEC to invoke the z/OS C++ compiler. These REXX EXECs share the same syntax:



where

%
: ensures that the REXX EXEC CC is invoked, not the z/OS UNIX System Services cc utility.

option
: is any valid compiler option

filename
: can be one of the following:

1. A sequential data set
2. A member of a partitioned data set
3. All members of a partitioned data set
4. A hierarchical file system (HFS) file
5. All HFS files in a directory

If *filename* is not immediately recognizable as an HFS file or data set, it is assumed to be a data set. Prefix the file name with // to identify it as a data set, and with ./ or / to identify it as an HFS file. For more information on file naming considerations refer to the *z/OS C/C++ Programming Guide*.

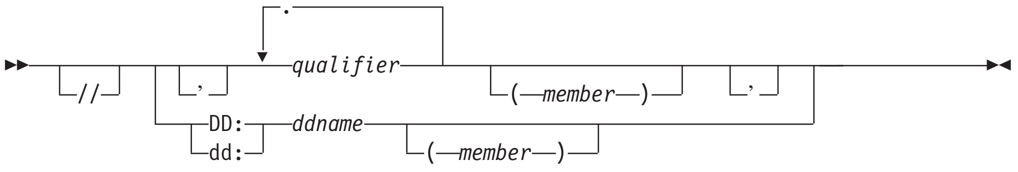
If you invoke either CC or CXX with no arguments or with only a single question mark, the appropriate preceding syntax diagram is displayed.

If you are using #pragma options to specify a compile-time option that generates an output file, you must use a ddname to specify the output file name. The compiler will not automatically generate file names for output that is created by #pragma options.

Unless CBC3C00E has been customized, the default SYSLIB for CC is CEE.SCEEH.H, and CEE.SCEEH.SYS.H concatenated. If you want to override the default SYSLIB that is allocated by the CC exec, you must allocate the ddname SYSLIB **before** you invoke CC. If you did not allocate the ddname SYSLIB before you invoked CC EXEC, the CC EXEC allocates the default SYSLIB.

Specifying Sequential and Partitioned Data Sets

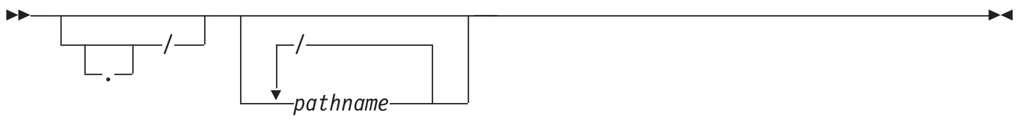
To specify a sequential or partitioned data set for your source file use the following syntax:



Note: If you use the leading single quote to indicating a fully qualified data set name, you must also use the trailing single quote.

Specifying HFS Files or Directories

You can use the `CC` or `CXX` REXX EXECs to compile source code that is stored in HFS files and directories. Use the following syntax when specifying HFS file or directory as your input or output file:



If you specify an HFS directory, all the source files in that directory are compiled. In the following example all the files in `/u/david/src` are compiled:

```
CC /u/david/src
```

When the file name contains the special characters double quote, blank or backslash, you must precede these characters with a backslash, as follows:

```
CC  /u/david/db\ 1.c
CC  file\"one
```

When you use the `CC` or `CXX` REXX EXEC, you must use unambiguous HFS source file names. For example, the following input files are HFS files:

```
CXX ./test/hello.c
CC  /u/david/test/hello.c
CXX test/hello.c
CC  ///hello.c
CC  ../test/hello.c
```

If you specify a filename that does not include pathnames with single slashes, the compiler treats the file as a non-HFS file. The compiler treats the following input files as non-HFS files:

```
CXX hello.c
CC  //hello.c
```

Using Special Characters

When HFS file names contain the special characters blank, backslash, and double quote, you must precede the special character with a backslash(\).

When suboptions contain the special characters left bracket `(`, right bracket `)`, comma, backslash, blank and double quote, you must precede these characters with a double backslash(\\) to ensure that they are interpreted correctly, as in:

```
    def(errno=\\(*__errno\\(\\)\\))
```

Note: Under TSO, you must precede special characters by a backslash \ in both file names and options.

Specifying Compiler Options under TSO

When you use REXX EXECs supplied by IBM, you can override the default compiler options by specifying the options directly on the invocation line after an open left parenthesis `(`. The following example specifies, multiple compiler options with the sequential file `STUDENT.GRADES.CXX`:

```
       CXX 'STUDENT.GRADES.CXX'
              ( LIST,TEST,
               LSEARCH(MASTER.STUDENT,COURSE.TEACHER),
               SEARCH(VGM9.FINANCE,SYSABC.REPORTS),
               OBJ('GRADUATE.GRADES.OBJ(REPORT)')
```

See “Summary of Compiler Options” on page 66 for more information on compiler options.

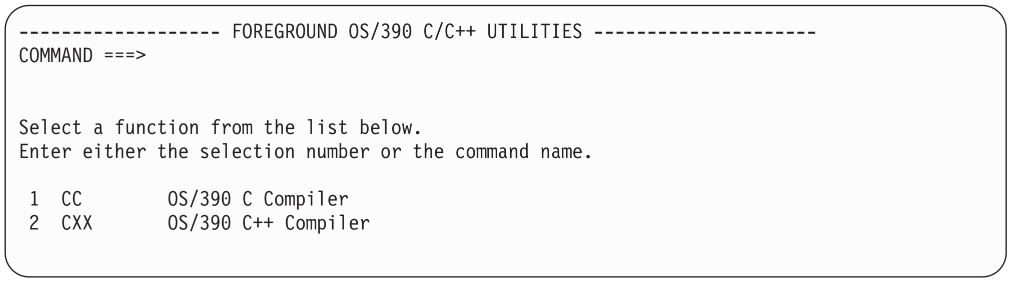
Using ISPF to Invoke the Compiler

Under TSO, you can use ISPF foreground and batch compile panels to start the z/OS C/C++ compiler. You can use online help with these panels.

Note: You cannot use ISPF to invoke the IPA Link step.

Foreground Processing

1. Select the `Foreground` option (4) from the `ISPF-PDF PRIMARY OPTION MENU`. The `FOREGROUND SELECTION PANEL` is presented.
2. Select the `C/C++` option (19). The `FOREGROUND UTILITIES` panel is presented, as shown in Figure 22.



```
------------------- FOREGROUND OS/390 C/C++ UTILITIES ---------------------
COMMAND ===>


Select a function from the list below.
Enter either the selection number or the command name.

 1  CC        OS/390 C Compiler
 2  CXX       OS/390 C++ Compiler
```

Figure 22. Foreground IBM z/OS C/C++ Utility Panel

3. Select `1` to get the `FOREGROUND z/OS C COMPILE` panel, or select `2` to get the `FOREGROUND z/OS C++ COMPILE` panel as shown in Figure 23 on page 297.
4. Enter information such as your source data, password, object data set name, compiler options, and additional input libraries (as necessary).

```
------------------------- FOREGROUND z/OS C++ COMPILE ------------------
Command ===>

ISPF Library:           1
   Project ===> USERID
   Group   ===> DEV        ===>          ===>          ===>
   Type    ===> CXX
   Member  ===> *         (Blank or pattern for member selection list)
                          (* for entire PDS)

Other Partitioned or Sequential Data Set:
   Data Set Name  ===>          2

Data Set Password ===>          (If password protected)  3

Compiler Options:
           ===>                 4
           ===> SEARCH('CEE.SCEEH.+','CBC.SCLBH.+')
           ===> USERPATH(/USERID/DEV/INCL)
           ===>
           ===> OPT
           ===>
```

Figure 23. Foreground IBM z/OS C++ Compile Panel

1 The `ISPF Library` field is used if you do not specify a data set in **2**. Input to the compiler is either a member of an ISPF library, a member of a partitioned data set, or a sequential data set. Fill in the `Project`, `Group`, `Type`, and `Member` fields. To can compile the entire PDS, place an asterisk (*) in the member field.

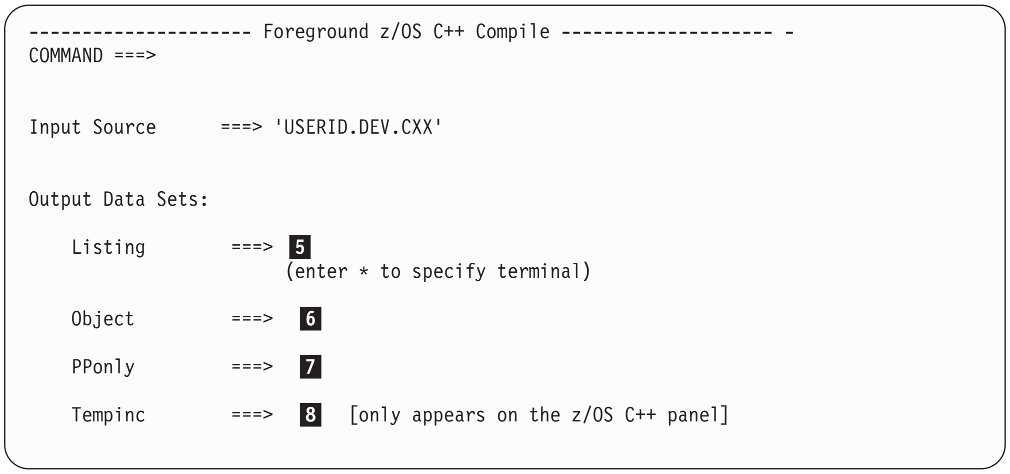
Note: If the source data set is partitioned and you did not specify a member, you are presented with a member list from which to choose the desired member.

2 Use the `Other Partitioned or Sequential Data Set` field if your input source is one of the following:
- a sequential data set
- a PDS with a number of qualifiers not equivalent to three.

If you specify data sets in both **1** and **2**, the data set that you specified in this field is used. To compile an entire PDS of source instead of an individual PDS member, enter the PDS name followed by (*). You can specify a member of a PDS by entering the member name in parentheses after the data set name.

3 If any of your data sources are password protected, you must specify the password in the `Data Set Password` field.

4 Use the `Compiler Options` field, specify the compiler options that you want to use. For a complete list of compiler options, see "Compiler Option Defaults" on page 63.



```
-------------------- Foreground z/OS C++ Compile -------------------- -
COMMAND ===>


Input Source      ===> 'USERID.DEV.CXX'


Output Data Sets:

    Listing        ===> 5
                        (enter * to specify terminal)

    Object         ===>  6

    PPonly         ===>  7

    Tempinc        ===>  8   [only appears on the z/OS C++ panel]
```

Figure 24. Foreground IBM z/OS C++ Compile Panel (2)

Note for z/OS C: The only difference in the appearance of the panels for z/OS C is in the heading, and the absence of the `Tempinc` option.

5. Enter names of the desired output data sets.

   5 Use the `Listing` field to specify a name for the listing data set. If you leave this field blank, the compiler generates a default name. See "Specifying Output Files" on page 284 for information on the defaults. To generate a listing data set, you must specify the compiler option `SOURCE` or `LIST` under `Compiler Options`.

   6 Use the `Object Data Set` field to specify a name for the object data set. If you leave this field blank, the compiler generates a default name. See "Specifying Output Files" on page 284 for information on the defaults.

   7 Use the `PPonly` field to specify a name for the PPONLY data set. If you leave this field blank, the compiler generates a default name. You can also specify the `LINES` or `COMMENTS` suboptions in this field. See "PPONLY | NOPPONLY" on page 160 for more information.

   8 For z/OS C++, use the `Tempinc` field to specify a PDS name for the template instantiation files. If you leave this field blank, the PDS is given the name `TEMPINC`.

6. Press `Enter` to invoke the foreground processing program.

HFS Note: You cannot use HFS files as input to the ISPF panels, but you can target your output to HFS files through the compiler options.

Batch Processing

Use the `batch` option to invoke the compiler as a batch job. JCL is generated for the job on the basis of the information you enter on the batch processing panels, and the job is submitted for execution.

When you choose the `batch` option from the `ISPF-PDF PRIMARY OPTION MENU`, the `BATCH SELECTION PANEL` is shown. Notice the `SOURCE DATA ONLINE` option and the `JOB STATEMENT INFORMATION` area at the bottom of this panel.

The `SOURCE DATA ONLINE` option specifies whether or not to check if the data set is available. If you specify `YES`, `ISPF` checks to see if the data set exists. If it does not exist, you receive an ISPF message to indicate that the data set was not catalogued. If you specify NO, ISPF does not check for the data set.

The JOB STATEMENT INFORMATION consists of four lines for JCL card images. These lines are submitted as part of the batch job, so you must follow all the rules of JCL.

Alternatively, choose the IBM z/OS C/C++ Compiler option to show the BATCH IBM z/OS C/C++ COMPILE panels. These panels are similar to the FOREGROUND IBM z/OS C/C++ COMPILE panels. Most of the fields, such as the ISPF library, Other Partitioned, or Sequential Data Set, and Compiler Options behave the same way. See "Foreground Processing" on page 296 for descriptions.

The Batch option does not support passwords. If your input or output data sets are password protected, use the Foreground option. If you submit a job that includes a password-protected data set, the system operator is requested to enter the required password.

Use the Listing Data Set and SYSOUT Class fields to send the compiler's list output directly to a SYSOUT queue or into a data set. If you fill in both fields, the value for SYSOUT Class is used.

Enter the source information and other parameters that this panel requires, and press <ENTER>. This generates the JCL, and submits the job.

If your system programmer has not provided a default search option for the C++ compiler, variable CBCCXOPT in CBC.SCBCUTL(CBC3C00E), or you want to modify it, you should enter it under Compiler Options. For example, "SEARCH('CEE190.SCEEH.+')".

Compiling and Binding in the z/OS Shell

z/OS UNIX C/C++ programs with source code in HFS files or data sets must be compiled to create output object files residing either in HFS files or data sets.

You can compile and bind application source code at one time, or compile the source and then bind at another time with other application source files or compiled objects.

The c89, c++, and cc utilities invoke the binder by default, unless the output file of the link-editing phase (-o option) is a PDS, in which case they use the Prelinker.

For information on customizing your environment to compile and bind in the z/OS UNIX System Services environment, see "Environment Variables" on page 567.

Use the c89 utility to compile and bind a C application program from the z/OS shell. The syntax is:

```
c89 [-options ...] [file.c ...] [file.a ...] [file.o ...] [-l libname]
```

where:

options	are c89 options.
file.c	is a source file. Note that C source files have a file extension of lowercase c.
file.o	is an object file.
file.a	is an archive file.

libname	is an archive library.

The c89 utility supports IPA. For information on how to invoke the IPA Compile step from c89, refer to "Invoking IPA from the c89 Utility" on page 302.

You can also use the cc utility to compile a C application program from the z/OS shell. For more information, see "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555.

Use the c++ utility to compile and bind a C++ application program from the z/OS shell. The syntax for c++ is:

```
c++ [-options ...] [file.C ...] [file.a ...] [file.o ...] [-l libname]
```

where:

options	are C++ options.
file.C	is a source file. Note that C++ files have a file extension of uppercase C.
file.o	is an object file.
file.a	is an archive file.
libname	is an archive library.

Another name for the c++ utility is cxx. The cxx utility and the c++ utility are identical. You can use cxx instead of c++ in all the examples that are shown in this section.

For a complete list of c++ options, and for more information on cxx, see "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555.

Note: You can compile and bind application program source and objects from within the shell using the c89 or c++ utility. If you use either of these utilities, you must keep track of and maintain all the source and object files for the application program. You can use the make utility to maintain your z/OS UNIX System Services application source files and object files automatically when you update individual modules. The make utility runs c89 and c++ for you.

For more information on using the make utility, see "Chapter 20. Archive and Make Utilities" on page 453 and *z/OS UNIX System Services Programming Tools*.

Compiling without Binding with c89/CC++

To compile source files without binding them, enter the c89 or c++ command with the -c option to create object file output. Use the -o option to specify placement of the application program executable file to be generated. The placement of the intermediate object file output depends on the location of the source file:

- If the z/OS C/C++ source module is an HFS file, the object file is created in the working directory.
- If the z/OS C/C++ source module is a data set, the object file is created as a data set. The object file is placed in a data set with the qualified name of the source and identified as an object.

  For example, if the z/OS C/C++ source is in the sequential data set LANE.APPROG.USERSRC.C, the object is placed in the data set

LANE.APPROG.USERSRC.OBJ. If the source is in the partitioned data set (PDS) member 'OLSEN.IPROGS.C(FILSER)', the object is placed in the PDS member 'OLSEN.IPROGS.OBJ(FILSER)'.

Note: When the z/OS C/C++ source is located in a PDS member, you should specify double-quote characters around the qualified data set name. For example:

```
c89 -c "//'OLSEN.IPROGS.C(FILSER)'"
```

If the filename is not bracketed by quotes, the parentheses around the member name in the fully qualified PDS name would be subject to special shell parsing rules.

Since the data set name is always converted to uppercase, you can specify it in lowercase or mixed case.

- Compiling z/OS C application source to produce only object files. c89 recognizes that a file is a C source file by the .c suffix for HFS files, and the .C low-level qualifier for data sets. c89 recognizes that a file is an object file by the .o suffix for HFS files, and the .OBJ low-level qualifier for data sets.
  - To compile z/OS C source to create the default object file usersource.o in your working HFS directory, specify:
    ```
    c89 -c usersource.c
    ```
  - To compile z/OS C source to create an object file as a member in the PDS 'KENT.APPROG.OBJ', specify:
    ```
    c89 -c "//'kent.approg.c(usersrc)'"
    ```
- Compiling z/OS C++ application source to produce only object files. c++ recognizes that a file is a C++ source file by the .C suffix for HFS files, and the .CXX low-level qualifier for data sets. c++ recognizes that a file is an object file by the .o suffix for HFS files, and the .OBJ low-level qualifier for data sets.
  - To compile z/OS C++ source to create the default object file usersource.o in your working HFS directory, specify:
    ```
    c++ -c usersource.C
    ```
  - To compile z/OS C++ source to create an object file as a member in the PDS 'JONATHAN.APPROG.OBJ', specify:
    ```
    c++ -c "//'jonathan.approg.CXX(usersrc)'"
    ```
- Compiling and binding application source to produce an application executable file.
  - To compile an application source file to create the object file usersource.o in the HFS working directory and the executable file mymod.out in the /app/bin directory, specify:
    ```
    c89 -o /app/bin/mymod.out usersource.c
    ```
  - To compile the z/OS C source member MAINBAL in the PDS 'CLAUDIO.PGMS.C', and bind it to produce the application executable file /u/claudio/myappls/bin/mainbal.out, specify:
    ```
    c89 -o /u/claudio/myappls/bin/mainbal.out "//'claudio.pgms.C(MAINBAL)'"
    ```

z/OS C++ Note:

To use the TSO utility OGET to copy a C++ HFS listing file to a VBA data set, you must add a blank to any null records in the listing file. Use the awk command as follows:

```
c++ -cV mypgm.C | awk '/^[^$]/ {print} /^$/ {printf "%s \n", $0}'
  > mypgm.lst
```

Compiling and Binding in One Step with c89 and c++ (or cxx)

To compile and bind a C/C++ application program in one step to produce an executable file, specify `c89` or `c++` *without* specifying the `-c` option. You can use the `-o` option with the command to specify the name and location of the application program executable file to be created. The `c++` and cxx utilities are identical. You can use `cxx` instead of `c++` in all the examples that are shown in this section.

The `c89`, `c++`, and cc utilities invoke the binder by default, unless the output file of the link-editing phase (-o option) is a PDS, in which case they use the Prelinker.

- To compile and bind an application source file to create the default executable file `a.out` in the HFS working directory, specify:

  ```
  c89 usersource.c
  c++ usersource.C
  ```

- To compile and bind an application source file to create the `mymod.out` executable file in your `/app/bin` directory, specify:

  ```
  c89 -o /app/bin/mymod.out usersource.c
  c++ -o /app/bin/mymod.out usersource.C
  ```

- To compile and bind several application source files to create the `mymod.out` executable file in your `/app/bin` directory, specify:

  ```
  c89 -o /app/bin/mymod.out usrsrc.c otsrc.c "//'MUSR.C(PWAPP)'"
  c++ -o /app/bin/mymod.out usrsrc.C otsrc.C "//'MUSR.C(PWAPP)'"
  ```

- To compile and bind an application source file to create the `MYLOADMD` member of your `'APPROG.LIB'` PDS, specify:

  ```
  c89 -o "//'APPROG.LIB(MYLOADMD)'" usersource.c
  c++ -o "//'APPROG.LIB(MYLOADMD)'" usersource.C
  ```

- To compile and bind an application source file with several previously compiled object files to create the executable file `zinfo` in your `/prg/lib` HFS directory, specify:

  ```
  c89 -o /prg/lib/zinfo usrsrc.c xstobj.o "//'MUSR.OBJ(PWAPP)'"
  c++ -o /prg/lib/zinfo usrsrc.C xstobj.o "//'MUSR.OBJ(PWAPP)'"
  ```

- To compile and bind an application source file and capture the listings from the compile and bind steps into another file, specify:

  ```
  c89 -V barryl.c > barryl.lst
  c++ -V barryl.C > barryl.lst
  ```

Building an Application with XPLINK from the c89 Utility

To build an application with `XPLINK` from the c89 utility you must use the `XPLINK` compiler option (i.e., –Wc,xplink) and the `XPLINK` link-edit option (i.e., –Wl,xplink). The binder option is not actually passed to the binder. It is used by `c89` to set up the appropriate link data sets.

Invoking IPA from the c89 Utility

You can invoke the IPA Compile step, the IPA Link step, or both from the `c89` utility. The step that you invoke depends upon the invocation parameters and type of files specified. To invoke IPA, you must specify the `I` phase indicator along with the `W` option of the `c89` utility. You can specify `IPA` suboptions as comma-separated keywords.

If you invoke the `c89` utility by specifying the `-c` compiler option and at least one source file, c89 automatically specifies `IPA(NOLINK)` and automatically invokes the IPA Compile step. For example, the following command invokes the IPA Compile step for the source file `hello.c`:

```
c89 -c -WI,noobject hello.c
```

If you invoke `c89` with at least one source file for compilation and any number of object files, and do not specify the `-c` option, `c89` invokes the IPA Compile step once for each compilation unit. It then invokes the IPA Link step once for the entire program, and then invokes the binder. For example, the following command invokes the IPA Compile step and the bind while creating program foo:

```
c89 -o foo -WI,object foo.c
```

Refer to “Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file” on page 555 for more information about the `c89` utility.

Specifying Options for the IPA Compile Step

When using the `c89` utility, you can pass options to the IPA Compile step, as follows:

- You can pass `IPA` compiler option suboptions by specifying `-WI,`, followed by the suboptions.
- You can pass compiler options by specifying `-Wc,`, followed by the options.

Using IPA(OBJECTONLY) with the c89 Utility

A compilation using `IPA(OBJONLY)` is simply a standard non-IPA compilation with this option added. Do not use the `-WI` flag, as this would convert the compilation into an IPA Compile step.

For example, the following command results in an `OPT(2) IPA(OBJONLY)` compilation for the source file `hello.c`:

```
c89 -c -Wc,ipa\(objonly\) -2 hello.c
```

Using the make Utility

You can use the `make` utility to control the build of your z/OS UNIX System Services C/C++ applications. The `make` utility calls the `c89` utility by default to compile and bind the programs that the previously created makefile specifies.

For example, to create `myappl` you compile and bind two source parts `mymain.c` and `mysub.c`. This dependency is captured in makefile `/u/jake/myappl/Makefile`. No recipe is specified, so the default makefile rules are used. If `myappl` was built and a subsequent change was made only to `mysub.c`, you would specify:

```
cd /u/jake/myappl
make
```

The `make` utility sees that `mysub.c` has changed, and invokes the following commands for you:

```
c89  -O -c mysub.c
c89  -o myappl mymain.o mysub.o
```

Note: The `make` utility requires that application program source files that are to be "maintained" through use of a makefile reside in HFS files. To compile and bind z/OS C/C++ source files that are in data sets, you must use the `c89` utility directly.

See the *Z/OS UNIX System Services Command Reference* for a description of the `make` utility. For a detailed discussion on how to create and use makefiles to manage application parts, see the *z/OS UNIX System Services Programming Tools*.

Compiling with IPA

If you request Interprocedural Analysis (IPA) through the `IPA` compiler option, the compilation process changes significantly. IPA instructs the compiler to optimize your z/OS C/C++ program across compilation units, and to perform optimizations that are not otherwise available with the z/OS C/C++ compiler. You should refer to the *z/OS C/C++ Programming Guide* for an overview of IPA processing before you invoke the compiler with the `IPA` compiler option.

Differences between the IPA compilation process and the regular batch or `c89` compilation process are noted throughout this chapter.

Figure 25 shows the flow of processing for a regular compilation:

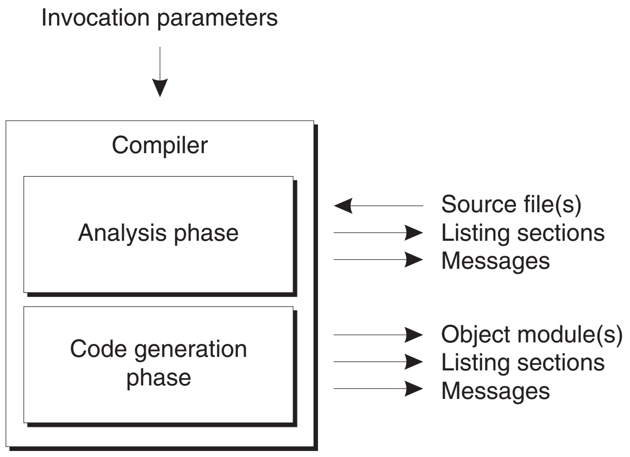


Figure 25. Flow of regular compiler processing

IPA processing consists of two separate steps, called the IPA Compile step and the IPA Link step.

The IPA Compile Step

The IPA Compile step is similar to a regular compilation.

You invoke the IPA Compile step for each source file in your application by specifying the `IPA(NOLINK)` compiler option. The output of the IPA Compile step is an object file which contains IPA information, or both IPA information and conventional object code and data. The IPA information is an encoded form of the compilation unit with additional IPA-specific compile-time optimizations.

Figure 26 on page 305 shows the flow of IPA Compile step processing.

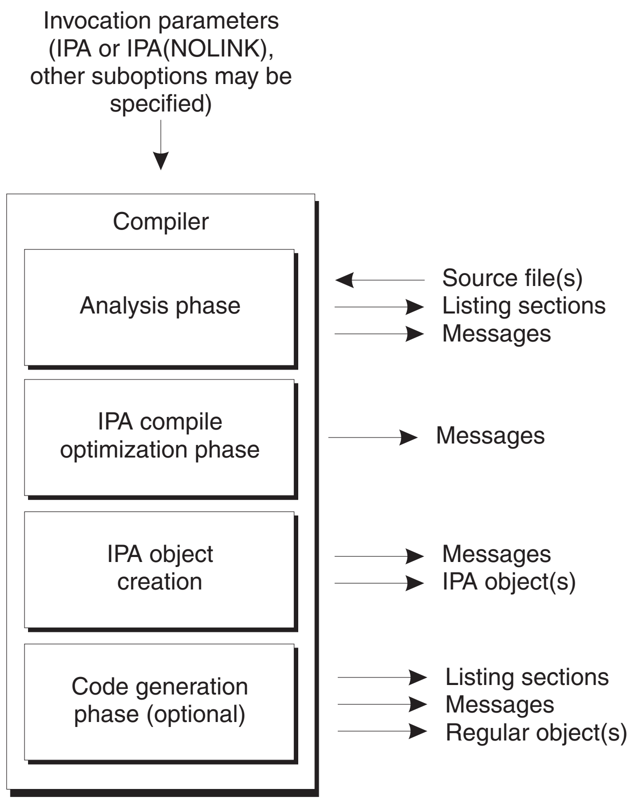


Figure 26. IPA Compile step processing

The same environments that support a regular compilation also support the IPA Compile step.

The IPA Link Step

The IPA Link step is similar to the binding process.

You invoke the IPA Link step by specifying the `IPA(LINK)` compiler option. This step links the user application program together by combining object files with IPA information, object files with conventional object code and data, and load module members. It merges IPA information, performs IPA Link-time optimizations, and generates the final object code and data.

Each application program module must be built with a single invocation of the IPA Link step. All parts must be available during the IPA Link step; missing parts may result in termination of IPA Link processing.

Figure 27 on page 306 shows the flow of IPA Link step processing:

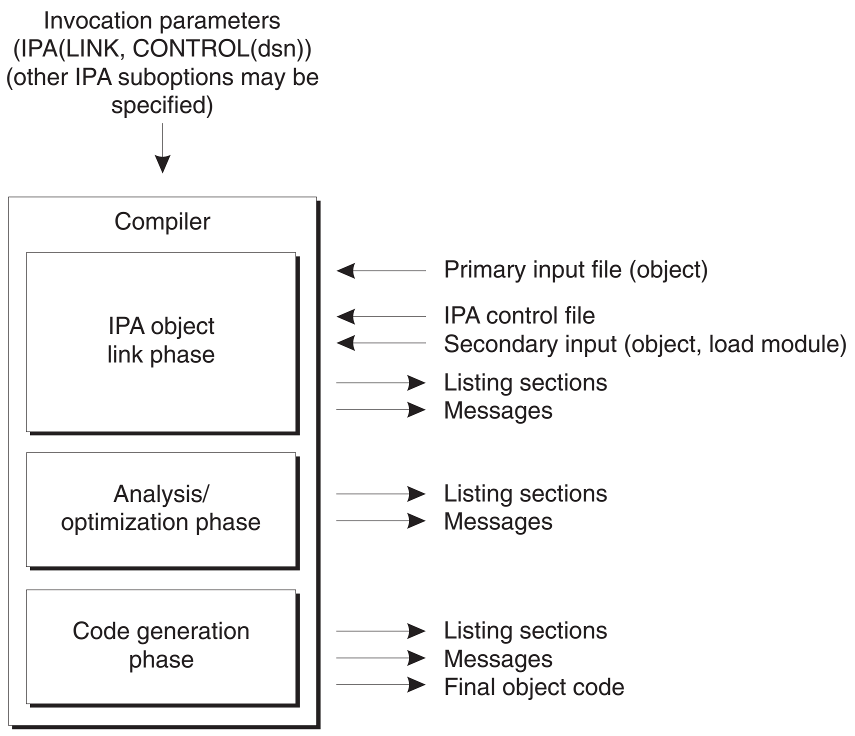

Invocation parameters (IPA(LINK, CONTROL(dsn)) (other IPA suboptions may be specified)
Compiler
IPA object link phase
Analysis/ optimization phase
Code generation phase
Primary input file (object)
IPA control file
Secondary input (object, load module)
Listing sections
Messages
Listing sections
Messages
Listing sections
Messages
Final object code

Figure 27. IPA Link step processing

Only `c89`, `c++` and z/OS batch (without the ISPF interface) support the IPA Link step. Refer to “Chapter 11. Using the IPA Link Step with z/OS C/C++ Programs” on page 329 for information about the IPA Link step.

Compiling with IPA(OBJONLY)

The full Interprocedural Analysis using the IPA Compile and IPA Link steps performs significant optimizations beyond those which are available using regular compilation. If problems occur, diagnosis may take significant time and effort.

The `IPA(OBJONLY)` compilation is an intermediate level of optimization. This results in a modified regular compile, not an IPA Compile step. Unlike the IPA Compile step, no IPA information is written to the object file.

During compilation, this step performs the same IPA-specific compile-time optimizations as the IPA Compile step, performs the requested non-IPA optimizations, and then generates optimized object code and data.

You invoke the compiler for each source file in your application by specifying the `IPA(OBJONLY)` compiler option.

The object file may be used by an IPA Link step, a prelink/link, or a bind. If it is used as input to an IPA Link step, no IPA link-time optimizations can be performed for this compilation unit because no IPA information is available.

Figure 28 on page 307 shows the flow of processing for an `IPA(OBJONLY)` compilation.

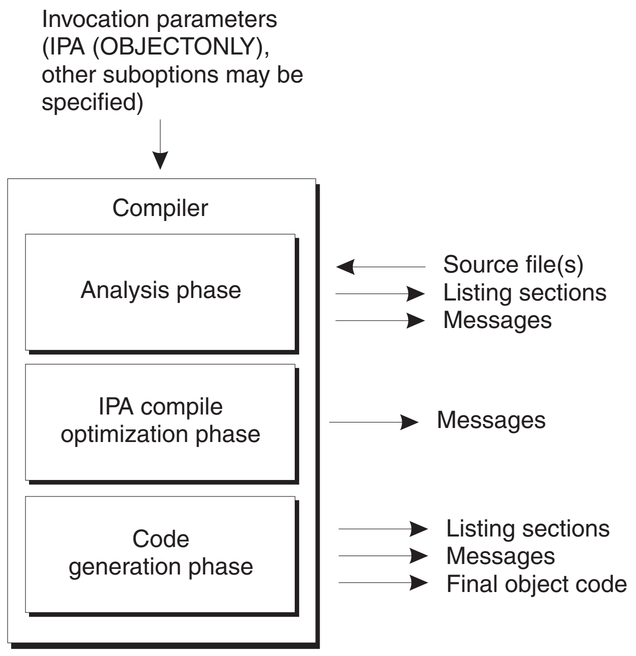


Figure 28. Compiling with IPA(OBJECTONLY)

Working With Object Files

S/390 object files are composed of a stream of 80 byte records. These may be binary object records, or link control statements. It is useful to be able to browse the contents of an object file, so that some basic information can be determined. For more information on object files, see “Object File Formats” on page 336.

Browsing Object Files

Object files, which are sequential datasets or are members of a PDS or PDSE object library, can be browsed directly using the PDF edit and browse options.

Object files, which are files in an HFS file system, can be browsed using the PDF `obrowse` command. To force display in F 80 record mode, one would issue the following sequence of operations:

1. Enter the command: `obrowse file.oo`

   Note that the file name is deliberately typed with an extra character. This will result in the display of an obrowse dialog panel with an error message that the file is not found. After pressing <Enter>, a second obrowse dialog is displayed to allow the file name to be corrected. This panel has an entry field for the record length.
2. Correct the file name and enter 80 in the record length entry field.
3. Browse the object records as you would a F 80 dataset.

The hex display mode (enabled by the HEX ON primary command) allows the value of each byte to be displayed.

Identifying Object File Variations

Browse the object file and scroll to the end of the file. The last few records contain a character string, which lists the options used during compilation.

In addition, it is possible to identify the compiler mode used to generate the object file, as follows:

1. NOIPA

   Option text has ″NOIPA″.
2. IPA(NOOBJECT)

   Option text has ″IPA (NOLINK, NOOBJ)″. Towards the beginning of the file, an ESD record will contain the symbol ″@@IPAOBJ″. A second ESD record will contain the symbol ″@@DOIPA″.
3. IPA(OBJECT)

   Option text has ″IPA (NOLINK, OBJ)″. Towards the beginning of the file, an ESD record will contain the symbol ″@@IPAOBJ″. The IPA information will be separated from the ″real″ code and data by a delimiter END record with the comment ″of IPA object″. After the real code and data, there will be a second delimiter END record with the comment ″of object″.
4. IPA(OBJONLY)

   Option text has ″IPA (OBJONLY)″.

Using Feature Test Macros

For information on how to use feature test macros, refer to the *z/OS C/C++ Run-Time Library Reference*.

Using Include Files

The `#include` preprocessor directive allows you to retrieve source statements from secondary input files and incorporate them into your C/C++ program.

z/OS C/C++ Language Reference describes the `#include` directive. Its syntax is:

```
►►──#include──┬─<──┬────┬──filename──>─┬──────────────────────────────►◄
              │    └─//─┘              │
              └─"──┬────┬──filename──"─┘
                   └─//─┘
```

The angle brackets specify system include files, and double quotation marks specify user include files.

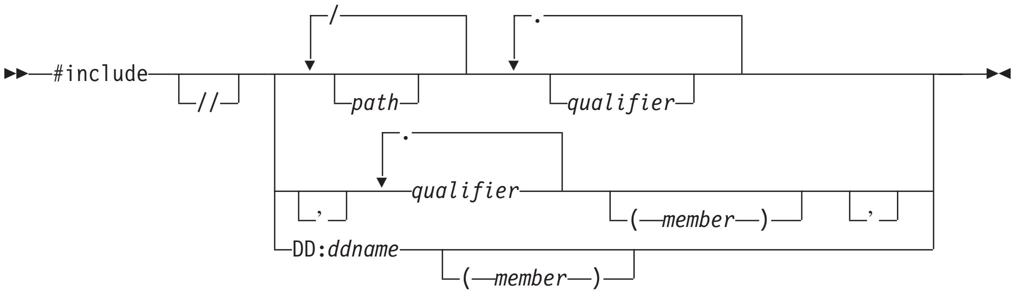
When you use the `#include` directive, you must be aware of the following:

- The *library search sequence*, the search order that C/C++ uses to locate the file. See “Search Sequences for Include Files” on page 316 for more information on the library search sequence.
- The file-naming conversions that C/C++ performs.
- The area of the input record that contains sequence numbers when you are including files with different record formats. See the *z/OS C/C++ Language Reference* for more information on `#pragma sequence`.

Specifying Include File Names

You can use the SEARCH and LSEARCH compiler options to specify search paths for system include files and user include files. For more information on these options, see "LSEARCH | NOLSEARCH" on page 138 and "SEARCH | NOSEARCH" on page 167.

You can specify *filename* of the #include directive in the following format:



The leading double slashes (//) not followed by a slash (in the first character of *filename*) indicate that the file is to be treated as a non-HFS file, hereafter called a data set.

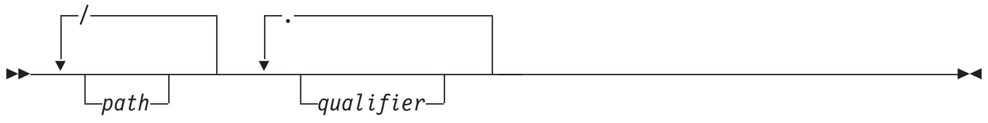
Note:

1. *filename* immediately follows the double slashes (//) without spaces.
2. Absolute data set names are specified by putting single quotation marks (') around the name. Refer to the above syntax diagram for this specification.
3. Absolute HFS filenames are specified by putting a leading slash (/) as the first character in the file name.
4. ddnames are always considered absolute.

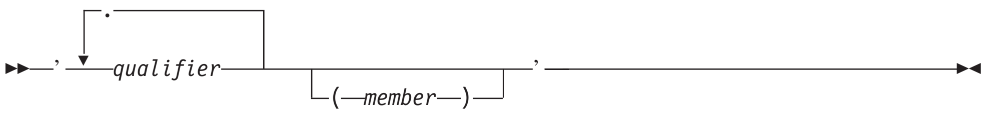
Forming File Names

Refer to "Determining whether the File Name is in Absolute Form" on page 313 for information on absolute file names. When the compiler performs a library search, it treats *filename* as either an HFS file name or a data set name. This depends on whether the library being searched is HFS or MVS. If the compiler treats *filename* as an HFS file name, it does not perform any conversions on it. If it treats *filename* as a data set name (DSN), it performs the following conversion:

- For the first DSN format:

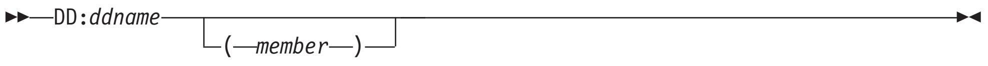


  The compiler:
  1. Uppercases *qualifier* and *path*
  2. Truncates each *qualifier* and *path* to 8 characters
  3. Converts the underscore character (which is invalid for a DSN) to hex 7c, viewed as an '@' (at sign) in code page 1047
- For the second DSN format:



The compiler:
1. Uppercases the *qualifier* and *member*
2. Converts the underscore character (which is invalid for a DSN) to hex `7c`, viewed as an ’@’ (at sign) in code page 1047

- For the third DSN format:



The compiler:
1. Uppercases the `DD:`, *ddname*, and *member*
2. Converts the underscore character (which is invalid for a DSN) to hex `7c`, viewed as an ’@’ (at sign) in code page 1047

Forming Data Set Names with LSEARCH | SEARCH Options

When the *filename* specified in the `#include` directive is not in absolute form, the compiler combines it with different types of libraries to form complete data set specifications. These libraries may be specified by the `LSEARCH` or `SEARCH` compiler options. When the `LSEARCH` or `SEARCH` *opt* indicates a data set then depending on whether it is a `ddname`, sequential data set, or PDS, different parts of *filename* are used to form the `ddname` or data set name.

Forming DDname

The leftmost qualifier of the *filename* in the `#include` directive is used when the *filename* is to be a `ddname`. For example:

Invocation:
```
SEARCH(DD:SYSLIB)
```

Include directive:
```
#include "sys/afile.g.h"
```

Resulting `ddname`**:**
```
DD:SYSLIB(AFILE)
```

In the above example, if your header file includes an underscore (_), for example, `#include "sys/afile_1.g.h"`, the resulting ddname is `DD:SYSLIB(AFILE@1)`.

Forming Sequential Data Set Names

You specify libraries in the SEARCH | LSEARCH options as sequential data sets by using a trailing period followed by an asterisk (.*), or by a single asterisk (*). See “Specifying Sequential Data Sets and PDSs” on page 141 to understand how to specify sequential data sets. All *qualifier*s and periods (.) in *filename* are used for sequential data set specification. For example:

Invocation:
```
SEARCH(AA.*)
```

Include directive:
```
#include "sys/afile.g.h"
```

Resulting fully qualified data set name:
`userid.AA.AFILE.G.H`

Forming PDS Name with LSEARCH | SEARCH + Specification

To specify libraries in the `SEARCH` and `LSEARCH` options as PDSs, use a period that is followed by a plus sign (.+), or a single plus sign (+). See "Specifying Sequential Data Sets and PDSs" on page 141 to understand how PDSs are specified. When this is the case then all the *path*s, slashes (replaced by periods), and any *qualifier*s following the leftmost *qualifier* of the *filename* are appended to form the data set name. The leftmost *qualifier* is then used as the member name. For example:

Invocation:
`SEARCH('AA.+')`

Include directive:
`#include "sys/afile.g.h"`

Resulting fully qualified data set name:
`AA.SYS.G.H(AFILE)`

and

Invocation:
`SEARCH('AA.+')`

Include directive:
`#include "sys/bfile"`

Resulting fully qualified data set name:
`AA.SYS(BFILE)`

Forming PDS with LSEARCH | SEARCH Options With No +

When the `LSEARCH` or `SEARCH` option specifies a library but it neither ends with an asterisk (*) nor a plus sign (+), it is treated as a PDS. The leftmost qualifier of the *filename* in the `#include` directive is used as the member name. For example:

Invocation:
`SEARCH('AA')`

Include directive:
`#include "sys/afile.g.h"`

Resulting fully qualified data set name:
`AA(AFILE)`

Examples Of Forming Data Set Names

The following table gives the original format of the *filename* and the resulting converted name when you specify the `NOOE` option:

Table 38. Include filename Conversions When NOOE Is Specified

`#include` Directive	Converted Name
Example 1. This *filename* is absolute because single quotation marks (') are used. It is a sequential data set. A library search is not performed. `LSEARCH` is ignored.	
`#include "'USER1.SRC.MYINCS'"`	USER1.SRC.MYINCS
Example 2. This *filename* is absolute because single quotation marks (') are used. The compiler attempts to open data set COMIC/BOOK.OLDIES.K and fails because it is not a valid data set name. A library search is not performed when *filename* is in absolute form. `SEARCH` is ignored.	
`#include <'COMIC/BOOK.OLDIES.K'>`	COMIC/BOOK.OLDIES.K
Example 3.	

Table 38. Include filename Conversions When NOOE Is Specified (continued)

`#include` Directive	Converted Name
`SEARCH(LIB1.*,LIB2.+,LIB3) #include "sys/abc/xx"`	• first *opt* in SEARCH SEQUENTIAL FILE = *userid*.LIB1.XX • second *opt* in SEARCH PDS = *userid*.LIB2.SYS.ABC(XX) • third *opt* in SEARCH PDS = *userid*.LIB3(XX)
Example 4.	
`SEARCH(LIB1.*,LIB2.+,LIB3) #include "Sys/ABC/xx.x"`	• first *opt* in SEARCH SEQUENTIAL FILE = *userid*.LIB1.XX.X • second *opt* in SEARCH PDS = *userid*.LIB2.SYS.ABC.X(XX) • third *opt* in SEARCH PDS = *userid*.LIB3(XX)
Example 5.	
`SEARCH(LIB1.*,LIB2.+,LIB3) #include <sys/name_1>`	• first *opt* in SEARCH SEQUENTIAL FILE = *userid*.LIB1.NAME@1 • second *opt* in SEARCH PDS = *userid*.SYS(NAME@1) • third *opt* in SEARCH PDS = *userid*.LIB3(NAME@1)
Example 6.	
`SEARCH(LIB1.*,LIB2.+,LIB3) #include <Name2/App1.App2.H>`	• first *opt* in SEARCH SEQUENTIAL FILE = *userid*.LIB1.APP1.APP2.H • second *opt* in SEARCH PDS = *userid*.LIB2.NAME2.APP2.H(APP1) • third *opt* in SEARCH PDS = *userid*.LIB3(APP1)
Example 7. The PDS member named YEAREND of the library associated with the `ddname` PLANLIB is used. A library search is not performed when *filename* in the `#include` directive is in absolute form (`ddname` is used). `SEARCH` is ignored.	
`#include <dd:planlib(YEAREND)>`	DD:PLANLIB(YEAREND)

Search Sequence

The following diagram describes the compiler's file searching sequence:

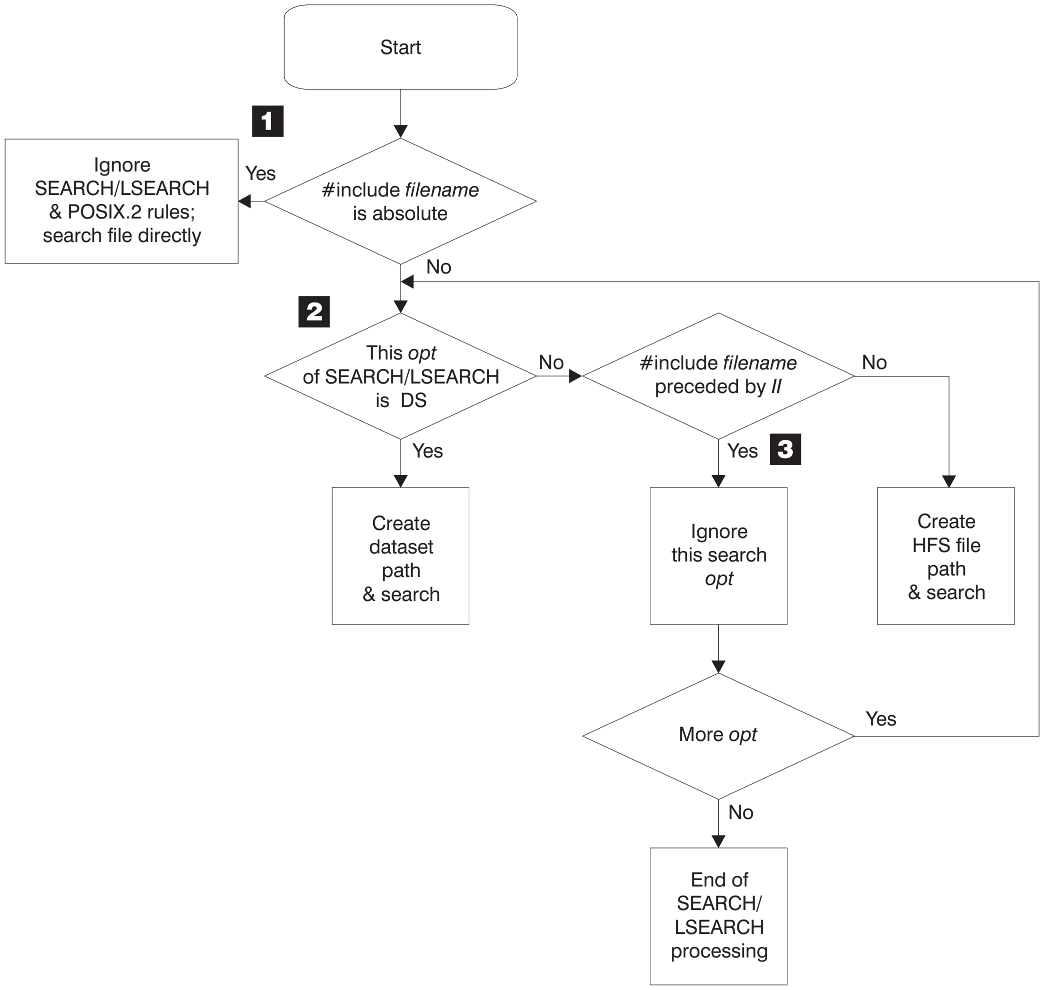

Start
1
#include *filename* is absolute
Yes
Ignore SEARCH/LSEARCH & POSIX.2 rules; search file directly
No
2
This *opt* of SEARCH/LSEARCH is DS
No
#include *filename* preceded by //
No
Yes
Create dataset path & search
Yes
3
Ignore this search *opt*
Create HFS file path & search
More *opt*
Yes
No
End of SEARCH/ LSEARCH processing

Figure 29. Overview of Include File Searching

1 The compiler opens the file without library search when the file name that is specified in `#include` is in absolute form. This also means that it bypasses the rules for the `SEARCH` and `LSEARCH` compiler options, and for POSIX.2. See Figure 30 on page 314 for more information on absolute file testing.

2 When the file name is not in absolute form, the compiler evaluates each option in `SEARCH` and `LSEARCH` to determine whether to treat the file as a data set or an HFS file search. The LSEARCH/SEARCH *opt* testing here is described in Figure 31 on page 315.

3 When the `#include` file name is not absolute, and is preceded by exactly two slashes (//), the compiler treats the file as a data set. It then bypasses all HFS file options of the `SEARCH` and `LSEARCH` options in the search.

Determining whether the File Name is in Absolute Form

The compiler determines if the file name that is specified in `#include` is in absolute form as follows:

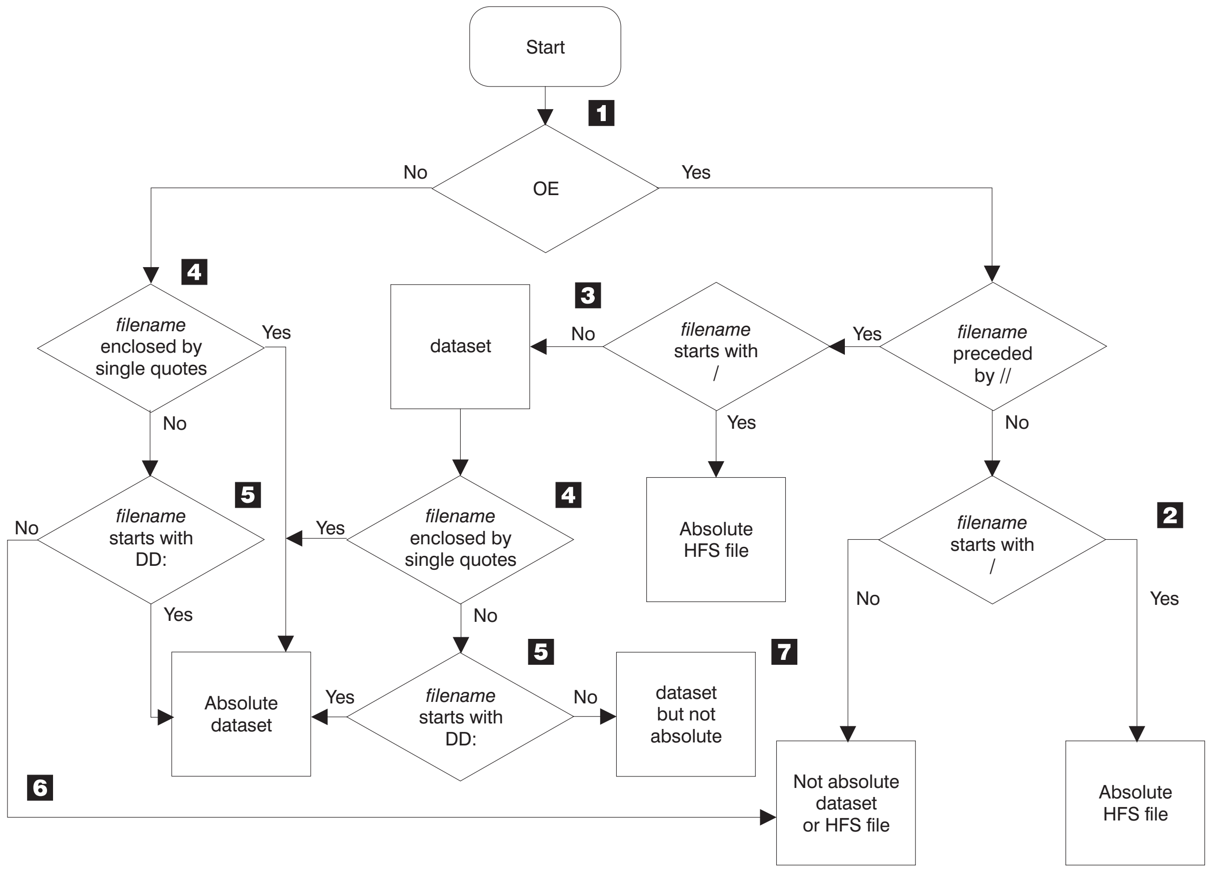


Figure 30. Testing If filename Is In Absolute Form

1 The compiler first checks whether you specified `OE`.

2 When you specify `OE`, if double slashes (//) do not precede *filename*, and the file name starts with a slash (/), then *filename* is in absolute form and the compiler opens the file directly as an HFS file. Otherwise, the file is not an absolute file and each *opt* in the `SEARCH` or `LSEARCH` compiler option determines if the file is treated as an HFS or data set in the search for the include file.

3 When `OE` is specified, if double slashes (//) precede *filename*, and the file name starts with a slash (/), then *filename* is in absolute form and the compiler opens the file directly as an HFS file. Otherwise, the file is a data set, and more testing is done to see if the file is absolute.

4 If *filename* is enclosed in single quotation marks ('), then it is an absolute data set. The compiler directly opens the file and ignores the libraries that are specified in the `LSEARCH` or `SEARCH` options.

5 If you used the ddname format of the `#include` directive, the compiler uses the file associated with the ddname and directly opens the file as a data set. The libraries that are specified in the `LSEARCH` or `SEARCH` options are ignored.

6 If none of the above conditions are true then *filename* is not in absolute format and each *opt* in the `SEARCH` or `LSEARCH` compiler option determines if the file is an HFS or a data set and then searched for the include file.

7 If none of the above conditions are true, then *filename* is a data set, but it is not in absolute form. Only *opt*s in the `SEARCH` or `LSEARCH` compiler option that are in data set format are used in the search for include file.

For example:

```
 Options specified:

  OE

Include Directive:

  #include "apath/afile.h"       NOT absolute, HFS/MVS (no starting slash)
  #include "/apath/afile.h"      absolute HFS, (starts with 1 slash)
  #include "//apath/afile.h.c"   NOT absolute, MVS (starts with 2 slashes)
  #include "a.b.c"               NOT absolute, HFS/MVS (no starting slash)
  #include "///apath/afile.h"    absolute HFS, (starts with 3 slashes)
  #include "DD:SYSLIB"           NOT absolute, HFS/MVS (no starting slash)
  #include "//DD:SYSLIB"         absolute, MVS (DD name)
  #include "a.b(c)"              NOT absolute, HFS/MVS (no starting slash)
  #include "//a.b(c)"            NOT absolute, OS/MVS (PDS member name)
```

Using SEARCH and LSEARCH

When the file name in the #include directive is not in absolute form, the *opt*s in `SEARCH` are used to find system include files and the *opt*s in `LSEARCH` are used to find user include files. Each *opt* is a library path and its format determines if it is an HFS path or a data set path:

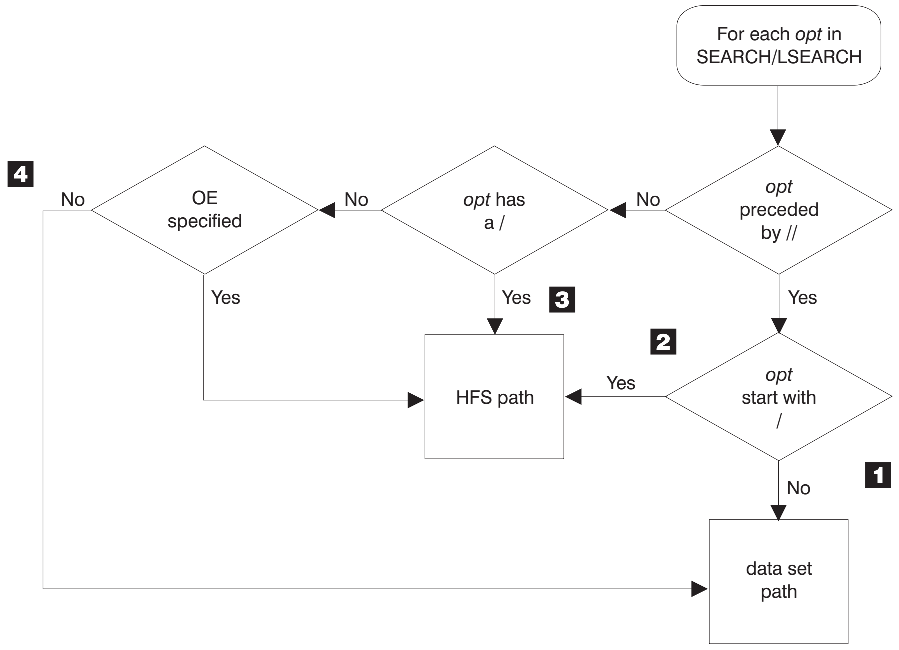


Figure 31. Determining if the SEARCH/LSEARCH opt is an HFS path

Note:

1. If *opt* is preceded by double slashes (//) and *opt* does not start with a slash (/), then this path is a data set path.
2. If *opt* is preceded by double slashes (//) and *opt* starts with a slash (/), then this path is an HFS path.
3. If *opt* is **not** preceded by double slashes (//) and *opt* starts with a slash (/), then this path is an HFS path.

4. If *opt* is **not** preceded by double slashes (//), *opt* does not start with a slash (/) and NOOE is specified then this path is a data set path.

For example:

`SEARCH(./PATH)`	is an explicit HFS path
`OE SEARCH(PATH)`	is treated as an HFS path
`NOOE SEARCH(PATH)`	is treated as a non-HFS path
`NOOE SEARCH(//PATH)`	is an explicit non-HFS path.

When combining the library with the file name specified on the `#include` directive, it is the form of the library that determines how the include file name is to be transformed. For example:

```
Options specified:

  NOOE LSEARCH(Z, /u/myincs, (*.h)=(LIB(mac1)))

Include Directive:

  #include "apath/afile.h"

Resulting fully qualified include names:

1. userid.Z(AFILE)  (Z is non-HFS so filename is treated as non-HFS)
2. /u/myincs/apath/afile.h   (/u/myincs is HFS so filename is treated as HFS)
3. userid.MAC1.H(AFILE)             (afile.h matches *.h)
```

An HFS path specified on a `SEARCH` or `LSEARCH` option only combines with the file name specified on an `#include` directive if the file name is not explicitly stated as being MVS only. A file name is explicitly stated as being MVS only if two slashes (//) precede it, and *filename* does not start with a slash (/). For example:

```
Options specified:

  OE LSEARCH(/u/myincs, q, //w)

Include Directive:

  #include "//file.h"

Resulting fully qualified include names

 userid.W(FILE)
```

`/u/myincs` and `q` would not be combined with `//file.h` because both paths are HFS and `//file.h` is explicitly MVS.

The order in which options on the `LSEARCH` or `SEARCH` option are specified is the order that is searched.

See “LSEARCH | NOLSEARCH” on page 138 and “SEARCH | NOSEARCH” on page 167 for more information on these compiler options.

Search Sequences for Include Files

The status of the `OE` option affects the search sequence.

With the NOOE option

Search Sequences for include files are used when the include file is **not** in absolute form. “Determining whether the File Name is in Absolute Form” on page 313 describes the absolute form of include files.

If the include filename is not absolute, the compiler performs the library search as follows:

- For system include files:
  1. The search order as specified on the SEARCH option, if any.
  2. The libraries specified on the SYSLIB DD statement
- For user include files:
  1. The directory of the file that contains the #include directive
  2. When the containing file is HFS, the search order as specified on the LSEARCH option, if any
  3. The libraries specified on the USERLIB DD statement
  4. The search order for system include files

The example below shows an excerpt from a JCL stream, that compiles a C program for a user whose user prefix is JONES:

```
//COMPILE  EXEC PROC=EDCC,
//              CPARM='SEARCH(''''BB.D'''',BB.F),LSEARCH(CC.X)'
//SYSLIB   DD DSN=JONES.ABC.A,DISP=SHR
//         DD DSN=ABC.B,DISP=SHR
//USERLIB  DD DSN=JONES.XYZ.A,DISP=SHR
//         DD DSN=XYZ.B,DISP=SHR
//SYSIN    DD DSN=JONES.ABC.C(D),DISP=SHR
   .
   .
   .
```

The search sequence that results from the preceding JCL statements is:

Table 39. Order of Search for Include Files

Order of Search	For System Include Files	For User Include Files
First	BB.D	JONES.CC.X
Second	JONES.BB.F	JONES.XYZ.A
Third	JONES.ABC.A	XYZ.B
Fourth	ABC.B	BB.D
Fifth		JONES.BB.F
Sixth		JONES.ABC.A
Seventh		ABC.B

With the OE option

Search Sequences for include files are used when the include file is **not** in absolute form. “Determining whether the File Name is in Absolute Form” on page 313 describes the absolute form of an include file.

If the include filename is not absolute, the compiler performs the library search as follows:

- For system include files:
  1. The search order as specified on the SEARCH option, if any
  2. The libraries specified on the SYSLIB DD statement

- For user include files:
  1. If you specified `OE` with a file name and the file being processed is an HFS file and a main source file, the directory of the file containing the #include directive
  2. The search order as specified on the `LSEARCH` option, if any
  3. The libraries specified on the `USERLIB DD` statement
  4. The search order for system include files

For example, given a file `/r/you/cproc.c` that contains the following #include directives:

```
#include "/u/usr/header1.h"
#include "//aa/bb/header2.x"
#include "common/header3.h"
#include <header4.h>
```

And the following options:

```
OE(/u/crossi/myincs/cproc)
SEARCH(//V.+, /new/inc1, /new/inc2)
LSEARCH(//(*.x)=(lib(AAA)), /c/c1, /c/c2)
```

The include files would be searched as follows:

Table 40. Examples of Search Order for z/OS UNIX

`#include` Directive Filename	**Files in Search Order**
Example 1. This is an absolute pathname, so no search is performed.	
`#include "/u/usr/header1.h"`	1. /u/usr/header.h
Example 2. This is a data set (starts with //) and is treated as such.	
″//aa/bb/header2.x″	1. *userid*.AAA(HEADER2) 2. DD:USERLIB(HEADER2) 3. *userid*.V.AA.BB.X(HEADER2) 4. DD:SYSLIB(HEADER2)
Example 3. This is a system include file with a relative path name. The search starts with the directory of the parent file or the name specified on the OE option if the parent is the main source file (in this case the parent file is the main source file so the OE suboption is chosen i.e. /u/crossi/myincs).	
″common/header3.h″	1. /u/crossi/myincs/common/header3.h 2. /c/c1/common/header3.h 3. /c/c2/common/header3.h 4. DD:USERLIB(HEADER3) 5. *userid*.V.COMMON.H(HEADER3) 6. /new/inc1/common/header3.h 7. /new/inc2/common/header3.h 8. DD:SYSLIB(HEADER3)
Example 4. This is a system include file with a relative path name. The search follows the order of suboptions of the SEARCH option.	
<header4.h>	1. *userid*.V.H(HEADER4) 2. /new/inc1/common/header4.h 3. /new/inc2/common/header4.h 4. DD:SYSLIB(HEADER4)

Compiling z/OS C Source Code Using the SEARCH option

The following data sets contain the commonly-used system header files for C: [3]
- `CEE.SCEEH.H` (standard header files)
- `CEE.SCEEH.SYS.H` (standard system header files)
- `CEE.SCEEH.ARPA.H` (standard internet operations headers)
- `CEE.SCEEH.NET.H` (standard network interface headers)
- `CEE.SCEEH.NETINET.H` (standard internet protocol headers)

To specify that the compiler search these data sets, code the option:

```
SEARCH('CEE.SCEEH.+')
```

These header files are also in the HFS in the directory `/usr/include`. To specify that the compiler search this directory, code the option:

```
SEARCH(/usr/include/)
```

This option is the default for the `c89`, `cc` and `cxx` z/OS UNIX System Services utilities.

IBM supplies this option as input to the Installation and Customization of the compiler. Your system programmer can modify it as required for your installation.

The cataloged procedures, REXX EXECs, and panels that are supplied by IBM for C specify the following data sets for the SYSLIB ddname by default:
- `CEE.SCEEH.H` (standard header files)
- `CEE.SCEEH.SYS.H` (standard system header files)

This is supplied for compatibility with previous releases, and will be overridden if `SEARCH()` is used as described above.

Compiling z/OS C++ Source Code Using the SEARCH option

The following data sets contain the commonly-used system header files for z/OS C++: [3]
- `CEE.SCEEH.H` (standard header files)
- `CEE.SCEEH.SYS.H` (standard system header files)
- `CEE.SCEEH.ARPA.H` (standard internet operations headers)
- `CEE.SCEEH.NET.H` (standard network interface headers)
- `CEE.SCEEH.NETINET.H` (standard internet protocol headers)
- `CBC.SCLBH.H` (class library header files)
- `CBC.SCLBH.HPP` (class library header files)
- `CBC.SCLBH.C` (class library template definition files)
- `CBC.SCLBH.INL` (class library inline definition files)

To specify that the compiler search these data sets, code the option:

```
SEARCH('CEE.SCEEH.+','CBC.SCLBH.+')
```

These header files are also in the HFS in the directory `/usr/include`. To specify that the compiler search this directory, code the option:

```
SEARCH(/usr/include/)
```

This option is the default for the `c89`, `cc` and `cxx` z/OS UNIX System Services utilities. The class library datasets are in the HFS directory `/usr/lpp/ioclib/include`.

3. The high-level qualifier may be different at your installation.

IBM supplies this option as input to the installation and customization of the compiler. Your system programmer can modify it as required for your installation.

Chapter 10. Using Precompiled Headers

You can improve your compile time by using precompiled headers (PCH). Use the options GENP and USEP together to automatically create and maintain precompiled header files for your application. If you use these options consistently, the compiler creates precompiled header files if they do not exist, and attempts to use them if they do. When you change a source file, the compiler automatically regenerates the precompiled version the next time you compile your program.

The compiler generates a precompiled object for the first **initial sequence** of #include directives. The next time you compile, this object can be used wherever that initial sequence appears. The precompiled object is not re-interpreted every time it is included, since the precompiled object is only used in cases where the context is the same. For example, same language, same beginning sequence of #include directives, same options, and macro definitions.

To get the most benefit from this method, use the same initial sequence of headers wherever possible. The more files that share the same initial sequence, the greater the improvement in your compile time. See “Organizing Your Source Files” on page 327 for tips on getting the most improvement.

Note: A precompiled header may not necessarily be reused although a matching initial sequence is found. Usage also depends on the availability of consistent address locations between compilations as described in “Restrictions” on page 326. Compile time improvement is of a statistical nature. For example, when doing a large number of compilations in an application build, you can obtain overall improvement even though individual compiles may not benefit to the same degree.

Determining the Initial Sequence

The initial sequence of headers which consists of directives in the primary source file, can consist of the following:

- #include directives
- Comments
- #error directives
- Null directives
- False conditional compilation blocks beginning with #elif or #else. In the following example, the headers a.h, b.h, and c.h are included in the initial sequence. The header d.h is not.

```
#define foo
#undef goo
#if foo
  #include "a.h"
  #include "b.h"
#elif
  else
    .
    .
    .
#else
  else
    .
    .
    .
#endif
```

```
#include "c.h"
#if goo
  #include "d.h"
...
```

- `#endif` directives

Only comments, `#define`, `#undef`, and `#if` can precede the first `#include` directive. For C programs, `#pragma` directives are also allowed. If anything else precedes the `#include` directive, the compiler will not create or attempt to use precompiled headers with that source file.

Any one of the following terminates the initial sequence:
- The compiler detects any construct not in the initial sequence list as described above.
- The compiler detects `#pragma hdrstop` after a `#include` directive. In this instance all `#include` directives that follow `#pragma hdrstop` directive will not be part of the initial sequence.
- The compiler detects `#pragma hdrstop` before the first `#include` directive. In this instance, there is no initial sequence, and the compiler does not create a precompiled header.

Any `#include` directives after the initial sequence are not precompiled: they will be compiled every time you compile the source file.

When a header contains conditional compilation directives to prevent it from being included a second time, it is only counted once in the initial sequence, even if it appears multiple times. Figure 32 on page 323 illustrates how the `initial sequence` can vary, depending on whether any macros are defined on the command line.

Consider the following code sequence:

h1.h

```
int h1;
include "h3.h"
```

h2.h

```
int h2;
```

h3.h

```
#ifndef H3_H
#define H3_H
  int h3;
#endif
```

main.c

```
/* Comments are OK */
#define M 1
#undef N
#if F
  int f(int);
#endif
#if STDIO
  #include <stdio.h>
#endif
#include "h1.h"

/* Comments are OK */
#include "h2.h"
#include "h3.h"
main() {
.
.
.
}
```

Figure 32. Initial Sequence Based on Defined Macros

The following table shows three different initial sequences as a result of different compile-time options as input.

Table 41. Initial Sequence Based on Macros

Macros Defined	Resulting Initial Sequence
None	″h1.h″, ″h2.h″, ″h3.h″
STDIO	<stdio.h>, ″h1.h″, ″h2.h″, ″h3.h″
F	No initial sequence (because the prototype `int f(int)`; occurs before any #include directives)

Although `h3.h` is included twice (once in `main.c` and once in `h1.h`), only the first `#include` directive is considered in the initial sequence. The second `#include` directive does not take effect because of the conditional compilation directive in `h3.h`.

Matching the Initial Sequence

Once the precompiled initial sequence is created, other compilation units in subsequent compiles can use it. Another compilation unit can use the precompiled initial sequence under the following conditions:

- The compilation unit has a matching initial sequence of `#include` directives. The compilation unit can have a longer initial sequence, as long as the first part of the sequence matches. Any `#include` directives beyond the initial matching portion are compiled normally.
- The include files that make up the precompiled header object have not changed. The compiler checks the modification date of each include file.
- Any macros that were expanded or tested while generating the precompiled header object are defined with the same replacement tokens. The compiler checks macro names that are:
  - Defined before the start of the initial sequence, using the `#define` directive or the `def` compile time option.
  - Undefined before the start of the initial sequence, with the `#undef` directive or the `undef` compile time option.
  - Predefined by the compiler.

  If the macro was not expanded or tested during the precompile, then its status does not matter, and does not have to match.
- No additional macros have been defined.
- Compiler option specifications must match exactly. You must specify them in the same way during both compilations. The only exceptions are the `GENP` and `USEP` options; in this case, the filename suboption must also be the same.
- Under z/OS C, the same `#pragma` directives, if any, before the first `#include`. The specifications and order of the `#pragma` directives must be the same.

Example - Reusing Sequences

Given the following two compilation units, `prog1.c`, and `prog2.c` and two header files `h1.h` and `h2.h`:

h1.h

```
#if TEST
  int h1;
#endif
```

h2.h

```
int h2 = M+5;
```

prog1.c

```
#undef X
#include "h1.h"
#include "h2.h"
func1() {
.
.
.
}
```

prog2.c

```
#define X 1
#include "h1.h"
#include "h2.h"
func2() {
.
.
.
}
```

Figure 33. Example of Reusing Initial Sequence

The file `prog2.c` can use the precompiled header object from `prog1.c` under the following conditions:

- The macro `TEST` has the same definition in both `prog1.c` and `prog2.c`, or is not defined in both.
- Macro `M` has the same definition in both `prog1.c` and `prog2.c`, or is not defined in both.
- No additional macros have been defined in `prog2.c` (whether they are used or not).

The different definitions of macro `X` in `prog1.c` and `prog2.c` do not matter, since `X` is never tested or expanded.

Using the GENP and USEP Compiler Options

You can specify `GENP` or `USEP` with a `suboption`. If you do not specify a `suboption`, and did not allocate `DD SYSCPCH`, a filename is generated based on the source file name. The default suffixes are as follows:

Source file type	MVS File	HFS File
C Source file	PCH	pch
C++ Source file	PCHPP	pchpp

When you specify GENP and USEP together, the last file name that is specified will be used. If you compile your program as follows, the compiler uses the PCH name MY.PCH:

```
CC HELLO.PDSSRC(MEM1) (GENPCH(TEST.PCH) GENPCH(MY.PCH) USEPCH(MY.PCH)
```

The compiler then does the following:
- If MY.PCH exists and is current, the compiler uses it if the initial sequence matches the source initial sequence.
- If MY.PCH exists and is not current, the compiler regenerates it.
- If MY.PCH does not exist, the compiler generates it.

Depending on how source files are organized, a header file can be part of more than one PCH file. It could be tedious to keep track of changes to the header files and to keep the corresponding PCH files up-to-date. By consistently using GENP and USEP options together as described above, you can automatically maintain and use a current precompiled header.

Notes:
1. You cannot use the same precompiled header files for C and C++ programs.
2. To create a precompiled header file, you must have write access to the data sets or directories you specify. To use a precompiled header, you must have read permission for that file.
3. Precompiled header files do not appear in any listing files. An informational message is printed if the compiler does not use the precompiled header file.

Using an Alternative Initial Sequence

Because of the restrictions on reusing precompiled headers (the same sequence of headers, and the same context in terms of macro names and options), you can create and keep more than one precompiled header object. You can then use the one that suits your particular compilation.

You can specify the name of an alternate precompiled header file to use, or an alternate directory to search. Use the filename suboption of the GENP and USEP compiler options on the command line, or on the CPARM parameter of your JCL.

Restrictions

To use an existing precompiled header file, the compiler needs the same address location that was used when the file was created. If this address location is not available, the compiler will not use the precompiled header file. It will compile all the #include files, ignoring the USEP option.

You can increase the probability of accessing the required address location by following these rules:
- The compile time options must be the same during the generation and reuse of precompiled header files. Put the options into a file, and use the compiler option OPTFILE to ensure that you use consistent options to generate and reuse precompiled header files.
- For C, any #pragma directives that appear before the Initial Sequence must be the same during the generation and reuse of precompiled headers. To ensure that your #pragma directives are consistent for all compilations, put them inside a header file whenever possible. These #pragma directives need not be processed again in the reuse compile, since they are inside a header file that is part of the Initial Sequence. This improves the compile time.
- For C++, you cannot use a #pragma directive before the Initial Sequence.

- Maintain a consistent runtime environment when you invoke the compiler. You can do this by using the same runtime options (i.e. the compiler as an application, options like `HEAP`, `STACK`, etc), and the same `region size`. If you are working from OE by using the OMVS shell, start the shell up directly from TSO, instead of from ISPF. This will free more memory for the compiler.

There is no guarantee that you can use precompiled header files between different runtime environments. For example, between TSO and batch, or between different user sites or z/OS installations. The available address locations may also change after a system reconfiguration.

In addition, timestamp information must be available for the `#include` header files; otherwise the compiler may not create or use the precompiled header file. This is because the compiler needs the timestamp information to check whether a precompiled header file is up-to-date. Make sure that timestamp information is available in the system headers, which reside in system partition data sets and in the HFS directory `/usr/lpp` .

Because timestamp information is not available in sequential data sets, avoid using these for header files if you want to use the precompiled header feature. The precompiled header itself can be a sequential data set.

Organizing Your Source Files

To take full advantage of precompiled headers, you may need to reorganize your source files. There are two strategies that you can use to organize your source files. Use the one that best suits your application environment:
- A long initial sequence of headers. This method limits the number of source files that can reuse it, but provides significant improvement for those files.
- A short initial sequence of headers that are shared by many source files. This method increases the number of source files that can reuse it, but the performance improvement for any one compile is not as significant.

If your source has an `initial sequence` of header files which is common to all members in a PDS or directory, then generating and using one PCH file for the entire PDS of directory has the greatest potential performance benefit.

Set up the PCH as follows:
1. Compile a single member from the PDS or directory with `GENP` and `USEP(filename(member))`. This can be a dummy source file with only the Initial Sequence of header files. The purpose here is to let the compiler check whether the precompiled header file exists and is up-to-date, and to create or refresh it if necessary.
2. Compile the entire directory with `USEP(filename(member))`. You must specify the PCH file name, including the member name if the PCH is in a PDS, with the `USEP` option.

Note: If you specify `GENP` at this point, a the compiler generates a PCH for each member in the directory.

Use the following hints and suggestions to organize your source files.

Common Header File

Create a common header file which has `#include` directives for those header files that are shared by many different compilation units. `#include` this common header file as the first header file in each primary source file, followed immediately by `#pragma hdrstop`. The common headers will participate in the precompiled header file while the other headers will be compiled normally.

Global PCH File for the Entire Directory

Create a global header file which has `#include` directives for *every* header in your application, and include it in each primary source file. This means that a particular source file may not use all headers in the global header file.

One PCH file for Each Member of the Directory

If your source has a different `initial sequence` of header files for each member in a directory, you can generate and use one PCH file for each member by always compiling with `GENP` and `USEP`. The first time you compile this directory, the compilation will create a PCH file for each member. Subsequent compiles will reuse the PCH files, thereby improving your compile time.

You do not have to specify the PCH filename; you can use the defaults. If you do specify a PCH file name, it should be a PDS or an HFS directory. If you specify a sequential file, PDS member or HFS file, the compiler uses this name for the output of each PCH. If this happens, the PCH for each compile unit overwrites the previous PCH, and only the PCH for the last compile unit remains at the end.

Chapter 11. Using the IPA Link Step with z/OS C/C++ Programs

This chapter shows how to use the IPA (Interprocedural Analysis) Link step with your z/OS C/C++ program. Before reading this chapter, refer to the *z/OS C/C++ Programming Guide* for an overview of IPA.

The `IPA(LINK)` option triggers IPA Link step processing.

IPA Linking Your Program

The IPA Link step combines IPA object files that are created by the IPA Compile step with non-IPA object files and information from load module library members. The IPA Link step optionally performs IPA and code generation optimizations, and generates the final code and data for your program. You must bind the resulting object module to create the executable program.

The entry point of your application must be an IPA object file.

Typically, z/OS C/C++ applications contain references to z/OS Language Environment library functions, as well as interface routines for products such as CICS and DB2. These object module and load module libraries must be available to the IPA Link step for symbol resolution. The IPA Link step extracts all required object information from these libraries to form part of the object module it generates. If external references remain unresolved after the link portion of the IPA Link step has completed, processing terminates before optimization or code generation of the final object code. OS/390 Version 2 Release 4 has introduced the SCEELKEX library, which is a LONGNAME object version of a large portion of the Language Environment function library. When you IPA Link your application program, place the SCEELKEX library ahead of the SCEELKED library in the search order. This will preserve long runtime function names in the object module and listings that IPA Link generates.

You should specify the libraries that are described in the previous paragraph in your bind step. During IPA Link step processing with `IPA(NONCAL)` in effect, IPA resolves object information for explicit runtime symbols. The IPA Link step produces additional, implicit references to external runtime symbols during code generation. Although the IPA Link step will search for explicit runtime references, it does not search for implicit runtime references.

To avoid problems with unresolved implicit runtime references, ensure that the runtime object module and load module libraries are available to the binder. Also, check the binder listings and messages to make sure that all your symbols are resolved.

If you use the prelinker, make sure that the runtime object module libraries are available to the prelinker, and that the runtime object module and load module libraries are available to the Linkage Editor. The Object Resolution Warnings section of the Prelinker Map and the Linkage Editor Map display unresolved references, as follows:

```
=======================================================
|            Object Resolution Warnings               |
=======================================================

WARNING EDC4015: Unresolved references are detected:
CEEBETBL CEEROOTA EDCINPL
```

IPA object modules contain longnames, and may be included in object libraries for easy automatic library call resolution.

For information on creating object libraries in z/OS C/C++, refer to “Chapter 15. Object Library Utility” on page 411. For information on binding object modules under z/OS UNIX System Services, refer to “Chapter 12. Binding z/OS C/C++ Programs” on page 353.

Using DD Statements for the Standard Data Sets

The IPA Link step uses certain ddnames. Table 42 lists these ddnames, along with their types and functions. For details on the attributes of specific data sets see “Description of Data Sets Used” on page 533.

Table 42. Data Sets Used by the IPA Link Step

ddname	Type	Function
SYSIN[1]	Input	Primary input
STEPLIB[1],[5]	Utility Library	Location of the z/OS C/C++ compiler (which provides the IPA Link step) and the z/OS Language Environment data sets
SYSLIB	Library	Data set for runtime library (SCEELKEX,SCEELKED) [2] Optional data sets for secondary input [4]
SYSLIN[1]	Output	Output data set for the object module, if the `OBJECT` compiler option is specified
SYSPUNCH[1]	Output	Output data set for the object module, if the `DECK` and `NOOBJECT` compiler options are specified
SYSOUT[4]	Output	Destination of diagnostic messages generated by the IPA Link step
SYSCPRT[4]	Output	IPA Link step listing, generated if the `IPA(MAP)`, `LIST`, or `XREF` option is specified.
User-specified[3]	Input	Additional object modules and load modules
SYSUT1, SYSUT4-9, SYSUT14[1]	Output	Work data sets

Notes:

1 Required data set

2 Required for library runtime routines

3 As required by the program:

- Program parts in object library or load module library format
- DLL IMPORT side-decks generated by the binder, which define function or variable interfaces of a DLL referenced by the current application

4 Optional data set

5 Optional data sets, if the compiler and runtime library are installed in the LPA or ELPA. To save resources and improve compile time, especially in z/OS UNIX System Services, do not unnecessarily specify data sets on the STEPLIB DD name.

Primary Input (SYSIN)

Primary input to the IPA Link step must be one or more separately compiled object modules or IPA Link control statements. You can specify this input in a sequential data set, a member of a partitioned data set, or an in-line object module (DD *).

Location of Compiler and z/OS Language Environment Library (STEPLIB)

To IPA Link your program, the system must find the data sets that contain the compiler, and the data sets that contain the z/OS Language Environment runtime library. If the runtime library is installed in the LPA or ELPA, it is found automatically. Otherwise, `SCEERUN` must be in the `JOBLIB` or `STEPLIB`. For information on the search order, see "Chapter 14. Running a C or C++ Application" on page 401.

Secondary Input (SYSLIB)

Secondary input to the IPA Link step consists of object modules, or load modules that are not part of the primary input data set but are to be included in the user executable program. These may be included either:

- Explicitly, as a result of processing an IPA Link control INCLUDE statement.
- Implicitly, as a result of automatic call library processing. This can be due to either
  - Processing a library specified on an IPA Link control LIBRARY statement
  - Searching the libraries that are allocated to SYSLIB (once the IPA Link step has processed all primary input)

The automatic call library is used to resolve external symbols that are currently unresolved.

The call libraries that are used as input to the IPA Link step normally include the z/OS Language Environment libraries. If required, include additional call libraries such as SYS1.LINKLIB, a private program library, or a subroutine library to resolve all external references to your application.

If you are IPA Linking an application that imports symbols from a DLL, you must INCLUDE its definition side-deck on the SYSLIB or other user DD name. The IPA Link step uses the definition side-deck to resolve external symbols for functions and variables that your application imports. If you call more than one DLL, you need to `INCLUDE` a definition side-deck for each.

You can use the SYSLIB DD statement to concatenate multiple object module libraries and load module libraries. For more information on concatenating data sets, see page 292.

Notes:

1. All secondary input data sets for the IPA Link step must be cataloged.
2. The IPA Link step supports PDS format load module libraries only. It does not support Program Objects that are in PDSE format, or z/OS UNIX System Services HFS executable files.

Output (SYSLIN or SYSPUNCH)

The IPA Link step generates a single object module. If you specify the `OBJECT` compiler option, the IPA Link step stores the object module in the data set that is referenced by the SYSLIN DD name. If you specify the `DECK` and `NOOBJECT` compiler options, the IPA Link step stores the object module in the data set that is referenced by the SYSPUNCH DD name.

Destination of Errors Generated by the IPA Link Step (SYSOUT)

If the IPA Link step encounters problems, it generates diagnostic messages and places them in the `SYSOUT` data set.

Listing (SYSCPRT)

If you specify the `ATTRIBUTE`, `IPA(MAP)`, `LIST`, or `XREF` compiler option, the IPA Link step writes a listing to the `SYSCPRT` file name. The options have the following purposes:

`ATTRIBUTE`	Causes IPA Link to generate an External Symbol Cross-Reference listing section for each partition. The IPA Link step may also generate a Storage Offset Listing if you specified the `XREF`, `IPA(ATTRIBUTE)`, or `IPA(XREF)` option specified during the IPA Compile step.
`IPA(MAP)`	Provides information about the object and source files that are included as input to the IPA Link step, and information about the partitions that it generates.
`LIST`	Causes IPA Link to generate a Pseudo Assembly listing for each partition, showing the code and data that are generated in each partition.
`XREF`	Causes an IPA Link to generate an External Symbol Cross-Reference listing section to each partition. The IPA Link step may also generate a Storage Offset Listing if you specify the `XREF`, `IPA(ATTRIBUTE)`, or `IPA(XREF)` option specified during the IPA Compile step.

Refer to “Using the IPA Link Step Listing” on page 256 for more information about listings that the IPA Link step generates.

Temporary Workspaces for the IPA Link Step (SYSUTx)

The IPA Link step requires data sets for use as temporary workspaces. You define these data sets by DD statements with the names SYSUT1, SYSUT4—9, and SYSUT14. These data sets must be on direct access devices.

IPA Link Step Input

Input to the IPA Link step can be:

- Object records, which can be:
  - One or more IPA object modules
  - IPA Link control statements
  - z/OS Language Environment stub routines
  - Other object libraries and load module libraries
- The IPA Link step control file

Unresolved references or undefined writable static objects often result if you give the IPA Link step object modules produced with a mixture of inconsistent options. For example, `RENT`, `NORENT`, or `DLL`.

Objects processed with the `IPA(OBJONLY)` option are processed the same as objects with the `NOIPA` option.

Note: The IPA Link step will not accept as input a program object that is produced by the binder.

Primary Input

Primary input to the IPA Link step consists of a sequential data set (file) that contains one or more separately compiled object modules or IPA Link control statements. Specify the primary input data set through the `SYSIN` DD name.

Note: If you used the OS/390 Release 2 C/C++ compiler to create an IPA or combined IPA/conventional object module, and specified the `OPTIMIZE(0)` and `IPA(NOOPTIMIZE)` compiler options, your object module is incompatible with a later release of the z/OS C/C++ IPA Link step. You must recompile your source code with a later release of the C/C++ IPA Compile step before attempting to use the current release of the z/OS C/C++ IPA Link step.

Refer to "Object Record Formats" on page 336 for more information about the different types of object records.

IPA Linking Multiple Object Modules

z/OS C/C++ generates a CEESTART CSECT at the beginning of the object module in two situations:

- For a source program that contains the `main()` function, as long as you have not specified the `NOSTART` compiler option.
- For a source program containing a function for which a `#pragma linkage (`*name*`, FETCHABLE)` preprocessor directive applies.

When you IPA Link multiple object modules into a single object module, the binder resolves the entry point of the resulting object module to the external symbol CEESTART. If you want to control the entry point of the object module, use the `ENTRY` binder control statement or the c89 ″-e″ option.

For the IPA Link step, object modules containing the `main()` function or `#pragma fetchable` function must be IPA object files. If these object files are IPA Linked with other object modules produced by C, assembler, or other languages, the IPA object file containing the `main()` or `#pragma fetchable` function must be the first module to receive control. You must also ensure that the entry point of the resulting load module is resolved to the external symbol `CEESTART`. To ensure this, you can include the following binder `ENTRY` control statement in the input to the binder:

```
ENTRY CEESTART
```

If you are building a DLL with IPA, you must use the `ENTRY` control statement as described above.

Secondary Input

Secondary input to the IPA Link step consists of object modules, or load modules that are not part of the primary input data set but are to be included in the object module. They may be included either:

- Explicitly, as a result of processing an IPA Link control INCLUDE statement.
- Implicitly, as a result of automatic call library processing. This can be due to either
  – Processing a library specified on an IPA Link control LIBRARY statement

– Searching the libraries that are allocated to SYSLIB (once the IPA Link step has processed all primary input)

The automatic call library is used to resolve external symbols that are currently unresolved. The IPA Link step locates the library member in which the external symbols are defined, extracts the corresponding object information, and incorporates it in the output object module.

The automatic call library may include:

- Object module libraries. These may contain IPA object files or non-IPA object modules, and may contain the records of IPA Link control statements.

  These libraries may be:

  – PDS libraries

  – PDSE libraries

  – archive libraries

  Note: You do not normally use control statement records within secondary input with the `c89` utility. The `c89` utility allocates libraries that are passed in the `c89` invocation. You cannot allocate additional user autocall libraries with user-specified DD names.
- Load module libraries
- z/OS Language Environment libraries, if any of the z/OS Language Environment library functions are needed to resolve external references

Refer to “Object Record Formats” on page 336 for more information about the different types of object records.

Note: You can concatenate PDS, PDSE, and load module libraries together. However, you cannot concatenate archive libraries to other library types.

Specify the standard secondary input data sets with a SYSLIB DD statement. You can also explicitly reference secondary input, through IPA Link control statements.

Additional Object Modules and Load Modules as Input

You can explicitly reference secondary input through `INCLUDE` or `LIBRARY` control statements.

Use the `INCLUDE` statement to specify additional object information from object modules or load modules that you want included in the final object module.

Use the `LIBRARY` statement to specify additional libraries to be searched for object information from object modules or load modules to be included in the final object module. The IPA Link step only uses data sets that are specified by the `LIBRARY` statement if there are unresolved references once it has processed all other input.

When the IPA Link step encounters an `INCLUDE` statement, it incorporates the data sets that the statement specifies. If you specify the `IPA(NONCAL)` option, the IPA Link step performs a library search for currently unresolved symbols when it encounters a `LIBRARY` statement. If the processing of subsequent `INCLUDE` or `LIBRARY` statements results in new or unresolved symbols, the IPA Link step does not search a previously encountered library again. You need to specify another `LIBRARY` statement that points to the same library so that IPA Link searches it again.

Uppercase Name Resolution with the IPA(UPCASE) Option

If you specify the `IPA(UPCASE)` option, the IPA Link step makes an additional automatic library call pass against the SYSLIB DD statement. In this situation, symbol matching is case-insensitive. The purpose of this `IPA(UPCASE)` option is to provide support for linking assembler object routines without source changes. It is preferable to add `#pragma map` definitions for these symbols, so that IPA Link finds the correct symbols during normal automatic library call processing.

Processing the IPA Link Automatic Library Call

The IPA Link step uses the following process to resolve a referenced and currently undefined symbol, if you have specified the `IPA(NONCAL)` compiler option:

- If the data set contains a `C370LIB` directory created using the z/OS C/C++ Object Library Utility, and the `C370LIB` directory shows that a defined symbol by that name exists (with a case-insensitive exact match), the IPA Link step reads the PDS member containing that symbol.
- If the data set does not contain a `C370LIB` directory created using the z/OS C/C++ Object Library Utility and the reference is not to static external data, the IPA Link step reads the member or alias with the same name (with a case-sensitive exact match).

If unresolved symbols remain after IPA Link step has processed user input, and you specified the `NONCAL` option, the IPA Link step searches the files allocated to the SYSLIB DD name, as follows:

1. It searches for a case-insensitive exact match in the `C370LIB` and non-`C370LIB` libraries that are concatenated to the SYSLIB DD name, as described above.
2. If the symbol remains unresolved, IPA searches z/OS Language Environment for a library function with the same name as the symbol. (You must include the Language Environment stub library in the SYSLIB concatenation).
3. If the symbol is still unresolved, and you have specified the `IPA(UPCASE)` option, IPA searches using the uppercased name.

For more information about the z/OS C/C++ Object Library Utility, see “Chapter 15. Object Library Utility” on page 411.

References to Currently Undefined Symbols (External References)

If the IPA Link step finds unresolved references to external symbols after it has completed the link portion of its processing, it issues a diagnostic message and terminates processing.

Library Routine Considerations

z/OS Language Environment contains runtime libraries for all Language Environment-enabled languages: C, C++, COBOL, FORTRAN, and PL/I. For detailed instructions on linking and running z/OS C/C++ programs under z/OS Language Environment, refer to the *z/OS Language Environment Programming Guide*.

z/OS Language Environment is dynamic. That means that many of the functions, such as library functions, are not physically stored as a part of your executable program. Instead, only a small portion of code, known as a stub routine, is stored with your executable program. This results in a smaller executable module. There is a stub routine for each library function. Each stub routine has:
- The same name as the library function that it represents
- Enough code to locate the actual library function at run time

The C stub routines are in the file CEE.SCEELKED, or CEE.SCEELKEX. For detailed information on the runtime libraries see *Z/OS UNIX System Services Command Reference*.

Using DLLs

If you are building an application that imports symbols from a DLL, your input to the IPA Link step must include the definition side-deck that the binder produced when the DLL was built.

The IPA Link step uses longnames to resolve exported and imported symbols when it generates an object module for an application that is compiled with the DLL compiler option.

For information on how to create a DLL or an application that uses DLLs, see the *z/OS C/C++ Programming Guide*.

Object File Formats

The High Level Assembler (HLASM) and other z/OS compilers and language translators generate two object file formats:

Object File Format
: The standard S/370 "TEXT" object format, packaged as fixed-length 80 byte records. Extensions to the basic format support long external symbols when the z/OS C/C++ compiler LONGNAME option is in effect. IPA Link accepts input in object file format. The z/OS C/C++ compiler only produces files that are in object file format.

Generalized Object File Format (GOFF)
: A hierarchical object file format that was introduced with HLASM R2, and the z/OS Binder. IPA Link does NOT support this format as input.

Refer to *z/OS DFSMS Program Management* for more information on object file formats.

Object Record Formats

There are two basic types of object records which may be present in a file of object file format.

Binary Object Records

Binary object records provide information about your program. The records may include IPA object information, or code and data generated through the OBJECT suboption of the IPA compiler option during the IPA Compile step.

The records include the following types:

- ESD
- XSD
- TXT
- END
- RLD

The z/OS C/C++ compiler or an equivalent language translator may generate these object records.

IPA Link Control Statements

You can also specify control statement records as input. These statements can include the following types:

- `INCLUDE`
- `LIBRARY`
- `RENAME`
- `IMPORT`
- `ALIAS`
- `ENTRY`
- `NAME`

The `INCLUDE` and `LIBRARY` control statements explicitly identify secondary input files.

IPA Link control statements are contiguous records that you can specify in an object file or in a DD * stream. The syntax and format of these control statements are similar to those that the binder uses. The logical records can span multiple fixed-block, 80 column wide physical records.

You can specify blank records and comment control statements (those starting with an asterisk in column 1), but the IPA Link step ignores them.

The following table shows the format of records:

Table 43. IPA Link Control Statements

Start Column	End Column	Field Length	Description
1	1	1	Record type indicator • blank-Control • 0X02-Binary • "*"-Comment
2	71	70	Record data
72	72	1	Control statement continuation indicator • EBCDIC blank character-No continuation (required for last record) • non-blank EBCDIC character-Continuation
73	80	8	Record sequence number (optional field, contents not verified)

You can delimit character strings with blanks, commas, or parentheses. If character strings contain embedded blanks, you must enclose the strings in single quotes. If you want to enclose a name in single quotation marks, and it contains a single quotation mark, replace the single quotation mark with two adjacent ones. For example, if you want the name `SymbolNameWithAQuote'InTheMiddle`, specify it as follows: `'SymbolNameWithAQuote''InTheMiddle'`.

All 70 data characters of a control statement are significant. Control statements continue in column 2 (IPA conforms to the same convention as the Program Management Binder).

The IPA Link step performs syntax checking on the object records. If it finds an error, it issues a diagnostic message and indicates the location of the error. Records cannot continue past the end of an object file.

The following sections describe the IPA Link control statements.

ENTRY Control Statement: The `ENTRY` control statement has the following syntax:

```
►►──ENTRY──┬─function─────────┬──────────────────────────────────────────────►◄
           └─'──function──'───┘
```

function	An external symbol that will be used as the program entry point. Mixed-case longnames must be enclosed in quotes.

The IPA Link step processes `ENTRY` statements. It passes the statement to the binder.

IMPORT Control Statement: The `IMPORT` control statement has the following syntax:

```
►►──IMPORT──┬─CODE──┬─dll-name─────────┬──┬─function─────────┬─┬──────────────►◄
            │       └─'──dll-name──'───┘  └─'──function──'───┘ │
            └─DATA──┬─dll-name─────────┬──┬─variable─────────┬─┘
                    └─'──dll-name──'───┘  └─'──variable──'───┘
```

dll-name	The directory name (primary member or alias) or HFS filename of the load module or program object that contains the imported function or variable. The maximum length of a dll-name is 1024 characters. The maximum length of an HFS filename is 255 bytes.
variable	An exported variable name. It is a mixed-case longname.
function	An exported function name. It is a mixed-case longname.

The IPA Link step processes `IMPORT` statements. It passes the binder the statements that represent entry points that are present within the DLL.

INCLUDE Control Statement: The `INCLUDE` control statement has the following syntax:

```
►►──INCLUDE──┬─ddname─────────┬──(──┬─member──────────────┬──)──────────────►◄
             └─'──ddname──'───┘     │  ┌─,─────────────┐  │
                                    │  ▼               │  │
                                    └───'──member──'───┴──┘
```

ddname	a ddname associated with a file to be included.
member	the member of the `DD` to be included.

The IPA Link step attempts to read the `DD` or member of the `DD` (whichever you specify), and if successful, resolves the INCLUDE request.

Note: The IPA Link step removes the `INCLUDE` control statement and does not place it in the IPA Link output object module.

LIBRARY Control Statement: The LIBRARY control statement has the following syntax:

```
►►──LIBRARY──┬─name─────┬─────────────────────────────────────────────►◄
             └─'──name──'─┘
```

name	The name of a DD that defines a library. This could be a concatenation of one or more libraries that were created with or without the Object Library Utility.

Note: The IPA Link step removes the LIBRARY control statement and does not place it in the IPA Link output object module.

IPA Link Step Control File

The IPA Link Step control file is a fixed-length or variable-length format file that contains additional IPA processing directives. The CONTROL suboption of the IPA compiler option identifies this file.

The IPA Link step issues an error message if any of the following conditions exist in the control file:

- The control file directives have invalid syntax.
- There are no entries in the control file.
- Duplicate names exist in the control file.

You can specify the following directives in the control file. Note that in the listed directives, *name* can be a regular expression. Thus, *name* can match multiple symbols in your application through pattern matching. For more information on regular expressions, see "Using Regular Expressions" on page 342.

csect=csect_names_prefix

Supplies information that the IPA Link step uses to name the CSECTs for each partition that it creates. The *csect_names_prefix* parameter is a comma-separated list of tokens that is used to construct CSECT names.

The behavior of the IPA Link steps varies depending upon whether you specify the CSECT option with a qualifier.

- **If you do not specify the CSECT option with a qualifier**, the IPA Link step does the following:
  - Truncates each name prefix or pads it at the end with @ symbols, if necessary, to create a 7 character token
  - Uppercases the token
  - Adds a suffix to specify the type of CSECT, as follows:

C	code
S	static data
T	test

- **If you specify the CSECT option with a non-null qualifier**, the IPA Link step does the following:
  - Uppercases the token
  - Adds a suffix to specify the type of CSECT, as follows where *qualifier* is the qualifier you specified for CSECT and *nameprefix* is the name you specified in the IPA Link Step Control File:

qualifier#nameprefix#C	code

qualifier#nameprefix#S	static data
qualifier#nameprefix#T	test

- **If you specify the `CSECT` option with a null qualifier**, the IPA Link step does the following:
  - Uppercases the token
  - Adds a suffix to specify the type of `CSECT`, as follows where *nameprefix* is the name you specified in the IPA Link Step Control File:

nameprefix#C	code
nameprefix#S	static data
nameprefix#T	test

The IPA Link step issues an error message if you specify the `CSECT` option but no control file, or did not specify any `csect` directives in the control file. In this situation, IPA generates a CSECT name and an error message for each partition.

The IPA Link step issues a warning or error message (depending upon the presence of the `CSECT` option) if you specify CSECT name prefixes, but the number of entries in the `csect_names` list is fewer than the number of partitions that IPA generated. In this situation, for each unnamed partition, the IPA Link step generates a CSECT name prefix with format `@CSnnnn`, where `nnnn` is the partition number. If you specify the `CSECT` option, the IPA Link step also generates an error message for each unnamed partition. Otherwise, the IPA Link step generates a warning message for each unnamed partition.

noexports
Removes the ″export″ flag from all symbols (functions and variables) in IPA and non-IPA input files.

export=*name[,name]*
Specifies a list of symbols (functions and variables) to export by setting the symbol ″export″ flag. Note: Only symbols defined within IPA objects can be exported using this directive.

inline=*name[,name]*
Specifies a list of functions that are desirable for the compiler to inline. The functions may or may not be inlined.

inline=*name[,name]* **from** *name[,name]*
Specifies a list of functions that are desirable for the compiler to inline, if the functions are called from a particular function or list of functions. The functions may or may not be inlined.

noinline=*name[,name]*
Specifies a list of functions that the compiler will not inline.

noinline=*name[,name]* **from** *name[,name]*
Specifies a list of functions that the compiler will not inline, if the functions are called from a particular function or list of functions.

exits=*name[,name]*
Specifies names of functions that represent program exits. Program exits are calls that can never return, and can never call any procedure that was compiled with the IPA Compile step.

lowfreq=*name[,name]*

Specifies names of functions that are expected to be called infrequently. These functions are typically error handling or trace functions.

partition=small|*medium***|large|unsigned-integer**

Specifies the size of each program partition that the IPA Link step creates. The size of the partition is directly proportional to the time that is required to perform code generation, and the quality of the generated code. When partition sizes are large, it usually takes longer to complete the code generation, and the quality of the generated code is usually better.

For a finer degree of control, you can use an *unsigned-integer* value to specify the partition size. The integer is in ACUs (Abstract Code Units), and its meaning may change between releases. You should only use this integer for very short term tuning efforts, or when the number of partitions (and therefore the number of CSECTs in the output object module) must remain constant.

The size of a CSECT cannot exceed 16 MB.

The default for this directive is `medium`.

partitionlist=*partition_number***[,***partition_number***]**

Used to reduce the size of an IPA Link listing. If the IPA Link control file contains this directive and the `LIST` option is active, a pseudo-assembly listing is generated for only these partitions.

partition_number is a decimal number representing an `unsigned int`.

safe=*name[,name]*

Specifies a list of "safe" functions that are not compiled as IPA objects. These are functions that do not indirectly call a visible (not missing) function either through a direct call or a function pointer. Safe functions can modify global variables, but may not call functions that are not compiled as IPA objects.

isolated=*name[,name]*

Specifies a list of "isolated" functions that are not compiled as IPA objects. Neither isolated functions nor functions within their call chain can refer to global variables. IPA assumes that functions that are bound from shared libraries are isolated.

pure=*name[,name]*

Specifies a list of "pure" functions that are not compiled as IPA objects. These are functions that are ″safe″ and ″isolated″ and do not indirectly alter storage accessible to visible functions. A "pure" function has no observable internal state nor has side-effects, defined as potentially altering any data visible to the caller. This means that the returned value for a given invocation of a function is independent of any previous or future invocation of the function.

unknown=*name[,name]*

Specifies a list of "unknown" functions that are not compiled as IPA objects. These are functions that are not safe, isolated, or pure. This is the default for all functions defined within non-IPA objects. Any function specified as ″unknown″ can make calls to other parts of the program compiled as IPA objects and modify global variables and dummy arguments. This option greatly restricts the amount of interprocedural optimization for calls to ″unknown″ functions.

missing=*attribute*

Specifies the characteristics of "missing" functions. There are two types of "missing" functions:

- Functions dynamically linked from another DLL (defined using an IPA Link IMPORT control statement)
- Functions that are statically available but not compiled with the `IPA` option

IPA has no visibility to the code within these functions. You must ensure that all user references are resolved at IPA Link time with user libraries or runtime libraries.

The default setting for this directive is `unknown`. This instructs IPA to make pessimistic assumptions about the data that may be used and modified through a call to such a missing function, and about the functions that may be called indirectly through it.

You can specify the following attributes for this directive:

safe	Specifies that the missing functions are "safe". See the description for the `safe` directive, above.
isolated	Specifies that the missing functions are "isolated". See the description for the `isolated` directive, above.
pure	Specifies that the missing functions are "pure". See the description for the `pure` directive, above.
unknown	Specifies that the missing functions are "unknown". See the description for the `unknown` directive, above. This is the default attribute.

retain=*symbol-list*

Specifies a list of exported functions or variables that the IPA Link step retains in the final object module. The IPA Link step does not prune these functions or variables during optimization.

LIBANSI Option and Symbol Attributes

The IPA Link step has attribute information for the Language Environment runtime entry points. When the `LIBANSI` option is active and Level (2) IPA optimization is selected during the IPA Link step, this information will be used to improve the generated code.

Using Regular Expressions

You can specify various attributes about functions such as controlling whether a function is inline or not through the IPA control file. For example, the following directives directs IPA not to inline functions `foo1()`, `foo2()`, or `foo3()`. It directs IPA to inline functions `bar1()`, `bar2()`, and `bar3()`. It also declares that `trace1()`, `trace2()`, and `trace3()` are low frequency functions:

```
noinline=foo1,foo2,foo3
inline=bar1,bar2,bar3
lowfreq=trace1,trace2,trace3
```

Regular expressions are a common form of expressing pattern matching. The most common forms of regular expressions are listed below. Using regular expressions, you can describe the list of functions to which these attributes and inline directives refer. For example, you could rewrite the above directives as:

```
noinline=foo[123]
inline=bar[123]
lowfreq=trace[123]
```

The directives for which regular expressions are supported in the IPA control file are:

- inline
- noinline
- unknown
- lowfreq
- exits
- pure
- safe

You can also find additional information on using regular expressions in the z/OS UNIX shell in the *Z/OS UNIX System Services Command Reference*.

`string`
: A regular string of characters matches the same string of characters in the item you are searching. Thus you can search for all occurrences of the string `test` by using the regular expression `test`. This will also match lines containing the strings `testimony`, `latest` and `intestine`.

`start (ˆ)`
: Indicates a *beginning of line* in a match. For example `ˆtest` matches all lines that begin with `test`. Note that this must appear as the left most character in the expression.

`end ($)`
: Indicates an *end of line* in a match. The regular expression `test$` will match those lines that end with `test`, and `ˆtest$` will match those lines that contain only `test`. Note that to work as the end of line, the `$` must be the last character in the expression.

`single character (.)`
: The period matches any character. Matches for `t.st` includes `test`, `tast`, `tZst`, and `t1st`.

`escape (\)`
: Use the back slash character to *escape* special characters if you want to search for them in the expression. For example, if you wanted to find those lines ending with a period, the expression `.$` shows all lines that have at least one character in them. You need to escape the `.` as `\.$`. Similarly to find those places with the two characters period and dollar sign next to each other, use `\.\`. You also have to escape the back slash character to match it by using `\\`.

`character set ([ ])`
: Represents a set of characters to compare against a single character. Ranges of characters can be represented by the first character in the series, followed by a hyphen, then the last character in the series. For example, [abcdefg] is the same as [a–g]. The pattern `t[a–g123]st` matches `tast`, `test`, and `t2st`, but not `t–st`, `taast` nor `tAst`. Note that the special characters `\`, `]`, `–`,

and ^, must be escaped. See the escape character, (\), above. The special character] can appear as the first character in the range without being escaped.

`complemented character set ([^ ])`
: The special character ^ placed at the beginning of a character set indicates that the character set must not match the single character. For example, `t[^a-zA-Z]st` matches `t1st`, `t–st`, or `t,st`, but not `test` or `tYst`.

`grouping (( ))`
: You can use parentheses to group expressions. This is useful when you want to use a string as basis for a pattern. For example, `(test)+` matches multiple instances of `test`.

`alternation (|)`
: This is an *or* operator. It allows the extension of a pattern to include alternative matches. For example, to match those lines beginning with `Page` or ending with `Foot`, you could use the expression `^Page|Foot$`.

`replication (*)`
: When a character pattern is followed by a `*`, the pattern will match zero or more instances of that pattern. For example, the pattern `te*st` matches `tst`, `test`, `teeeeeest`, and so on.

`replication (+)`
: This modifier works the same way as `*`, but it matches at least one instance of the pattern. For example, `t(es)+t` will match `test`, `tesest`, and so on, but not `tt`.

`replication (?)`
: The question mark is used in a fashion similar to the two previous patterns, but matches exactly one or zero matches. Thus, `te?st` will matter either `tst` or `test`.

`replication ({m,n})`
: For more rigid replicated pattern matches, the braces can be used to indicate an exact number, a minimum number, or a range of numbers of replications. To match exactly `m` copies, use the form `{`*m*`}`, where *m* is a number between 1 and 255. To match a minimum of *m* copies, use the form `{`*m*`}`. To match between *m* and *n* copies, use the form `{`*m*`,`*n*`}}` where *n* must be between *m* and 255. For example, `a{2}` matches `aa`, `b{1,4}` matches `b`, `bb`, `bbb` and `bbbb`.

Output from the IPA Link Step

You can specify output from the IPA Link step as one of the following:
1. A sequential data set
2. A member of a partitioned data set
3. A partitioned data set
4. A hierarchical file system (HFS) file
5. An HFS directory

Output may be either an object module or a listing.

For valid combinations of input and output file types, refer to Table 37 on page 287.

Specifying Output Files

You can use compiler options to specify the output files for IPA Link, as follows:

Table 44. Compiler Options That Provide Output File Names

Output File Type	**Compiler Option**
Object Module	OBJECT(filename)
Listing File	LIST(filename)

If you specify compiler options that generate output files but do not specify the suboptions to identify the output files or allocate the ddnames, the IPA Link step generates the output file names based on the input file name. For data sets, the IPA Link step uses the userid under which the compiler is running as the high-level qualifier. It generates the low-level qualifier by appending a suffix, as shown in Table 45. z/OS creates HFS files in the current working directory.

The IPA Link step uses the following default suffixes:

Table 45. Default Suffixes for Output File Types

Output File Type.	**MVS File**	**HFS File**
Output from IPA Compile Step	OBJ	`o`
Listing File	LIST	`lst`
Output from IPA Link Step	IPA	`I` (for c89 batch) or `o` (otherwise)

Refer to the *Z/OS UNIX System Services Command Reference* for more information about default suffixes.

Note: Output files default to the HFS directory if the input resides in the HFS, or to the MVS file if the input resides in a data set.

If you use the c89 utility to compile HFS source files and perform an IPA Link in one invocation, and do not specify output filenames in the compiler options, the compiler writes output files to the current working directory. It generates output file names by:
- Appending a suffix, if it does not exist
- Replacing the suffix, if it exists

as shown in Table 45. For example, the following command generates the IPA Compile step object file `./hello.o` and the IPA Link step object file `./hello.I` :

```
cc  /user/tullio/hello.c
```

The IPA Link step object file `./hello.I` is temporary, but you can use environment variables to make it permanent. Refer to the z/OS Shells and Utilities manual for more information.

Notes:
1. If you have specified the `OE` option, see "OE | NOOE" on page 150 for a description of the default naming convention.
2. If you supply the primary input file inline in your JCL, you must provide a file name for the output, or route it to the job log. The compiler will not generate an output file name automatically. You can specify a file name either as a suboption for a compiler option, or on a ddname in your JCL.

Listing Output

To create a listing file that contains source, object or inline reports, use one of the following:

- the `MAP` suboption of the `IPA` option
- the `AGGREGATE` option
- the `LIST` option
- the `INLINE(,REPORT,,)` option
- the `XREF` option

The IPA Link Step listing contains several individual listing sections that are only generated if required. Unresolved requests generate error or warning messages in the listing.

The listing includes the results of the default or specified options of the `IPARM` parameter (that is, the diagnostic messages and the object code listing). If you specified *filename* with two or more of these compile options, the IPA Link step combines the listings and writes them to the last file named. If you did not specify any suboptions, the IPA Link step writes the listing to the `SYSCPRT` DD name, if you have allocated it. Otherwise, the IPA Link step generates a default file name as described in “LIST | NOLIST” on page 133.

Object Module Output

To create an object module and store it on disk or tape, you can use either the `OBJECT` or `DECK` compiler option.

If you do not specify a *filename* with the `OBJECT` compiler option, the IPA Link step stores the object code in the file that you defined in the `SYSLIN DD` statement. With the `DECK` and `NOOBJECT` compiler options, the IPA Link step uses the file that you defined in the `SYSPUNCH DD`. If you did not specify any suboptions, and did not allocate SYSLIN, the IPA Link step generates a default file name as described in “OBJECT | NOOBJECT” on page 148.

You must use the binder (or the prelinker, followed by the linkage-editor) to process the object module from the IPA Link step. You should not use the output object module from one IPA Link step as input to another IPA Link step.

Mapping Static Symbol Names

Static symbols (such as C static functions) within a compilation unit are not exposed as external symbols if an application program is built using the non-IPA compilation process.

The IPA Link merges and optimizes the IPA object information, and splits it into partitions for final code and data generation. The partitioning process must flexibly assign the code and data from the original Compilation Units to their final partition based on how they are used within the application.

As the IPA Link step reads IPA object modules, it assigns each static static symbol a unique name and promotes it to an external symbol. This prevents static symbols from constraining the partitioning. IPA Link generates the unique name by adding a prefix to the original static name, as follows:

@n@original_name
: where *n* is the object file id number. Refer to the Object File Map section of the “Using the IPA Link Step Listing” on page 256 for details.

If an object file defines multiple static symbols with the same name, IPA Link generates the unique name for the subsequent symbols as follows:

@n@m@original_name
where:

n	is the object file id number.
m	is the collision counter, starting with 1.

Running the IPA Link Step Under z/OS Batch

The following diagram shows the basic IPA Link step process for your C/C++ application.

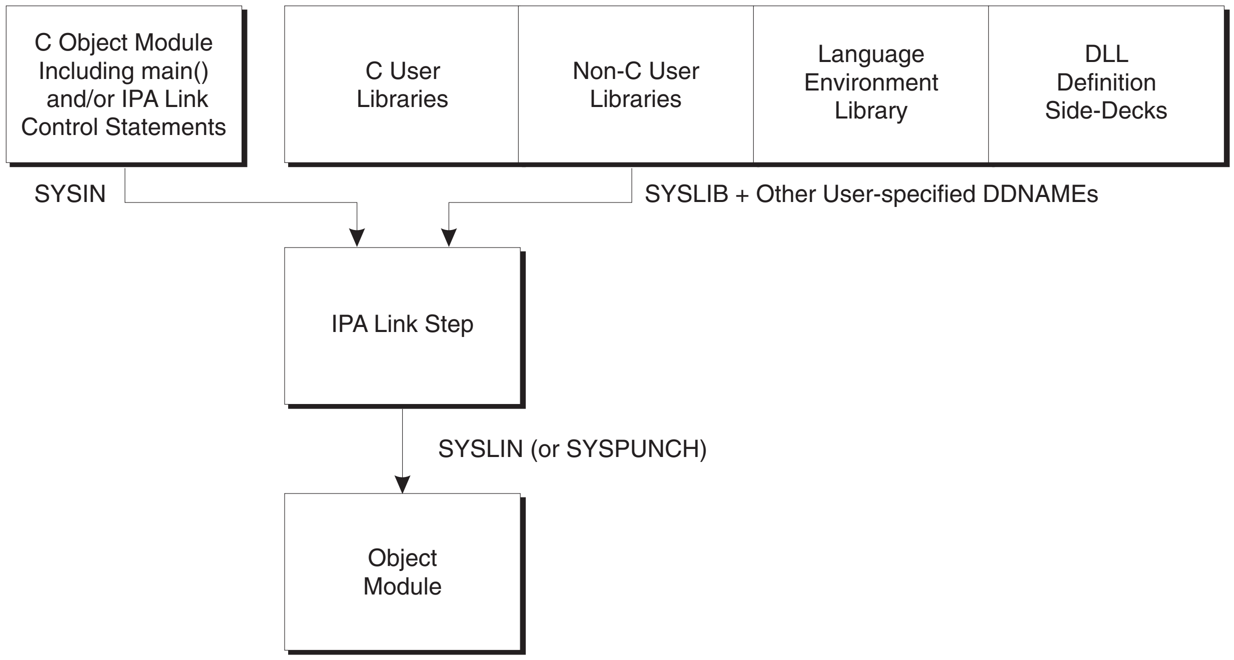

C Object Module Including main() and/or IPA Link Control Statements
C User Libraries
Non-C User Libraries
Language Environment Library
DLL Definition Side-Decks
SYSIN
SYSLIB + Other User-specified DDNAMEs
IPA Link Step
SYSLIN (or SYSPUNCH)
Object Module

Figure 34. Basic IPA Link Step Processing

Use the SYSIN DD statement to specify your primary input. This may be object modules or IPA Link step control statements.

Use the SYSLIB DD statement to specify your secondary input. Your secondary input may be C/C++ user libraries, non-C/C++ user libraries, or the Language Environment library. Also, if you are creating an application that imports symbols from DLLs, you must INCLUDE the definition side-deck for each DLL from the SYSLIB DD statement.

You can specify additional secondary input through user-specified ddnames.

The IPA Link step stores the final object module that it generates in the data set that is referenced by the SYSLIN or SYSPUNCH DD name.

Using the EDCI and CBCI Cataloged Procedures

You can use the IBM-supplied cataloged procedures EDCI and CBCI to perform IPA Link step processing on your program. The two procedures are the same. IBM provides CBCI to conform to the procedure naming conventions of C++, and CBCI is aliased to EDCI. Note that by default, the EDCI procedure does not save the generated object module.

The EDCI procedure specifies the `IPA(LINK)` option for you.

The following example shows the general job control procedure for IPA linking a program under z/OS batch:

```
// jobcard
//*
//* IN THE FOLLOWING STEP, THE MEMBERS TESTFILE AND DECODE FROM
//* THE LIBRARIES USERID.WORK.OBJECT AND USERID.LIBRARY.OBJECT ARE
//* IPA LINKED, AND THE GENERATED OBJECT MODULE IS PLACED
//* IN USERID.WORK.IPAOBJ(TEST).
//* AN IPA LINK LISTING IS GENERATED AND DIRECTED TO SYSOUT=*.
//*
//IPALINK EXEC EDCI,
//     INFILE='SEE.SYSIN.OVERRIDE',
//     OUTFILE='USERID.WORK.IPAOBJ(TEST),DISP=SHR',
//     IPARM='IPA(MAP,LIST,DUP,ER,NONCAL)',
//     IREGSIZ=64M
//SYSLIB   DD  DSNAME=CEE.SCEELKED,DISP=SHR
//OBJECT   DD  DSNAME=USERID.WORK.OBJECT,DISP=SHR
//LIBRARY  DD  DSNAME=USERID.LIBRARY.OBJECT,DISP=SHR
//SYSOUT   DD  SYSOUT=*
//SYSCPRT  DD  SYSOUT=*
//SYSIN DD DATA,DLM=@@
 INCLUDE OBJECT(TESTFILE)
 INCLUDE LIBRARY(DECODE)
@@
```

Figure 35. IPA Linking a Program under z/OS Batch

Specifying IPA Link Options

Use the IPARM parameter to specify the IPA Link options. The format of the parameter is:

```
IPARM='"ipa-link-options" '
```

where *ipa-link-options* is a list of IPA Link options, separated by commas.

Specifying Region Size

Use the IREGSIZ parameter to specify the IPA Link step region. The format of the parameter is:

```
IREGSIZ=region-size
```

Specifying Secondary Input under z/OS Batch

Specify the secondary input data sets with the SYSLIB DD statement. Add `LIBRARY` and `INCLUDE` control statements to reference object and load module library data sets. If you have multiple secondary input data sets, concatenate them as shown in the following example:

```
//SYSLIB DD DSNAME=CEE.SCEELKED,DISP=SHR
//       DD DSNAME=AREA.SALESLIB,DISP=SHR
```

To specify additional object modules or libraries, code `INCLUDE` and `LIBRARY` statements after your `DD` statements as part of your job control procedure, as follows:

```
   ⋮
//SYSIN   DD DSNAME=myid.IPAOBJ,DISP=SHR
//        DD DSNAME=...
   ⋮
//        DD *
  INCLUDE ddname(member)
     LIBRARY ADDLIB(CPGM10)
/*
```

Figure 36. IPA Link Control Statements

Using Your Own JCL

The following example shows sample JCL for running the IPA Link step:

```
//jobname    JOB   acctno,name...
//COMPILE    EXEC  PGM=CBCDRVR,PARM='/IPA(MAP,LIST,DUP,ER,NONCAL)'
//STEPLIB    DD    DSNAME=CEE.SCEERUN,DISP=SHR
//           DD    DSNAME=CBC.SCBCCMP,DISP=SHR
//SYSLIN     DD    DSNAME=userid.MYPROG.OBJ,DISP=SHR
//SYSLIB     DD    DSNAME=userid.SECOND.LOAD,DISP=SHR
//           DD    DSNAME=CEE.SCEELKED,DISP=SHR
//OBJECT     DD    DSNAME=userid.WORK.OBJECT,DISP=SHR
//LIBRARY    DD    SYSOUT=userid.LIBRARY.OBJECT,DISP=SHR
//SYSOUT     DD    SYSOUT=*
//SYSCPRT    DD    SYSOUT=*
//SYSUT1     DD    DSN=...
//SYSUT4     DD    DSN=...
...
//SYSIN      DD *
 INCLUDE OBJECT(TESTFILE)
 INCLUDE LIBRARY(DECODE)
/*
```

Figure 37. JCL for Running the IPA Link Step

Running the IPA Link Step in z/OS UNIX

Processing your application under z/OS UNIX System Services is the same as processing it under z/OS batch.

Using JCL

The example JCL, below, uses archive libraries and data sets. INCLUDE files may be PDS members, sequential files, or HFS files. Libraries may be partitioned data sets or archive libraries.

```
// jobcard
//*
//* THE FOLLOWING STEP IPA LINKS THE OBJECT FILES DEFINED BY DDOBJ1,
//* AND DDOBJ2 AND PLACES THE GENERATED OBJECT MODULE IN
//* USERID.WORK.IPAOBJ(TEST). AN IPA LINK LISTING IS GENERATED AND
//* DIRECTED TO SYSOUT=*.
//*
//IPALINK EXEC EDCI,
//     INFILE='SEE.SYSIN.OVERRIDE',
//     OUTFILE='USERID.WORK.IPAOBJ(TEST),DISP=SHR',
//     IPARM='IPA(MAP,LIST,DUP,ER,NONCAL)',
//     IREGSIZ=64M
//SYSLIB   DD  DSNAME=CEE.SCEELKED,DISP=SHR
//* object file
//DDOBJ1   DD PATH='/u/myuserid/callfoogoohoo.o',
//            PATHOPTS=(ORDONLY),
//            PATHDISP=(KEEP,KEEP)
//* PDS member
//DDOBJ2   DD DISP=SHR,DSN=MYUSERID.QAPARTNR.OBJ(MEM1)
//* archive library
//DDLIB3   DD PATH='/u/myuserid/mylibrary.a',
//            PATHOPTS=(ORDONLY),
//            PATHDISP=(KEEP,KEEP)
//* PDS Library
//DDLIB4   DD DISP=SHR,DSN=MYUSERID.QAPARTNR.OBJ
//SYSLIN   DD DISP=SHR,DSN=USERID.WORK.IPAOBJ(TEST)
//SYSOUT   DD  SYSOUT=*
//SYSCPRT  DD  SYSOUT=*
//SYSIN    DD *
  INCLUDE  DDOBJ1
  INCLUDE  DDOBJ2
  LIBRARY  DDLIB3
  LIBRARY  DDLIB4
/*
```

Figure 38. Using JCL for IPA Linking z/OS UNIX Applications

Invoking IPA from the c89 Utility

The `c89` utility supports IPA. You can invoke the IPA Compile step, the IPA Link step, or both. The step that `c89` invokes depends upon the invocation parameters and type of files you specify. You must specify the `I` phase indicator along with the `W` option of the `c89` utility. You can specify `IPA` suboptions as comma-separated keywords.

If you invoke `c89` with a source file and the `-c` option, `c89` automatically specifies the `IPA(NOLINK)` option and invokes the IPA compile step. For example, the following command invokes the IPA Compile step for source file hello.c:

```
c89 -c -WI hello.c
```

If you invoke `c89` with an object file, do not specify the `-c` option and do not specify any source files, `c89` automatically specifies `IPA(LINK)` and invokes the IPA Link step, and the binder. For example, the following command invokes the IPA Link step and the binder to create a program called hello:

```
c89 -o hello -WI hello.o
```

If you invoke `c89` with at least one source file and any number of object files, and do not specify the `-c` option, `c89` automatically invokes the IPA Compile step once for each compilation unit and the IPA Link step once for the entire program. For example, the following command invokes the IPA Compile step, the IPA Link step, and the binder while creating program foo:

```
c89 -o foo -WI,object foo.c
```

Specifying Options

When using `c89`, you can pass options to IPA, as follows:

- If you specify `-WI,` followed by `IPA` suboptions, c89 passes those suboptions to both the IPA Compile step and the IPA Link step.
- If you specify `-Wc,` followed by compiler options, c89 passes those options only to the IPA Compile step.
- If you specify `-Wl,I,` followed by compiler options, c89 passes those options only to the IPA Link step.

The following is an example of passing options:

```
c89 -2 -WI,noobject -Wc,source -Wl,I,"maxmem(2048)" file.c
```

In this example, you pass the `IPA(NOOBJECT)` option to both the IPA Compile and IPA Link steps, the `SOURCE` option only to the IPA Compile step, and the `MAXMEM(2048)` option only to the IPA Link step.

Using IPA Link with Archive Files

The IPA Link step supports all archive files, including those which are empty.

Other Considerations

The compiler (which includes IPA) is packaged in MVS load module format, not in HFS executable format.

Refer to “Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file” on page 555 for more information about the `c89` utility.

Chapter 12. Binding z/OS C/C++ Programs

This chapter describes how to bind your programs using the binder (the DFSMS/MVS program management binder) in the z/OS batch, z/OS UNIX System Services, and TSO environments.

When You Can Use the Binder

The output of the binder is a program object. You can store program objects in a PDSE member or in an HFS file. Depending on the environment you use, you can produce binder program objects as follows:

- For c89:

  If the targets of your executables are HFS files, you can use the binder. If the targets of your executables are PDSs, you must use the prelinker, followed by the binder. If the targets of your executables are PDSEs, you can use the binder alone.
- For z/OS batch or TSO:

  If you can use PDSEs, you can use the binder. If you want to to use PDSs, you must use the prelinker for the following:
  - C++ code
  - C code compiled with the `LONGNAME`, `RENT`, or `DLL` options
- For `GOFF` and `XPLINK`:

  If you have compiled your program with the `GOFF` and/or `XPLINK` compiler options, you must use the binder.

For more information on the prelinker, see “Appendix A. Prelinking and Linking z/OS C/C++ Programs” on page 461.

When You Cannot Use the Binder

The following are the restrictions to using the binder to produce a program object.

Your Output is a PDS, not a PDSE

If you are using z/OS batch or TSO, and your output must target a PDS instead of a PDSE, you cannot use the binder.

CICS

Prior to CICS 1.3, PDSEs are not supported. From CICS Transaction Server 1.3 onwards, there is support in CICS for PDSEs. Please refer to *CICS Transaction Server for OS/390 Release Guide*, where there are several references to PDSEs, and a list of prerequisite APAR fixes.

MTF

MTF does not support PDSEs. If you have to target MTF, you cannot use the binder.

IPA

Object files that are generated by the IPA Compile step using the compiler option `IPA(NOLINK,OBJECT)` may be given as input to the binder. Such an object file is a combination of an IPA object module, and a regular compiler object module. The binder processes the regular compiler object module, ignores the IPA object module, and no IPA optimization is done.

Object files that are generated by the IPA Compile step using compiler option `IPA(NOLINK,NOOBJECT)` should not be given as input to the binder. These are IPA only object files, and do not contain a regular compiler object module.

The IPA Link step will not accept a program object as input.

Using Different Methods to Bind

This section shows you how to use different methods to bind your application:

Single Final Bind
: Compile all your code and then perform a single final bind of all the object modules.

Bind Each Compile Unit
: Compile and bind each compilation unit, then perform a final bind of all the partially bound program objects.

Build and Use DLLs
: Build DLLs and programs that use those DLLs.

Rebind a Changed Compile Unit
: Recompile only changed compile units, and rebind them into a program object without needing other unchanged compile units.

Single Final Bind

You can use the method that is shown in Figure 39 on page 355 to build your application's executable for the first time. With this method, you compile each source code unit separately, then bind all of the resultant object modules together to produce an executable program object.

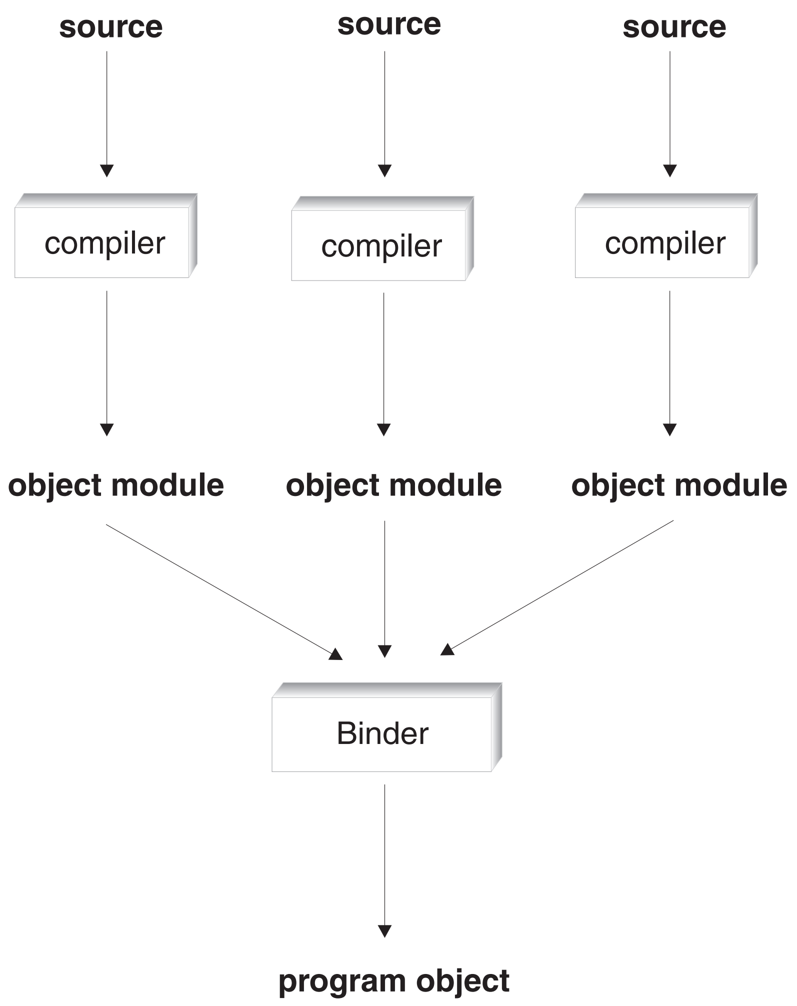


Figure 39. Single Final Bind

Bind Each Compile Unit

If you have changed the source in a compile unit, you can use the method that is shown in Figure 40 on page 356. With this method, you compile and bind your changed compile unit into an intermediate program object, which may have unresolved references. Then you bind all your program objects together to produce a single executable program object.

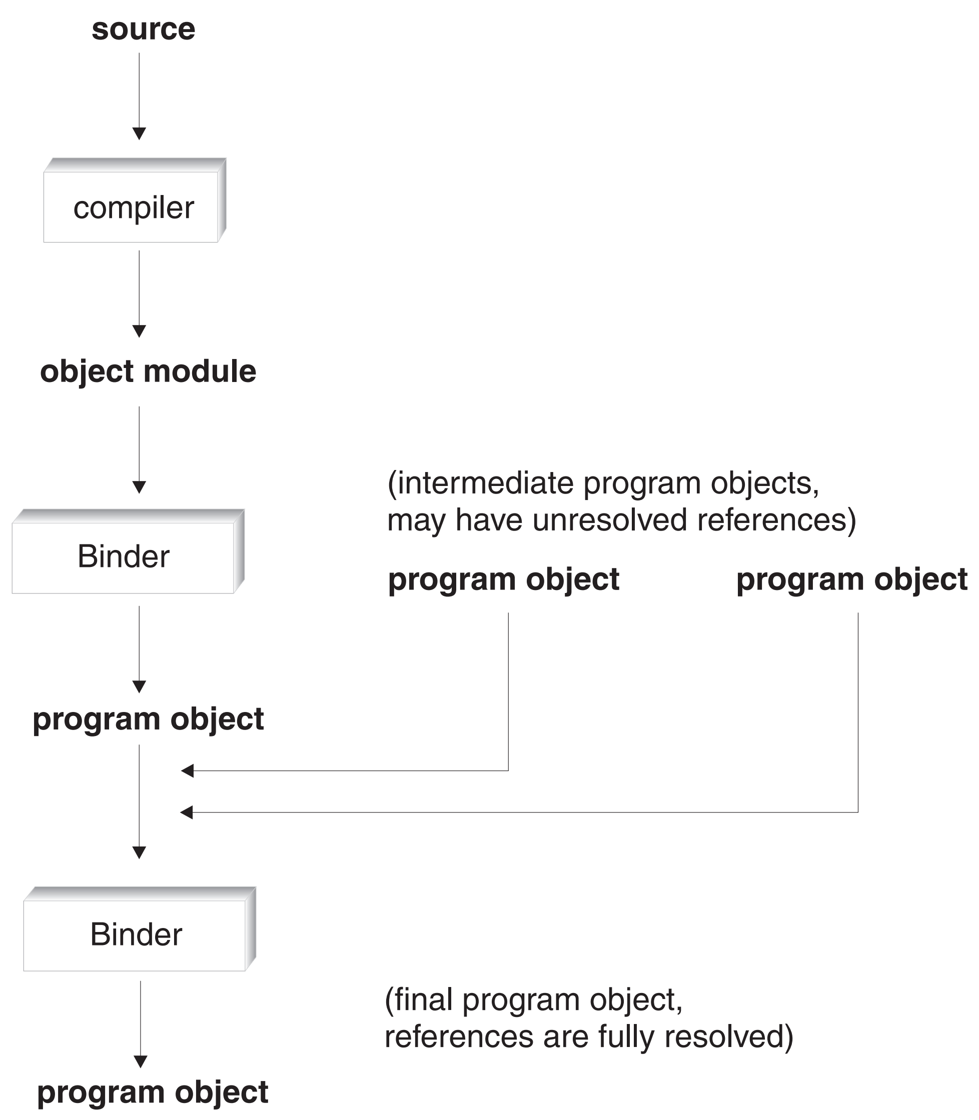


Figure 40. Bind Each Compile Unit

Build and Use a DLL

You can use the method that is shown in Figure 41 on page 357 to build a DLL. To build a DLL, the code that you compile must contain symbols which indicate that they are exported. You can use the compiler option `EXPORTALL` or the `#pragma export` directive to indicate symbols in your C or C++ code that are to be exported. For C++, you can also use the `_Export` keyword.

When you build the DLL, the bind step generates a DLL and a file of IMPORT control statements which lists the exported symbols. This file is known as a definition side deck. The binder writes one `IMPORT` control statement for each exported symbol. The file that contains `IMPORT` control statements indicates symbol names which may be imported and the name of the DLL from which they are imported.

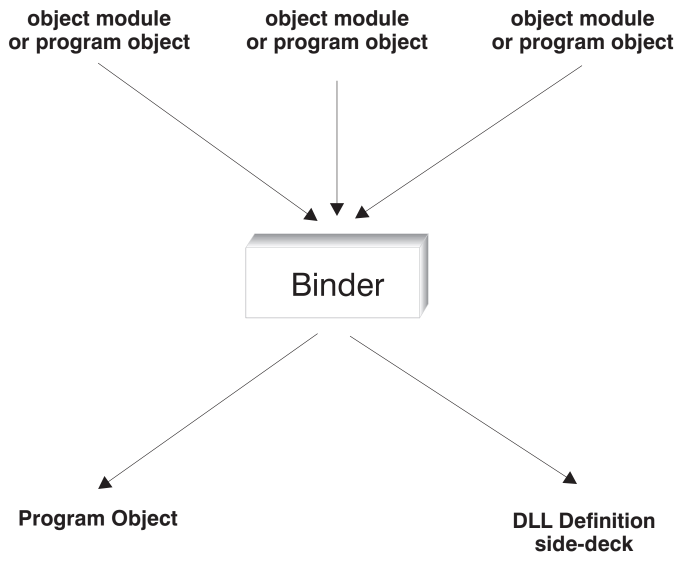

object module or program object
object module or program object
object module or program object
Binder
Program Object
DLL Definition side-deck

Figure 41. Build a DLL

You can use the method that is shown in Figure 42 to build an application that uses a DLL. To build a program which dynamically links symbols from a DLL during application run time, you must have C++ code, or C code that is compiled with the `DLL` option. This allows you to import symbols from a DLL. You must have an `IMPORT` control statement for each symbol that is to be imported from a DLL. The `IMPORT` control statement controls which DLL will be used to resolve an imported function or variable reference during execution. The bind step of the program that imports symbols from the DLL must include the definition side-deck of `IMPORT` control statements that the DLL's build generated.

The binder does not take an incremental approach to the resolution of DLL-linkage symbols. When binding or rebinding a program that uses a DLL, you must always specify the `DYNAM(DLL)` option, and must provide all `IMPORT` control statements. The binder does not retain these control statements for subsequent binds.

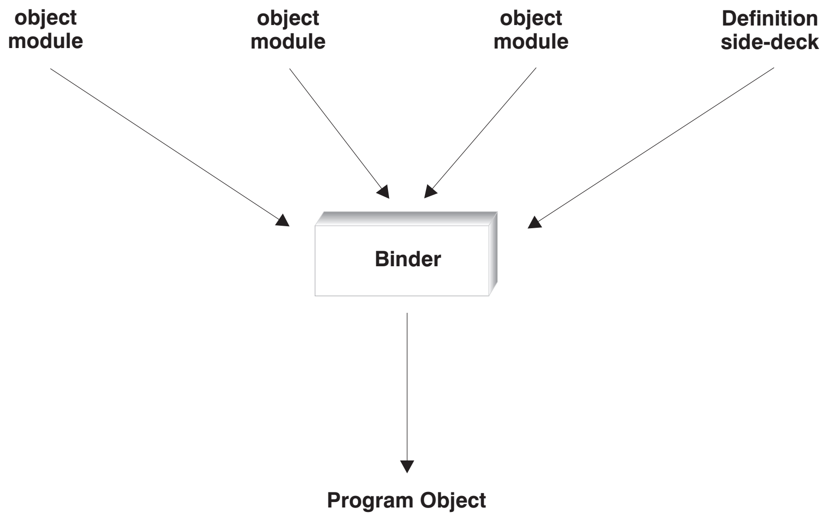


Figure 42. Build an application that uses a DLL

Rebind a Changed Compile Unit

You can use the method shown in Figure 43 to rebind an application after making changes to a single compile unit. Compile your changed source file and then rebind the resultant object module with the complete program object of your application. This will replace the binder sections that are associated with the changed compile unit in the program.

You can use this method to maintain your application. For example, you can change a source file and produce a corresponding object module. You can then ship the object module to your customer, who can bind the new object module with the application's complete program object. If you use this method, you have fewer files to maintain: just the application's program object and your source code.

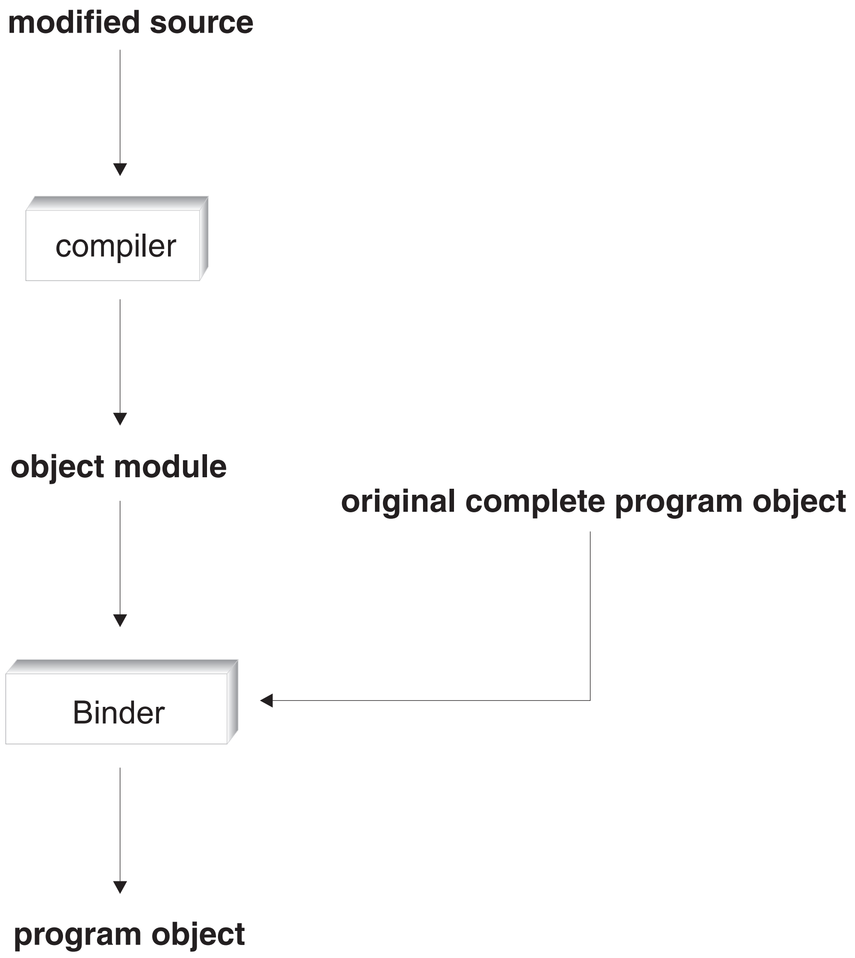


Figure 43. Rebinding a Changed Compile Unit

Binding Under z/OS UNIX

The `c89` and `c++` utilities are the interface to the compiler and the binder for z/OS UNIX System Services C/C++ applications. You can use `c89` and `c++`, to compile and bind a program in one step, or to bind application object modules after compilation.

Since OS/390 V2R4, the default, for the above utilities, is to invoke the binder alone, without first invoking the prelinker. That is, since the OS/390 V2R4 level of

OS/390 Language Environment and DFSMS 1.4, if the output file (`-o` executable) is not a PDS member, then the binder will be invoked. To modify your environment to run the prelinker, refer to the description of the **{_STEPS}** environment variable in “Environment Variables” on page 567.

Typically, you invoke the `c89` and `c++` utilities from the z/OS shell. For more information on these utilities, see “Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file” on page 555 or the *Z/OS UNIX System Services Command Reference*.

z/OS UNIX Example

The example source files `unit0.c`, `unit1.c`, and `unit2.c` that are shown in Figure 44, are used to illustrate all of the z/OS UNIX System Services examples that follow.

```
 /* file: unit0.c */
 #include <stdio.h>
 extern int f1(void);
 extern int f4(void);
 int main(void) {
   int rc1;
   int rc4;
   rc1 = f1();
   rc4 = f4();
   if (rc1 != 1) printf("fail rc1 is %d\n",rc1);
   if (rc4 != 40) printf("fail rc4 is %d\n",rc4);
   return 0;
 }

 /* file: unit1.c */
 int f1(void) { return 1; }

/* file: unit2.c */
int f2(void) { return 20;}
int f3(void) { return 30;}
int f4(void) { return f2()*2; /* 40 */ }
```

Figure 44. Example Source Files

Single Final Bind Using c89

Note that the following steps are similar for the `cc` and `c++` utilities. You must specify the static C++ IBM Open Class Library for the link-edit phase of `c++`. If you are statically linking the relevant class library object code, you must override the PLKED.SYSLIB concatenation to include the SCLBCPP data set, as follows:

```
c++ ... -l "//'cbc.sclbcpp'"
```

For more information, refer to “Prelinking and Linking Under z/OS Batch” on page 480.

Compile each source file, then perform a final single bind of everything as follows:

1. Compile each source file to generate the object modules `unit0.o`, `unit1.o`, and `unit2.o` as follows:
   ```
   c89 -c -W c,"CSECT(myprog)"  unit0.c
   c89 -c -W c,"CSECT(myprog)"  unit1.c
   c89 -c -W c,"CSECT(myprog)"  unit2.c
   ```
2. Perform a final single bind to produce the executable program myprog. Use the `c89` utility as follows:

```
c89 -o myprog unit0.o unit1.o unit2.o
```

The `-o` option of the `c89` command specifies the name of the output executable. The `c89` utility recognizes from the file extension `.o` that `unit0.o`, `unit1.o` and `unit2.o` are not to be compiled but are to be included in the bind step.

The following is an example of a makefile to perform a similar build:

```
PGM = myprog
SRCS = unit0.c unit1.c unit2.c
OBJS = $(SRCS:b:+".o")
COPTS = -W c,"CSECT(myprog)"
$(PGM) : ($OBJS)
        c89 -o $(PGM) $(OBJS)
%.o : %.c
        c89 -c -o $@ $(COPTS) $<
```

For more information on makefiles, see *z/OS UNIX System Services Programming Tools*.

Advantage

This method is simple, and is consistent with existing methods of building applications, such as makefiles.

Bind Each Compile Unit Using c89

Compile each source file and also bind it, then perform a final bind of all the partially bound units as follows:

1. Compile each source file to its object module (.tmp). Bind each object module into a partially bound program object (.o), which may have unresolved references. In this example, references to f1() and f4() in `unit0.o` are unresolved. When the partially bound programs are created, remove the object modules as they are no longer needed. Use `c89` to compile each source file, as follows:

   ```
   c89 -c -W c,"CSECT(myprog)" -o unit0.tmp unit0.c
   c89 -r -o unit0.o unit0.tmp
   rm unit0.tmp

   c89 -c -W c,"CSECT(myprog)" -o unit1.tmp unit1.c
   c89 -r -o unit1.o unit1.tmp
   rm unit1.tmp

   c89 -c -W c,"CSECT(myprog)" -o unit2.tmp unit2.c
   c89 -r -o unit2.o unit2.tmp
   rm unit2.tmp
   ```

   The -r option supports rebindability by disabling autocall processing.
2. Perform the final single bind to produce the executable program myprog by using `c89`:

   ```
   c89 -o myprog unit0.o unit1.o unit2.o
   ```

The following is an example of a makefile to perform a similar build:

```
_C89_EXTRA_ARGS=1
.EXPORT : _C89_EXTRA_ARGS       1
PGM = myprog                       2
SRCS = unit0.c unit1.c unit2.c       3
OBJS = $(SRCS:b:+".o")              4
COPTS = -W c,"CSECT(myprog)"
$(PGM) : $(OBJS)                     5
        c89 -o $(PGM) $(OBJS)
%.tmp : %.c      6
        c89 -c -o $@ $(COPTS) $<
%.o : %.tmp                          7
        c89 -r -o $@ $<
```

1	Export the environment variable _C89_EXTRA_ARGS so `c89` will process files with non-standard extensions. Otherwise `c89` will not recognize `unit0.tmp`, and the makefile will fail
2	name of executable
3	list of source files
4	list of partly bound parts
5	executable depends on parts
6	make `.tmp` file from `.c`
7	make `.o` from `.tmp`

In this example, `make` automatically removes the intermediate .tmp files after the makefile completes, since they are not marked as PRECIOUS. For more information on makefiles, see *z/OS UNIX System Services Programming Tools*.

Advantage

Binding a set of partially bound program objects into a fully bound program object is faster than binding object modules into a fully bound program object for NOGOFF objects. For example, a central build group can create the partially bound program objects. Developers can then use these program objects and their changed object modules to create a development program object.

Build and Use a DLL Using c89

Build `unit1.c` and `unit2.c` into DLL `onetwo`, which exports functions f1(), f2(), f3(), and f4(). Then build `unit0.c` into a program which dynamically links to functions f1() and f4() defined in the DLL.

1. Compile `unit1.c` and `unit2.c` to generate the object modules `unit1.o` and `unit2.o` which have functions to be exported. Use the `c89` utility as follows:
   ```
   c89 -c -W c,"EXPORTALL,CSECT(myprog)" unit1.c
   c89 -c -W c,"EXPORTALL,CSECT(myprog)" unit2.c
   ```
2. Bind `unit1.o` and `unit2.o` to generate the DLL onetwo:
   ```
   c89 -Wl,dll -o onetwo unit1.o unit2.o
   ```

   When you bind code with exported symbols, you should specify the DLL binder option (-W l,dll).

   In addition to the DLL onetwo being generated, the binder writes a list of `IMPORT` control statements to onetwo.x. This list is known as the definition side-deck. One `IMPORT` control statement is written for each exported symbol. These generated control statements will be included later as input to the bind step of an application that uses this DLL, so that it can import the symbols.

3. Compile `unit0.c` with the DLL option `-W c,DLL`, so that it can import unresolved symbols. Bind the object module, with the definition side-deck `onetwo.x` from the DLL build:

   ```
   c89 -c -W c,DLL unit0.c
   c89 -o dll12usr unit0.o onetwo.x
   ```

Advantage

The bind time advantage of using DLLs is that you only need to rebuild the DLL with the changed code in it. You do not need to rebuild all applications that use the DLL in order to use the changed code.

Rebind a Changed Compile Unit Using c89

Rebuild an application after making a change to a single source file. Recompile the single changed source file and make a replacement of its binder sections in the program.

1. Recompile the single changed source file. Use the compile time option `CSECT` to ensure that each section is named for purposes of rebindability. For example, assume that you have made a change to `unit1.c`. Recompile `unit1.c` by using `c89` as follows:

   ```
   c89 -o unit1.o -W c,"CSECT(myprog)" unit1.c
   ```

2. Rebind only the changed compile unit into the executable program, which replaces its corresponding binder sections in the program object:

   ```
   cp -m myprog myprog.old
   c89 -o myprog unit1.o myprog
   ```

   The `cp` command is optional. It saves a copy of the old executable in case the bind fails in such a way as to damage the executable. `myprog` is overwritten with the result of the bind of `unit1.o`. Like named sections in `unit1.o` replace those in the `myprog` executable.

The following is an example of a makefile to perform a similar build:

```
_C89_EXTRA_ARGS=1
.EXPORT : _C89_EXTRA_ARGS     1
SRCS = unit0.c unit1.c unit2.c      2
myprog.PRECIOUS : $(SRCS)   3
        @if [ -e $@ ]; then OLD=$@; else OLD=; fi;\
        CMD="$(CC) -Wc,csect $(CFLAGS) $(LDFLAGS) -o $@ $? $$OLD";\  4

        echo $$CMD; $$CMD;
        -@rm -f $(?:b+"$O")
```

1	allow non-conventional filenames
2	list of source files
3	do not delete `myprog` if the make fails
4	compile source files newer than the executable, and bind

The attribute .PRECIOUS ensures that such parts are not deleted if `make` fails. $? are the dependencies which are newer than the target.

Note:

- You need the .PRECIOUS attribute to avoid removing the current executable, since you depend on it as subsequent input.
- If more than one source part changes, and any compiles fail, then on subsequent makes, all compiles are redone.

For a complete description of all `c89` options see "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555. For a description of make, see *Z/OS UNIX System Services Command Reference* and for a make tutorial, see *z/OS UNIX System Services Programming Tools*.

Advantage

Rebinds are fast because most of the program is already bound, and none of the intermediate object modules are retained.

Binding with XPLINK Under z/OS UNIX

To bind your `XPLINK` module, specify –Wl,xplink on the c89/c++ command.

Binding under z/OS Batch

You can use the following procedures, which the z/OS C/C++ compiler supplies, to invoke the binder:

Procedure name	**Description**
CEEXL	C Bind an XPLINK Program
CEEXLR	C Bind and run an XPLINK program
EDCCB	C Compile and Bind steps
EDCCBG	C Compile, Bind, and Go steps
EDCXCB	C Compile and Bind an XPLINK Program
EDCXCBG	C Compile, Bind, and Go steps for an XPLINK Program
EDCXLDEF	Create C Source from a Locale, Compile, and Bind the XPLINK Program
CBCB	C++ Bind step
CBCBG	C++ Bind and Go steps
CBCCB	C++ Compile and Bind steps
CBCCBG	C++ Compile, Bind, and Go steps
CBCXB	C++ Bind an XPLINK Program
CBCXBG	C++ Bind and Go steps for an XPLINK Program
CBCXCB	C++ Compile and Bind an XPLINK Program
CBCXCBG	C++ Compile, Bind, and Go steps for an XPLINK Program

If you want to generate DLL code, you must use the binder `DYNAM(DLL)` option. All the z/OS C/C++ supplied cataloged procedures that invoke the binder use the `DYNAM(DLL)` option. For C++, these cataloged procedures use the DLL versions of the IBM-supplied class libraries by default; the IBM-supplied definition side-deck data set for class libraries, SCLBSID, is included in the `SYSLIN` concatenation.

z/OS Batch Example

Figure 45 on page 364 shows the example source files `USERID.PLAN9.C(UNIT0)`, `USERID.PLAN9.C(UNIT1)`, and `USERID.PLAN9.C(UNIT2)`, which are used to illustrate all of the z/OS batch examples that follow.

```
/* file: USERID.PLAN9.C(UNIT0) */
#include <stdio.h>
extern int f1(void);
extern int f4(void);
int main(void) {
int rc1;
int rc4;
rc1 = f1();
rc4 = f4();
if (rc1 != 1) printf("fail rc1 is %d\n",rc1);
if (rc4 != 40) printf("fail rc4 is %d\n",rc4);
return 0;
}

/* file: USERID.PLAN9.C(UNIT1) */
int f1(void) { return 1; }

/* file: USERID.PLAN9.C(UNIT2) */
int f2(void) { return 20;}
int f3(void) { return 30;}
int f4(void) { return f2()*2; /* 40 */ }
```

Figure 45. Example Source Files

Single Final Bind under z/OS Batch

Compile each source file, then perform a final single bind of everything as follows:

1. Compile each source file to generate the object modules `USERID.PLAN9.OBJ(UNIT0)`, `USERID.PLAN9.OBJ(UNIT1)`, and `USERID.PLAN9.OBJ(UNIT2)`. Use the `EDCC` procedure as follows:

```
//COMP0  EXEC EDCC,
//       INFILE='USERID.PLAN9.C(UNIT0)',
//       OUTFILE='USERID.PLAN9.OBJ,DISP=SHR',
//       CPARM='LONG,RENT'
//COMP1  EXEC EDCC,
//       INFILE='USERID.PLAN9.C(UNIT1)',
//       OUTFILE='USERID.PLAN9.OBJ,DISP=SHR',
//       CPARM='LONG,RENT'
//COMP2  EXEC EDCC,
//       INFILE='USERID.PLAN9.C(UNIT2)',
//       OUTFILE='USERID.PLAN9.OBJ,DISP=SHR',
//       CPARM='LONG,RENT'
```

2. Perform a final single bind to produce the executable program `USERID.PLAN9.LOADE(MYPROG)`. Use the `CBCB` procedure as follows:

```
//BIND EXEC CBCB,OUTFILE='USERID.PLAN9.LOADE,DISP=SHR'
//OBJECT DD DSN=USERID.PLAN9.OBJ,DISP=SHR
//SYSIN DD *
  INCLUDE OBJECT(UNIT0)
  INCLUDE OBJECT(UNIT1)
  INCLUDE OBJECT(UNIT2)
  NAME MYPROG(R)
/*
```

   The `OUTFILE` parameter along with the `NAME` control statement specify the name of the output executable to be created.

Advantage
This method is simple, and is consistent with existing methods of building applications, such as makefiles.

Bind Each Compile Unit under z/OS Batch

Compile each source file and also bind it, then perform a final bind of all the partially bound units as follows:

1. Compile and bind each source file to generate the partially bound program objects `USERID.PLAN9.LOADE(UNIT0)`, `USERID.PLAN9.LOADE(UNIT1)`, and `USERID.PLAN9.LOADE(UNIT2)`, which may have unresolved references. In this example, references to f1() and f4() in `USERID.PLAN9.LOADE(UNIT0)` are unresolved. Compile and bind each unit by using the `EDCCB` procedure as follows:

```
//COMP0  EXEC EDCCB,
//       CPARM='CSECT(MYPROG)',
//       BPARM='LET,CALL(NO),ALIASES(ALL)',
//       INFILE='USERID.PLAN9.C(UNIT0)',
//       OUTFILE='USERID.PLAN9.LOADE(UNIT0),DISP=SHR'
//COMP1  EXEC EDCCB,
//       CPARM='CSECT(MYPROG)',
//       BPARM='LET,CALL(NO),ALIASES(ALL)',
//       INFILE='USERID.PLAN9.C(UNIT1)',
//       OUTFILE='USERID.PLAN9.LOADE(UNIT1),DISP=SHR'
//COMP2  EXEC EDCCB,
//       CPARM='CSECT(MYPROG)',
//       BPARM='LET,CALL(NO),ALIASES(ALL)',
//       INFILE='USERID.PLAN9.C(UNIT2)',
//       OUTFILE='USERID.PLAN9.LOADE(UNIT2),DISP=SHR'
```

   The `CALL(NO)` option prevents autocall processing.

2. Perform the final single bind to produce the executable program `MYPROG` by using the `CBCB` procedure:

   You have two methods for building the program.

   a. Explicit include: In this method, when you invoke the `CBCB` procedure, you use include cards to explicitly specify all the program objects that make up this executable. Automatic library call is done only for CEE.SCEELKED and CEE.SCEELKEX, because those are the only libraries pointed to by `DD SYSLIB`. `DD SYSLIB` does not point to any of your user code. For example:

```
//BIND   EXEC CBCB,
//         OUTFILE='USERID.PLAN9.LOADE,DISP=SHR'
//INPGM  DD DSN=USERID.PLAN9.LOADE,DISP=SHR
//SYSIN DD *
   INCLUDE INPGM(UNIT0)
   INCLUDE INPGM(UNIT1)
   INCLUDE INPGM(UNIT2)
   NAME MYPROG(R)
/*
```

   b. Library search: In this method, you specify the compile unit that contains your main() function, and allocate your object library to `DDname SYSLIB`. The binder performs a library search and includes additional members from your object library, and generates the output program object. You invoke the binder as follows:

```
//BIND   EXEC CBCB,
//       OUTFILE='USERID.PLAN9.LOADE,DISP=SHR'
//INPGM  DD DSN=USERID.PLAN9.LOADE,DISP=SHR
//SYSLIB DD
//         DD
//         DD
//         DD DSN=USERID.PLAN9.LOADE,DISP=SHR
//SYSIN DD *
   INCLUDE INPGM(UNIT0)
   NAME MYPROG(R)
/*
```

Advantage

Binding a set of partially bound program objects into a fully bound program object is faster than binding object modules into a fully bound program object. For example, a central build group can create the partially bound program objects. Developers can then use these program objects and their changed object modules to create a development program object.

Build and Use a DLL under z/OS Batch

Build `USERID.PLAN9.C(UNIT1)` and `USERID.PLAN9.C(UNIT2)` into DLL `USERID.PLAN.LOADE(ONETWO)`, which exports functions f1(), f2(), f3() and f4(). Build `USERID.PLAN9.C(UNIT0)` into a program which dynamically links to functions f1() and f4() defined in the DLL.

1. Compile `USERID.PLAN9.C(UNIT1)` and `USERID.PLAN9.C(UNIT2)` to generate the object modules `USERID.PLAN9.OBJ(UNIT1)` and `USERID.PLAN9.OBJ(UNIT2)`, which define the functions to be exported. Use the `EDCC` procedure as follows:

```
//* Compile UNIT1
//CC1 EXEC EDCC,
//         CPARM='OPTF(DD:OPTIONS)',
//         INFILE='USERID.PLAN9.C(UNIT1)',
//         OUTFILE='USERID.PLAN9.OBJ(UNIT1),DISP=SHR'
//COMPILE.OPTIONS DD *
   LIST RENT LONGNAME EXPORTALL
*/
//* Compile UNIT2
//CC2 EXEC EDCC,
//         CPARM='OPTF(DD:OPTIONS)',
//          INFILE='USERID.PLAN9.C(UNIT2)',
//          OUTFILE='USERID.PLAN9.OBJ(UNIT2),DISP=SHR'
//COMPILE.OPTIONS DD *
   LIST RENT LONGNAME EXPORTALL
*/
```

2. Bind `USERID.PLAN9.OBJ(UNIT1)` and `USERID.PLAN9.OBJ(UNIT2)` to generate the DLL `ONETWO`:

```
//* Bind the DLL
//BIND1    EXEC CBCB,
//           BPARM='CALL,DYNAM(DLL)',
//           OUTFILE='USERID.PLAN9.LOADE(ONETWO),DISP=SHR'
//INOBJ    DD DISP=SHR,DSN=USERID.PLAN9.OBJ
//SYSDEFSD DD DISP=SHR,DSN=USERID.PLAN9.IMP(ONETWO)
//SYSLIN   DD *
   INCLUDE INOBJ(UNIT1)
   INCLUDE INOBJ(UNIT2)
   NAME ONETWO(R)
/*
```

When you bind code with exported symbols, you must specify the binder option DYNAM(DLL). You must also allocate the definition side-deck DD SYSDEFSD to define the definition side-deck where the IMPORT control statements are to be written.

In addition to the DLL being generated, a list of IMPORT control statements is written to DD SYSDEFSD. One IMPORT control statement is written for each exported symbol. These generated control statements will be included later as input to the bind step of an application that uses this DLL, so that it can import the symbols.

3. Compile USERID.PLAN9.C(UNIT0) so that it may import unresolved symbols, and bind with the file of IMPORT control statements from the DLL's build:

```
//* Compile the DLL user
//CC1 EXEC EDCC,
//         CPARM='OPTF(DD:OPTIONS)',
//         INFILE='USERID.PLAN9.C(UNIT0)',
//         OUTFILE='USERID.PLAN9.OBJ(UNIT0),DISP=SHR'
//COMPILE.OPTIONS DD *
   LIST RENT LONGNAME DLL
/*
//* Bind the DLL user with input IMPORT statements from the DLL build
//BIND1    EXEC CBCB,
//         BPARM='CALL,DYNAM(DLL)',
//         OUTFILE='USERID.PLAN9.LOADE,DISP=SHR'
//INOBJ    DD DISP=SHR,DSN=USERID.PLAN9.OBJ
//IMP      DD DISP=SHR,DSN=USERID.PLAN9.IMP
//SYSLIN   DD *
  INCLUDE INOBJ(UNIT0)
  INCLUDE IMP(ONETWO)
  ENTRY CEESTART
  NAME DLL12USR(R)
/*
```

Advantage

The bind time advantage of using DLLs is that you only need to rebuild the DLL with the changed code in it. You do not need to rebuild all applications that use the DLL in order to use the changed code.

Rebind a Changed Compile Unit under z/OS Batch

Rebuild an application after making a change to a single source file. Recompile the single changed source file and make a replacement of its binder sections in the program.

1. Recompile the single changed source file. Use the compile time option CSECT to ensure that each section is named for purposes of rebindability. For example, assume that you have made a change to USERID.PLAN9.C(UNIT1). Recompile the source file using the EDCC procedure as follows:

```
//* Compile UNIT1 user
//CC EXEC EDCC,
//             CPARM='OPTF(DD:OPTIONS)',
//             INFILE='USERID.PLAN9.C(UNIT1)',
//             OUTFILE='USERID.PLAN9.OBJ(UNIT1),DISP=SHR'
//COMPILE.OPTIONS DD *
   LIST RENT LONGNAME DLL CSECT(MYPROG)
/*
```

2. Rebind only the changed compile unit into the executable program, which replaces its corresponding binder sections in the program object:

```
//BIND   EXEC CBCB,
//       OUTFILE='USERID.PLAN9.LOADE,DISP=SHR'
//OLDPGM  DD DSN=USERID.PLAN9.LOADE,DISP=SHR
//NEWOBJ  DD DSN=USERID.PLAN9.OBJ,DISP=SHR
//SYSIN DD *
   INCLUDE NEWOBJ(UNIT1)
   INCLUDE OLDPGM(MYPROG)
   NAME NEWPGM(R)
/*
```

Advantage

Rebinds are fast because most of the program is already bound, and none of the intermediate object modules are retained.

Writing JCL for the binder

You can use cataloged procedures rather than supply all the JCL required for a job step. However, you can use JCL statements to override the statements of the cataloged procedure.

Use the EXEC statement in your JCL to invoke the binder. The EXEC statement to invoke the binder is:

```
      //BIND EXEC PGM=IEWL
```

Use the EXEC statement's PARM parameter to select one or more of the optional facilities that the binder provides. For example, you can specify the OPTIONS option on the PARM parameter to read binder options from the DD name OPTS, as follows:

```
//BIND1 EXEC PGM=IEWL,PARM='OPTIONS=OPTS'
//OPTS     DD *
     AMODE=31,MAP
     RENT,DYNAM=DLL
     CASE=MIXED,COMPAT=CURR
/*
//SYSLIB   DD DISP=SHR,DSN=CEE.SCEELKEX
//         DD DISP=SHR,DSN=CEE.SCEELKED
//         DD DISP=SHR,DSN=CEE.SCEECPP
//SYSLIN   DD DISP=SHR,DSN=USERID.PLAN9.OBJ(P1)
//         DD DISP=SHR,DSN=CBC.SCLBSID(IOSTREAM)
//SYSLMOD  DD DISP=SHR,DSN=USERID.PLAN9.LOADE(PROG1)
//SYSPRINT DD SYSOUT=*
```

In the example above, object module P1 is bound using the IOSTREAM DLL definition sidedeck. The Language Environment runtime libraries SCEELKED, SCEELKEX, and SCEECPP are statically bound to produce the program object PROG1.

The binder always requires three standard data sets. You must define these data sets on DD statements with the DDnames SYSLIN, SYSLMOD, and SYSPRINT.

A typical sequence of job control statements for binding an object module into a program object is shown below. The binder control statement NAME puts the program object into the PDSE USER.LOADE with the member name PROGRAM1.

```
//BIND     EXEC  PGM=IEWL,PARM='MAP'
//SYSPRINT DD *                          << out: binder listing
//SYSDEFSD DD DUMMY                      << out: generated IMPORTs
//SYSLMOD  DD DISP=SHR,DSN=USERID.PLAN9.LOADE << out: PDSE of executables
//SYSLIB   DD DISP=SHR,DSN=CEE.SCEELKED   << in:  autocall libraries to search
//         DD DISP=SHR,DSN=CEE.SCEELKEX
//         DD DISP=SHR,DSN=CEE.SCEECPP
//INOBJ    DD DISP=SHR,DSN=USERID.PLAN.OBJ       << in: compiler object code
//SYSLIN DD*
   INCLUDE INOBJ(UNIT0)
   INCLUDE INOBJ(UNIT1)
   INCLUDE INOBJ(UNIT2)
   ENTRY CEESTART
   NAME  PROGRAM1(R)
  /*
```

You can explicitly include members from a data set like `USERID.PLAN.OBJ`, as is done above. If you want to be more flexible and less explicit, include only one member, typically the one that contains the entry point (e.g. `main()`). Then you can add `USERID.PLAN.OBJ` to the `SYSLIB` concatenation so that a library search brings in the remaining members.

Binding Under TSO Using CXXBIND

This section describes how to bind your z/OS C++ or z/OS C program in TSO by invoking the `CXXBIND` REXX EXEC. This REXX EXEC invokes the binder and creates an executable program object.

If you specify a data set name in an option, and the high-level qualifier of the data set is not the same as your user prefix, you must use the fully qualified name of the data set and place single quotation marks around the entire name.

If you specify an HFS filename in an option, it must be an absolute filename: that is, it must begin with a slash (/). You can include commas and special characters in filenames, but you must enclose filenames that contain special characters or commas in single quotes. If a single quote is part of the filename, you must specify the quote twice.

The syntax for the `CXXBIND EXEC` is:

```
                        ┌─,──────────────────┐
                        ▼                    │
►►──CXXBIND──OBJ──(──────┬─input-object───┬──┴──)──────────────────────────►
                         └─'input-object'─┘

►──┬─────────────────────────────────────────┬──────────────────────────────►
   │              ┌─,────────────────┐       │
   │              ▼                  │       │
   └─OPT──(───────binder_option;─────┴──)────┘
```

```
►─┬──────────────────────────────────────────────┬──────────────────────────────►
  │            ┌─,─────────────────────────┐      │
  │            ▼                           │      │
  └─LIB──(──┬─search-library-name───┬─┴──)─┘
            └─'search-library-name'─┘

►─┬────────────────────────────────────────────┬────────────────────────────────►
  └─LOAD──(──┬─output_program_object───┬──)──┘
             └─'output_program_object'─┘

►─┬─────────────────────────────────────────────────┬───────────────────────────►
  └─IMP──(──┬─file_of_generated_imports───┬──)──┘
            └─'file_of_generated_imports'─┘

►─┬────────────────────────────────────┬──┬──────────┬──────────────────────────►◄
  └─LIST──(──┬─output_listing───┬──)──┘  └─XPLINK───┘
             └─'output_listing'─┘
```

OBJ
: You must **always** specify the input file names by using the OBJ keyword parameter. Each input file must be one of the following:
- An object module that can be a PDS member, a sequential data set, or an HFS file
- A load module that is a PDS member
- A program object that can be a PDSE member or an HFS file
- A text file that contains binder statements. The file can be a PDS member, a sequential data set, or an HFS file

OPT
: Use the OPT keyword parameter to specify binder options. For example, if you want the binder to use the MAP option, specify the following:

```
CXXBIND OBJ(PLAN9.OBJ(PROG3)) OPT('MAP')...
```

LIB
: Use the LIB keyword parameter to specify the PDS and PDSE libraries that the binder should search to resolve unresolved external references during a library search of the DD SYSLIB.

The default libraries that are used when the XPLINK option is not specified are the C/C++ libraries CEE.SCEELKED, CEE.SCEELKEX, CEE.SCEECPP and the C++ class library CBC.SCLBSID. The default libraries that are used when the XPLINK option is specified are the C/C++ libraries CEE.SCEEBIND, CEE.SCEELIB and the C++ class library CBC.SCLBSID. The default library names are added to the DD SYSLIB concatenation if library names are specified with the LIB keyword parameter.

LOAD
: Use the LOAD keyword parameter to specify where the resultant executable program object (which must be a PDSE member, or an HFS file) should be stored.

IMP
: Use the IMP keyword parameter to specify where the generated IMPORT control statements should be written.

LIST
: Use the LIST keyword parameter to specify where the binder listing should be written. If you specify *, the binder directs the listing to your console.

`XPLINK`	Use the `XPLINK` keyword parameter when you are building an XPLINK executable program object. Specifying `XPLINK` will change the default libraries as described under the LIB option.

TSO Example

Figure 46 shows the example source files `PLAN9.C(UNIT0)`, `PLAN9.C(UNIT1)`, and `PLAN9.C(UNIT2)`, that are used to illustrate all of the TSO examples that follow.

```
/* file: USERID.PLAN9.C(UNIT0) */
#include <stdio.h>
extern int f1(void);
extern int f4(void);
int main(void) {
int rc1;
int rc4;
rc1 = f1();
rc4 = f4();
if (rc1 != 1) printf("fail rc1 is %d\n",rc1);
if (rc4 != 40) printf("fail rc4 is %d\n",rc4);
return 0;
}

/* file: USERID.PLAN9.C(UNIT1) */
int f1(void) { return 1; }

/* file: USERID.PLAN9.C(UNIT2) */
int f2(void) { return 20;}
int f3(void) { return 30;}
int f4(void) { return f2()*2; /* 40 */ }
```

Figure 46. Example Source Files

Single Final Bind Under TSO

Compile each source file, then perform a final single bind of everything as follows:

1. Compile each unit to generate the object modules `PLAN9.OBJ(UNIT0)`, `PLAN9.OBJ(UNIT1)`, and `PLAN9.OBJ(UNIT2)`. Use the `CC` REXX exec as follows:
   ```
   CC PLAN9.C(UNIT0) OBJECT(PLAN9.OBJ) CSECT(MYPROG)
   CC PLAN9.C(UNIT1) OBJECT(PLAN9.OBJ) CSECT(MYPROG)
   CC PLAN9.C(UNIT2) OBJECT(PLAN9.OBJ) CSECT(MYPROG)
   ```
2. Perform a final single bind to produce the executable program `PLAN9.LOADE(MYPROG)`. Use the `CXXBIND` REXX exec as follows:
   ```
   CXXBIND OBJ(PLAN9.OBJ(UNIT0),PLAN9.OBJ(UNIT1),PLAN9.OBJ(UNIT2))
      LOAD(PLAN9.LOADE(MYPROG))
   ```

Advantage

This method is simple, and is consistent with existing methods of building applications, such as makefiles.

Bind Each Compile Unit Under TSO

Compile and bind each source file, then perform a final bind of all the partially bound units as follows:

1. Compile and bind each source file to generate the partially bound program objects `PLAN9.LOADE(UNIT0)`, `PLAN9.LOADE(UNIT1)`, and `PLAN9.LOADE(UNIT2)`, which may have unresolved references. In this example, references to f1() and f4() in `PLAN9.LOADE(UNIT0)` are unresolved. Compile and bind each unit by using the CC and `CXXBIND` REXX execs as follows:

```
CC PLAN9.C(UNIT0) OBJECT(PLAN9.OBJ) CSECT(MYPROG)
CXXBIND OBJ(PLAN9.OBJ(UNIT0)) OPT('LET,CALL(NO)')
   LOAD(PLAN9.LOADE(UNIT0))

CC PLAN9.C(UNIT1) OBJECT(PLAN9.OBJ) CSECT(MYPROG)
CXXBIND OBJ(PLAN9.OBJ(UNIT1)) OPT('LET,CALL(NO)')
   LOAD(PLAN9.LOADE(UNIT1))

CC PLAN9.C(UNIT2) OBJECT(PLAN9.OBJ) CSECT(MYPROG)
CXXBIND OBJ(PLAN9.OBJ(UNIT2)) OPT('LET,CALL(NO)')
   LOAD(PLAN9.LOADE(UNIT1))
```

The `CALL(NO)` option prevents autocall processing.

2. Perform the final single bind to produce the executable program `MYPROG` by using the `CXXBIND` REXX exec:

   ```
   CXXBIND OBJ(PLAN9.LOADE(UNIT0), PLAN9.LOADE(UNIT1), PLAN9.LOADE(UNIT2))
      LOAD(PLAN9.LOADE(MYPROG))
   ```

Advantage

Binding a set of partially bound program objects into a fully bound program object is faster than binding object modules into a fully bound program object. For example, a central build group can create the partially bound program objects. Developers can then use these program objects and their changed object modules to create a development program object.

Build and Use a DLL under TSO

Build `PLAN9.C(UNIT1)` and `PLAN9.C(UNIT2)` into DLL `PLAN9.LOADE(ONETWO)` which exports functions f1(), f2(), f3() and f4(). Then build `PLAN9.C(UNIT0)` into a program which dynamically links to functions f1() and f4() defined in the DLL.

1. Compile `PLAN9.C(UNIT1)` and `PLAN9.C(UNIT2)` to generate the object modules `PLAN9.OBJ(UNIT1)` and `PLAN9.OBJ(UNIT2)` which have functions to be exported. Use the `CC` REXX exec as follows:

   ```
   CC PLAN9.C(UNIT1) OBJECT(PLAN9.OBJ) EXPORTALL,LONGNAME,DLL,CSECT(MYPROG)
   CC PLAN9.C(UNIT2) OBJECT(PLAN9.OBJ) EXPORTALL,LONGNAME,DLL,CSECT(MYPROG)
   ```

2. Bind `PLAN9.OBJ(UNIT1)` and `PLAN9.OBJ(UNIT2)` to generate the DLL `PLAN9.LOADE(ONETWO)`:

   ```
   CXXBIND OBJ(PLAN9.LOADE(UNIT0), PLAN9.LOADE(UNIT1)) IMP (PLAN9.IMP(ONETWO))
         LOAD(PLAN9.LOADE(ONETWO))
   ```

   When you bind code with exported symbols, you must specify the binder option `DYNAM(DLL)`. You must also use the `CXXBIND IMP` option to define the definition side-deck where the `IMPORT` control statements are to be written.

3. Compile `PLAN9.C(UNIT0)` so that it may import unresolved symbols, and bind with `PLAN9.IMP(ONETWO)`, which is the definition side-deck containing `IMPORT` control statements from the DLL's build:

   ```
   CC PLAN9.C(UNIT0) OBJECT(PLAN9.OBJ) CSECT(MYPROG),DLL
   CXXBIND OBJ(PLAN9.LOADE(UNIT0), PLAN9.IMP(ONETWO)) LOAD(PLAN9.LOADE(DLL12USR))
   ```

Advantage

The bind time advantage of using DLLs is that you only need to rebuild the DLL with the changed code in it. You do not need to rebuild all applications that use the DLL in order to use the changed code.

Rebind a Changed Compile Unit Under TSO

Rebuild an application after making a change to a single source file. Recompile the single changed source file and make a replacement of its binder sections in the program.

1. Recompile the single changed source file. Use the compile time option `CSECT` to ensure that each section is named for purposes of rebindability. For example, assume that you have made a change to `PLAN9.C(UNIT1)`. Recompile `PLAN9.C(UNIT1)` by using the `CC` REXX exec as follows:

   ```
   CC PLAN9.C(UNIT1) OBJECT(PLAN9.OBJ) CSECT(MYPROG)
   ```

2. Rebind only the changed source file into the executable program, which replaces its corresponding binder sections in the program object:

   ```
   CXXBIND OBJ(PLAN9.OBJ(UNIT1), PLAN9.LOADE(MYPROG))
        LOAD(PLAN9.LOADE(NEWPROG)
   ```

Advantage

Rebinds are fast because most of the program is already bound, and none of the intermediate object modules are retained.

Chapter 13. Binder Processing

You can bind any z/OS C/C++ object module or program object.

Object files with longname symbols, reentrant writable static symbols, and DLL-style function calls require additional processing to build global data for the application. You can always rebind if you don't require this additional processing. You can also re-bind if you used the binder for this additional processing and produced a program object (in other words, you didn't use the prelinker). If you used the prelinker and performed this additional processing, you cannot later rebind. If you have done additional processing and output it to a PDS, you cannot rebind it. For further information, refer to "Chapter 3. About Prelinking, Linking, and Binding" on page 31.

Various limits have been increased from the linkage-editor. For example, the binder supports variable and function names up to 1024 characters long.

For the Writable Static Area (WSA), the binder assigns relative offsets to objects in the Writable Static Area and manages initialization information for objects in the Writable Static Area. The Writable Static Area is not loaded with the code. Language Environment runtime requests it.

For C++, the binder collects constructor calls and destructor calls for static C++ objects across multiple compile units. C++ linkage names appear with the full signature in the binder listing. A cross reference of mangled versus demangled names is also provided.

For DLLs, the binder collects static DLL initialization information across multiple compile units. It then generates a function descriptor in the Writable Static Area for each DLL-referenced function, and generates a variable descriptor for each DLL-referenced variable. It accepts `IMPORT` control statements in its input to resolve dynamically linked symbols, and generates an `IMPORT` control statement for each exported function and variable.

z/OS UNIX System Services HFS support allows library search of archive libraries that were created with the `ar` utility. HFS files can be specified on binder control statements.

C/C++ code is rebindable, provided all the sections are named. You can use the `CSECT` compiler option or the `#pragma csect` directive to name a section. If the `GOFF` option is active, then your CSECTs will automatically be named. See "CSECT | NOCSECT" on page 92.

Note: If you do not name all the sections and you try to rebind, the binder cannot replace private or unnamed sections. The result is a permanent accumulation of dead code and of duplicate functions.

The `RENAME` control statement may rename specified unresolved function references to a definition of a different name. This is especially helpful when matching function names that should be case insensitive. The `RENAME` statement does not apply to rebinds. If you rebind updated code with the original name, you will need another `RENAME` control statement to make references match their definitions.

The binder starts its processing by reading object code from primary input (`DD SYSLIN`). It accepts the following inputs:

- Object modules (compiler output from C/C++ and other languages)
- Load modules (previously link-edited by the Linkage-Editor)
- Program Objects (previously bound by the binder)
- Binder control statements
- Generalized Object File Format (GOFF) files

During the processing of primary input, control statements can control the binder's processing. For example, the `INCLUDE` control statement will cause the binder to read and include other code.

Among other processing, the binder records whether or not symbols (external functions and variables) are currently defined. During the processing of primary input, the `AUTOCALL` control statement causes a library to be immediately searched for members that contain an unresolved symbol's definition. If such a member is found, the binder reads it as autocall input before it processes more primary or secondary input.

After the binder processes primary input, it searches the libraries that are included in `DD SYSLIB` for definitions of unresolved symbols, unless you specified the options `NOCALL` or `NORES`. This is final autocall processing. The binder may read library members that contain the sought definition as autocall input.

Final autocall processing drives `DD SYSLIB` autocall resolution one or two times. After the first `DD SYSLIB` autocall resolution is complete, symbols that are still unresolved are subject to renaming. If renaming is done, `DD SYSLIB` autocall is driven a second time to resolve the renamed symbols.

After the binder completes final autocall (if autocall takes place), it processes the `IMPORT` control statements that were read in to match unresolved DLL type references. It then marks those symbols as being resolved from DLLs.

Finally, the binder generates an output program object. It stores the program object in an HFS file, or as a member of the program library (PDSE) specified on the `DD SYSLMOD` statement. The Program Management Loader can load this program object into virtual storage to be run. The binder can generate a listing. It can also generate a file of `IMPORT` control statements for symbols exported from the program that are to be used to build other applications that use this DLL.

Linkage Considerations

The binder will check that a statically bound symbol reference and symbol definition have compatible attributes. If a mismatch is detected, the binder will issue a diagnostic message. This attribute information is contained within the binder input files, such as object files, program objects, and load modules.

For C and C++, the default attribute is based on the `XPLINK` and `NOXPLINK` options. Individual symbols can have a different attribute than the default by using the #pragma `OS_UPSTACK`, #pragma `OS_DOWNSTACK`, and #pragma `OS_NOSTACK`.

The attributes can also be set for assembly language. Refer to the *HLASM Language Reference*, SC26-4940 for further information.

Primary Input Processing

The binder obtains its primary input from the contents of the data sets that are defined by the `DD SYSLIN`.

Primary input to the binder can be a sequential data set, a member of a partitioned data set, or an instream data set. The primary input must consist of one or more separately compiled program objects, object modules, load modules or binder control statements.

C or C++ Object Module as Input

The binder accepts object modules generated by the C or C++ compiler (as well as other compilers or assemblers) as input. All initialization information and relocation information for both code and the Writable Static Area is retained, which makes each compile unit fully rebindable.

Secondary Input Processing

Secondary input to the binder consists of files that are not part of primary input but are included as input due to the `INCLUDE` control statement.

The binder obtains its secondary input by reading the members from libraries of object modules (which may contain control statements), load modules, or program objects.

Load Module as Input

The binder accepts a load module that was generated by the Linkage-Editor input, and converts it into program object format on output.

Note: Object modules that define or refer to writable static objects that were processed by the prelinker and link-edited into a load module do not contain relocation information. You cannot rebind these compile units, or use them as input to the IPA Link step. See “Code That Has Been Prelinked” on page 399 for more information on prelinked code and the binder.

Program Object as input

The binder accepts previously bound program objects as input. This means that you can recompile only a changed compile unit, and rebind it into a program without needing other unchanged compile units. See “Rebind a Changed Compile Unit” on page 358 and “Rebindability” on page 392.

You can compile and bind each compile unit to a program object, possibly with unresolved references. To build the full application, you can then bind all the separate program objects into a single executable program object.

Autocall Input Processing (Library Search)

The library search process is also known as automatic library call, or autocall for short. Unresolved symbols, including unresolved DLL-type references, may have their definitions within a library member that is searched during library search processing.

The library member that is expected to contain the definition is read. This may resolve the expected symbol, and also other symbols which that library member may define. Reading in the library member may also introduce new unresolved symbols.

Incremental Autocall Processing (AUTOCALL Control Statement)

Traditionally, autocall has been considered part of the final bind process. However, through the use of the `AUTOCALL` control statement, you can invoke autocall at any time during the include process.

The binder searches the libraries that occur on `AUTOCALL` control statements immediately for unresolved symbols and DLL references, before it processes more primary or secondary input. See “AUTOCALL Control Statement” on page 275. After processing the `AUTOCALL` statement, if new unresolved symbols are found that cannot be resolved from within the library being processed, the library will not be searched again. To search the library again, another `AUTOCALL` statement or `SYSLIB` must indicate the same library.

Final Autocall Processing (SYSLIB)

The binder performs final autocall processing of `DD SYSLIB` in addition to incremental autocall. It performs this processing after it completes the processing of `DD SYSLIN`.

`DD SYSLIB` defines the libraries of object modules, load modules, or program objects that the binder will search after it processes primary and secondary input.

The binder searches each library (PDS or PDSE) in the `DD SYSLIB` concatenation in order. The rules for searching for a symbol definition in a PDS or PDSE are as follows:
- If the library contains a C370LIB directory (@@DC370$) that was created using the C/C++ Object Library Utility, and the directory points to a member containing the symbol’s definition, that member is read.
- If the library has a member or alias with the same name as the symbol that is being searched, that member of the library is read.

You can use the LIBRARY control statement to suppress the search of `SYSLIB` for certain symbols, or to search an alternate library.

Non-XPLINK LIBRARIES

The libraries described here are to be used only for binding non-XPLINK program modules.

For C and C++, you should include CEE.SCEELKEX and CEE.SCEELKED in your `DD SYSLIB` concatenation when binding your program. Those libraries contain the Language Environment resident routines, which include those for callable services, initialization, and termination. CEE.SCEELKED has the uppercase (`NOLONGNAME`), 8-byte-or-less versions of the standard C library routines. ’PRINTF’.CEE.SCEELKEX has the equivalent case-sensitive longnamed routines; for example ’printf’, ’pthread_create’.

For C++, you should also include the C++ base library in data set CEE.SCEECPP in your `DD SYSLIB` concatenation when binding your program. It contains the C++ base routines such as global operator new.

XPLINK LIBRARIES

The libraries described here are to be used only for binding XPLINK program modules.

For C and C++, you must include CEE.SCEEBIND in your `DD SYSLIB` concatenation when binding your program. This library contains the Language Environment resident routines, which include those for initialization and termination.

XPLINK C run-time and C++ base libraries are packaged as DLLs. Therefore, the bindings for those routines resolve dynamically. This is accomplished by providing definition side-decks (object modules containing IMPORT control statements). This is done using INCLUDE control statements in the Binder primary or secondary input. Language Environment side-decks reside in the CEE.SCEELIB dataset.

The Language Environment routine definitions for callable services are contained in the CELHS001 member of the data set CEE.SCEELIB. For example, CEEGTST is contained here.

The C run-time library routine definitions are contained in the CELHS003 member of the data set CEE.SCEELIB, which contains NOLONGNAME and case-sensitive long-named routines (for example, 'printf', 'PRINTF', and 'pthread_create' are contained here). It also contains the C run-time library global variables; for example 'environ'.

For C++, you should also include the C++ base library side-deck (member CELHSCPP in data set CEE.SCEELIB). It contains the C++ base routines such as global operator new.

Rename Processing

Rename processing is performed at the end of the first pass of final autocall processing of `DD SYSLIB`, when all possible references have been resolved with the names as they were on input. The binder renaming logic permits the conversion of unresolved non-DLL external function references and drives the final autocall process again.

The binder maps names according to the following hierarchy:
1. If the name has ever been mapped due to a pragma map in C++ code, the name is not renamed.
2. If the name has ever been mapped due to a pragma map in C code that was compiled with the `LONGNAME` option, the name is not renamed.
3. If a valid `RENAME` control statement was read for an unresolved function name, new-name specified on the applied `RENAME` statement is chosen, provided that old-name did not already appear on an applied `RENAME` statement as either a new or old name. Syntactically correct `RENAME` control statements that are not applied are ignored. See "RENAME Control Statement" on page 278.
4. If the name corresponds to a Language Environment function, the binder may map the name according to C/C++ run-time library rules.
5. If the `UPCASE(YES)` option is in effect and the name is 8 bytes or less, and not otherwise renamed by any of the previous rules, the name chosen is the same name but with all alphabetic characters mapped to uppercase, and '_' mapped to '@'. The binder maps names with the initial characters IBM, CEE, or PLI to initial characters of IB$, CE$, and PL$, respectively. All names that are different only in case will map to the same name.

If renamed, the original name is replaced. The original name and the generated new name appear in the rename table of the binder listing. See "Renamed Symbol Cross Reference" on page 385.

Generating Aliases for Automatic Library Call (Library Search)

For library search purposes, a member of a library (PDS, PDSE, or archive) can be an object module, a load module, or a program object. It has one member name, but may define multiple symbols (variables or functions) within it. To make library search successful, you must expose these defined symbols as aliases to the binder.

When the binder searches for an unresolved reference, it can find, through the member name or an alias, the member which contains the definition. It then reads that member.

You can create aliases in the following ways:
- `ALIAS` binder control statement
- `ALIASES(ALL)` binder option
- `ar` utility for object module archives
- `EDCALIAS` utility for object module PDS and PDSEs

Note: Aliases that the `EDCALIAS` utility generates are supported only for migration purposes. Use the `EDCALIAS` utility only if you need to provide autocall libraries to both prelinker and binder users. Otherwise, you should use the `ALIASES(ALL)` option, and bind separate compile units.

Dynamic Link Library (DLL) Processing

The binder supports the code that is generated by C++, and by C with the `DLL` compiler option, as well as code that is generated by C and C++ with the `XPLINK` option. Code generated with the `XPLINK` compiler option, like code generated by C++ and code generated by C with the DLL option, is always DLL-enabled (that is, references can be satisfied by IMPORT control statements). The binder option `DYNAM(DLL)` controls DLL processing. You must specify `DYNAM(DLL)` if the program object is to be a DLL, or if it contains DLL-type references. This section assumes that you specified the `DYNAM(DLL)` option. See “DYNAM(DLL | NO)” on page 271 for more information on the `DYNAM(DLL)` binder option. You must also specify CASE(MIXED) in order to preserve the case sensitivity of symbols on IMPORT control statements.

If you are building an application that imports symbol definitions from a DLL, you must include an `IMPORT` control statement for each symbol to which your application expects to dynamically link. Typically, the input to your application’s bind step should include the definition side-deck of `IMPORT` control statements that the binder generated when the DLL was built. For compatibility, the binder accepts definition side-decks of `IMPORT` control statements that the Language Environment Prelinker generated. To use the definition-side decks that are distributed with IBM Class libraries, you must specify the binder option `CASE(MIXED)`.

After final autocall processing of `DD SYSLIB` is complete, all DLL-type references that are not statically resolved are compared to `IMPORT` control statements. Symbols on `IMPORT` control statements are treated as definitions, and cause a matching unresolved symbol to be considered dynamically rather than statically resolved. A dynamically resolved symbol causes an entry in the binder class B_IMPEXP to be created. If the symbol is unresolved at the end of DLL processing, it is not accessible at run time.

Addresses of statically bound symbols are known at application load time, but addresses of dynamically bound symbols are not. Instead, the run-time library that loads the DLL that exports those symbols finds their addresses at application run time. The run-time library also fixes up the importer’s linkage blocks (descriptors) in C_WSA during program execution.

The binder builds tables of imported and exported symbols in the class B_IMPEXP, section IEWBCIE. This element contains the necessary information about imported and exported symbols to support run-time library dynamic linking and loading.

Statically bound functions

For each DLL-referenced function, the binder will generate a function linkage block (descriptor) of the same name as a part in the class C_WSA.

Some of the linkage descriptors for `XPLINK` code are generated by the compiler rather than the binder. Compiler-generated descriptors are not visible as named entities at bind time. For `XPLINK`:

- Functions, which are referenced exclusively in the compilation unit, have descriptors which are generated by the compiler and have no visible names.
- Functions, which are possibly referenced outside of the compilation unit (either by function pointer, or because they are exported), have descriptors which are generated by Language Environment when the DLL is loaded. They are not part of C_WSA. There will be a pointer to the function descriptor in C_WSA.
- For all other DLL-referenced functions, function descriptors are generated by the binder as a part with the same name in the class C_WSA (with the exception that for NORENT compiles, the descriptor will be in B_DESCR rather than C_WSA).

All C++ code and `XPLINK` code generate DLL references. C code generates DLL references if you used the `DLL` compiler option. If a DLL reference to an external function is resolved at the end of final autocall processing, the binder generates a function linkage block of the same name in the Writable Static Area, and initialize it to point to the resolved function. If the DLL reference is to a static function, the binder generates a function linkage block with a private name, which is initialized to point to the resolved static function.

Imported Variables

For each DLL-referenced external variable in C_WSA that is unresolved at the end of final autocall processing (`DD SYSLIB`), if a matching `IMPORT` control statement was read in, the variable is considered to be resolved via dynamic linking from the DLL named on the `IMPORT` control statement. The binder will generate a variable linkage block (descriptor) of the same name, as a part in the class C_WSA.

Imported Functions

For each DLL-referenced external function that is unresolved at the end of final autocall processing, if a matching `IMPORT` control statement was read in, the function is considered to be resolved via dynamic linking from the DLL named on the `IMPORT` control statement. The binder will generate a function linkage block (descriptor) of the same name, as a part in the class C_WSA.

Output Program Object

The `DD SYSLMOD` defines where the binder stores its output program object. You can store the output program object in one of the following:

- A PDSE member, where the binder stores a single program object
- A PDSE where the binder stores its output program objects (one program object for each `NAME` control statement)
- An HFS file or directory

The PDSE must have the attribute `RECFM=U`.

Output IMPORT Statements

The DD SYSDEFSD defines the output sequential data set where the binder writes out IMPORT control statements. The binder writes one control statement for each exported external symbol (function or variable), if you specify the option DYNAM(DLL). The data set must have the attributes RECFM=F or RECFM=FB, and LRECL=80.

You can mark symbols for export by using the #pragmaexport directive or the EXPORTALL compiler option, or the C++ _Export keyword.

Output Listing

This section contains an overview of the binder output listing. The binder creates the listing when you use the LIST binder option. It writes the listing to the data set that you defined by the DD SYSPRINT.

The listing consist of a number of categories. Some categories always appear in the listing, and others may appear depending on the options that you selected, or that were in effect.

Names that the binder generated appear as $PRIVxxxxxx rather than $PRIVATE. Private names that appear in the binder listing do not actually have that name in the program object. Their purpose in the listing is to permit association between various occurrences of the same private name within the listing. For purposes of rebindability, it is crucial that no sections have private names.

C++ names that appear in messages and listings are mangled names.

For the example listings in this section, the files USERID.PLAN9.OBJ(CU1) and /u/userid/plan9/cu2.o were bound together using the JCL shown in Figure 48 on page 383. Figure 47 on page 383 shows the corresponding source files:

```
/* file: USERID.PLAN9.C(CU1) */
/* compile with: LONGNAME RENT EXPORTALL CSECT("cu1")*/
#include <stdio.h>
int Ax=10;                                                    /* exported */
int ALongNamedThingVVWhichIsExported=11;      /* exported */
static int Az=12;
static int A1(void) {
   return Ax;
}
int ALongNamedThingFFWhichIsExported(void) {   /* exported */
  return Ax;
}
int A3(void) {              /* exported */
  return Ax + Az;
}
extern int b1(void);   /* statically bound, defined in plan9/cu2.C */
main() {
  int i;
  i = b1() + call_a3() + call_b1_in_cu2();
  printf("now returning\n");     /* printf statically bound from SCEELKEX */
  return i;
}
```

```
 /* file: cu2.C (C++ file) */
 /* compile with: CSECT(PROJ9) */
 extern b2(void);
 extern "C" c2(void);                                 /* imported from DLLC */
 extern     c3(void);                                  /* imported from DLLC */
 extern "C" int b1(void) {                           /* called from cu1.c */
   return b2();
 }
 int b2(void) {
   return c2() + c3();
 }
```

Figure 47. Source Files for Listing Example

```
//BIND1    EXEC CBCB,
//         BPARM='LIST(ALL),MAP,XREF',
//         OUTFILE='USERID.PLAN9.LOADE(HELLO1),DISP=SHR'
//INOBJ    DD DISP=SHR,DSN=USERID.PLAN9.OBJ
//SYSDEFSD DD DISP=SHR,DSN=USERID.PLAN9.IMP
//SYSPRINT DD DISP=SHR,DSN=USERID.PLAN9.LISTINGS(CU1CU2R)
//SYSLIN   DD *
 INCLUDE INOBJ(CU1)
 INCLUDE '/u/userid/plan9/cu2.o'
 IMPORT CODE,DLLC,c1
 IMPORT CODE,DLLC,c2
 IMPORT CODE,DLLC,c3__Fv
 RENAME 'call_a3' 'A3'
 RENAME 'call_b1_in_cu2' 'b1'
 ENTRY CEESTART
 NAME CU1CU2(R)
/*
```

Figure 48. Listing Example JCL

Header

The heading always appears at the top of each page. It contains the product number, the binder version and release number, the date and the time the bind step began, and the entry point name. The heading also appears at the top of each section.

The following example header was produced using the batch emulator:

```
OS/390 V2 R10 BINDER         09:08:20 WEDNESDAY MAY 10, 2000
BATCH EMULATOR  JOB(USERIDXX) STEP(BIND1   ) PGM= IEWL      PROCEDURE(BIND    )
```

Input Event Log

This section is a chronological log of events that took place during the input phase of binding. The binder LIST option controls its presence. See "LIST(OFF | STMT | SUMMARY | NOIMP | ALL)" on page 272 for more information on the LIST option.

```
IEW2278I B352 INVOCATION PARAMETERS - AMODE=31,MAP,RENT,DYNAM=DLL,CASE=MIXED,
COMPAT=CURR,ALIASES=ALL,LIST(ALL),MAP,XREF
IEW2322I 1220  1    INCLUDE INOBJ(CU1)
IEW2308I 1112 SECTION CEESTART HAS BEEN MERGED.
IEW2308I 1112 SECTION PROJ9#CU1#C HAS BEEN MERGED.
IEW2308I 1112 SECTION ALongNamedThingVVWhichIsExported HAS BEEN MERGED.
IEW2308I 1112 SECTION Ax HAS BEEN MERGED.
IEW2308I 1112 SECTION PROJ9#CU1#S HAS BEEN MERGED.
IEW2308I 1112 SECTION CEEMAIN HAS BEEN MERGED.
IEW2308I 1112 SECTION PROJ9#CU1#T HAS BEEN MERGED.
IEW2322I 1220  2    INCLUDE '/u/userid/plan9/cu2.o'
IEW2308I 1112 SECTION PROJ9#cu2.C#C HAS BEEN MERGED.
IEW2308I 1112 SECTION PROJ9#cu2.C#S HAS BEEN MERGED.
IEW2308I 1112 SECTION PROJ9#cu2.C#T HAS BEEN MERGED.
IEW2322I 1220  3    IMPORT CODE 'DLLC' 'c1'
IEW2322I 1220  4    IMPORT CODE 'DLLC' 'c2'
IEW2322I 1220  5    IMPORT CODE 'DLLC' 'c3__Fv'
IEW2322I 1220  6    RENAME 'call_a3' 'A3'
IEW2322I 1220  7    RENAME 'call_b1_in_cu2' 'b1'
IEW2322I 1220  8    ENTRY CEESTART
IEW2322I 1220  9    NAME CU1CU2(R)
 :
 :
```

Module Map

The Module Map is printed only if you specify the binder MAP option. It displays the attributes of each loadable binder class, along with the storage layout of the parts in that class.

For C/C++ programmers who use constructed reentrancy, two classes are of special interest: C_CODE and C_WSA. The C_CODE class exists if C++ code is encountered or if C code is compiled with LONGNAME or RENT. The C_WSA class exists if any defined writable static objects are encountered.

```
                         *** M O D U L E  M A P ***

---------------
CLASS  C_CODE           LENGTH =      5E4  ATTRIBUTES = CAT,   LOAD, RMODE=ANY
---------------

 SECTION    CLASS                                      ------- SOURCE --------
  OFFSET   OFFSET  NAME               TYPE    LENGTH  DDNAME   SEQ  MEMBER

                0  PROJ9#CU1#C       CSECT       330  INOBJ     01  CU1
       0        0     PROJ9#CU1#C       LABEL
      D0       D0     ALongName-ported  LABEL
     190      190     A3                LABEL
     248      248     main              LABEL




---------------
CLASS  C_WSA            LENGTH =       68  ATTRIBUTES = MRG, DEFER , RMODE=ANY
---------------

      CLASS
     OFFSET  NAME               TYPE          LENGTH

          0  c3()               DESCRIPTOR        20
         20  c2                 DESCRIPTOR        20
         40  ALongName#000001   PART               4
         44  Ax                 PART               4
         48  $PRIV000011        PART              18
         60  $PRIV000014        PART               8
```

Data Set Summary

The Module Map ends with a data set summary table, which associates input files with a corresponding DD name and concatenation number.

The binder creates a dummy DDname for each unique HFS file when it processes HFS pathnames from control statements. For example, on an INCLUDE control statement. The dummy DDname has the format ″/nnnnnnn″, where nnnnnnn is an integer assigned by binder, and appears in messages and listings in place of the HFS filename.

```
          ***  DATA SET SUMMARY  ***

DDNAME    CONCAT   FILE IDENTIFICATION
/0000001    01     /u/userid/plan9/cu2.o
INOBJ       01     USERID.PLAN9.OBJ
SYSLIB      01     CEE.CEE180.SCEELKEX
SYSLIB      02     CEE.CEE180.SCEELKED
SYSLIB      03     CEE.CEE180.SCEECPP
```

Renamed Symbol Cross Reference

The renamed symbol cross reference is printed only if a name was renamed for library search purposes, and you specified the MAP binder option.

The binder normally processes symbols exactly as received. However, it may remove certain symbolic references if they are not resolved by the original name during autocall. See “Rename Processing” on page 379. During renaming, the original reference is replaced. Such replacements, whether resolved or not, appear in the Rename Table.

The rename table is a listing of each generated new name and its original old name.

```
***  RENAMED SYMBOL CROSS REFERENCE  ***
---------------------
RENAMED SYMBOL
      SOURCE SYMBOL
---------------------

A3
      call_a3

b1
      call_b1_in_cu2

***  END OF RENAMED SYMBOL CROSS REFERENCE  ***

*** E N D  O F  M O D U L E  M A P ***
```

Cross Reference Table

The listing contains a cross-reference table of the program object if you specify the XREF binder option. Each line in the table contains one address constant in the program object. The left half of the table shows the location (OFFSET) and reference type (TYPE) within a defined part (SECT/PART) where a reference occurs. The right half of the table describes the symbol being referenced.

```
                                         C R O S S - R E F E R E N C E  T A B L E
                                        __________________________________________


TEXT CLASS = C_CODE

--------------- R E F E R E N C E --------------------------  T A R G E T -------------------------------
CLASS                        ELEMENT     |                                              ELEMENT
OFFSET SECT/PART(ABBREV)      OFFSET  TYPE | SYMBOL(ABBREV)   SECTION (ABBREV)           OFFSET CLASS NAME
                                         |
    68 PROJ9#CU1#C                68 Q-CON | Ax               $NON-RELOCATABLE               44 C_WSA
    70 PROJ9#CU1#C                70 A-CON | CEESTART         CEESTART                        0 B_TEXT
   138 PROJ9#CU1#C               138 Q-CON | Ax               $NON-RELOCATABLE               44 C_WSA
   204 PROJ9#CU1#C               204 Q-CON | $PRIV000011      $NON-RELOCATABLE               48 C_WSA
   208 PROJ9#CU1#C               208 Q-CON | Ax               $NON-RELOCATABLE               44 C_WSA
   2E4 PROJ9#CU1#C               2E4 Q-CON | $PRIV000011      $NON-RELOCATABLE               48 C_WSA
   2E8 PROJ9#CU1#C               2E8 V-CON | b1               PROJ9#cu2.C#C                   0 C_CODE
   2EC PROJ9#CU1#C               2EC V-CON | A3               PROJ9#CU1#C                   190 C_CODE
   2F0 PROJ9#CU1#C               2F0 V-CON | b1               PROJ9#cu2.C#C                   0 C_CODE
   2F4 PROJ9#CU1#C               2F4 V-CON | printf           printf                          0 B_TEXT
   33C CEEMAIN                     4 A-CON | main             PROJ9#CU1#C                   248 C_CODE
   340 CEEMAIN                     8 A-CON | EDCINPL          EDCINPL                         0 B_TEXT
   3C8 PROJ9#cu2.C#C              78 V-CON | b2()             PROJ9#cu2.C#C                  E0 C_CODE
   3D0 PROJ9#cu2.C#C              80 A-CON | CEESTART         CEESTART                        0 B_TEXT
   4CA PROJ9#cu2.C#C             17A Q-CON | $PRIV000014      $NON-RELOCATABLE               60 C_WSA
   588 PROJ9#cu2.C#C             238 Q-CON | $PRIV000014      $NON-RELOCATABLE               60 C_WSA
   58C PROJ9#cu2.C#C             23C Q-CON | c2               $NON-RELOCATABLE               20 C_WSA
   590 PROJ9#cu2.C#C             240 Q-CON | c3()             $NON-RELOCATABLE                0 C_WSA
```

Imported and Exported Symbols Listing

The imported and exported symbols listing is part of the Module Summary Report, and is printed before other module summary information. This section will not appear if you do not specify the DYNAM(DLL) option, or if you are not importing or exporting any symbols.

This section follows the cross-reference table in the binder map. The listing shows the imported or exported symbols, and whether they name code or data. It also shows the DLL member name for imported symbols.

Descriptors are identified as such in the listing. One of the following generates an object module that exports symbols:

- Code that is compiled with the C, C++, or COBOL EXPORTALL compiler option
- C/C++ code that contains the #pragma export directive
- C++ code that contains the _Export keyword

The listing format is shown below. All imported symbols appear first, followed by all exported symbols. Within each group, symbol names appear in alphabetical order. There are some differences between the two groups:

- The member name or HFS filename for `IMPORT` is derived from the `IMPORT` control statement.
- The member name for exports is always the same as the DLL's member name and does not appear in the listing.
- Symbol and member names that are longer than 16 bytes are abbreviated in the listing, using a hyphen. If there are duplicates, they are abbreviated using a number sign and a number. The abbreviation table shows the mapping from the abbreviated names to the actual names. See "Long Symbol Abbreviation Table" on page 389.

In the example above, you can see that c2 and c3 are to be dynamically linked

```
  ***  I M P O R T E D   A N D   E X P O R T E D   S Y M B O L S  ***

 IMPORT/EXPORT     TYPE    NAME                MEMBER
 -------------     ----    ----------------    ----------------
    IMPORT         CODE    c2                  DLLC
    IMPORT         CODE    c3()                DLLC

    EXPORT         DATA    Ax
    EXPORT         CODE    ALongName-ported
    EXPORT         DATA    ALongName#000001
    EXPORT         CODE    A3

                          *** END OF IMPORT/EXPORT ***
```

from a DLL named DLLC. Also, this program exports variables Ax and ALongNamedThingVVWhichIsExported, and functions A3 and ALongNamedThingFFWhichIsExported.

Mangled to Demangled Symbol Cross Reference

The mangled to demangled name table is similar to the rename table. It cross-references demangled C++ names in object modules with their corresponding mangled names.

Note: Mangling is name encoding for C++, which provides type safe linkage. Demangling is decoding of a mangled name into a human readable format.

```
  ***  SHORT MANGLED NAMES ***
  ---------------------
  MANGLED NAME
        DE-MANGLED NAME
  ---------------------

  b2__Fv
        b2()

  c3__Fv
        c3()

  ***  END OF MANGLED TO DEMANGLED CROSS REFERENCE ***
```

The following example is for long mangled names.

```
          ** A B B R E V I A T I O N / D E M A N G L E D   N A M E S **

ABBR/MANGLE NAME    LONG SYMBOL


__javCls1-ension            :=
__javCls18_java/awt/Dimension
  $$DEMANGLED$$    ==    java.awt.Dimension

__javCls1-nuItem          :=
__javCls17_java/awt/MenuItem
  $$DEMANGLED$$    ==    java.awt.MenuItem

__jav15_j-ame()V          :=
__jav15_java/awt/Button9_buildName()V
  $$DEMANGLED$$    ==    void java.awt.Button.buildName()
```

Processing Options

The processing options section of the module summary lists values of the binder options that were in effect during the bind process.

```
PROCESSING OPTIONS:

   ALIASES             ALL
   ALIGN2              NO
   AMODE               31
   CALL                YES
   CASE                MIXED
   COMPAT              PM3
   DCBS                NO
   DYNAM               DLL
   :
   :
   ***END OF OPTIONS***
```

Save Operation Summary

The save summary for a save to a program object lists the `blocksize` of the target PDSE. If you specified `DYNAM(DLL)`, and are exporting symbols, the save operation summary shows the data set name or the HFS pathname of the side file. For example:

```
SAVE OPERATION SUMMARY:

   MEMBER NAME         CU1CU2
   LOAD LIBRARY        USERID.PLAN9.LOADE
   PROGRAM TYPE        PROGRAM OBJECT(FORMAT 3)
   VOLUME SERIAL       M06001
   DISPOSITION         REPLACED
   TIME OF SAVE        11.13.40  JUN  3, 1997
   SIDEFILE            USERID.PLAN9.IMP(CU1CU2)
```

Save Module Attributes

The save module attributes section displays the attributes of the program object. These attributes are saved in the PDSE directory along with the program name, or saved in the HFS file.

```
SAVE MODULE ATTRIBUTES:

   AC                 000
   AMODE               31
   DC                 NO
   EDITABLE           YES
   EXCEEDS 16MB       NO
   EXECUTABLE         YES
   MIGRATABLE         NO
   OL                 NO
   OVLY               NO
   PACK,PRIME         NO,NO
   PAGE ALIGN         NO
   REFR               NO
   RENT               YES
   REUS               YES
   RMODE              ANY
   SCTR               NO
   SSI
   SYM GENERATED      NO
   TEST               NO
   XPLINK             NO
   MODULE SIZE (HEX)  00001360
```

Entry Point and Alias Summary

The entry point and alias summary will show an entry type of ″HIDDEN″ for hidden aliases. Hidden aliases may not be visible to some system utilities, and are marked as ″not executable″, to prevent an unintentional load and execution. They are for autocall purposes only. If you specify the option ALIASES(ALL), the binder generates hidden aliases.

```
 ENTRY POINT AND ALIAS SUMMARY:

 NAME:            ENTRY TYPE AMODE C_OFFSET CLASS NAME        STATUS

 CEESTART          MAIN_EP      31 00000000 B_TEXT
 b1                HIDDEN          00000350 C_CODE           REASSIGNED
 b2()              HIDDEN          00000430 C_CODE           REASSIGNED
 main              HIDDEN          00000248 C_CODE           REASSIGNED
 Ax                HIDDEN          00000044 C_WSA            REASSIGNED
 ALongName-ported  HIDDEN          000000D0 C_CODE           REASSIGNED
 ALongName#000001  HIDDEN          00000040 C_WSA            REASSIGNED
 A3                HIDDEN          00000190 C_CODE           REASSIGNED
 CEEMAIN           HIDDEN          00000338 C_CODE           REASSIGNED
 PROJ9#cu2.C#C     HIDDEN          00000350 C_CODE           REASSIGNED
 PROJ9#cu2.C#S     HIDDEN          000005D8 C_CODE           REASSIGNED
 PROJ9#cu2.C#T     HIDDEN          000005E0 C_CODE           REASSIGNED
 PROJ9#CU1#C       HIDDEN          00000000 C_CODE           REASSIGNED
 PROJ9#CU1#S       HIDDEN          00000330 C_CODE           REASSIGNED
 PROJ9#CU1#T       HIDDEN          00000348 C_CODE           REASSIGNED

                                  ***** E N D  O F  R E P O R T *****
```

Long Symbol Abbreviation Table

The long symbol abbreviation table lists symbol names that do not fit in the space that is allocated to them in the listing. This is a cross reference of abbreviations to the actual name. The abbreviation table is printed for symbols greater than 16 bytes in length, if you specify the MAP(YES) and XREF(YES) binder options.

```
*** L O N G   S Y M B O L   A B B R E V I A T I O N   T A B L E ***

      ABBREVIATION        LONG SYMBOL

      ALongName-ported := ALongNamedThingFFWhichIsExported
      ALongName#000001 := ALongNamedThingVVWhichIsExported


*** E N D   O F   L O N G   S Y M B O L   A B B R E V .   T A B L E ***
```

DDname vs Pathname Cross Reference Table

This section appears only if you specified pathnames on control statements.

The binder creates a dummy DDname for each unique HFS file when it processes HFS pathnames from control statements. For example, on an INCLUDE control statement. The dummy DDname has the format ″/nnnnnnn″, where nnnnnnn is an integer assigned by the binder. The integer nnnnnnn appears in messages and listings in place of the HFS filename.

The DDname vs pathname cross reference table shows the correspondence between the dummy DDname and its corresponding HFS filename. The table appears only if there is a generated DDname. Pathnames that you specified on JCL have user-assigned DDnames, and do not appear in this table. The following is the format of the DDname vs pathname cross reference table.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|D D N A M E   V S   P A T H N A M E   C R O S S   R E F E R E N C E   |
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 DDNAME   PATHNAME
--------  ----------------------------------------------------------------

/0000001  /u/userid/plan9/cu2.o

                         *** END OF DDNAME VS PATHNAME ***
```

Message Summary Report

The binder generates a message summary report at the conclusion of each bind operation. The summary contains information on the types and severity of the messages that were issued during the bind process. You can search other parts of the listing to find where the messages were issued.

```
----------------------
MESSAGE SUMMARY REPORT
----------------------
 SEVERE MESSAGES        (SEVERITY = 12)
 NONE

 ERROR MESSAGES         (SEVERITY = 08)
 NONE

 WARNING MESSAGES       (SEVERITY = 04)
 NONE

 INFORMATIONAL MESSAGES (SEVERITY = 00)
 2008  2278  2308  2322


 **** END OF MESSAGE SUMMARY REPORT ****
```

Binder Processing of C/C++ Object to Program Object

The binder recognizes C/C++ object modules and performs special processing for them.

C/C++ categorizes reentrant programs as natural or constructed. The binder supports both natural reentrancy and C/C++ constructed reentrancy. However, programs that contain constructed reentrancy need additional run-time library for support while executing.

C code is naturally reentrant if it contains no data in the Writable Static Area. Modifiable data can be one of the following:
- External variables
- Static variables
- Writable strings
- DLL linkage blocks (descriptors) for variables
- DLL linkage blocks (descriptors) for functions

C++ code always has DLL type references for all function references that require a function descriptor in C_WSA. This means that all C++ programs are made reentrant via constructed reentrancy.

Programs with constructed reentrancy have two areas:
- A modifiable area that contains modifiable objects, seen in the binder class C_WSA
- A constant or reentrant area that contains executable code and constant data, seen in the binder classes B_TEXT or C_CODE.

Each user running the program receives a private copy of the C_WSA demand load class, which is mapped by the binder and is loaded by the run-time library. Multiple spaces or sessions can share the second part only if it is installed in the link pack area (LPA) or extended link pack area (ELPA). You must install PDSEs dynamically in the LPA.

To generate reentrant C/C++ code, follow these steps:
1. Compile your source files to generate code with constructed reentrancy as follows:
   - Compile your C source files with the `RENT` compiler option to generate code with constructed reentrancy.

    - Compile your C++ source files with whatever options you require. The compiler will generate C++ code with constructed reentrancy.
2. Use the binder to combine all input object modules into a single output program object.

Each compile unit maps to a number of sections, which belong to the C_CODE, C_WSA, or B_TEXT binder classes. Named binder sections may be replaced and make the code potentially rebindable. You can name your C/C++ sections with either the CSECT compiler option, or with the use of the #pragma csect directive. The name of a section should not be the same as one of your functions or variables, as this will cause duplicate symbols.

Each section owns one or more parts. The names of the parts are the names that resolve references. The names of functions appear as labels, which also resolve references. Some parts that are owned by a section may be unnamed. Each part belongs to a binder class.

Each externally named object in the Writable Static Area appears as a part that is owned by a section of the same name in the program object. Such parts belong to the C_WSA binder class. The binder section that owns an object also owns the object's initialization information in the Writable Static Area. A rebind replaces this initialization information.

The code parts belong to the binder class of C_CODE or B_TEXT. The code parts consist of assembly instructions, constants and literals, and potentially read only variables that are not in the Writable Static Area. The following example will produce two sections, i and CODE1:

```
#pragma code(csect,"CODE1")
int i=10;
int foo(void) { return i; }
```

- The section named i is in class C_WSA, and has associated with it the initialization information to initialize 'i' to 10.
- The section named CODE1 is in class C_CODE, and has associated with it the entry point for function foo() and the machine instructions for the function.

When rebound, both sections i and CODE1 are replaced along with any information that is associated with them.

The names in the C_WSA class and in the C_CODE class are in the same namespace. A variable and a function cannot have the same name.

C++ constructor calls and destructor calls that need to be collected across compile units are collected in the class C_@@STINIT.

DLL initialization information, which needs to be collected across compile units, is collected in the class C_@@DLLI.

Note: The information in this section is applicable to XOBJ object modules and is not applicable to GOFF.

Rebindability

If the binder processes duplicate sections, it keeps **only the first one**. This feature is particularly important when rebinding. You must include the changed parts first and the old program object second. This is how you replace the changed sections.

The binder can process each object module separately so that you only need to recompile and rebind the modules that you have modified. You do not need to recompile or include the object module for any unchanged modules.

When the binder replaces a named section, it also replaces all of its parts (named or unnamed). If a section does not have the name you desire, you can change it with the `#pragma csect` directive or with the `csect` compiler option. Unnamed parts typically come from the following:
- Unnamed modifiable static parts in C_WSA (static variables, strings)
- Unnamed static parts in C_CODE that may not be modifiable (static variables, strings)
- Unnamed code, static, or test part in C_CODE

You should name all sections if you want to rebind. If a section is unnamed (has a private name) and you attempt to replace it on a rebind, the unnamed section is not replaced by the updated but corresponding unnamed section. Instead, the binder keeps both the old and new unnamed sections, causing the program module to grow in size. All references to functions that are defined by both the old section and the new section are resolved first to functions in the new section. The program may run correctly, but you will get warnings about duplicate function definitions at bind time. These duplicates will never go away on future rebinds because you cannot replace or delete unnamed sections. You will also accumulate dead code in the duplicate functions which can never be accessed. This is why it is important to name all sections if you want to rebind your code.

For example, suppose that our DLL consists of two compile units, `cu3.c` and `cu4.c`, that are bound using the JCL in Figure 49:

```
/* file: cu3.c */
/* compile with: LONGNAME RENT EXPORTALL*/
#pragma csect(code,"CODE3")
func3(void) { return 4; }
int int3 = 3;

/* file: cu4.c */
/* compile with: LONGNAME RENT EXPORTALL */
#pragma csect(code,"CODE4")
func4(void) { return 4; }
int int4 = 4;

//BIND1    EXEC CBCB,
//         BPARM='CALL,MAP,DYNAM(DLL)',
//         OUTFILE='USERID.PLAN9.LOADE,DISP=SHR'
//INOBJ    DD DISP=SHR,DSN=USERID.PLAN9.OBJ
//SYSLIN   DD *
 INCLUDE INOBJ(CU3)
 INCLUDE INOBJ(CU4)
 ENTRY CEESTART
 NAME BADEXE(R)
/*
```

Figure 49. JCL to bind cu3.c and cu4.c

Later, you discover that func3 is in error and should return 3. Change the source code in `cu3.c` and recompile. Rebind as follows:

```
//BIND1    EXEC CBCB,
//         BPARM='LIST(ALL),CALL,XREF,LET,MAP,DYNAM(DLL)',
//         OUTFILE='USERID.PLAN9.LOADE,DISP=SHR'
//INOBJ    DD DISP=SHR,DSN=USERID.PLAN9.OBJ
//INPOBJ   DD DISP=SHR,DSN=USERID.PLAN9.LOADE
//SYSLIN   DD *
 INCLUDE INOBJ(CU3)
 INCLUDE SYSLMOD(BADEXE)
 ENTRY CEESTART
 NAME GOODEXE(R)
/*
```

The input event log in the binder listing shows:

```
IEW2322I 1220  1    INCLUDE INOBJ(CU3)
IEW2308I 1112 SECTION CODE3 HAS BEEN MERGED.
IEW2308I 1112 SECTION int3 HAS BEEN MERGED.
IEW2322I 1220  2    INCLUDE INPOBJ(BADEXE)
IEW2308I 1112 SECTION CODE4 HAS BEEN MERGED.
IEW2308I 1112 SECTION int4 HAS BEEN MERGED.
IEW2308I 1112 SECTION CEESTART HAS BEEN MERGED.
IEW2308I 1112 SECTION CEESG003 HAS BEEN MERGED.
IEW2308I 1112 SECTION CEEBETBL HAS BEEN MERGED.
IEW2308I 1112 SECTION CEEBPUBT HAS BEEN MERGED.
IEW2308I 1112 SECTION CEEBTRM HAS BEEN MERGED.
IEW2308I 1112 SECTION CEEBLLST HAS BEEN MERGED.
IEW2308I 1112 SECTION CEEBINT HAS BEEN MERGED.
IEW2308I 1112 SECTION CEETGTFN HAS BEEN MERGED.
IEW2308I 1112 SECTION CEETLOC HAS BEEN MERGED.
IEW2322I 1220  3    ENTRY CEESTART
IEW2322I 1220  4    NAME GOODEXE(R)
```

`BADEXE` defines sections int3, CODE3, int4, and CODE4. If the binder sees duplicate sections, it uses the first one that it reads. Since `CU3` defines sections CODE3 and int3, and is included before `BADEXE`, both sections are replaced by the newer ones in `CU3` when program object `GOODEXE` is created.

DLL Considerations

Any IMPORT control statements used in the original bind must also be input to the re-bind.

Error recovery

This section describes common errors in binding.

Unresolved Symbols

Inconsistent reference vs. definition types

A common error is to compile one part of the code with `RENT` and another with `NORENT`. A `RENT` type reference (Q-CON in the binder listing) must be resolved by a Writable Static Area definition of a PART or a DESCRIPTOR in class C_WSA. A `NORENT` reference (V-CON or A-CON in the binder listing) must be resolved by CSECT or a LABEL typically in class C_CODE or B_TEXT.

Check the binder map to ensure that objects appear as parts in the expected classes (C_CODE, B_TEXT, C_WSA ...).

Inconsistent Name usage

Another problem is the case sensitivity of the symbol names. Objects in the Writable Static Area cannot be renamed, but unresolved function references may be renamed to find a definition of a different name. See "Rename Processing" on page 379 page 379. Such inconsistencies arise from inconsistent usage of the LONGNAME and NOLONGNAME compiler options, and from multi-language programs that make symbol names uppercase. For example, compile the file main.c with the options LONG, NORENT, and other.c with the options NOLONG, RENT:

```
 /* file: main.c */
 /* compile with LONG, NORENT */
 extern int I2;
 extern int func2(void);
 main() {
   int i;
   i = i2 + func2();
   return i;
 }
```

```
/* file: other.c */
/* compile with NOLONG,RENT */
int I2 = 2;
int func2(void) { return 2; }
```

When you bind the object modules together, the following errors will occur:

- An inconsistent use of the RENT | NORENT C compiler option causes symbol I2 to be unresolved. The definition of I2 from other.c is a writable static object because of the RENT option. But a writable static object cannot resolve the reference to I2 from main.c because it is a NORENT reference. The binder messages show:

  ```
  IEW2308I 1112 SECTION I2 HAS BEEN MERGED.

  IEW2456E 9207 SYMBOL I2 UNRESOLVED.
  ```

- An inconsistent use of the LONG | NOLONG C compiler option causes the symbol func2 to be unresolved. The function definition in other.c is in uppercase because of the NOLONG option. But the reference to func2 from main.c is in lowercase because of the LONG option. The binder listing shows that 'FUNC2' is a LABEL, that is a defined entry point; yet the binder messages show:

  ```
  IEW2456E 9207 SYMBOL func2 UNRESOLVED.
  ```

Significance of Library Search Order

The order in which the libraries in SYSLIB are concatenated is significant. For example suppose that functions f1() and f4() are resolved from SYSLIB:

```
  /* file: unit0.c */
  extern int f1(void);  /* from member UNIT1 of library LIB1 */
  extern int f4(void);  /* from member UNIT2 of library LIB2 */
  int main() {
    int rc1, rc4;
    rc1 = f1();
    rc4 = f4();
    if (rc1 !=  1) printf("fail rc1 is %d-n", rc1);
    if (rc4 != 40) printf("fail rc1 is %d-n", rc4);
    return 0;
  }
```

SYSLIB defines the libraries USERID.LIB1 with members UNIT1 and UNIT2, and USERID.LIB2 with members of the same name but different contents.

The library members are compiled from the following:

```
/* member UNIT1 of library LIB1 */
int f1(void) { return 1; }

/* member UNIT2 of library LIB1 */
int f2(void) { return 2; }
```

```
/* member UNIT1 of library LIB2 */
int f1(void) { return 10; }

/* member UNIT2 of library LIB2 */
int f2(void) { return 20; }
int f3(void) { return 30; }
int f4(void) { return f2()*2; /* 40 */ }
```

When bound with ALIASES(ALL), or when the EDCALIAS utility is used, all defined symbols are seen in a library directory as aliases that indicate the library member that contains their definition.

There are two definitions of f1(), but library search of SYSLIB for f1 searches library LIB1 first, and finds alias f1 of member UNIT1. It reads in that member, and the call to f1() returns 1. Library search of SYSLIB for f4 searches LIB1 first, and does not find a definition. It then searches LIB2, and finds alias f4 of member UNIT2 of library LIB2. So UNIT2 of library LIB2 is read in resolving not only f4, but also f2 and f3, and the call to f4() returns 40. UNIT2 of library LIB1 is not read by mistake because an alias indicates not only the member name, but also the library in which that member resides.

If the order of LIB1 and LIB2 is reversed, LIB2 is searched first, and f1() is obtained from LIB2 instead.

If changing the library search order cannot work for you, use the LIBRARY control statement. See “LIBRARY Control Statement” on page 276.

Duplicates

If the binder processes duplicate sections, it keeps the first one and ignores subsequent ones, without giving a warning. This feature is used to replace named sections when rebinding by replacing only changed sections.

If the binder processes functions that have duplicate names, it keeps all definitions, but all references resolve to the first one. An exception is in the case of C++ template instantiation. The binder takes the first user-defined function (if any) of the same signature rather than the first compiler-generated definition via template instantiation.

For example, compile the following source files doit1.c and doit2.c:

```
#include <stdio.h>
/* file: doit1.c */
int int1 = 1;
#pragma csect(code,"DO1")
int func2(void) { return 2; }
int func3(void) { return 3; }
extern int func4(void);
int main() {
  int i1,i2,i3,i4;
  i1 = int1;
  i2 = func2();
  i3 = func3();
  i4 = func4();
  printf("%d %d %d\n",i1,i2,i3,i4);
  return 0;
}
```

```
/* file: doit2.c */
int int1 = 11;
#pragma csect(code,"DO2")
int func3(void) { return 33; }
int func4(void) { return 44; }
```

Use the LONGNAME compiler option, and bind. The binder sections are int1, DO1 and int1, DO2. The binder keeps one of the duplicate sections, int1, and does not issue a warning. But uniquely named sections contain the functions. Section DO1 contains the functions func2 and func3. Section DO2 contains the functions func3 and func4. The binder retains both sections DO1 and DO2, but because both sections contain function func3, it issues a warning message as follows:

```
IEW2480W A711 EXTERNAL SYMBOL func3 OF TYPE LD WAS ALREADY DEFINED AS A
SYMBOL OF TYPE LD IN SECTION DO1.
```

It is easier to find the object code with the duplicate if you use multiple INCLUDE statements rather than DD concatenation. For example, if you use:

```
  //INOBJ  DD DISP=SHR,DSN=USERID.PLAN9.OBJ
  //SYSLIN DD *
   INCLUDE INOBJ(DOIT1)
   INCLUDE INOBJ(DOIT2)
   ENTRY CEESTART
  /*
```

The members in the binder listing are separated logically. The messages in the binder listing are:

```
  :
  :
IEW2322I 1220  1    INCLUDE INOBJ(DOIT1)
IEW2308I 1112 SECTION CEESTART HAS BEEN MERGED.
IEW2308I 1112 SECTION DO1 HAS BEEN MERGED.
IEW2308I 1112 SECTION int1 HAS BEEN MERGED.
IEW2308I 1112 SECTION CEEMAIN HAS BEEN MERGED.
IEW2322I 1220  2    INCLUDE INOBJ(DOIT2)
IEW2480W A711 EXTERNAL SYMBOL func3 OF TYPE LD WAS ALREADY DEFINED AS A
SYMBOL OF TYPE LD IN SECTION DO1.
IEW2308I 1112 SECTION DO2 HAS BEEN MERGED.
```

From the informational messages, it is clear that section DO1 is from INOBJ(DOIT1), and that DO2 is from INOBJ(DOIT2). But if you use DD concatenation as follows:

```
  //INOBJ  DD DISP=SHR,DSN=USERID.PLAN9.OBJ
  //SYSLIN DD DISP=SHR,DSN=USERID.PLAN9.OBJ(DOIT1)
  //       DD DISP=SHR,DSN=USERID.PLAN9.OBJ(DOIT2)
  //       DD *
   ENTRY CEESTART
  /*
  :
  :
```

Now the messages are:

```
IEW2308I 1112 SECTION CEESTART HAS BEEN MERGED.
IEW2308I 1112 SECTION DO1 HAS BEEN MERGED.
IEW2308I 1112 SECTION int1 HAS BEEN MERGED.
IEW2308I 1112 SECTION CEEMAIN HAS BEEN MERGED.
IEW2480W A711 EXTERNAL SYMBOL func3 OF TYPE LD WAS ALREADY DEFINED AS A
SYMBOL OF TYPE LD IN SECTION DO1.
IEW2308I 1112 SECTION DO2 HAS BEEN MERGED.
```

It is no longer clear which input file defines which section, and this makes tracking down duplicates to the originating compile unit more difficult.

Duplicate functions from autocall

If a library member that is expected to contain the definition of a symbol is read, it may resolve the expected symbol. It may also resolve other symbols because the library member may define multiple functions. These unexpected definitions that are pulled in through library search may cause duplicates. Since you cannot always be sure which one of the duplicate symbols you will resolve with, you should remedy the situation that is causing the duplicate symbols.

Hunting down references to unresolved symbols

Unresolved requests generate error or warning messages in the binder listing. If a function or variable is unresolved at the end of binder processing, it can be resolved at a later rebind.

If you did not expect a symbol to remain unresolved, you can look at the binder listing to see which parts reference the symbol. If your `DD: SYSLIN` has a large concatenation, the input is logically concatenated before the binder processes it. Since the compile units are not logically separated, it is hard to tell which compile unit defines the part that has the reference. For example:

```
//SYSLIN DD DISP=SHR,DSN=USERID.PLAN9.OBJ(MEM1)
//       DD DISP=SHR,DSN=USERID.PLAN9.OBJ(MEM2)
//       DD DISP=SHR,DSN=USERID.PLAN9.OBJ(MEM3)
```

You should consider using multiple `INCLUDE` control statements, which will logically separate the compile units for the binder informational messages in the listing. You can then find the compile unit with the unresolved reference (similar to finding duplicate function definitions). For example:

```
//INOBJ  DD DISP=SHR,DSN=USERID.PLAN9.OBJ
//SYSLIN DD *
 INCLUDE INOBJ(DOIT1)
 INCLUDE INOBJ(DOIT2)
 ENTRY CEESTART
/*
```

Incompatible Linkage Attributes

The binder will check that a statically bound symbol reference and symbol definition have compatible attributes. If a mismatch is detected, the binder will issue a diagnostic message. This attribute information is contained within the binder input files, such as object files, program objects, and load modules.

For C and C++, the default attribute is based on the `XPLINK` and `NOXPLINK` options. Individual symbols can have a different attribute than the default by using the #pragma `OS_UPSTACK`, #pragma `OS_DOWNSTACK`, and #pragma `OS_NOSTACK`.

The attributes can also be set for assembly language. Refer to the *HLASM Language Reference*, SC26-4940 for further information.

Non-reentrant DLL Problems

If you bind a DLL with the option `REUS(NONE)`, each load of the DLL causes a separate load of the code area and the data area (C_WSA). If you split a statically bound program into mutually dependent DLLs, you will probably not get the desired result. Function pointers that used to compare the same may not be the same anymore, because the multiple loads of a DLL have more than one copy of the function in memory.

The same is true for data. A separate copy of C_WSA is loaded. So, data objects that are exported from a DLL and modified are not seen as modified by the new program that uses the DLL. You should bind all DLLs with `REUS(RENT)`, or `REUS(SERIAL)` so that a new C_WSA is loaded only once per enclave.

Code That Has Been Prelinked

You cannot bind code that refers to objects in the Writable Static Area and has been prelinked, and code which refers to objects in the Writable Static Area and has not been prelinked, in the same program object. This is because the z/OS Prelinker and the binder use different methods to manage the Writable Static Area. The z/OS Prelinker removes relocation information about objects in the Writable Static Area, making them invisible to the binder. The binder keeps relocation information and manages the Writable Static Area in the binder class C_WSA.

Chapter 14. Running a C or C++ Application

This chapter gives an overview of how to run z/OS C/C++ programs under z/OS batch, TSO, and the z/OS Shell.

z/OS Language Environment provides a common runtime environment for C, C++, COBOL, PL/I, and FORTRAN. For detailed instructions on running existing and new z/OS C/C++ programs under z/OS Language Environment, refer to *z/OS Language Environment Programming Guide*. *z/OS C/C++ Programming Guide* also describes how to run z/OS C/C++ programs in a CICS environment.

Setting the Region Size for z/OS C/C++ Applications

Prior to running your applications, ensure that you have the required region size to run the compiler and to run your application.

Note: Our current default region size is 48M, but depending on your program and the degree of optimization you're using (i.e., OPT(2) and/or IPA), you may require significantly more space.

If your installation does not change the IBM-supplied default limits in the IEALIMIT or IEFUSI exit routine modules, different values for the region size have the following results:

Region Size Value	Result
`0K` or `0M`	Provides the job step with all the storage that is available below and above 16 MB. The resulting size of the region below and above 16 MB is unpredictable.
> `0K` or `0M`, and ≤ `16384K` or `16M`	Establishes the size of the private area below 16 MB. If the region size specified is not available below 16 MB, the job step terminates abnormally. The extended region size is the default value of 32 MB.
> `16384K` or `16M`, and ≤ `32768K` or `32M`	Provides the job step all the storage available below 16 MB. The resulting size of the region below 16 MB is unpredictable. The extended region size is the default value of 32 MB.
> `32768K` or `32M`, and ≤ `2096128K` or `2047M`	Provides the job step all the storage available below 16 MB. The resulting size of the region below 16 MB is unpredictable. The extended region size is the specified value. If the region size specified is not available above 16 MB, the job step abnormally terminates.

Assuming that you do not use your own IEFUSI exit to override this, a specification of `REGION=4M` provides 4 MB below 16 MB, and a default of 32 MB above 16MB for a total of 36 MB of available virtual memory and not just 4 MB.

Specifying `REGION=40M` provides all available private virtual memory below 16 MB, most likely around 8 MB to 10 MB, and 40 MB above 16 MB for a total of around 48 MB. This means that a JCL change from `REGION=4M` to `REGION=40M` does not change the virtual storage available to the compiler from 4MB to 40MB, but rather from 36MB to 48MB. If the only storage use increase is above 16 MB, then the actual increase is 8 MB.

Running an Application Under z/OS Batch

You must have the Language Environment Library SCEERUN available before you try to run your application under z/OS batch.

If your application was compiled using the XPLINK compiler option you must have the Language Environment Library SCEERUN2 available before you try to run your application under z/OS batch.

If your application was bound with the DLL Class Libraries, you must supply SCLBDLL at run time. The DLL data set can be in the system libraries, your JOBLIB statement, or your STEPLIB statement.

The search sequence for library files is in the following order: STEPLIB, JOBLIB, LINKPACK, and LINKLIST.

Specifying Runtime Options under z/OS Batch

When you run a C or C++ application, you can override the default values for a set of z/OS C/C++ runtime options. These options affect your application's execution, including its performance, its error-handling characteristics, and its production of debugging and tuning information.

For your application to recognize runtime options, either the EXECOPS compiler option, or the #pragma runopts(execops) directive must be in effect. The default compiler option is EXECOPS.

You can specify runtime options under z/OS batch as follows:

- In your JCL; in the PARM parameter of the EXEC statement. For more information, refer to “Specifying Runtime Options in the EXEC Statement” on page 403.
- On the GPARM parameter of the cataloged procedures that are supplied by IBM. Refer to “Using Cataloged Procedures” on page 403.
- The #pragma runopts statement in your source code.
- The CEEUOPT facility that is provided by z/OS Language Environment.
- In the assembler user exit. For more information, refer to the *z/OS C/C++ Programming Guide*.

If EXECOPS is in effect, use a slash '/' to separate runtime options from arguments passed to the application. For example:

```
GPARM='STORAGE(FE,FE,FE)/PARM1,PARM2,PARM3'
```

Language Environment interprets the character string that precedes the slash as runtime options. The character string following the slash is passed to your application's main() function as arguments. If a slash does not separate the arguments, Language Environment interprets the entire string as an argument.

If the NOEXECOPS option is in effect, none of the preceding runtime options will take effect. In fact, any arguments and options that you specify in the parameter string (including the slash, if present) are passed as arguments to the main() function. For a description of runtime options see “Specifying Runtime Options” on page 279.

You should establish the required settings of the options for all z/OS C/C++ programs that you execute on a production basis. Each time the program is run, the default runtime options that were selected during z/OS C/C++ installation apply, unless you override them by using one of the following:

- Coding a `#pragma` runopts directive in your source
- Creating a `CEEUOPT` csect with the `CEEXOPT` macro and linking this csect into the program module.
- Specifying runtime options in the `EXEC` or `GPARM` statements

The following example shows you how to run your program under z/OS batch. Partitioned data set member `MEDICAL.ILLNESS.LOAD(SYMPTOMS)` contains your z/OS C/C++ executable program. The program was compiled with the `EXECOPS` compiler option in effect. If you want to use the runtime option `RPTOPTS(ON)`, and to pass `TESTFUNCT` as an argument to the function, use the JCL stream as follows:

```
//JOBname  JOB...
//STEP1    EXEC  PGM=SYMPTOMS,PARM='RPTOPTS(ON)/TESTFUNCT'
   ⋮
//STEPLIB  DD    DSN=MEDICAL.ILLNESS.LOAD,DISP=SHR
//         DD    DSN=CEE.SCEERUN,DISP=SHR
```

Figure 50. Running your program under z/OS Batch

Specifying Runtime Options in the EXEC Statement

You can specify runtime options in the `PARM` parameter of the `EXEC` statement as follows:

```
//[stepname] EXEC PGM=program_name,
//           PARM='[runtime options/][program parameters]'
```

For example, if you want to generate a storage report and runtime options report for the application `PROGRAM1`, specify the runtime option `RPTOPTS(ON)` as follows:

```
//GO1   EXEC PGM=PROGRAM1,PARM='RPTOPTS(ON) / '
```

Note that the runtime options that are passed to the main routine are followed by a slash (/) to separate them from program parameters.

Using Cataloged Procedures

You can use one of the following cataloged procedures that are supplied with the z/OS C/C++ compiler to run your program. Each procedure listed below includes an execution step:

For z/OS C programs:

`EDCCBG`	Compile, bind, and run.
`EDCXCBG`	Compile, bind, and execute an XPLINK C Program.

For z/OS C++ programs:

`CBCBG`	Bind and run.
`CBCCBG`	Compile, bind, and run.
`CBCG`	Run
`CBCXBG`	Bind and run an `XPLINK` z/OS C++ program.
`CBCXCBG`	Compile, bind, and run an `XPLINK` z/OS C++ program.
`CBCXG`	Run an `XPLINK` z/OS C++ program.

For more information on these cataloged procedures, see “Appendix D. Cataloged Procedures and REXX EXECs” on page 529.

If you are using an IBM-supplied cataloged procedure, you must specify the runtime options on the `GPARM` parameter of the `EXEC` statement. Ensure that the `EXECOPS` runtime option is in effect. For example:

```
//STEP   EXEC EDCCBG,INFILE='...',
//            GPARM='STACK(10K)/'
```

You can also use the GPARM parameter to pass arguments to the z/OS C/C++ main() function. Place the argument, preceded by a slash, after the runtime options. For example:

```
//GO  EXEC EDCCBG,INFILE=...,
//         GPARM='STACK(10K)/ARGUMENT'
```

If you want to pass an argument without specifying runtime options and EXECOPS is in effect (this is the default), precede it with a slash. For example:

```
//GO  EXEC  EDCCBG,...GPARM='/ARGUMENT'
//GO  EXEC  EDCCBG,...GPARM='/HFS file:/u/mike/cloudy.C'
```

If you want to pass parameters which contain slashes, and you are not providing runtime options, you must precede the parameters with a slash, as follows:

```
//GO  EXEC  EDCCBG,...GPARM='/HFS file:/u/mike/cloudy.C'
```

See also “Specifying Runtime Options” on page 279.

Running an Application under TSO

Before you run your program under TSO, you must have access to the runtime library CEE.SCEERUN. To ensure that you have access to the runtime library, do one of the following:

- If you are running under ISPF in the foreground, concatenate the libraries to ISPLLIB.
- Have your system programmer add the libraries to the LPALST or LPA.
- Have your system programmer add the libraries to the LNKLST.
- Have your system programmer change the LOGON PROC so the libraries are added to the STEPLIB for the TSO session.
- If your application was compiled using the XPLINK compiler option, you must have the Language Environment Library SCEERUN2 available before you try to run your application under TSO.
- If you are using IBM Open Class Libraries, concatenate the SCLBDLL data set to C++ STEPLIB, or add it to the LPA.

The TSO CALL command runs a load module under TSO. If *data-set-name* is the partitioned data set member that holds the load module, the command to load and run a specified load module is:

```
CALL 'data-set-name' ['parameter-string'];
```

For example, if the load module is stored in partitioned data set member ’SAMPLE.CPGM.LOAD(TRICKS)’, and the default runtime options are in effect, run your program as follows:

```
CALL 'SAMPLE.CPGM.LOAD(TRICKS)'
```

If you specify the unqualified name of the data set, the system assumes the descriptive qualifier LOAD. If you do not specify a member name, the system assumes the name TEMPNAME.

You do not need to use the CALL command if the STEPLIB DD name includes the data set that contains your program. For example, you could call a program PROG1 with two required parameters PARM1 and PARM2 from the command line:

```
PROG1 PARM1 PARM2
```

See the appropriate manual listed in *z/OS Information Roadmap* for more information on STEPLIB.

Specifying Runtime Options under TSO

You can specify runtime options in a #pragma runopts directive or in the *'parameter-string'* of the TSO CALL command. The 'parameter-string' contains two fields that are separated by a slash(/), and takes the form:

```
'[runtime options/][arguments to main]'
```

The first field is passed to the program initialization routine as a runtime option list; the second field is passed to the main() function.

To allow your application to recognize runtime options, EXECOPS must be in effect. You can specify your additional runtime options on the command line as follows: specify the options followed by a slash (/), followed by the parameters you want to pass to the main() function.

For example, to run a load module that is stored in the partitioned data set member GINGER.HOURLY.LOAD(CHECK), with the runtime option RPTOPTS(ON), use the following command:

```
CALL 'GINGER.HOURLY.LOAD(CHECK)' 'RPTOPTS(ON)/'
```

If the NOEXECOPS compiler or runtime option is in effect, what you specify on the command line (including the slash, if present) is passed as arguments to the main() function. For a description of runtime options see “Specifying Runtime Options” on page 279 .

If you want to pass your parameters as mixed case, you must use the ASIS runtime option. See “Passing Arguments to the z/OS C/C++ Application” for more information on passing mixed case parameters.

Passing Arguments to the z/OS C/C++ Application

The arguments passed to main() are argc and argv. argc is an integer whose value is the number of arguments that are given when the program is run. argv is an array of pointers to null terminated character strings, which contain the arguments for the program. The first argument is the name of the program being run on the TSO command line. For more information on argc, argv, and main() see “ARGPARSE | NOARGPARSE” on page 83 or the *z/OS C/C++ Language Reference*.

The case of the characters in argv depends on you invoked how your z/OS C/C++ program, as shown in the following table.

Table 46. Case sensitivity of arguments under TSO

How the z/OS C/C++ program is invoked	Example	Case of argument
As TSO command	`program args`	Mixed case (However, if you pass the arguments entirely in upper case, the argument will be changed to lower case.)
By CALL command (with or without ASIS)	`CALL program args`	Lower case

Table 46. Case sensitivity of arguments under TSO (continued)

How the z/OS C/C++ program is invoked	Example	Case of argument
By `CALL` command with control arguments `ASIS`	`CALL program Args ASIS`	Mixed case (However, if you pass the arguments entirely in upper case, the argument will be changed to as lower case.)
By `CALL` command with control `ASIS`	`CALL program ARGS ASIS`	The arguments will be changed to lower case following ANSI/ISO C standards.

Running an Application under z/OS UNIX

This section discusses how to run your z/OS UNIX System Services C/C++ application.

z/OS UNIX Application Environments

You can run your z/OS UNIX System Services C/C++ application programs from the following environments:

- z/OS shell
- z/OS ISPF Shell (ISHELL)
- TSO/E

  To call an application program that resides in an HFS file from the TSO/E `READY` prompt, you must use the `BPXBATCH` utility.
- z/OS batch

  To run an application program that resides in an HFS file, you must use the `BPXBATCH` utility with the JCL `EXEC` statement.
- z/OS shell through z/OS batch or TSO

  By using the IBM-supplied `BPXBATCH` program, you can run an application program that resides in an HFS file. You supply the name of the program as an argument to the `BPXBATCH` program, which invokes the shell environment. The `BPXBATCH` runs under the z/OS batch environment or under TSO.

Specifying Runtime Options under z/OS UNIX

When invoking a program from the z/OS shell, slash-separated runtime options arguments syntax is not used. All the arguments always go to the `main()` routine. Specify runtime options by using the exported environment variable `_CEE_RUNOPTS`. The runtime will only use `_CEE_RUNOPTS` if the `EXECOPS` option is in effect.

Restriction on Using 24-bit AMODE Programs

You cannot run a 24-bit AMODE z/OS C/C++ application program that resides in an HFS file. Any programs you intend to run from the file system must be 31-bit AMODE, problem program state, PSW key 8 programs. If you plan to run a 24-bit AMODE z/OS C/C++ program from within an application, ensure that the executable resides in a PDS or PDSE member.

Any new z/OS UNIX System Services z/OS C/C++ applications you develop should be 31-bit AMODE.

Copying Applications between a PDS and HFS

If you have a C/C++ application as a PDS member and want to place it in the HFS, you can use the z/OS UNIX System Services TSO/E command `OPUTX` to copy the member into an HFS file.

If you have a C/C++ application as an HFS file and want to place it in a PDS, you can use the z/OS UNIX System Services TSO/E command `OGETX` to copy the HFS file into a PDS.

You can also bind directly into a data set member with the `c89` or `c++` utility by specifying a data set member name on the `-o` option, as in:

```
c89 -o"//loadlib(foo)"
```

For a description of these commands, see "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555. For examples of using these commands to copy data sets to HFS files, see *z/OS UNIX System Services User's Guide*.

Running a Data Set Member from the z/OS Shell

If your z/OS UNIX System Services C/C++ program resides in data sets and you must run the executable member from within the shell, you can pass a call to the program to TSO/E. Type the TSO/E `CALL` command with the name of the executable data set member on the shell command line and press the TSO/E function key to pass the command to TSO/E. Alternatively, you can use the `tso` command from under the shell. Just precede the `CALL` with `tso` on the command line and press the `ENTER` key.

When the program completes, the shell session is restored.

Running z/OS UNIX Applications under z/OS Batch

Using the BPXBATCH Utility

Use the IBM-supplied `BPXBATCH` program to run a C/C++ application under z/OS batch from an HFS file. You can invoke the `BPXBATCH` utility from TSO/E, or by using JCL. The `BPXBATCH` utility submits a batch job and performs an initial user login to run a specified program from the shell environment.

Before you invoke `BPXBATCH`, you must have the appropriate authority to read from and write to HFS files. You should also allocate `stdout` and `stderr` HFS files for writing program output such as error messages. Allocate the standard files using the PATH options on TSO/E `ALLOCATE` command or the JCL DD statement.

For more information on the `BPXBATCH` program, refer to "Chapter 21. BPXBATCH Utility" on page 455.

Invoking BPXBATCH from TSO/E

From TSO/E, you can invoke `BPXBATCH` several ways:
- From the TSO/E `READY` prompt
- From a `CALL` command
- From a REXX exec

Figure 51 on page 408 shows a REXX EXEC that does the following:
1. runs the application program `/myap/base_comp` from your user ID
2. directs output to the file `/myap/std/my.out`

3. writes error messages to the file /myap/std/my.err
4. copies the output and error data to data sets

```
/* base_comp REXX exec */
"Allocate File(STDOUT) Path('/u/myu/myap/std/my.out') Pathopts(OWRONLY,OCREAT,OTRUNC)
         Pathmode(SIRWXU) Pathdisp(DELETE,DELETE)"
"Allocate File(STDERR) Path('/u/myu/myap/std/my.err') Pathopts(OWRONLY,OCREAT,OTRUNC)
         Pathmode(SIRWXU) Pathdisp(DELETE,DELETE)"

"BPXBATCH PGM /u/myu/myap/base_comp"

"Allocate File(output1) Dataset
('MYAPPS.STD(BASEOUT)')"
"Ocopy Indd(STDOUT) Outdd(output1) Text Pathopts(OVERRIDE)"

"Allocate File(output2) Dataset('MYAPPS.STD(BASEERR)')"
"Ocopy Indd(STDERR) Outdd(output2) Text Pathopts(OVERRIDE)"
```

Figure 51. REXX EXEC to Run a Program

To invoke BPXBATCH, enter the name of the REXX exec from the TSO/E READY prompt. When the REXX exec completes, the stdout and stderr allocated files are deleted.

Invoking BPXBATCH Using JCL

To invoke BPXBATCH using JCL, submit a job that executes an application program and allocates the standard files using DD statements. For example, to run the application program /myap/base_comp from your user ID, direct its output to the file /myap/std/my.out, and write error messages to the file /myap/std/my.err, code the JCL statements as follows:

```
//jobname  JOB ...
//stepname EXEC PGM=BPXBATCH,PARM='PGM /u/myu/myap/base_comp'
//STDOUT    DD PATH='/u/myu/myap/std/my.out',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//STDERR    DD PATH='/u/myu/myap/std/my.err',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
```

Submitting a non-HFS z/OS UNIX Executable to Run under z/OS Batch

If your program requires z/OS UNIX System Services, but has been link edited into a load module (PDS member) or bound into a non-HFS program object (PDSE member), it may be executed in the z/OS batch environment. Use the JCL ″EXEC″ statement to submit the executable to run under the batch environment. You must have the runtime option POSIX in effect, either as #pragma runopts(POSIX(ON)), or as PARM='POSIX(ON)/'.

Part 4. Utilities and Tools

This section contains information about the utilities and tools that you can use under z/OS.

Chapter 15. Object Library Utility

This chapter describes how to use the Object Library Utility to update libraries of object modules. On z/OS, a library is a PDS or PDSE with object modules as members.

Object libraries provide convenient packaging of object modules. With the Object Library Utility, a library can contain object modules with long names, short names, or writable static data. The Object Library Utility creates information, such as the members that contain defined long names, short names, or writable static data. This information is stored in a special member of the library that this chapter refers to as the `C370LIB` or EDCALIAS directory.

Commands are available to add object modules to a library, to delete object modules from a library, or to build the `C370LIB` directory for a library. Use the `DIR` command to build the `C370LIB` directory for a library of object modules. Use the `MAP` command to list the contents of the `C370LIB` directory.

You can create an object library under z/OS batch and TSO.

Creating an Object Library Under z/OS Batch

Under z/OS batch, the following cataloged procedures include an Object Library Utility step:

`EDCLIB`	Maintain an object library
`EDCCLIB`	Compile and maintain an object library. (`C only`)

For more information on the data sets that you use with the Object Library Utility, see "Description of Data Sets Used" on page 533.

To compile the z/OS C program `WALTER.SOURCE(SUB1)` for long names and add to `WALTER.SOURCE.OBJ(SUB1)`, use the following JCL. The Object Library Utility directory for the library, `WALTER.SOURCE.OBJ`, is updated in the process.

```
//COMPILE EXEC EDCCLIB,INFILE='WALTER.SOURCE(SUB1)',CPARM='LO',
//      LIBRARY='WALTER.SOURCE.OBJ',MEMBER='SUB1'
```

If you request a map for the library `WALTER.SOURCE.OBJ`, use the following:

```
//OBJLIB EXEC EDCLIB,OPARM='MAP',LIBRARY='WALTER.SOURCE.OBJ'
```

For z/OS C++, use the `EDCLIB` cataloged procedure. You can specify options for the Object Library Utility step. These options can generate a library directory, add members or delete members of a directory, or generate a map of library members and defined external symbols. This section shows you how to specify these options under z/OS batch.

The following example creates a new `C370LIB` directory. If the directory already exists, it is updated.

```
   //DIRDIR   EXEC EDCLIB,
   //         LIBRARY='LUCKY13.CXX.OBJMATH',
   //         OPARM='DIR'
```

To create a map:

```
   //MAPDIR   EXEC EDCLIB,
   //         LIBRARY='LUCKY13.CXX.OBJMATH',
   //         OPARM='MAP'
```

To add new members to an object library, use the `ADD` option to update the directory. For example, to add a new member named `MA191`:

```
//ADDDIR   EXEC EDCLIB,
//         LIBRARY='LUCKY13.CXX.OBJMATH',
//         OPARM='ADD MA191',
//         OBJECT='DSNAME=LUCKY13.CXX.OBJ(OBJ191),DISP=SHR'
```

To delete a member from an object library, use the `DEL` option to keep the directory up to date. For example, to delete a member named `OLDMEM`:

```
//DELDIR   EXEC EDCLIB,
//         LIBRARY='LUCKY13.CXX.OBJMATH',
//         OPARM='DEL OLDMEM'
```

Creating an Object Library Under TSO

The Object Library Utility has the following syntax:

```
►►──C370LIB──┬─ADD──LIB──(libname(membername))──┬──────────────────┬─┬──────────►◄
             │                                  └─OBJ──(objname)───┘ │
             ├─DEL──LIB──(libname(membername))───────────────────────┤
             ├─MAP──LIB──(libname)───────────────────────────────────┤
             │ ┌──────────────────┐                                  │
             │ └─LIST──(map)──────┘                                  │
             └─DIR──LIB──(libname)───────────────────────────────────┘
```

where:

`ADD`	Adds (or replaces) an object module to an object library. If you use `ADD` to insert an object module to a member of a library that already exists, the previous member is deleted prior to the insert. If the source data set is the same as the target data set, `ADD` does not delete the member, and only updates the Object Library Utility directory.
`DEL`	Deletes an object module from an object library.
`MAP`	Lists the names (entry points) of object library members.
`DIR`	Builds the Object Library Utility-directory member. The Object Library Utility-directory contains the names (entry points) of library members.
`LIB` *(libname(membername))*	Specifies the target data set for the `ADD` and `DEL` functions. The data set name must contain a member specification to indicate which member Object Library Utility should create, replace, or delete.
`OBJ(`*objname*`)`	Specifies the source data set that contains the object module that is to be added to the library. If you do not specify a data set name, Object Library Utility uses the target data set that you specified in `LIB`(libname(membername)) as the source.
`LIB(`*libname*`)`	Specifies the object library for which a map is to be

	produced or for which a Object Library Utility-directory is to be built.
`LIST(`*map*`)`	Specifies the data set that is to contain the library map. If you specified an asterisk (*), the library map is directed to your terminal. If you do not specify a data set name, a name is generated using the library name and the qualifier `MAP`. If `TEST.OBJ` is the input library data set, and your user prefix is `FRANK`, the map's data set name is `FRANK.TEST.OBJ.MAP`.

Under TSO, for z/OS C you can use either the `C370LIB` REXX EXEC or the `CC` REXX EXEC with the parameter `C370LIB`. The `C370LIB` parameter of the `CC` REXX EXEC specifies that, if the object module from the compile is directed to a PDS member, the Object Library Utility-directory is to be updated. This step is the equivalent to a compile and `C370LIB ADD` step. If the `C370LIB` parameter is specified, and the object module is not directed to a member of a PDS, the `C370LIB` parameter is ignored.

Object Library Utility Map

The Object Library Utility produces a listing for a given library when you specify the `MAP` command. The listing contains information on each member of the library.

```
==============================================================================
|                      Object Library Utility Map                            |
|1                                                                           |
|C370LIB:5647-A01 V1 R9 M00 IBM Language Environment 1998/06/22 11:46:49|
==============================================================================
  Library Name: MYUSRID.A.OBJECT                   1997/07/24 15:46:39

*------------------------------------------------------------------------*
*   Member Name: ASMSTUFF                     (D) 1996/02/14 11:46:39 *
* 2                                        569623400   R01 M01 *
*------------------------------------------------------------------------*

     (S) External Name: CSECT1
     (S) External Name: ENTRY1
```

```
*------------------------------------------------------------------------*
*   Member Name: CSTUFF                          (D) 1996/03/17 12:37:39 *
* 2                                         5688216     R32 M00 *
*------------------------------------------------------------------------*

     (L) Function Name: foo
    (WL) External Name: this_int_is_in_writable_static_and_its_name_will
                        _wrap_because_it_is_too_long

*------------------------------------------------------------------------*
*   Member Name: CXXSTUFF                        (D) 1996/01/06 10:21:39 *
*  2                                        5688216     R32 M00 *
*------------------------------------------------------------------------*
   3
   User Comment: This is a user comment in CXXSTUFF
   4
     (L) Function Name: testeh()
     (L) Function Name: f1()
     (L) Function Name: operator++(U&)
    (WL) External Name: i1
    (WL) External Name: i2


=========  E N D   O F   O B J E C T   L I B R A R Y   M A P  ==========
```

1 Map Heading

The heading contains the product number, the library version and release number, and the date and the time the Object Library Utility step began. The name of the library immediately follows the heading. To the right of the library name is the start time of the last Object Library Utility step that updated the Object Library Utility-directory.

2 Member Heading

The product number of the processor that produced the object module follows the name of the object module member. If the `END` record in the object module does not have the processor information in the appropriate format, the `Processor ID` field does not appear.

The `Timestamp` field appears in *yyyy/mm/dd* format. A letter that is enclosed in parentheses indicates the meaning of the timestamp. That is, the Object Library Utility retains a timestamp for each member and selects the time according to the following hierarchy:

(P) indicates that the timestamp is extracted from the object module from the date form or the timestamp form of `#pragma comment`, whichever comes first.

(D) indicates that the timestamp is based on the time that the Object Library Utility `DIR` command was last issued.

(T) indicates that the timestamp is the time that the `ADD` command was issued for the member.

3 User Comments

Displays the user form of comments that `#pragma comment` generated. These comments are extracted from the `END` record. You can add such comments on multiple `END` records and have them displayed in the listing. See the *z/OS C/C++ Language Reference* for more information on the `END` record.

4 Symbol Information

Immediately following `Member Heading` and user comments is a list of the defined objects that the member contains. Each symbol is prefixed by `Type` information that is enclosed in parentheses and either `External Name` or `Function Name`. `Function Name` will appear, provided the object module was compiled with the `LONGNAME` option and the symbol is the name of a defined external function. In all other cases, `External Name` is displayed. The `Type` field gives additional information on each symbol. That is

'L' indicates that the name is a long name. An long name is an external C++ name in an object module or an external non-C++ name in an object module produced by compiling with the `LONGNAME` option.

'S' indicates that the name is a short name. A short name is an external non-C++ name in an object module produced by compiling with the `NOLONGNAME` option. Such a name is up to 8 characters long and single case.

'W' indicates that this is a writable static object. If it is not present, then this is not a writable static object.

Note: `WL` indicates that the symbol is both an L-name and in writable static.

Chapter 16. DLL Rename Utility

This chapter describes the DLL Rename utility, which is part of z/OS Language Environment. You can use the DLL Rename utility to package and redistribute DLLs with your application.

As of OS/390 Version 1 Release 3, the C/C++ IBM Open Class Library component is licensed with the z/OS base and can be used without enablement of the C/C++ features. If your application uses C++ Class Library DLLs for execution on OS/390 Version 1 Release 3 or a later release, you are not required to rename the IBM-supplied DLLs that are shipped with your application.

If your application uses the C++ Class Library DLLs for execution on a system prior to OS/390 Version 1 Release 3, you MUST use the DLL Rename utility to rename the IBM-supplied DLLs, and ship the renamed DLLs with your application.

Note: The DLL Rename utility does not support `GOFF` (`XPLINK`) created DLLs.

With the DLL Rename utility, you can modify an executable application or a DLL to change the names of any DLLs that are loaded at execution time. The DLL Rename utility also provides a report which you can use to understand the DLL dependencies of your application.

Note: This utility does not change the names of variables or functions that are exported by the DLL or imported by your application.

You can use the DLL Rename utility under z/OS batch, TSO, and z/OS UNIX System Services. CICS and IMS do not support it.

Note: If you want to use the DLL Rename Utility, do not specify the Linkage editor option NE when you link-edit the DLL Rename Utility load module. This option removes the information the DLL Rename Utility requires to rename the DLL.

For information on building and using DLLs, see “Using DLLs” on page 470, and the *z/OS C/C++ Programming Guide*.

DLL Redistribution Scenario

Here is an example of a DLL redistribution situation. This example only applies if the application is intended for a release of OS/390 prior to OS/390 Version 1 Release 3.

Your C++ application is targetted to run on OS/390 Version 1 Release 2 C++, and references the `iostream` class library. You do not want your customers to license the C/C++ feature of OS/390 in order to access `IOSTREAM DLL` through the C++ Class Library DLLs. The following steps outline the process for renaming the IBM-supplied `IOSTREAM DLL` so that you can repackage it with your product. For details on the exact steps see “Using the DLL Rename Utility under z/OS Batch” on page 420 or “Using the DLL Rename Utility under TSO” on page 421.

1. Copy the DLL member `IOSTREAM` from the IBM-supplied library to your product library by using the `IEBCOPY` utility with the `COPYMOD` command. You should retain the original version of `IOSTREAM DLL`, especially if other applications use it.

2. Run the DLL Rename utility and rename references to IOSTREAM to a new name, PAHZIOST, so that your program will reference the new name. The name PAHZIOST is an example, you can use any valid PDS member name or a member in a PDSE built using OS/390 Version 2 Release 4 of the Binder.

   ```
   //* Assumption is that PAH are the
   characters used for your product
   //QUERY  EXEC PGM=EDCDLLRN
   //SYSIN  DD *
      'userid.product.load(PAHPGM1)'
      'userid.product.load(IOSTREAM)'   IOSTREAM=PAHZIOST
   /*
   ```

   When the DLL Rename utility has completed, you will notice that member IOSTREAM has been renamed to PAHZIOST, and all references to IOSTREAM in your application are changed to PAHZIOST. You can verify this by running the DLL rename utility again without any rename cards.

   Note: If you want to be able to rebuild your application in the future, and you are required to rename IBM-supplied DLLs, you should copy the IOSTREAM definition side-deck into a private library, and change the name of the DLL on the IMPORT cards from IOSTREAM to PAHZIOST. You can then rebuild your application and bind it with the new definition side-deck. Otherwise, you will have to run the DLL Rename utility each time you rebuild.
3. You should also copy the members ICLBMSGT, CLB3MSGE, and CLB3MSGK from the PDS that contains the IBM-supplied class library DLLs. These members are used to display error messages in the event of failures in the class library code. You should rename these members to new names starting with PAHZ with an alias to the old name. For example:
   - ICLBMSGT - determines which error message member should be loaded based on the language level (LANGLVL) run-time option. Rename it to PAHZMSGT with an alias to ICLBMSGT
   - ICLBMSGE - English error messages, rename it to PAHZMSGE with an alias to ICLBMSGE
   - ICLBMSGK - Kanji error messages, rename it to PAHZMSGK with an alias to ICLBMSGK
4. Ship your product, renamed class library DLLs, and error messages load modules to your customers.

Inputs and Outputs

Input to the DLL Rename utility is a set of one or more programs or DLLs in either of the following:

- Any PDS
- A PDSE compiled with z/OS C/C++ compiler and bound with the binder.

Note: The DLL Rename utility does not support PDSE members built using the z/OS Language Environment Prelinker.

The modules can be applications that call DLLs or DLL modules themselves. Your applications and all the DLLs referenced (either by the application or by another DLL) can be all modified in a single step. Specify a DLL if it may call other DLLs. If the DLL does not reference any DLLs being renamed and is not itself being renamed, no modifications are performed.

Note: If the DLL being renamed is also specified as an input module, the DLL Rename utility attempts to rename the module itself. Copy the DLL first if it may be used by other applications using the old name.

Restriction

The input and output load-library modules must be PDS or PDSE members with the following attributes:

```
RECFM=U, 256<=BLKSIZE<=32760
```

If you run this utility under z/OS batch, input comes from the `SYSIN` data set. All the specified data gets passed to the DLL Rename utility including any sequence numbers resulting from using the ISPF editor. You must ensure that there are no sequence numbers in columns 73 to 80 in the data passed to the DLL Rename utility, otherwise it will fail.

If you run this utility interactively with TSO, the output goes to your terminal. If you use z/OS batch or TSO batch, output goes to the first of these data sets for which there is a definition:
- `DD:SYSPRINT`
- `DD:SYSTERM`
- `DD:SYSERR`

If you have not defined any of these data sets, output goes to `SYSOUT = *`.

When you specify rename statements, the old DLL name and the new DLL name must be different. The DLL names must be 8 characters or less in length. You must ensure that the name is a valid name for a load library.

The DLL Rename utility generates a report. For each program or DLL, it shows a list of DLLs that may be loaded or referenced. This information may help you understand the DLL dependencies of your application. The report contains the following:
- The fully qualified name of the input
- A list of DLLs that the module imports
- Any renaming of those DLLs that was performed

The following are examples of the DLL Rename utility reports:

```
DLLRNAME Report:                      1997/06/20  15:20:00

USERID.PROJECT.LOAD(PAHZAPP1)
  The following is a list of DLLs that are imported:

     IOSTREAM
```

Figure 52. Example of Output from DLLRNAME Utility - Query Only

Note: For any input module that does not reference a DLL, the report lists the module name, but does not list any information about it.

The DLL Rename utility also provides a report when it successfully renames any DLLs.

```
DLLRNAME Report:                     1997/06/20  15:45:00

USERID.PROJECT.LOAD(PAHZAPP1)
  The following is a list of DLLs that are imported:

     PAHZIOST which was renamed from IOSTREAM
```

Figure 53. Example of Output from DLLRNAME Utility

Using the DLL Rename Utility under z/OS Batch

The following is an example of JCL that you can use to rename a DLL under z/OS batch.:

```
//RENAME EXEC EDCDLLRN,GOPARM='LEopts / options'
//SYSIN DD *
     modname1 modname2
     modname3 modname4   oldname=newname
/*
```

where:

EDCDLLRN	is an IBM-supplied procedure that runs the DLL rename utility. The procedure is shipped in the data set CEE.SCEEPROC.
LEopts	refers to z/OS Language Environment runtime options. If you want to receive messages in Kanji, use the NATLANG option. For detailed information about z/OS Language Environment runtime options, see the *z/OS Language Environment Programming Reference*.
options	you can enter valid options in uppercase or lowercase. The following are valid options:

NOREPORT	Does not generate output, unless you are performing a query.
FORCE	If the *newname* specified for a DLL is the same as an existing DLL member name, the renamed DLL erases and replaces the existing DLL.

modname	is an existing program or DLL. You can enter more than one value for modname. The modules can be fully qualified or can assume a high-level qualifier of the current user prefix.
oldname	is the member name of the existing DLL being referenced.
newname	is the new DLL member name.

The following list shows the order for determining the default output data set name:
- DD:SYSPRINT, if defined
- DD:SYSTERM, if defined
- DD:SYSERR, if defined
- SYSOUT=* appended to the JOB log.

You can use an input file instead of specifying it in instream JCL. The input file must contain the module names for each application or DLL and the corresponding oldname=newname strings. The input file must also be assigned to DD SYSIN.

Notes:

1. If rename statements `oldname=newname` are not specified in the input, DLL Rename utility queries the input `modnames` and lists the DLLs that they load.
2. If you are renaming a DLL that is shared by many applications, you should copy the DLL (by using the COPYMOD command of IEBCOPY) to preserve the old DLL and create the new DLL.
3. For the `SYSIN DD *` statement, ensure that there are no sequence numbers in columns 73 through to 80 that is passed as input to DLL Rename utility, otherwise it will fail.

Example of Renaming a DLL under z/OS Batch

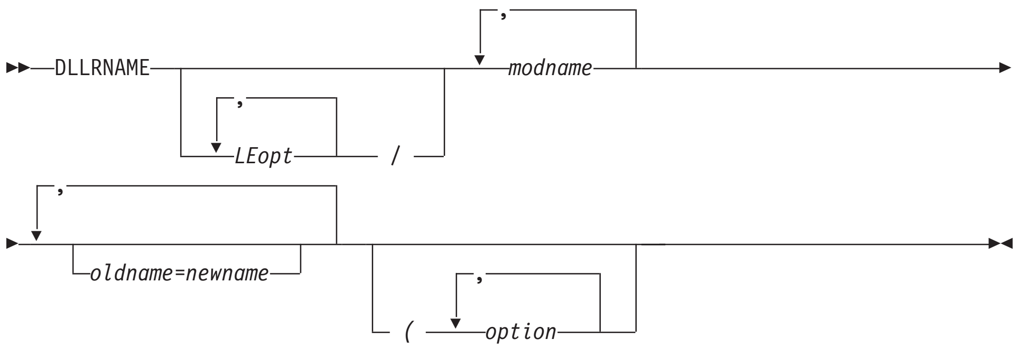
To rename the DLL described in "DLL Redistribution Scenario" on page 417,

1. Query your application to find all imported DLLs
```
//QUERY   EXEC PGM=EDCDLLRN
//SYSIN   DD   *
  'userid.product.LOAD(PAHPGM1)'
/*
```
2. Run the DLL Rename utility to rename the class library DLL
```
//RENAME  EXEC PGM=EDCDLLRN
//SYSIN   DD   *
 'userid.product.load(PAHPGM1)' IOSTREAM=PAHZIOST
 /*
```
3. Ship `PAHPGM1` to your customers with the `PAHZIOST` DLL.

Using the DLL Rename Utility under TSO

The following is the syntax diagram for specifying all parameters directly with the DLL Rename utility

Specifying DLLRNAME Parameters Directly



where:

`LEopt`	refers to z/OS Language Environment runtime options. If you want to receive messages in Kanji, use the `NATLANG` option. For detailed information about z/OS Language Environment runtime options, see the *z/OS Language Environment Programming Reference*.
modname	is an existing program or DLL. You can enter more than one value for modname. The modules can be fully qualified or can assume a high-level qualifier of the current user prefix.
oldname	is the member name of the existing DLL being referenced.

newname is the new DLL member name.

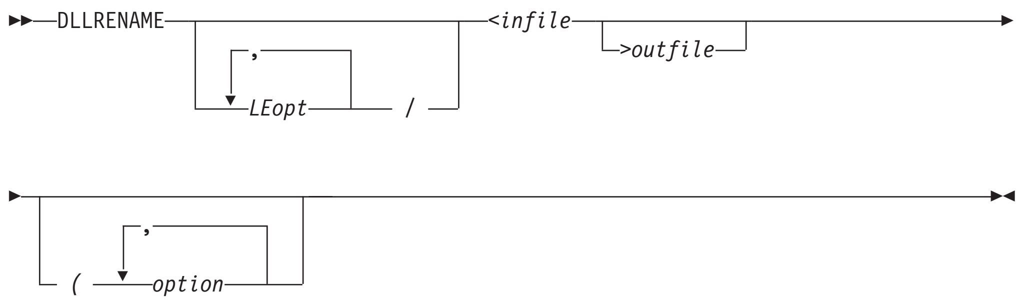
options you can enter valid options in uppercase or lowercase. The following are valid options:

`NOREPORT` Does not generate output, unless you are performing a query.

`FORCE` If the *newname* specified for a DLL is the same as an existing DLL member name, the renamed DLL erases and replaces the existing DLL.

Specifying DLLRNAME Parameters Using an Input File

The following is the syntax diagram for using an input file to provide parameters to the DLL Rename utility under TSO.



where:

`LEopt` refers to z/OS Language Environment runtime options. If you want to receive messages in Kanji, use the `NATLANG` option. For detailed information about z/OS Language Environment runtime options, see the *z/OS Language Environment Programming Reference*.

infile the name of the file that supplies input parameters to the DLL Rename utility. It contains the module name for each application or DLL and the corresponding `oldname=newname` strings.

If you do not specify an input file, the default is `SYSIN`.

outfile the file name for the output from the DLL Rename utility. The following list shows the order for determining the default outfile under TSO batch:
- `DD:SYSPRINT`, if defined
- `DD:SYSTERM`, if defined
- `DD:SYSERR`, if defined
- `SYSOUT=*` appended to the JOB log

Under TSO interactive, the default outfile destination is the terminal.

option You can enter valid options in uppercase or lowercase. These are the valid options:

`NOREPORT` Does not generate output, unless you are performing a query.

`FORCE` If the *newname* specified for a DLL is the same as an existing DLL member name, the renamed DLL erases and replaces the existing DLL.

Note: If there are no `oldname=newname` strings in the input, DLLRNAME queries the input `modname`s and lists the DLLs that they load.

The z/OS Language Environment runtime load library and the load library that contains DLLRNAME must be allocated to the `STEPLIB` DD name. This data set is called `CEE.SCEERUN`.

If you have not allocated the output load-library module, the data set is allocated with the attributes of the input load-library module.

Example of Renaming a DLL under TSO

To rename the DLL described in "DLL Redistribution Scenario" on page 417, run the Utility:

```
DLLRNAME 'userid.product.load(PAHPGM1)' IOSTREAM=PAHZIOST
```

Note: If you receive an error, run DLLRNAME as a query (without any *oldname=newname* parameters) to see if any DLLs were renamed.

Chapter 17. Filter Utility

This chapter describes how to use the CXXFILT utility to convert mangled names to demangled names.

When z/OS C++ compiles a program, it has the ability to encode function names. It also has the ability to encode other identifiers to include type and scoping information. This encoding process is called *mangling*. Mangled names ensure type-safe linking.

Use the CXXFILT utility to convert these mangled names to demangled names. The utility copies the characters from either a given file or from standard input, to standard output. It replaces all mangled names with their corresponding demangled names.

The CXXFILT utility demangles any of the following classes of mangled names when the appropriate options are specified.

regular names
: Names that appear within the context of a function name or a member variable. For example, the mangled name `__ls__7ostreamFPCc` is demangled as `ostream::operator<<(const char*)`.

class names
: Includes stand-alone class names that do not appear within the context of a function name or a member variable. For example, the stand-alone class name `Q2_1X1Y` is demangled as `X::Y`.

special names
: Special compiler-generated class objects. For example, the compiler-generated symbol name `__vft1X` is demangled as `X::virtual-fn-table-ptr`.

```
►►──CXXFILT──┬──────────┬──┬──────────────────────────────┬──────────────────►◄
             └─filename─┘  │      ┌──────┬─────────┐      │
                           │      │      └─,─┘     │      │
                           │      ▼                │      │
                           └─ (───┤ options ├──────┴──────┘
```

options:

```
   ┌─NOSYMMAP─┐  ┌─NOSIDEBYSIDE─┐  ┌─NOWIDTH──────┐  ┌─NOREGULARNAME─┐
├──┼──────────┼──┼──────────────┼──┼──────────────┼──┼───────────────┼──────────►
   └─SYMMAP───┘  └─SIDEBYSIDE───┘  └─WIDTH(width)─┘  └─REGULARNAME───┘

   ┌─NOCLASSNAME─┐  ┌─NOSPECIALNAME─┐
►──┼─────────────┼──┼───────────────┼──────────────────────────────────────┤
   └─CLASSNAME───┘  └─SPECIALNAME───┘
```

The *filename* refers to the files that contain the mangled names to be demangled. You may specify more than one file name, which can be a sequential file, or a PDS member. If you specify no file name, CXXFILT reads from `stdin`.

The following section describes the options that you can use with the CXXFILT utility.

CXXFILT Options

You can use the following options with CXXFILT.

SYMMAP | NOSYMMAP

Default: NOSYMMAP

Produces a symbol map on standard output. This map contains a list of the mangled names and their corresponding demangled names. The map only displays the first 40 bytes of each demangled name; it truncates the rest. Mangled names are not truncated.

If an input mangled name does not have a demangled version, the symbol mapping does not display it.

The symbol mapping is displayed after the end of the input stream is encountered, and after CXXFILT terminates.

SIDEBYSIDE | NOSIDEBYSIDE

Default: NOSIDEBYSIDE

Each mangled name that is encountered in the input stream is displayed beside its corresponding demangled name. If you do not specify this option, then only the demangled names are printed. In either case, trailing characters in the input name that are not part of a mangled name appear next to the demangled name. For example, if an extraneous xxxx is input with the mangled name pr__3FOOF, then the SIDEBYSIDE option would produce this result:

```
FOO::pr()       pr__3FOOFvxxxx
```

WIDTH(width) | NOWIDTH

Default: NOWIDTH

Prints demangled names in fields, *width* characters wide. If the name is shorter than *width*, it is padded on the right with blanks; if longer, it is truncated to *width*. The value of *width* must be greater than 0. If *width* is greater than the record width, then the output is wrapped.

REGULARNAME | NOREGULARNAME

Default: REGULARNAME

This option demangles regular names such as pr__3FOOFv.

The mangled name that is supplied to CXXFILT is treated as a regular name by default. Specifying the NOREGULARNAME option will turn the default off. For example, specifying the CLASSNAME option without the NOREGULARNAME option will cause CXXFILT to treat the mangled name as either a regular name or stand-alone class name.

CLASSNAME | NOCLASSNAME

Default: NOCLASSNAME

This option demangles stand—alone class names such as Q2_1X1Y.

To request that the mangled names be treated as stand-alone class names only, and never as a regular name, use both CLASSNAME and NOREGULARNAME.

SPECIALNAME | NOSPECIALNAME

Default: NOSPECIALNAME

Demangles special names, such as compiler-generated symbol names, for example __vft1X.

To request that the mangled names be treated as special names only, and never as regular names, use CXXFILT (SPECIALNAME NOREGULARNAME.

Unknown Type of Name

If you cannot specify the type of name, use CXXFILT (SPECIALNAME CLASSNAME. This causes CXXFILT to attempt to demangle the name in the following order:

1. Regular name
2. Stand-alone class name
3. Special name

Under z/OS Batch

The CXXFILT utility accepts input by two methods: from stdin or from a file.

The following example uses the CXXFILT cataloged procedure, from data set CBC.SCBCPRC. CXXFILT reads from stdin (sysin), treats mangled names as regular names, produces a symbol mapping, and uses a field width 15 characters. The JCL follows:

```
//RUN EXEC CXXFILT,CXXPARM='(SYMMAP WIDTH(15)'
⋮
//SYSIN DD *
pr__3FOOFvxxxx
__ls__7ostreamFPCc
__vft1X
/*
```

The output is:

```
FOO::pr()      xxxx
ostream::operator<<(const char*)
__vft1X


C++ Symbol Mapping

demangled                                  mangled
---------                                  -------
FOO::pr()                                  pr__3FOOFv
ostream::operator<<(const char*)           __ls__7ostreamFPCs
```

Notes:

1. Because the trailing characters xxxx in the input name pr__3FOOFvxxxx are not part of a valid mangled name, and the SIDEBYSIDE option is not on, the trailing characters are not demangled.

Note: In the symbol mappings, the trailing characters xxxx are *not* displayed.

2. The __vft1X input is not demangled and does not appear in the symbol mapping because it is a special name, and the SPECIALNAME option was not specified.

The second method of giving input to CXXFILT is to supply it in one or more files. Fixed and variable file record formats are supported. Each line of a file can have one or more names separated by space. In the example below, mangled names are treated either as regular names or as special names (the special names are compiler-generated symbol names). Demangled names are printed in fields 35 characters wide, and output is in side-by-side format.

The FILE1 contains the following two mangled names:

```
pr__3FOOFv
__vft1X
```

You can use the following JCL:

```
//RUN EXEC CXXFILT,CXXPARM='FILE1 (SPECIALNAME WIDTH(35) SIDEBYSIDE'
```

The CXXFILT utility terminates when it reads the end-of-file.

Under TSO

The CXXFILT utility accepts input by two methods: from `stdin` or from a file.

With the first method, enter names after invoking CXXFILT. You can specify one or more names on one or more lines. The output is displayed after you press Enter. Names that are successfully demangled, as well as those which are not demangled, are displayed in the same order as they were entered. To indicate end of input, enter `/*`.

In the following example, CXXFILT treats mangled names as regular names, produces a symbol mapping, and uses a field width 15 characters wide.

```
user>  CXXFILT (SYMMAP WIDTH(15)
user>  pr__3FOOFvxxxx
reply< FOO::pr()      xxxx
user>  __ls__7ostreamFPCc
reply> ostream::operator<<(const char*)
user>  __vft1X
reply> __vft1X
user>  /*

reply> C++ Symbol Mapping
reply>
reply> demangled                                  mangled
reply> ---------                                  -------
reply> FOO::pr()                                  pr__3FOOFv
reply> ostream::operator<<(const char*)           __ls__7ostreamFPCs
```

Notes:

1. Because the trailing characters xxxx in the input name pr__3FOOFvxxxx are not part of a valid mangled name, and the SIDEBYSIDE option is not on, the trailing characters are not demangled.

   In the symbol mappings, the trailing characters xxxx are *not* displayed.

2. The __vft1X input is not demangled and does not appear in the symbol mapping because it is a special name, and the SPECIALNAME option was not specified.

3. The symbol mapping is displayed only after /* requests CXXFILT termination

The second method of giving input to CXXFILT is to supply it in one or more files. CXXFILT supports fixed and variable file record formats. Each line of a file can have one or more names separated by space. In the example below, mangled names are treated either as regular names or as special names (the special names are compiler-generated symbol names). Demangled names are printed in fields 35 characters wide, and output is in side-by-side format.

The FILE1 contains the following two mangled names:

```
pr__3FOOFv
__vft1X
```

For example, enter the following command:

```
cxxfilt FILE1 (SPECIALNAME WIDTH(35) SIDEBYSIDE
```

The above command produces the following output:

```
FOO::pr()                          pr__3FOOFv
X::virtual-fn-table-ptr            __vft1X
```

CXXFILT terminates when it reads the end-of-file.

Chapter 18. DSECT Conversion Utility

This chapter describes how to use the DSECT conversion utility, which generates a structure to map an assembler DSECT. This utility is used when a C or C++ program calls or is called by an Assembler program, and a structure is required to map the area passed.

You assemble the source for the assembler DSECT by using the High Level Assembler, and specifying the ADATA option. (See *HLASM Programmer's Guide*, for a description of the ADATA option.) The DSECT utility then reads the SYSADATA file that is produced by the High Level Assembler and produces a file that contains the equivalent C structure structure according to the options specified.

DSECT Utility Options

The options that you can use to control the generation of the C or C++ structure are as follows. You can specify them in uppercase or lowercase, separating them by spaces or commas.

Table 47. DSECT Utility Options, Abbreviations, and IBM-Supplied Defaults

DSECT Utility Option	Abbreviated Name	IBM Supplied Default
SECT[(*name*,...)]	None	SECT(ALL)
BITF0XL | NOBITF0XL	BITF | NOBITF	NOBITF0XL
COMMENT[(*delim*,...)] | NOCOMMENT	COM | NOCOM	COMMENT
DEFSUB | NODEFSUB	DEF | NODEF	DEFSUB
EQUATE[(*suboptions*,...)] | NOEQUATE	EQU | NOEQU	NOEQUATE
HDRSKIP[(*length*)] | NOHDRSKIP	HDR(*length*) | NOHDR	NOHDRSKIP
LOCALE(*name*) | NOLOCALE	LOC | NOLOC	NOLOCALE
INDENT[(*count*)] | NOINDENT	IN(*count*) | NOIN	INDENT(2)
LOWERCASE | NOLOWERCASE	LC | NOLC	LOWERCASE
OPTFILE(*filename*) | NOOPTFILE	OPTF | NOOPTF	NOOPTFILE
PPCOND[(*switch*)] | NOPPCOND	PP(*switch*) | NOPP	NOPPCOND
SEQUENCE | NOSEQUENCE	SEQ | NOSEQ	NOSEQUENCE
UNNAMED | NOUNNAMED	UNN | NOUNN	NOUNNAMED
OUTPUT[(*filename*)]	OUT[(*filename*)]	OUTPUT(DD:EDCDSECT)
RECFM[(*recfm*)]	None	C/C++ Library defaults
LRECL[(*lrecl*)]	None	C/C++ Library defaults
BLKSIZE[(*blksize*)]	None	C/C++ Library defaults

SECT

DEFAULT: SECT(ALL)

The SECT option specifies the section names for which structures are to produced. The section names can be either CSECT or DSECT names. They must exist in the SYSADATA file that is produced by the Assembler. If you do not specify the SECT option or if you specify SECT(ALL), structures are produced for all CSECTs and DSECTs defined in the SYSADATA file, except for private code and unnamed DSECTs.

If the High Level Assembler is run with the BATCH option, only the section names defined within the first program can be specified on the SECT option. If you specify SECT(ALL) (or select it by default), only the sections from the first program are selected.

BITF0XL | NOBITF0XL

DEFAULT: NOBITF0XL

Specify the BITF0XL option when the bit fields are mapped into a flag byte as in the following example:

```
FLAGFLD   DS   F
          ORG  FLAGFLD+0
B1FLG1    DC   0XL(B'10000000')'00'    Definition for bit 0 of 1st byte
B1FLG2    DC   0XL(B'01000000')'00'    Definition for bit 1 of 1st byte
B1FLG3    DC   0XL(B'00100000')'00'    Definition for bit 2 of 1st byte
B1FLG4    DC   0XL(B'00010000')'00'    Definition for bit 3 of 1st byte
B1FLG5    DC   0XL(B'00001000')'00'    Definition for bit 4 of 1st byte
B1FLG6    DC   0XL(B'00000100')'00'    Definition for bit 5 of 1st byte
B1FLG7    DC   0XL(B'00000010')'00'    Definition for bit 6 of 1st byte
B1FLG8    DC   0XL(B'00000001')'00'    Definition for bit 7 of 1st byte
          ORG  FLAGFLD+1
B2FLG1    DC   0XL(B'10000000')'00'    Definition for bit 0 of 2nd byte
B2FLG2    DC   0XL(B'01000000')'00'    Definition for bit 1 of 2nd byte
B2FLG3    DC   0XL(B'00100000')'00'    Definition for bit 2 of 2nd byte
B2FLG4    DC   0XL(B'00010000')'00'    Definition for bit 3 of 2nd byte
```

When the bit fields are mapped as shown in the above example, you can use the following code to test the bit fields:

```
TM   FLAGFLD,L'B1FLG1         Test bit 0 of byte 1
Bx   label                    Branch if set/not set
```

When you specify the BITF0XL option, the length attribute of the following fields provides the mapping for the bits within the flag bytes.

The length attribute of the following fields is used to map the bit fields if a field conforms to the following rules:

- The field does not have a duplication factor of zero.
- The field has a length between 1 and 4 bytes and does not have a bit length.
- The field does not have more than one nominal value.

and the following fields conform to the following rules:

- Has a Type attribute of 'B', 'C', or 'X'.
- Has the same offset as the field (or consecutive fields have overlapping offsets).
- Has a duplication factor of zero.
- Does not have more than one nominal value.
- Has a length attribute between 1 and 255 and does not have a bit length.
- The length attribute maps one bit or consecutive bits. for example, B'10000000' or B'11000000', but not B'10100000'.

The fields must be on consecutive lines and must overlap a named field. If the fields above are used to define the bits for a field, EQU statements that follow the field are not used to define the bit fields.

COMMENT | NOCOMMENT

DEFAULT: COMMENT

The COMMENT option specifies whether the comments on the line where the field is defined will be placed in the structure produced.

If you specify the COMMENT option without a delimiter, the entire comment is placed in the structure.

If you specify a delimiter, any comments that follow the delimiter are skipped and are not placed in the structure. You can remove changes that are flagged with a particular delimiter. The delimiter cannot contain imbedded spaces or commas. The case of the delimiter and the comment text is not significant. You can specify up to 10 delimiters, and they can contain up to 10 characters each.

DEFSUB | NODEFSUB

DEFAULT: DEFSUB

The DEFSUB option specifies whether #define directives will be built for fields that are part of a union or substructure.

If the DEFSUB option is in effect, fields within a substructure or union have the field names prefixed by an underscore. A #define directive is written at the end of the structure to allow the field name to be specified directly as in the following example.

```
struct dsect_name {
  int          field1;
  struct {
    int           _subfld1;
    short int     _subfld2;
    unsigned char _subfld3[4];
    } field2;
  }
#define subfld1  field2._subfld1
#define subfld2  field2._subfld2
#define subfld3  field2._subfld3
```

If the DEFSUB option is in effect, the fields that are prefixed by an underscore may match the name of another field within the structure. No warning is issued.

EQUATE | NOEQUATE

DEFAULT: NOEQUATE

The EQUATE option specifies whether the EQU statements following a field are to be used to define bit fields, to generate #define directives, or are to be ignored.

The suboptions specify how the EQU statement is used. You can specify one or more of the suboptions, separating them by spaces or commas. If you specify more than one suboption, the EQU statements that follow a field are checked to see if they are valid for the first suboption. If so, they are formatted according to that option. Otherwise, the subsequent suboptions are checked to see if they are applicable.

If you specify the EQUATE option without suboptions, EQUATE(BIT) is used. If you specify NOEQUATE (or select it by default), the EQU statements that follow a field are ignored.

You can specify the following suboptions for the EQUATE option:

BIT
: Indicates that the value for an EQU statement is used to define the bits for a field where the field conforms to the following rules:
- The field does not have a duplication factor of zero.
- The field has a length between 1 and 4 bytes and has a bit length that is a multiple of 8.
- The field does not have more than one nominal value.

and the EQU statements that follow the field conform to the following rules:

- The value for the EQU statements that follow the field mask consecutive bits (for example, X’80’ followed by X’40’).
- The value for an EQU statement masks one bit or consecutive bits for example, B’10000000’ or B’11000000’, but not B’10100000’.
- Where the length of the field is greater than 1 byte, the bits for the remaining bytes can be defined by providing the EQU statements for the second byte after the EQU statement for the first byte.
- The value for the EQU statement is not a relocatable value.

When you specify EQUATE(BIT), the EQU statements are converted as in the following example:

```
FLAGFLD  DS   H
FLAG21   EQU  X'80'
FLAG22   EQU  X'40'
FLAG23   EQU  X'20'
FLAG24   EQU  X'10'
FLAG25   EQU  X'08'
FLAG26   EQU  X'04'
FLAG27   EQU  X'02'
FLAG28   EQU  X'01'
FLAG2A   EQU  X'80'
FLAG2B   EQU  X'40'
struct dsect_name {
  unsigned int flag21   : 1,
               flag22   : 1,
               flag23   : 1,
               flag24   : 1,
               flag25   : 1,
               flag26   : 1,
               flag27   : 1,
               flag28   : 1,
               flag2a   : 1,
               flag2b   : 1,
                        : 6;
  }
```

BITL Indicates that the length attribute for an EQU statement is used to define the bits for a field where the field conforms to the following rules:

- The field does not have a duplication factor of zero.
- The field has a length between 1 and 4 bytes and has a bit length that is a multiple of 8.
- The field does not have more than one nominal value.

and the EQU statements that follow the field conform to the following rules:

- The value that is specified for the EQU statement has the same or overlapping offset as the field.
- The length attribute for the EQU statement is between 1 and 255.
- The length attribute for the EQU statement masks one bit or consecutive bits, for example, B’10000000’ or B’11000000’, but not B’10100000’.
- The value for the EQU statement is a relocatable value.

When you specify EQUATE(BITL), the EQU statements are converted as in the following example:

```
BYTEFLD   DS   F
B1FLG1    EQU  BYTEFLD+0,B'10000000'
B1FLG2    EQU  BYTEFLD+0,B'01000000'
B1FLG3    EQU  BYTEFLD+0,B'00100000'
B1FLG4    EQU  BYTEFLD+0,B'00010000'
B1FLG5    EQU  BYTEFLD+0,B'00001000'
B1FLG6    EQU  BYTEFLD+0,B'00000100'
B1FLG7    EQU  BYTEFLD+0,B'00000010'
```

```
B1FLG8    EQU  BYTEFLD+0,B'00000001'
B2FLG1    EQU  BYTEFLD+1,B'10000000'
B2FLG2    EQU  BYTEFLD+1,B'01000000'
B2FLG3    EQU  BYTEFLD+1,B'00100000'
B2FLG4    EQU  BYTEFLD+1,B'00010000'
struct dsect_name {
  unsigned int b1flg1  : 1,
               b1flg2  : 1,
               b1flg3  : 1,
               b1flg4  : 1,
               b1flg5  : 1,
               b1flg6  : 1,
               b1flg7  : 1,
               b1flg8  : 1,
               b2flg1  : 1,
               b2flg2  : 1,
               b2flg3  : 1,
               b2flg4  : 1,
                       : 20;
  }
```

DEF Indicates that the EQU statements following a field are used to build #define directives to define the possible values for a field. The #define directives are placed after the end of the structure. The EQU statements should not specify a relocatable value.

When you specify EQUATE(DEF), the EQU statements are converted as in the following example:

```
FLAGBYTE DS   X
FLAG1    EQU  X'80'
FLAG2    EQU  X'20'
FLAG3    EQU  X'10'
FLAG4    EQU  X'08'
FLAG5    EQU  X'06'
FLAG6    EQU  X'01'
struct dsect_name {
  unsigned char flagbyte;
  }
/* Values for flagbyte field */
#define flag1 0x80
#define flag2 0x20
#define flag3 0x10
#define flag4 0x08
#define flag5 0x06
#define flag6 0x01
```

HDRSKIP | NOHDRSKIP

DEFAULT: NOHDRSKIP

The HDRSKIP option specifies that the fields within the specified number of bytes from the start of the section are to be skipped. Use this option where a section has a header that is not required in the structure produced.

The value that is specified on the HDRSKIP option indicates the number of bytes at the start of the section that are to be skipped. HDRSKIP(0) is equivalent to NOHDRSKIP.

In the following example, if you specify HDRSKIP(8), the first two fields are skipped and only the remaining two fields are built into the structure.

```
SECTNAME DSECT
PREFIX1  DS   CL4
PREFIX2  DS   CL4
FIELD1   DS   CL4
```

```
FIELD2   DS   CL4
struct sectname {
  unsigned char field1[4];
  unsigned char field2[4];
  }
```

If the value specified for the HDRSKIP option is greater than the length of the section, the structure is not be produced for that section.

INDENT | NOINDENT

DEFAULT: INDENT(2)

The INDENT option specifies the number of character positions that the fields, unions, and substructures are indented. Turn off indentation by specifying INDENT(0) or NOINDENT. The maximum value that you can specify for the INDENT option is 32767.

LOCALE | NOLOCALE

The LOCALE(name) specifies the name of a locale to be passed to the setlocale() function. Specifying LOCALE without the *name* parameter is equivalent to passing the NULL string to the setlocale() function.

The structure produced contains the left and right brace, and left and right square bracket, backslash, and number sign which have different code point values for the different code pages. When the LOCALE option is specified, and these characters are written to the output file, the code point from the LC_SYNTAX category for the specified locale is used.

The default is NOLOCALE.

You can abbreviate the option to LOC(*name*) or NOLOC.

LOWERCASE | NOLOWERCASE

DEFAULT: LOWERCASE

The LOWERCASE option specifies whether the field names within the C structure are to be converted to lowercase or left as entered. If you specify LOWERCASE, all the field names are converted to lowercase. If you specify NOLOWERCASE, the field names are built into the structure in the case in which they were entered in the assembler section.

OPTFILE | NOOPTFILE

The OPTFILE(*filename*) option specifies the filename that contains the records that specify the options to be used for processing the sections. The records must be as follows:

- The lines must begin with the SECT option, and only one section name must be specified. The options following determine how the structure is produced for the specified section. The section name must only be specified once.
- The lines may contain the options BITF0XL, COMMENT, DEFSUB, EQUATE, HDRSKIP, INDENT, LOWERCASE, PPCOND, and UNNAMED, separated by spaces or commas. These override the options that are specified on the command line for the section.

The OPTFILE option is ignored if the SECT option is also specified on the command line.

The default is NOOPTFILE.

You can abbreviate the option to OPTF(*filename*) or NOOPTF.

PPCOND | NOPPCOND

DEFAULT: NOPPCOND

The PPCOND option specifies whether preprocessor directives will be built around the structure definition to prevent duplicate definitions.

If you specify PPCOND, the following are built around the structure definition.

```
#ifndef switch
#define switch
   ⋮
  structure definition for section
   ⋮
#endif
```

where *switch* is the switch specified on the PPCOND option or the section name prefixed and suffixed by two underscores. For example, _ _name_ _.

If you specify a switch, the #ifndef and #endif directives are placed around all structures that are produced. If you do not specify a switch, the #ifndef and #endif directives are placed around each structure produced.

SEQUENCE | NOSEQUENCE

DEFAULT: NOSEQUENCE

The SEQUENCE option specifies whether sequence numbers will be placed in columns 73 to 80 of the output record. If you specify the SEQUENCE option, the structure is built into columns 1 to 72 of the output record, and sequence numbers are placed in columns 73 to 80. If you specify NOSEQUENCE (or select it by default), sequence numbers are not generated, and the structure is built within all available columns in the output record.

If the record length for the output file is less than 80 characters, the SEQUENCE option is ignored.

UNNAMED | NOUNNAMED

DEFAULT: NOUNNAMED

The UNNAMED option specifies that names are not generated for the unions and substructures within the main structure.

OUTPUT

DEFAULT: OUTPUT(DD:EDCDSECT)

The structures that are produced are, by default, written to the EDCDSECT DD statement. You can use the OUTPUT option to specify an alternative DD statement or data set name to write the structure. You can specify any valid file name up to 60 characters in length. The file name specified will be passed to fopen() as entered.

RECFM

DEFAULT: C/C++ Library default

The RECFM option specifies the record format for the file to be produced. You can specify up to 10 characters. If it is not specified, the C or C++ library defaults are used.

LRECL

DEFAULT: C/C++ Library default

The LRECL option specifies the logical record length for the file to be produced. The logical record length that is specified must not be greater than 32767. If it is not specified, the C or C++ library defaults will be used.

BLKSIZE

DEFAULT: C/C++ Library default

The BLKSIZE option specifies the block size for the file to be produced. The block size that is specified must not be greater than 32767. If it is not specified, the C or C++ library defaults will be used.

Generation of Structures

The structure is produced as follows according to the options in effect.

- The section name is used as the structure name. A #pragma pack(packed) is generated at the top of the file, and a #pragma pack(reset) is generated at the end to ensure that the structure matches the assembler section. For example:

  ```
  #pragma pack(packed)
  struct dsect_name {
     ⋮
    };
  #pragma pack(reset)
  ```
- Any nonalphanumeric characters in the section or field names are converted to underscores. Duplicate names may be generated when the field names are identical except for the national character. No warning is issued.
- Where fields overlap, a substructure or union is built within the main structure. A substructure is produced where possible. When substructures and unions are built, the DSECT utility generates the structure and union names.
- The substructures and unions within the main structure are indented according to the INDENT option unless the record length is too small to permit any further indentation.
- Fillers are added within the structure when required. The DSECT utility generates a filler name.
- Where there is no direct equivalent for an assembler definition within the C or C++ language, the field is defined as a character field.
- If a field has a duplication factor of zero, but cannot be used as a structure name, the field is defined as though the duplication factor of zero was eliminated.
- Where a line within the assembler input consists of an operand with a duplication factor of zero (for alignment), followed by the field definition, the first operand is skipped. For example:

  ```
  FIELDA    DS  OF,CLB
  ```

  is treated as though the following was specified.

  ```
  FIELDA    DS  CLB
  ```

- When the `COMMENT` option is in effect, the comment on the line that follows the definition of the field is placed in the structure. The comment is placed on the same line as the field definition where possible, or on the following line.

  /* is removed from the beginning of comments, and */ is removed from the end of comments. Any remaining instances of */ in the comment are converted to **.

Each field within the section is converted to a field within the structure, as the following examples show:

- Bit length fields

  If the field has a bit length that is not a multiple of 8, it is converted as follows. Otherwise, it is converted according to the field type.

DS CL.n	`unsigned int   name : n;` where `n` is from 1 to 31.
DS CL.n	`unsigned char  name[x];` where `n` is greater than 32. x will be the number of bytes that are required (that is, the bit length / 8 + 1).
DS 5CL.n	`unsigned char  name[x];` where x will be the number of bytes required (that is, the duplication factor * bit length / 8 + 1).

- Characters

DS C	`unsigned char name;`
DS CL2	`unsigned char name[2];`
DS 4CL2	`unsigned char name[4][2];`

- Graphic Characters

DS G	`wchar_t       name;`
DS GL1	`unsigned char name;`
DS GL2	`wchar_t       name;`
DS GL3	`unsigned char name[3];`
DS 4GL1	`unsigned char name[4];`
DS 4GL2	`wchar_t       name[4];`
DS 4GL3	`unsigned char name[4][3];`

- Hexadecimal Characters

DS X	`unsigned char name;`
DS XL2	`unsigned char name[2];`
DS 4XL2	`unsigned char name[4][2];`

- Binary fields

DS B	`unsigned char  name;`
DS BL2	`unsigned char  name[2];`
DS 4BL2	`unsigned char  name[4][2];`

- Half and Fullword Fixed-point

DS F	`int            name;`
DS H	`short int      name;`
DS FL1 or HL1	`char           name;`
DS FL2 or HL2	`short int      name;`
DS FL3 or HL3	`int            name : 24;`
DS FLn or HLn	`unsigned char  name[n];` where `n` is greater than 4.
DS 4F	`int            name[4];`
DS 4H	`short int      name[4];`
DS 4FL1 or 4HL1	`char           name[4];`
DS 4FL2 or 4HL2	`short int      name[4];`
DS 4FL3 or 4HL3	`unsigned char  name[4][3];`
DS 4FLn or 4HLn	`unsigned char  name[4][n];` where `n` is greater than 4.

- Floating Point

DS E	`float          name;`

DS D	`double         name;`
DS L	`long double    name;`
DS 4E	`float          name[4];`
DS 4D	`double         name[4];`
DS 4L	`long double    name[4];`
DS EL4 or DL4 or LL4	`float          name;`
DS EL8 or DL8 or LL8	`double         name;`
DS LL16	`long double    name;`
DS E, D or L	`unsigned char  name[n];` where `n` is other than 4, 8, or 16.

- Packed Decimal

DS P	`unsigned char  name;`
DS PL2	`unsigned char  name[2];`
DS 4PL2	`unsigned char  name[4][2];`

- Zoned Decimal

DS Z	`unsigned char  name;`
DS ZL2	`unsigned char  name[2];`
DS 4ZL2	`unsigned char  name[4][2];`

- Address

DS A	`void          *name;`
DS AL1	`unsigned char  name;`
DS AL2	`unsigned short name;`
DS AL3	`unsigned int   name : 24;`
DS 4A	`void          *name[4];`
DS 4AL1	`unsigned char  name[4];`
DS 4AL2	`unsigned short name[4];`
DS 4AL3	`unsigned char  name[4][3];`

- Y-type Address

DS Y	`unsigned short name;`
DS YL1	`unsigned char  name;`
DS 4Y	`unsigned short name[4];`
DS 4YL1	`unsigned char  name[4];`

- S-type Address (Base and displacement)

DS S	`unsigned short name;`
DS SL1	`unsigned char  name;`
DS 4S	`unsigned short name[4];`
DS 4SL1	`unsigned char  name[4];`

- External Symbol Address

DS V	`void          *name;`
DS VL3	`unsigned int   name : 24;`
DS 4V	`void          *name[4];`
DS 4VL3	`unsigned char  name[4][3];`

- External Dummy Section Offset

DS Q	`unsigned int   name;`
DS QL1	`unsigned char  name;`
DS QL2	`unsigned short name;`
DS QL3	`unsigned int   name : 24;`
DS 4Q	`unsigned int   name[4];`
DS 4QL1	`unsigned char  name[4];`
DS 4QL2	`unsigned short name[4];`
DS 4QL3	`unsigned char  name[4][3];`

- Channel Command Words

When a CCW, CCW0, or CCW1 assembler instruction is present within the section, a typedef ccw0_t or ccw1_t is defined to map the format of the CCW.

The CCW, CCW0, or CCW1 is built into the structure as follows:

CCW cc,addr,flags,count	`ccw0_t`	`name;`
CCW0 cc,addr,flags,count	`ccw0_t`	`name;`
CCW1 cc,addr,flags,count	`ccw1_t`	`name;`

Under z/OS Batch

You can use the IBM-supplied cataloged procedure EDCDSECT to execute the DSECT utility as in the following example.

```
KNOWN:   - The assembler source name is FRED.SOURCE(TESTASM).
         - The structure is to be written to FRED.INCLUDE(TESTASM).
         - The required DSECT Utility options are EQU(BIT).

USE THE FOLLOWING JCL:
         //DSECT    EXEC PROC=EDCDSECT,
         //         INFILE='FRED.SOURCE(TESTASM)',
         //         OUTFILE='FRED.INCLUDE(TESTASM)',
         //         DPARM='EQU(BIT)'
```

Figure 54. Running the DSECT Utility under z/OS Batch

EDCDSECT invokes the High Level Assembler to assemble the source that is provided with the ADATA option. It then executes the DSECT utility to produce the structure. It writes the structure to the data set that is specified by the OUTFILE parameter, unless the OUTPUT option is also specified. A report that indicates the options in effect and any error messages is written to SYSOUT.

If the assembler source requires macros or copy members from a macro library, include them on the SYSLIB DD for the ASSEMBLY step.

The parameters to the EDCDSECT procedure are:

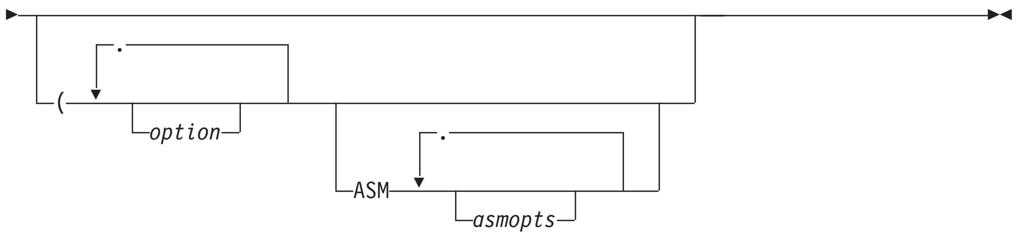
Table 48. EDCDSECT Procedure Parameters

Parameter	Description
INFILE	Input assembler source data set name. This option must be provided.
OUTFILE	The data set name for the file into which the structure is written. If you do not specify an OUTFILE name, a temporary data set is generated.
APARM	High Level Assembler options.
DPARM	DSECT Utility options.

Under TSO

If you have REXX installed, you can run the DSECT utility under TSO by using the CDSECT EXEC. The format of the parameters for the CDSECT EXEC is:

```
►►──CDSECT──infile──outfile──────────────────────────────────────►
```



where *infile* specifies the file name of the assembler source program containing the required section. *outfile* specifies the file that the structure produced is written to, and *options* are any valid DSECT utility options. If you specify ASM, any following options must be High-Level Assembler options. The ADATA is specified by default.

```
  KNOWN:   - The assembler source name is FRED.SOURCE(TESTASM).
           - The structure is to be written to FRED.INCLUDE(TESTASM).
           - The required DSECT Utility options are EQU(BIT).

  USE THE FOLLOWING COMMAND:
           CDSECT 'FRED.SOURCE(TESTASM)' 'FRED.INCLUDE(TESTASM)' ( EQU(BIT)
```

Figure 55. Running the DSECT Utility under TSO

When the CDSECT command is executed, the High Level Assembler is executed with the required options. The DSECT utility is then executed with the specified options. A report of the options and any error messages will be displayed on the terminal.

If the assembler source requires macros or copy members from a macro library, issue the ALLOCATE command to allocate the required macro libraries to the SYSLIB DD statement before issuing the CDSECT command.

Chapter 19. Coded Character Set and Locale Utilities

This chapter describes the coded character set conversion utilities and the `localedef` utility. The coded character set conversion utilities help you to convert a file from one coded character set to another. The `localedef` utility allows you to define the language and cultural conventions that your environment uses.

Coded Character Set Conversion Utilities

These are the Coded Character Set Conversion utilities that you may find useful prior to compiling:

iconv Converts a file from one coded character set encoding to another. You can use iconv to convert C source code before compilation or to convert input files. The standard C library functions such as `iconv_open()`, `iconv()`, and `iconv_close()` are called from the iconv utility to perform coded character set translation. Any program that requires coded character set translation can call these functions. For more information on these functions, refer to the *Z/OS UNIX System Services Command Reference*.

genxlt Generates a translate table that the iconv utility and the iconv family of functions can use to convert coded character sets. It can be used to build code set converters for code pages that are not supplied with z/OS C/C++, or to build code set conversions for existing code pages.

The genxlt utility runs under z/OS batch and TSO. The iconv utility runs under z/OS Batch, TSO, and the z/OS shell. The `iconv_open()`, `iconv()`, and `iconv_close()` functions can be called under these environments and CICS/ESA.

iconv Utility

The iconv utility converts the characters from the input file from one coded character set (code set) definition to another code set definition, and writes the characters to the output file.

The iconv utility uses the `iconv_open()`, `iconv()`, and `iconv_close()` functions to perform the conversion requested. It creates one character in the output file for each character in the input file, and does not perform padding or truncation.

When conversions are performed between single-byte code pages, the output files are the same length as the input files. When conversions are performed between double-byte code pages, the output files may be longer or shorter than the input files because the shift-out and shift-in characters may be added or removed. If you are using the iconv utility under the z/OS shell, see *Z/OS UNIX System Services Command Reference* for details on syntax and uses. For more information on the `iconv()` function, refer to the *z/OS C/C++ Run-Time Library Reference*.

Under z/OS Batch

JCL procedure `EDCICONV` invokes the iconv utility to copy the input data set to the output data set and convert the characters from the input code page to the output code page.

The `EDCICONV` procedure has the following parameters:

INFILE	The data set name for the input data set
OUTFILE	The data set name for the output data set

FROMC	The name of the code set in which the input data is encoded
TOC	The name of the code set to which the output data is to be converted

For example:

```
//ICONV    EXEC PROC=EDCICONV,
//         INFILE='FRED.INFILE',
//         OUTFILE='FRED.OUTFILE',
//         FROMC='IBM-037',
//         TOC='IBM-1047'
```

The output data set must be pre-allocated. If the data set does not exist, `iconv` will fail. An output data set with a fixed record format may only be used if all the records created by the `iconv` utility will have the same record length as the output data set. No padding or truncation is performed. If the output data set has variable length records, the record length must be large enough for the longest record created. Because of these restrictions, when converting to or from a DBCS, the output data set must have variable length records. Otherwise the `iconv` utility will fail.

For more information, refer to the *z/OS C/C++ Programming Guide*.

Under TSO

TSO CLIST `ICONV` invokes the iconv utility to copy the input data set to the output data set and convert the characters from the input code page to the output code page.

The parameters of the `ICONV` CLIST are as follows:

```
►►──ICONV──infile──outfile──FROMCODE(──fromcode──)──TOCODE(──tocode──)───────────►◄
```

Where:

infile	The input data set name.
outfile	The output data set name.
fromcode	The name of the code set in which the input data is encoded.
tocode	The name of the code set to which the output data is to be converted.

For example,

```
ICONV INPUT.FILE OUTPUT.FILE FROMCODE(IBM-037) TOCODE(IBM-1047)
```

The output data set must be pre-allocated. If the data set does not exist, `iconv` will fail. An output data set with a fixed record format may only be used if all the records created by the `iconv` utility will have the same record length as the output data set. No padding or truncation is performed. If the output data set has variable length records, the record length must be large enough for the longest record created. Because of these restrictions, when converting to or from a DBCS, the output data set must have variable length records. Otherwise the `iconv` utility will fail.

For more information, refer to the *z/OS C/C++ Programming Guide*.

Under the z/OS Shell

iconv [–sc] –f *oldset* **–t** *newset [file ...]*

or

iconv –l[–v]

The `iconv` utility converts characters in `file` (or from `stdin` if you do not specify a file) from one code page set to another. It writes the converted text to `stdout`. See *z/OS C/C++ Programming Guide* for more information about the code sets that are supported for this command.

If the input contains a character that is not valid in the source code set, `iconv` replaces it with the byte `0xff` and continues, unless the `–c` option is specified.

If the input contains a character that is not valid in the destination code set, behavior depends on the system's `iconv()` function. See *z/OS C/C++ Run-Time Library Reference* for more information about the character that is used for converting incorrect characters.

See *z/OS C/C++ Programming Guide* for a list of code pages that the z/OS shell supports.

You can use `iconv` to convert singlebyte data or doublebyte data.

Options:

–c	Characters that contain conversion errors are not written to the output. By default, characters not in the source character set are converted to the value `0xff` and written to the output.
–f *oldset*	*oldset* can be either the code set name or a pathname to a file that contains an external code set. Specifies the current code set of the input.
–l	Lists code sets in the internal table. This option is not supported.
–s	Suppresses all error messages about faulty encodings.
–t *newset*	Specifies the destination code set for the output. *newset* can be either the code set name or a pathname to a file that contains an external code set.
–v	Specifies verbose output.

genxlt Utility

The `genxlt` utility creates translation tables, which are used by the iconv_open(), iconv(), and iconv_close() services of the runtime library. These services can be called from both non-XPLINK and XPLINK applications. The non-XPLINK and XPLINK versions have different names. The non-XPLINK version of the GENXLT table should always be generated. If any XPLINK applications will require one of these translation tables, then the XPLINK version should also be generated.

Under TSO, you specify the options on the command line. Under z/OS batch, the options are specified on the EXEC PARM, and may be separated by spaces or commas. If you specify the same option more than once, `genxlt` uses the last specification.

DBCS|NODBCS Specifies whether `genxlt` will convert the DBCS characters within shift-out and shift-in characters. You should only specify the DBCS option when you are converting an EBCDIC code page to a different EBCDIC code page.

If the DBCS option is specified, when a shift-out character is encountered in the input, the characters up to the shift-in character are copied to the output, and not converted. There must be an even number of characters between the shift-out and shift-in characters, and the characters must be valid DBCS characters.

If you specify the NODBCS option, `genxlt` treats all the characters as a single SBCS character, and does not perform a check of DBCS characters.

For more information, refer to the *z/OS C/C++ Programming Guide*.

Under z/OS Batch

JCL procedure EDCGNXLT invokes the genxlt utility to read the character conversion information and produce the conversion table. It invokes the system Linkage Editor to build the load module.

The EDCGNXLT procedure has the following parameters:

INFILE	The data set name for the file that contains the character conversion information.
OUTFILE	The data set name for the output file that is to contain the link-edited conversion table. The non-XPLINK version of this table should have EDCU as the first four characters. The XPLINK version of this table should have CEHU as the first four characters.
GOPT	Options for the genxlt utility.

For example:

```
//GENXLT    EXEC PROC=GENXLT,
//          INFILE='FRED.GENXLT.SOURCE(EDCUEAEY)',
//          OUTFILE='FRED.GENXLT.LOADLIB(EDCUEAEY)',
//          GOPT='DBCS'
```

Under TSO

TSO CLIST GENXLT invokes the genxlt utility to read the character conversion information and produce the conversion table. It then invokes the system Linkage Editor to build the load module.

The general parameters for GENXLT CLIST are as follows:

```
►►──GENXLT──infile──outfile──┬────────┬──────────────────────────►◄
                             ├─DBCS───┤
                             └─NODBCS─┘
```

Where:

infile	The file name for the file that contains the character conversion information.
outfile	The file name for the output file that is to contain the link-edited conversion table. The non-XPLINK version of the table should have EDCU as the first four characters. The XPLINK version of this table should have CEHU as the first four characters.

For example:

```
GENXLT GENXLT.SOURCE(EDCUEAEY) GENXLT.LOADLIB(EDCUEAEY) DBCS
```

localedef Utility

The `localedef` utility creates locale objects, which are used by the setlocale() service of the runtime library. This service can be called from both non-XPLINK and XPLINK applications. The non-XPLINK and XPLINK locale object versions have different names. Also, `localedef` can generate the locale objects into a PDS or PDSE under BATCH or TSO, or into the HFS under the z/OS shell. The non-XPLINK version of the locale object should always be generated. If any XPLINK applications will use the locale then the XPLINK version should also be generated.

A *locale* is a collection of data that defines language and cultural conventions. Locales consist of various categories, that are identified by name, that characterize specific aspects of your cultural environment.

The `localedef` utility generates locales according to the rules that are defined in the locale definition file. A user can create his own customized locale definition file.

The utility reads the locale definition file and produces a locale object that the locale-specific library functions can use. You invoke `localedef` using either a JCL procedure or a TSO CLIST, or by specifying the `localedef` command under z/OS UNIX System Services. To activate a locale during your application's execution, you call the runtime function `setlocale()`.

The options for the `localedef` utility in TSO or z/OS Batch are as follows. Spaces or commas can separate the options. If you specify the same option more than once, `localedef` uses the last option that you specified.

`CHARMAP(`*name*`)`	Specifies the member name of the file that contains the definition of the encoded character set. If you do not specify this option, the `localedef` utility assumes the encoded character set IBM-1047. The name that is specified for the CHARMAP is the member name within a partitioned data set, with the – (dash) sign converted to an @ (at) sign.
FLAG(W|E)	The `FLAG` option controls whether `localedef` issues warning messages. If you specify `FLAG(W)`, `localedef` issues warning and error messages. If you specify `FLAG(E)`, `localedef` issues only the error messages.
BLDERR|NOBLDERR	If you specify the `BLDERR` option, `localedef` generates the locale even if it detects errors. If you specify the `NOBLDERR` option, `localedef` does not generate the locale if it detects an error.

The following sections describe how you can invoke the `localedef` utility. For more information on locale source files, codeset definition files (CHARMAPs), and locale object names, refer to the *z/OS C/C++ Programming Guide*. For information on using the `localedef` utility under z/OS UNIX System Services, refer to the *Z/OS UNIX System Services Command Reference*.

Under z/OS Batch

Note: To build XPLINK optimized locales, use `EDCXLDEF`.

Under z/OS batch, JCL procedure `EDCLDEF` invokes the `localedef` utility. It does the following:

1. Invokes the `EDCLDEF` module to read the locale definition data set and produces the C code to build the locale
2. Invokes the z/OS C/C++ compiler to compile the C source generated
3. Invokes the Linkage Editor to build the locale into a loadable module

The `EDCLDEF` JCL procedure has the following parameters:

`INFILE`	The data set name for the file that contains the locale definition information.
`OUTFILE`	For non-XPLINK, it is the data set name for the output partitioned data set and member that is to contain the link-edited locale object. For XPLINK, it is the data set name for the output PDSE and member that is to contain the bound locale object. The non-XPLINK version of the locale object should have EDC$ or or EDC@ as the first four characters of the member name. The XPLINK version should have CEH$ or CEH@ as the first four characters of the member name.
`LOPT`	The options for the `localedef` utility

For example:

```
//LOCALDEF    EXEC PROC=EDCLDEF,
//            INFILE='FRED.LOCALE.SOURCE(EDC$EUEY)',
//            OUTFILE='FRED.LOCALE.LOADLIB(EDC$EUEM)',
//            LOPT='CHARMAP(IBM-297)'
```

Under z/OS batch, you specify the options on the EXEC PARM and separate them by spaces or commas.

Under TSO

Under TSO, `LOCALDEF` invokes the localedef utility. The name is shortened to 8 characters from LOCALEDEF because of the file naming restrictions. It does the following:

1. Invokes the `EDCLDEF` module to read the locale definition data set and produce the C code to build the locale
2. Invokes the z/OS C/C++ compiler to compile the C source generated
3. Invokes the Linkage Editor to build the locale into a loadable module

The invocation syntax for the `LOCALDEF` REXX EXEC is as follows:

```
►►──LOCALDEF──infile──outfile──┬──────────────────┬──┬────────┬──────────►◄
                               └─LOPT(──loptions)─┘  └─XPLINK─┘
```

where:

infile	The data set name for the data set that contains the locale definition information
outfile	For non-XPLINK, it is the data set name for the output partitioned data set and member that is to contain the link-edited locale object. For XPLINK, it is the data set name for the output PDSE and member that is to contain the bound locale object. The non-XPLINK version of the locale object should have EDC$ or EDC@ as the first

four characters of the member name. The XPLINK version should have CEH$ or CEH@ as the first four characters of the member name.

loptions The options for the `localedef` utility.

XPLINK Indicates that the locale to be built is an XPLINK locale.

In the following example, the input source is `LOCALE.SOURCE(EDC$EUEY)`, the output library is `LOCALE.LOADLIB(EDC$EUEM)` for `en_us.IBM-297`, and options are `CHARMAP(IBM-297)`:

```
LOCALEDEF LOCALE.SOURCE(EDC$EUEY) LOCALE.LOADLIB(EDC$EUEM) LOPT(CHARMAP(IBM-297))
```

Under TSO, you specify the options on the command line.

Under the z/OS Shell

Under z/OS UNIX System Services, use the `localedef` command to invoke the `localedef` utility. The following is the invocation syntax for the `localedef` command:

localedef [**–c**] [**–X**][**–f** *charmap*] [**–i** *sourcefile*] *name*

Options:

–c Creates permanent output even if there were warning messages. Normally, `localedef` does not create permanent output when it has issued warning messages.

–X Causes `localedef` to generate an XPLINK optimized locale object.

–f *charmap* Specifies a *charmap* file that contains a mapping of character symbols and collating element symbols to actual character encodings.

–i *sourcefile* Specifies the file that contains the source definitions. If there is no **–i**, `localedef` reads the source definitions from the standard input.

name Is the target locale. The HFS name for the non-XPLINK version of the locale can be arbitrarily assigned, but by convention the *name* is the same as the descriptive name of the locale. The HFS name for the XPLINK version of the locale is then formed by adding the suffix ".xplink" to the end of the non-XPLINK name. Locale descriptive names are described in the *z/OS C/C++ Programming Guide*. It is permitted to ignore these naming conventions, but you are then required to explicitly supply the full path name of the locale object on each setlocale() invocation. In any event, the non-XPLINK and XPLINK versions of the locale must have distinct names. The convention of .xplink at the end of the XPLINK locales satisfies this requirement. It is common for setlocale() to be given the descriptive locale name using environment variables. When the conventions are followed then the system can find both the non-XPLINK and XPLINK when needed and without having to change the environment variables to fully specify the HFS locale. See the *z/OS C/C++ Programming Guide* for more information about HFS resident locale object names.

z/OS ships two versions of the `localedef` utility:

- One is invocable under z/OS Batch and TSO, and is shipped with the z/OS C/C++ compiler.

- The other is invocable under z/OS UNIX System Services, and is shipped with z/OS UNIX System Services.

For more information, refer to *z/OS UNIX System Services Planning*.

The TSO REXX Exec `LOCALDEF`, included in the C/C++ compiler, is not supported in the z/OS shell environment. In that environment, use the z/OS UNIX System Services `localedef` command instead.

Part 5. z/OS UNIX Utilities

This part contains information about the z/OS UNIX System Services utilities.

Chapter 20. Archive and Make Utilities

This chapter describes the z/OS UNIX System Services `archive` (`ar`) and `make` utilities. There are several other useful z/OS UNIX System Services utilities such as `gencat` and `mkcatdefs`. For information on their syntax and use, refer to the *Z/OS UNIX System Services Command Reference*.

The z/OS Shell and Utilities provide two utilities that you can use to simplify the task of creating and managing z/OS UNIX System Services C/C++ application programs: `ar` and `make`. Use these utilities with the `c89` and `c++` utilities to build application programs into easily updated and maintained executable file.

Archive Libraries

The `ar` utility allows you to create and maintain a library of z/OS C/C++ application object files. You can specify the `c89` and `c++` command strings so that archive libraries are processed during the IPA Link step or binding.

The archive library file, when created for application program object files, has a special symbol table for members that are object files. The symbol table is read to determine which object files should be bound into the application program executable file. The binder processes archive libraries during the binding process. It includes any object file in the specified archive library that it can use to resolve external symbols. Use of this autocall library mechanism is analogous to the use of Object Libraries for object file for data sets. For more information, see "Chapter 15. Object Library Utility" on page 411.

By default, the `c89` and `c++` utilities require that archive libraries end in the suffix `.a`, as in `file.a`. For example; source file `dirsum.c` is in your working directory's subdirectory `src`, and the archive library `symb.a` is in your working directory. To compile `dirsum.c` and resolve external symbols from `symb.a`, and create the executable in `exfils/dirsum` enter:

```
c89 -o exfils/dirsum src/dirsum.c symb.a
```

Creating Archive Libraries

To create the archive library, use the `ar -r` option. For example, to create an archive library that is named `bin/libbrobompgm.a` from your working directory, and add the member `jkeyadd.o` to it, specify:

```
ar -rc ./bin/libbrobompgm.a jkeyadd.o
```

`ar` creates the archive library file `libbrobompgm.a` in the `bin` subdirectory of your HFS working directory. The `-c` option tells `ar` to suppress the message that it normally sends when it creates an archive library file.

For control purposes, when working interactively, you can use the `-v` option to generate a message as each member is added to the archive:

```
ar -rv ./bin/libbrobompgm.a jkeyadd.o
```

To display the object files that are archived in the `bin/libbrobompgm.a` library from your working directory, specify:

```
ar -t ./bin/libbrobompgm.a
```

For a detailed discussion of the `ar` utility, see *Z/OS UNIX System Services Command Reference*.

Creating Makefiles

The `make` utility maintains all the parts of and dependencies for your application program. It uses a *makefile*, which you create, to keep your application parts (listed in it) up to date with one another. If one part changes, `make` updates all the other files that depend on the changed part.

A makefile is a normal HFS text file. You can use any text editor to create and edit the file. It describes the application program files, their locations, dependencies on other files, and rules for building the files into an executable file. When creating a makefile, remember that tabbing of information in the file is important and not all editors support tab characters the same way.

The `make` utility uses `c89` or `c++` to call the z/OS C/C++ compiler, and the binder, to recompile and rebind an updated application program.

See the *z/OS UNIX System Services Programming Tools*, and the *Z/OS UNIX System Services Command Reference* for a detailed discussion of the shell `make` utility and how to best take advantage of its function.

Chapter 21. BPXBATCH Utility

This chapter provides a quick reference for the IBM-supplied `BPXBATCH` program. `BPXBATCH` makes it easy for you to run shell scripts and z/OS C/C++ executable files that reside in hierarchical file system (HFS) files through the z/OS batch environment. If you do most of your work from TSO/E, use `BPXBATCH` to avoid going into the shell to run your scripts and applications.

In addition to using `BPXBATCH`, a user who wants to perform a local spawn without being concerned about environment set-up (that is, without having to set specific environment variables, which could be overwritten if they are also set in the user's profile) can use `BPXBATSL`. `BPXBATSL` provides users with an alternate entry point into `BPXBATCH`, and forces a program to run using a local spawn instead of fork/exec as `BPXBATCH` does. This ultimately allows a program to run faster.

`BPXBATSL` is also useful when the user wants to perform a local spawn of their program, but also needs subsequent child processes to be fork/executed. Formerly, with `BPXBATCH`, this could not be done since `BPXBATCH` and the requested program shared the same environment variables. `BPXBATSL` is an alias of `BPXBATCH`.

BPXBATCH Usage

The `BPXBATCH` program allows you to submit z/OS batch jobs that run shell commands, scripts, or z/OS C/C++ executable files in hierarchical file system (HFS) files from a shell session. You can invoke `BPXBATCH` from a JCL job, from TSO/E (as a command, through a `CALL` command, from a REXX EXEC).

JCL: Use one of the following:
- `EXEC PGM=BPXBATCH,PARM='SH program-name'`
- `EXEC PGM=BPXBATCH,PARM='PGM program-name'`

TSO/E: Use one of the following:
- `BPXBATCH SH program-name`
- `BPXBATCH PGM program-name`

`BPXBATCH` allows you to allocate the z/OS standard files `stdin`, `stdout`, and `stderr` as HFS files for passing input, for shell command processing, and writing output and error messages. If you do allocate standard files, they must be HFS files. If you do not allocate them, `stdin`, `stdout`, and `stderr` default to `/dev/null`. You allocate the standard files by using the options of the data definition keyword `PATH`.

Note: The `BPXBATCH` utility also uses the `STDENV` file to allow you to pass environment variables to to the program that is being invoked. This can be useful when not using the shell, such as when using the `PGM` parameter.

For JCL jobs, specify `PATH` keyword options on DD statements. For example:

```
//jobname JOB ...

//stepname EXEC PGM=BPXBATCH,PARM='PGM program-name parm1 parm2'

//STDIN   DD  PATH='/stdin-file-pathname',PATHOPTS=(ORDONLY)
//STDOUT  DD  PATH='/stdout-file-pathname',PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//            PATHMODE=SIRWXU
//STDERR  DD  PATH='/stderr-file-pathname',PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//           PATHMODE=SIRWXU
⋮
```

You can also allocate the standard files dynamically through use of SVC 99.

For TSO/E, you specify PATH keyword options on the ALLOCATE command. For example:

```
ALLOCATE FILE(STDIN) PATH('/stdin-file-pathname') PATHOPTS(ORDONLY)
ALLOCATE FILE(STDOUT) PATH('/stdout-file-pathname')
         PATHOPTS(OWRONLY,OCREAT,OTRUNC) PATHMODE(SIRWXU)
ALLOCATE FILE(STDERR) PATH('/stderr-file-pathname')
         PATHOPTS(OWRONLY,OCREAT,OTRUNC) PATHMODE(SIRWXU)

BPXBATCH SH program-name
```

You must always allocate stdin as read. You must always allocate stdout and stderr as write.

Parameter

BPXBATCH accepts one parameter string as input. At least one blank character must separate the parts of the parameter string. The total length of the parameter string must not exceed 100 characters. If neither SH nor PGM is specified as part of the parameter string, BPXBATCH assumes that it must start the shell to run the shell script allocated by stdin.

SH | PGM
Specifies whether BPXBATCH is to run a shell script or command or a z/OS C/C++ executable file that is located in an HFS file.

SH
Instructs BPXBATCH to start the shell, and to run shell commands or scripts that are provided from stdin or the specified *program-name*.

Note: If you specify SH with no program-name information, BPXBATCH attempts to run anything read in from stdin.

PGM
Instructs BPXBATCH to run the specified *program-name* as a called program.

If you specify PGM, you must also specify program-name. BPXBATCH creates a process for the program to run in and then calls the program. The HOME and LOGNAME environment variables are set automatically when the program is run, only if they do not exist in the file that is referenced by STDENV. You can use STDENV to set these environment variables, and others.

program-name
Specifies the shell command name or the HFS pathname for the shell script or z/OS C/C++ executable file to be run. In addition, *program-name* can contain option information.

BPXBATCH interprets the program name as case-sensitive.

Note: When PGM and *program-name* are specified and the specified program name does not begin with a slash character (/), BPXBATCH prefixes the user's *initial* working directory information to the program pathname.

Usage Notes

1. BPXBATCH is an alias for the program BPXMBATC, which resides in the SYS1.LINKLIB data set.
2. BPXBATCH must be invoked from a user address space running with a program status word (PSW) key of 8.
3. BPXBATCH does not perform any character translation on the supplied parameter information. You should supply parameter information, including HFS pathnames, using only the POSIX portable character set. For information on the POSIX portable character set, see the *z/OS C/C++ Language Reference*.
4. A program that is run by BPXBATCH cannot use allocations for any files other than stdin, stdout, or stderr.
5. BPXBATCH does not close file descriptors other than 0, 1, or 2. Other file descriptors that are open and not defined as "marked to be closed" remain open when you call BPXBATCH. BPXBATCH runs the specified script or executable file.
6. BPXBATCH uses write-to-operator (WTO) routing code 11 to write error messages to either the JCL job log or your TSO/E terminal. Your TSO/E user profile must specify WTPMSG so that BPXBATCH can display messages to the terminal.

Files

- SYS1.LINKLIB(BPXMBATC) is the BPXBATCH program location.
- The stdin default is /dev/null.
- The stdout default is /dev/null.
- The STDENV default is /dev/null.
- The stderr default is the value of stdout. If all defaults are accepted, stderr is /dev/null.

Part 6. Appendixes

Appendix A. Prelinking and Linking z/OS C/C++ Programs

Instead of using the prelinker and linkage editor, you can use the binder. See "Chapter 12. Binding z/OS C/C++ Programs" on page 353 for more information.

This chapter shows how to prelink and link your programs under z/OS with the z/OS Language Environment. The z/OS Language Environment Prelinker combines the object modules that comprise a C or C++ application into a single object module. The linkage editor then processes this object module and generates a load module that can be retrieved for execution.

You do not need to prelink object modules that:
- do not refer to writable static
- do not contain long names
- do not contain DLL code

You must use the z/OS Language Environment Prelinker before linking your application when any of the following are true:
- Your application contains C++ code.
- Your application contains C code that is compiled with the `RENT`, `LONGNAME`, `DLL`, or `IPA` compiler options.
- Your application is compiled to run under z/OS UNIX System Services.

If you do not need to prelink your application, continue to the information in "Linking an Application" on page 466. For information on creating object libraries in z/OS C++, refer to "Chapter 15. Object Library Utility" on page 411. For information on prelinking and linking object modules under z/OS UNIX System Services, refer to "Prelinking and Link-Editing under the z/OS Shell" on page 491.

Restrictions on Using the Prelinker

You cannot use the prelinker if you specified either the `XPLINK` or `GOFF` compiler option when you compiled.

Prelinking an Application

To prelink multiple object modules and then link with a load module, you must run the multiple object modules through the prelinker and add the load module in the link step. For example, when prelinking and linking a CICS program.

You must prelink together all components that require prelinking prior to linking. For example, `LINK(PRELINK(XOBJ1,XOBJ2))` and `LINK(PRELINK(XOBJ1,XOBJ2),OBJ3)` are valid but `LINK(PRELINK(XOBJ1), PRELINK(XOBJ2))` is not. The prelinker only handles a subset of what the linker handles, in particular, it does not understand load modules (or program objects).

For object modules with writable static references:
- The prelinker combines writable static initialization information
- The prelinker assigns relative offsets to objects in writable static storage
- The prelinker removes writable static name and relocation information

For object modules that contain long names, the prelinker maps long names to short names on output. Long names are mixed-case external names of up to 1024 characters. Short names are eight character, uppercase external names.

For object modules that contain DLL code (C++ code, or C code that was compiled with the `DLL` compiler option), the prelinker does the following:

- It generates a function descriptor (linkage section) in writable static for each DLL referenced function
- It generates a variable descriptor (linkage section) for each unresolved DLL referenced variable
- It generates an `IMPORT` control statement in the `SYSDEFSD` data set for each exported function and variable
- It generates internal information for the load module that describes which symbols are exported and which symbols are imported from other load modules
- It combines static DLL initialization information

z/OS Language Environment Library functions are not included as part of automatic library calls. This omission can result in warning messages about unresolved references to C library functions or C library objects. These unresolved C library functions or objects will be resolved in a following link-edit step.

For C or C++ object modules from applications that were compiled with the `DLL` compiler option, the prelinker uses longnames to resolve exported and imported symbols. For information on how to create a DLL or an application that uses DLLs, see the *z/OS C/C++ Programming Guide*.

Using DD Statements for the Standard Data Sets - Prelinker

The prelinker always requires three standard data sets. You must define these data sets in DD statements with the ddnames `SYSIN`, `SYSMOD`, and `SYSMSGS`.

You may need five other data sets that are defined by DD statements with the names `STEPLIB`, `SYSLIB`, `SYSDEFSD`, `SYSOUT`, and `SYSPRINT`. For a list of the data sets and their usage see Table 49. For details on the attributes of specific data sets see “Description of Data Sets Used” on page 533.

Table 49. Data Sets Used for Prelinking

ddname	Type	Function
SYSIN	Input	Primary input data, usually the output of the compiler
SYSMSGS	Input	Location of prelinker message file
STEPLIB[2]	Utility Library	Location of prelinker and z/OS Language Environment runtime data sets
SYSLIB	Library	Secondary input
SYSDEFSD[1]	Output	Definition side-deck
SYSOUT	Output	Prelinker Map
SYSMOD	Output	Output data set for the prelinked object module
SYSPRINT	Output	Destination of error messages generated by the prelinker
User-specified[1]	Input	Obtain additional object modules and load modules

Notes:

1 Required output from the prelinker if you are exporting variables or functions.

2 Optional data sets, if the compiler and runtime library are installed in the LPA or ELPA. To save resources and improve compile time, especially in z/OS UNIX System Services, do not unnecessarily specify data sets on the STEPLIB DD name.

Primary Input (SYSIN)

Primary input to the prelinker consists of a sequential data set, a member of a partitioned data set, or an in-line object module. The primary input must consist of one or more separately compiled object modules or prelinker control statements. (See “INCLUDE Control Statement” on page 495.)

If you are prelinking an application that imports symbols from a DLL, you must include the definition side-deck for that DLL in SYSIN. The prelinker uses the definition side-deck to resolve external symbols for functions and variables that are imported by your application. If you call more than one DLL, you need to include a definition side-deck for each.

Prelinker Message File (SYSMSGS)

With this DD statement name you provide the prelinker with the information it needs to generate error messages and the prelinker map.

Prelinker and z/OS Language Environment Library (STEPLIB)

To prelink your program the system must be able to locate the data sets that contain the prelinker and z/OS Language Environment runtime library. The DD statement with the name STEPLIB points to these data sets. If the runtime library is installed in the LPA or ELPA, it is found automatically. Otherwise, SCEERUN must be in the JOBLIB or STEPLIB. For information on the search order, see “Chapter 14. Running a C or C++ Application” on page 401.

Secondary Input (SYSLIB)

Secondary input to the prelinker consists of object modules that are not part of the primary input data set, but are to be included in the output prelinked object module from the *automatic call library*. The automatic call library contains object modules that will be used as secondary input to the prelinker to resolve external symbols left undefined after all the primary input has been processed. Concatenate multiple object module libraries by using the DD statement with the name SYSLIB. For more information on concatenating data sets, see page 292.

Note: SYSLIB data sets that are used as input to the prelinker must be cataloged.

Definition Side-Deck (SYSDEFSD)

The prelinker generates a definition side-deck if you are prelinking an application that exports external symbols for functions and variables (a DLL). You must provide this side-deck to any user of your DLL. The users of the DLL must prelink the side-deck of the DLL with their other object modules.

Listing (SYSOUT)

If you specify the MAP prelinker option, the prelinker writes a map to the SYSOUT data set. This map provides you with warnings, files that are included in input to the prelinker, and names of external symbols.

Output (SYSMOD)

The prelinker produces a single prelinked object module, and stores it in the SYSMOD data set. The linkage editor uses this data set as input.

Prelinker Error Messages (SYSPRINT)

If the prelinker encounters problems in its attempt to prelink your program, it generates error messages and places them in the SYSPRINT data set.

Input to the Prelinker

Input to the prelinker can be:

- One or more object modules (not previously prelinked)

- Prelinker control statements (INCLUDE, LIBRARY ...)
- Object module libraries

The process of resolving or including input from these sources depends on the type of the source and the current input and prelink options.

Unresolved references or undefined writable static objects often result if you give the prelinker input object modules produced with a mixture of inconsistent compiler options. For example, RENT | NORENT, LONGNAME | NOLONGNAME, or DLL options. These options may expose symbol names in different ways in your object file, so that the prelinker may be unable to find the matching definition of a referenced symbol if the definition and the reference are exposed differently.

Primary Input

Primary input to the prelinker consists of a sequential data set (file) that contains one or more separately compiled object modules, possibly with prelinker control statements. Specify the primary input data set through the SYSIN ddname.

Secondary Input

Secondary input to the prelinker consists of object modules that are not part of the primary input data set but are to be included as a result of processing of primary input. Object modules that are brought in because of INCLUDE control statements are secondary input. Object modules brought in as a result of automatic call library (library search) processing of currently unresolved symbols through a LIBRARY control statement or through SYSLIB are also secondary input.

An automatic call library may be in the form of:

- PDS Libraries that contain object modules
- PDSE Libraries that contain object modules
- Archive Libraries that contain object modules (if you used OMVS prelinker option).

Prelinker Output

Writable static references that are not resolved by the prelinker cannot be resolved later. Only the prelinker can be used to resolve writable static. The output object module of the prelinker should not be used as input to another prelink.

Prelinker Map

When you use the MAP prelinker option, the z/OS Language Environment Prelinker produces a Prelinker Map. The default is to generate a listing file. The listing contains several individual sections that are only generated if they are applicable. Unresolved references generate error or warning messages to the prelinker map.

Mapping Long Names to Short Names

You can use the output object module of the prelinker as input to a system linkage editor.

Because system linkage editors accept only short names, the z/OS Language Environment Prelinker maps long names to short names on output. It does not change short names. long names can be up to 1024 characters in length. Truncation of the long names to the 8 character short name limit is therefore not sufficient because name collisions may occur.

The z/OS Language Environment Prelinker maps a given long name to a short name on output according to the following hierarchy:

1. If any occurrence of the long name is a reserved runtime name, or was caused by a `#pragma map` or C `#pragma CSECT` directive, then that same name is chosen for all occurrences of the name. This name must not be changed, even if a `RENAME` control statement for the name exists. For information on the `RENAME` control statement, see "RENAME Control Statement" on page 497.
2. If the long name was found to have a matching short name, the same name is chosen. For example, `DOTOTALS` is coded in both a C (or C++) and an assembler program. This name must not be changed, even if a `RENAME` statement for the name exists. This rule binds the long name to its short name.
3. If a valid `RENAME` statement for the long name is present, then the short name specified on the `RENAME` statement is chosen.
4. If the name corresponds to a Language Environment Library function or library object for which you did not supply a replacement, the name chosen is the truncated, uppercased version of the long name library name (with _ mapped to @).
5. If you specify the prelinker OMVS option and the name corresponds to a POSIX Language Environment Library function for which you did not supply a replacement, the name chosen is the internal Language Environment Library short name.

   This short name is not chosen, if either:
   - A valid `RENAME` statement renames another long name to this short name. For example, the `RENAME` statement `RENAME mybigname PRINTF` would make the library function `printf()` unavailable if `mybigname` is found in input.
   - Another long name is found to have the same name as this short name. For example, explicitly coding and referencing `SPRINTF` in the C or C++ source program would make the library function `sprintf()` unavailable.

   Avoid such practices to ensure that the appropriate Language Environment Library function is chosen.
6. If the `UPCASE` option is specified for a C application, names that are 8 characters or fewer are changed to uppercase, with _ mapped to @. Names that begin with `IBM` or `CEE` will be changed to `IB$`, and `CE$`, respectively. Because of this rule, two different names can map to the same name. You should therefore exercise care when using the `UPCASE` option. The prelinker issues a warning message is issued if it finds a collision, but it still maps the names.
7. If none of the above rules apply, a default mapping is performed. This mapping is the same as the one the compiler option `NOLONGNAME` uses for external names, taking collisions into account. That is, the name is truncated to 8 characters and changed to uppercase (with _ mapped to @). Names that begin with `IBM` or `CEE` will be changed to `IB$` and `CE$`, respectively. If this name is the same as the original name, it is always chosen. This name is also chosen if a name collision does not occur. A name collision occurs if either
   - The short name has already been seen in **any** input; that is, the name is not new.
   - After applying this default mapping, the same name is generated for at least two, previously unmapped, names.

   If a name collision occurs, a unique name is generated for the output name. For example, the name `@ST00033` is generated.

A C application that is compiled with the `NOLONGNAME` compiler option and link-edited, except for collisions, presents the linkage editor with the same names as when the application is compiled with the `LONGNAME` option and prelinked.

See the *z/OS Language Environment Debugging Guide* for a list of error messages that the prelinker returns.

Linking an Application

The linkage editor processes your compiled program (object module) and readies it for loading and execution. The processed object module becomes a load module which is stored in a program library or HFS directory and can be retrieved for execution at any time.

Using DD Statements for Standard Data Sets—Linkage Editor

The linkage editor always requires four standard data sets. You must define these data sets in DD statements with the ddnames `SYSLIN`, `SYSLMOD`, `SYSUT1`, and `SYSPRINT`.

A fifth data set, defined by a DD statement with the name `SYSLIB`, is necessary if you want to use the automatic call library. Table 50 shows the five data set names and their characteristics.

Table 50. Data Sets Used for Linking

ddname	Type	Function
SYSLIN	Input	Primary input data, the output of the prelinker, compiler, or assembler
SYSPRINT	Output	Diagnostic messages Informational messages Module map Cross-reference list
SYSLMOD	Output	Output data set for the linkage editor
SYSUT1	Utility	Temporary workspace
SYSLIB[1]	Library	Secondary input
User-specified	Input	Obtain additional object modules and load modules

Notes:
[1] Required for library runtime routines

Primary Input (SYSLIN)

Primary input to the linkage editor consists of a sequential data set, a member of a partitioned data set, or an in-line object module. The primary input must be composed of one or more separately compiled object modules or linkage control statements. A load module cannot be part of the primary input, although the control statement `INCLUDE` can introduced it. (See "INCLUDE Control Statement" on page 495.)

Listing (SYSPRINT)

The linkage editor generates a listing that includes reference tables that are related to the load modules that it produces. You must define the data set where you want the linkage editor to store its listing in a DD statement with the name `SYSPRINT`.

Output (SYSLMOD)

Output (one or more linked load modules) from the linkage editor is always stored in a partitioned data set that is defined by the DD statement with the name `SYSLMOD`, unless you specify otherwise. This data set is known as a library.

Temporary Workspace (SYSUT1)

The linkage editor requires a data set for use as a temporary workspace. The data set is defined by a DD statement with the name `SYSUT1`. This data set must be on a direct access device.

Secondary Input (SYSLIB)

Secondary input to the linkage editor consists of object modules or load modules that are not part of the primary input data set, but are to be included in the load module from the *automatic call library*. The automatic call library contains load modules or object modules that are to be used as secondary input to the linkage editor to resolve external symbols that remain undefined after all the primary input has been processed.

The call library used as input to the linkage editor or loader can be a system library, a private program library, or a subroutine library.

Input to the Linkage Editor

Input to the linkage editor can be:

- One or more object modules (created through the `DECK` or `OBJECT` compiler options)
- Linkage editor control statements (`NAME` and `ALIAS`) that are generated by the `ALIAS` compiler option
- Previously link-edited load modules that you want to combine into one load module
- z/OS Language Environment library stub routines (`SYSLIB`)
- Other libraries

Primary Input

Primary input to the linkage editor consists of a sequential data set that contains one or more separately compiled object modules, possibly with linkage editor control statements.

Specify the primary input data set with the `SYSLIN` statement. For more information on the data sets that are used with z/OS C/C++, refer to “Description of Data Sets Used” on page 533.

Secondary Input

Secondary input to the linkage editor consists of object modules or load modules that are not part of the primary input data set but are to be included in the load module as the *automatic call library*.

The automatic call library contains object modules to be used as secondary input to the linkage editor to resolve external symbols left undefined after all primary input has been processed.

The automatic call library may be in the form of:

- Libraries that contain object modules, with or without linkage editor control statements
- Libraries that contain load modules
- The Language Environment Library, if any of the library functions are needed to resolve external references.

Secondary input is either all object modules or all load modules, but it cannot contain both types.

Specify the secondary input data sets with a **SYSLIB** statement and, if the data sets are object modules, add the linkage editor LIBRARY and INCLUDE control statements.

Additional Object Modules as Input

You can use the INCLUDE and LIBRARY linkage editor control statements to do the following:

1. Specify additional object modules that you want included in the output load module (INCLUDE statement).
2. Specify additional libraries to be searched for object modules to be included in the load module (LIBRARY statement). This statement has the effect of concatenating any specified member names with the automatic call library.

Linkage editor control statements in the primary input must specify any linkage editor processing beyond the basic processing that is described above.

Output from the Linkage Editor

The output from the linkage editor can be a single load module, or multiple load modules, that are generated by using the linkage editor 's NAME control statement.

For more information on using linkage editor control statements, see *z/OS DFSMS Program Management*.

SYSLMOD and SYSPRINT are the data sets that are used for link-edit output. The output from the linkage editor varies, depending on the options you select, as shown in Table 51.

Table 51. Options for Controlling Link-Edit Output

To Get This Output	Use This Option
A map of the load modules generated by the linkage editor.	MAP
A cross-reference list of data variables	XREF
Informational messages	Default
Diagnostic messages	Default
Listing of the linkage editor control statements	LIST
One or more load modules (which you must assign to a library)	Default

By default, you receive diagnostic and informative messages as the result of link-editing. You can get the other output items by specifying options in the PARM parameter in the EXEC statement in your link-edit JCL.

The load modules that are created are written in the data set that is defined by the SYSLMOD DD statement in your link-edit JCL. All diagnostic output to be listed is written in the data set that is defined by the SYSPRINT DD statement.

Detecting Link-Edit Errors

You receive a listing of diagnostic messages in SYSPRINT. Check the linkage editor map to make sure that all the object and load modules you expected were included.

You can find a description of link-edit messages in *z/OS DFSMS Program Management* .

The instructions for link-edit processing vary, depending on whether you are running under z/OS batch or TSO.

Note: For information on link-editing modules for interlanguage calls, refer to the *z/OS Language Environment Programming Guide*.

Library Routine Considerations

The Language Environment Library consists of one runtime component that contains all Language Environment-enabled languages, such as C, C++, COBOL, and PL/I. For detailed instructions on linking and running z/OS C/C++ programs under z/OS Language Environment, refer to the *z/OS Language Environment Programming Guide*.

The Language Environment Library is *dynamic*. This means that many of the functions, such as library functions, available in z/OS C/C++ are not physically stored as a part of your executable program. Instead, only a small portion of code is stored with your executable program, resulting in a smaller executable module size. This portion of code is known as a stub routine The stub routine represents each required library function. Each of these stub routines has:

- The same name as the library function which it represents.
- Enough code to locate the true library function at run time.

The C stub routines are in the file `CEE.SCEELKED`, which is part of z/OS Language Environment and must be specified as one of the libraries to be searched during autocall.

Link-Editing Multiple Object Modules

z/OS C generates a `CEESTART` CSECT at the beginning of the object module for any source program that contains the function `main()` (and for which the `START` compiler option was specified) or a function for which a `#pragma linkage (`*name*`, FETCHABLE)` preprocessor directive applies. When multiple object modules are link-edited into a single load module, the entry point of the resulting load module is resolved to the external symbol `CEESTART.` Runtime errors occur if the load module entry point is forced to some other symbol by use of the linkage editor `ENTRY` control statement.

If a C `main()` function is link-edited with object modules produced by C, other language processors or by assembler, the module containing the C `main()` must be the first module to receive control. You must also ensure that the entry point of the resulting load module is resolved to the external symbol `CEESTART.` To ensure this, the input to the linkage editor can include the following linkage editor `ENTRY` control statement:

```
ENTRY CEESTART
```

If you are building a DLL, you may need to use the `ENTRY` control statement as described above.

Building DLLs

Note: This section does not describe all of the steps that are required to build a DLL. It only describes the prelink step. For a complete description of how to build DLLs, see *z/OS C/C++ Programming Guide*.

Except for the object modules you require for creating the DLL, you do not require additional object modules. The prelinker automatically creates a definition side-deck that describes the functions and the variables that DLL applications can import.

Note: Although some C applications may need only the linkage editor to link them, all DLLs require either the use of the binder with the DYNAM(DLL) option, or the prelinker before the linkage editor.

When you build a DLL, the prelinker creates a definition side-deck, and associates it with the `SYSDEFSD` ddname. You must provide the generated definition side-deck to all users of the DLL. Any DLL application which implicitly loads the `DLL` must include the definition side-deck when they prelink.

The following is an example of a definition side-deck generated by the prelinker when prelinking a C object module:

```
IMPORT CODE 'BASICIO'   bopen
IMPORT DATA 'BASICIO'   bclose
IMPORT DATA 'BASICIO'   bread
IMPORT DATA 'BASICIO'   bwrite
IMPORT DATA 'BASICIO'   berror
```

You can edit the definition side-deck to remove any functions or variables that you do not want to export. For instance, in the above example, if you do not want to expose function `berror`, remove the control statement `IMPORT DATA 'BASICIO' berror` from the definition side-deck.

Note: You should also provide a header file that contains the prototypes for exported functions and external variable declarations for exported variables.
The following is an example of a definition side-deck generated by the prelinker when prelinking a C++ object module:

```
IMPORT CODE 'TRIANGLE' getarea__8triangleFv
IMPORT CODE 'TRIANGLE' getperim__8triangleFv
IMPORT CODE 'TRIANGLE' __ct__8triangleFv
```

You can edit the definition side-deck to remove any functions and variables that you do not want to export. For instance, in the above example, if you do not want to expose `getperim()`, remove the control statement `IMPORT CODE 'TRIANGLE' getperim__8triangleFv` from the definition side-deck.

The definition side-deck contains mangled names, such as `getarea__8triangleFv`. If you want to know what the original function or variable name was in your source module, look at the compiler listing created. Alternatively, use the `CXXFILT` utility to see both the mangled and demangled names. For more information on the `CXXFILT` utility, see "Chapter 17. Filter Utility" on page 425.

Note: You should also provide users of your DLL with a header file that contains the prototypes for exported functions and `extern` variable declarations for exported variables.

Linking Your Code

When you link your code, ensure that you specify the RENT or REUS(SERIAL) options.

Using DLLs

The prelinker is used to build DLLs that export defined external functions and variables, and to build programs or DLLs that import external functions and variables from other DLLs.

To assign a name to a DLL, use either the DLLNAME() prelinker option, or the NAME control statement. If you do not assign a name, and the data set SYSMOD is a PDS member, the member name is used as the DLL name. Otherwise, the name TEMPNAME is used.

To build a DLL, you need to compile object code that exports external functions or variables, then prelink and link that code into a load module. During the prelink step you need to capture the definition side-deck which is written to the ddname SYSDEFSD. The definition side-deck is a list of IMPORT control statements that correspond to the external functions and variables exported by the DLL.

Include the IMPORT statements at prelink time for any program that imports variables or functions from the DLL.

In the following C example, EXPONLY is a DLL which only exports a single variable year:

```
/* EXPONLY.C */
int year = 2001;      /* exported from this DLL */
```

In the following example, IMPEXP is a DLL that both imports and exports external functions and variables. It imports the external variable year from DLL EXPONLY, and exports external functions next_year and get_year.

```
/* IMPEXP.C */
extern int year;       /* imported from DLL EXPONLY */

void next_year(void) { /* exported from this DLL    */
  ++year;              /* load DLL EXPONLY, modify 'year' in DLL */
}

int get_year(void) {   /* exported from this DLL    */
  return year;         /* get value of 'year' from DLL EXPONLY */
}
```

In the following example, IMPONLY is a program that only imports functions and variables. It imports the variable ’year’ from DLL EXPONLY, and it imports functions next_year and get_year from DLL IMPEXP.

```
/* IMPONLY.C */
#include <stdio.h>
extern int  get_year(void);    /* import from DLL IMPEXP */
extern void next_year(void);   /* import from DLL IMPEXP */
extern int year;               /* import from DLL EXPONLY */
int main(void)
{
  int y;
  next_year();         /* load DLL IMPEXP, call function from DLL */
  y = get_year();      /* call function in DLL IMPEXP */
  if (   y == 2002
      && year == 2002) /* get value of 'year' from DLL EXPONLY */
    printf("pass\n");
  else
    printf("fail\n");
  return 0;
}
```

The following JCL builds the DLLs EXPONLY, IMPEXP, and the program IMPONLY, and then runs IMPONLY:

```
//* ------------------------------------------------------------------
//CEXPONLY  EXEC EDCC,
//  INFILE='USERID.DLL.C(EXPONLY)',
//  OUTFILE='USERID.DLL.OBJECT(EXPONLY),DISP=SHR ',
//  CPARM='LONG RENT EXPORTALL'
//* ------------------------------------------------------------------
//CIMPEXP   EXEC EDCC,
//  INFILE='USERID.DLL.C(IMPEXP)',
//  OUTFILE='USERID.DLL.OBJECT(IMPEXP),DISP=SHR ',
//  CPARM='LONG RENT DLL EXPORTALL'
//* ------------------------------------------------------------------
//CIMPONLY  EXEC EDCC,
//  INFILE='USERID.DLL.C(IMPONLY)',
//  OUTFILE='USERID.DLL.OBJECT(IMPONLY),DISP=SHR ',
//  CPARM='LONG RENT DLL'
//* ------------------------------------------------------------------
//LINK1     EXEC CBCL,PPARM='DLLNAME(EXPONLY)',
//  OUTFILE='USERID.DLL.LOAD(EXPONLY),DISP=SHR '
//PLKED.SYSIN    DD DSN=USERID.DLL.OBJECT(EXPONLY),DISP=SHR
//PLKED.SYSDEFSD DD DSN=USERID.DLL.IMPORTS(EXPONLY),DISP=SHR
//* ------------------------------------------------------------------
//LINK2     EXEC CBCL,PPARM='DLLNAME(IMPEXP)',
//  OUTFILE='USERID.DLL.LOAD(IMPEXP),DISP=SHR '
//PLKED.SYSIN    DD DSN=USERID.DLL.OBJECT(IMPEXP),DISP=SHR
//               DD DSN=USERID.DLL.IMPORTS(EXPONLY),DISP=SHR
//PLKED.SYSDEFSD DD DSN=USERID.DLL.IMPORTS(IMPEXP),DISP=SHR
```

Figure 56. JCL to build DLLs (Part 1 of 2)

```
//* ------------------------------------------------------------------
//LINK3     EXEC CBCL,
//  OUTFILE='USERID.DLL.LOAD(IMPONLY),DISP=SHR '
//PLKED.SYSIN    DD DSN=USERID.DLL.OBJECT(IMPONLY),DISP=SHR
//               DD DSN=USERID.DLL.IMPORTS(EXPONLY),DISP=SHR
//               DD DSN=USERID.DLL.IMPORTS(IMPEXP),DISP=SHR
//* ------------------------------------------------------------------
//GO       EXEC PGM=IMPONLY
//STEPLIB  DD DSN=USERID.DLL.LOAD,DISP=SHR
//         DD DSN=CEE.SCEERUN,DISP=SHR
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
```

Figure 56. JCL to build DLLs (Part 2 of 2)

- Both EXPONLY and IMPEXP are compiled with the option EXPORTALL because they export external functions and variables.
- Both IMPEXP and IMPONLY are compiled with the option DLL because they import functions and variables from other DLLs.
- Step LINK1 generates a definition side-deck USERID.DLL.IMPORTS(EXPONLY) which is a list of external functions and variables that are exported by DLL EXPONLY.
- Step LINK2 uses the definition side-deck that is generated in step LINK1 as part of the prelinker input to import the variable year from DLL EXPONLY.
- Step LINK2 generates a definition side-deck USERID.DLL.IMPORTS(IMPEXP) that is a list of external functions and variables that are exported by DLL IMPEXP.
- Both steps LINK1 and LINK2 use the prelinker DLLNAME option to set the DLL name seen on IMPORT statements generated in the definition side-decks.

- Step LINK3 uses the definition side-decks generated in step LINK1 and LINK2 as part of the prelinker input to import the variable year from DLL EXPONLY and to import the functions get_year and set_year from DLL IMPEXP.
- Step LINK3 does not specify a definition side-deck; program IMPONLY does not export any functions or variables.
- If you explicitly specify link-time parameters, be sure to specify the RENT option. The IBM-supplied cataloged procedure CBCL does this by default.
- The load module name of a DLL must match the DLLNAME seen on the corresponding IMPORT statements.
- Step GO has the program IMPONLY and the DLLs. EXPONLY and IMPEXP in its STEPLIB concatenation so that the DLLs can be dynamically loaded at runtime.

To see which functions and variables are imported or exported use the prelinker map. The following is a portion of the prelinker map from step LINK2:

```
========================================================================
|                          Load Module Map  1                          |
========================================================================

MODULE ID  MODULE NAME

  00001    EXPONLY


========================================================================
|                         Import Symbol Map  2                         |
========================================================================

*TYPE    FILE ID  MODULE ID  NAME

   D      00001     00001    year

*TYPE:  D=imported data  C=imported code


========================================================================
|                         Export Symbol Map  3                         |
========================================================================

*TYPE    FILE ID  NAME

   C      00001   get_year
   C      00001   next_year

*TYPE:  D=exported data  C=exported code
```

1 Load Module Map
This section lists the load modules from which functions and variables are imported. The load module names come from the input IMPORT control statements processed.

2 Import Symbol Map
This section lists the imported functions and variables. The MODULE ID indicates the DLL from which the function or variable is imported. The FILE ID indicates the file in which the IMPORT control statement was processed that resulted in this import.

3 Export Symbol Map
This section lists the external functions and variables which are exported. For each symbol that is listed in this section, an IMPORT control statement is written out to the DDname SYSDEFSD, the definition side-deck.

Prelinking and Linking an Application Under z/OS Batch and TSO

Figure 57 shows the basic prelinking and linking process for your C or C++ application.

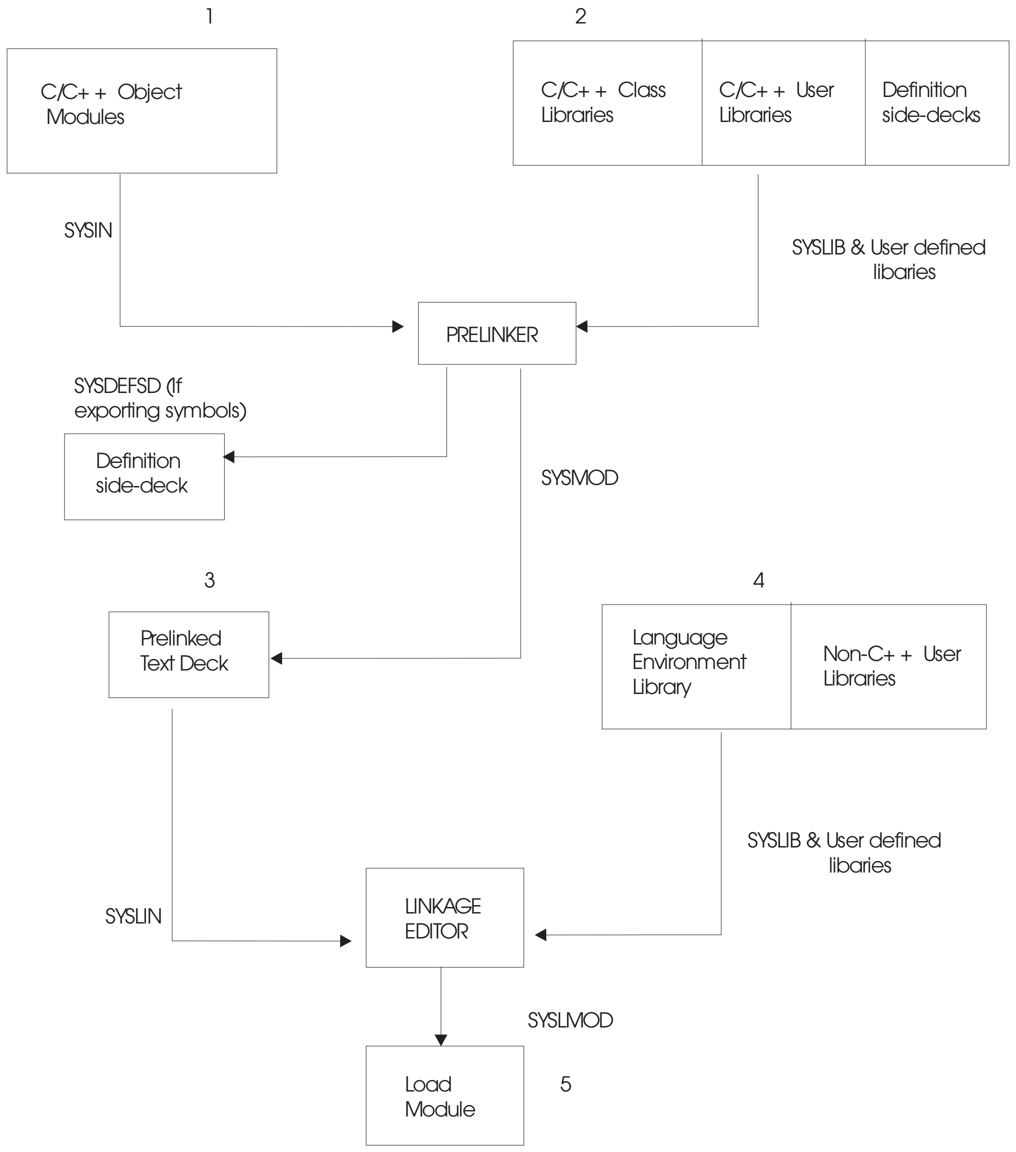


Figure 57. Basic Prelinker and Linkage Editor Processing

The data set `SYSIN`, **1** , that contains your object modules forms the prelinker's primary input.

Note: If you are creating an application that imports symbols from DLLs, you must provide the definition side-deck for each DLL referenced in `SYSIN`.

The prelinker uses its primary input, and its secondary input, **2** , from `SYSLIB` to produce a prelinked object module and, if you are exporting symbols, a definition side-deck. `SYSLIB` points to PDS libraries or PDSE libraries which may contain the following:
- Object modules with long names
- Object modules with writable static references

- C/C++ object module libraries
- DLL definition side-decks

The prelinked output object module is put in SYSMOD. If a definition side-deck is generated, it is put in SYSDEFSD, which is a sequential data set or a PDS member.

The linkage editor takes its primary input from SYSLIN which refers to the prelinked object module data set, **3** . The linkage editor uses the primary input and secondary input, **4** , to produce a load module, **5** . The secondary input consists of non-C++ user defined libraries, and the z/OS Language Environment runtime library (SCEELKED) specified using SYSLIB.

The load module, **5** , is put in the SYSLMOD data set. The load module becomes a permanent member of SYSLMOD. You can be retrieve it at any time to run in the job that created it, or in any other job.

z/OS Language Environment Prelinker Map

When you use the MAP prelinker option, the z/OS Language Environment Prelinker produces a Prelinker Map. The listing contains several individual sections that are only generated if they are applicable.

Consider the following example. The data set USERID.DLL.SOURCE(EXPONLY) contains

```
/* EXPONLY.C */
  int year = 2001;  /* exported from this DLL */
```

After step LINK0 in Figure 59 on page 476, the definition side-deck USERID.DLL.IMPORTS(EXPONLY) contains the record IMPORT DATA 'EXPONLY' year.

The map that is shown in Figure 60 on page 476 was created by compiling the program that is shown in Figure 58. Figure 60 on page 476 is the corresponding Prelinker Map from step LINK1 The linkage editor places the resulting load module in USERID.DLL.LOAD(IMPEXP2).

```
/* IMPEXP2.C */
#pragma variable(this_int_not_in_writable_static, NORENT)
int this_int_not_in_writable_static = 2001;
extern int year;
int this_int_is_in_writable_static = 1900;
int get_year(void) {
  return year;
}
void next_year(void) {
  year++;
}
void Name_Collision_In_First8(void) {
}
void Name_Collision_In_First_Eight(void) {
}
```

Figure 58. z/OS C++ Source File Used for the Example Prelinker Map

```
//*
//COMP0  EXEC  CBCC,CPARM='EXPORTALL',
//  INFILE='USERID.DLL.SOURCE(EXPONLY)',
//  OUTFILE='USERID.DLL.OBJECT(EXPONLY),DISP=SHR'
//LINK0  EXEC  CBCL,PPARM='DLLNAME(EXPONLY) NONCAL MAP',
//  OUTFILE='USERID.DLL.LOAD(EXPONLY),DISP=SHR'
//PLKED.SYSIN     DD  DSN=USERID.DLL.OBJECT(EXPONLY),DISP=SHR
//PLKED.SYSDEFSD  DD  DSN=USERID.DLL.DEFSD(EXPONLY),DISP=SHR
//*
//COMP1  EXEC  CBCC,CPARM='EXPORTALL',
//  INFILE='USERID.DLL.SOURCE(IMPEXP2)',
//  OUTFILE='USERID.DLL.OBJECT(IMPEXP2),DISP=SHR'
//LINK1  EXEC  CBCL,PPARM='DLLNAME(IMPEXP2) NONCAL MAP',
//  OUTFILE='USERID.DLL.LOAD(IMPEXP2),DISP=SHR'
//PLKED.SYSIN     DD  DSN=USERID.DLL.OBJECT(IMPEXP2),DISP=SHR
//                DD  DSN=USERID.DLL.DEFSD(EXPONLY),DISP=SHR
//PLKED.SYSDEFSD  DD  DSN=USERID.DLL.DEFSD(IMPEXP2),DISP=SHR
```

Figure 59. Example of JCL Used to Generate the Example Prelinker Map for a C++ program.

```
==========================================================================
|                          Prelinker Map    1                           |
|                                                                      |
| CPLINK:5647A01 V2 R10 M0 IBM Language Environment 2000/05/17 15:45:56|
 =========================================================================

 Command Options. . . . . : NONCAL   NOMEMORY ER       DUP      MAP
                          : NOOMVS   NOUPCASE


 =========================================================================
|                      Object Resolution Warnings    2                     |
 =========================================================================

 WARNING EDC4015: Unresolved references are detected:
 CEESTART CEESG003 @@TRGLOR
```

Figure 60. Prelinker Map (Part 1 of 3)

```
========================================================================
|                              File Map    3                              |
========================================================================

*ORIGIN  FILE ID  FILE NAME

   P      00001   DD:SYSIN
   A      00002   CEE210.SCEECPP(EDCHSG03)
   IN     00003   *** DESCRIPTORS ***

*ORIGIN:  P=primary input     PI=primary INCLUDE    SI=secondary INCLUDE
          A=automatic call     R=RENAME card         L=C Library
         IN=internal


========================================================================
|                      Writable Static Map    4                         |
========================================================================

  OFFSET    LENGTH  FILE ID  INPUT NAME

       0         4   00001   this_int_is_in_writable_static
       8        10   00003   <year>
      18         4   00001   @STATIC


========================================================================
|                         Load Module Map    5                          |
========================================================================

MODULE ID  MODULE NAME

  00001    EXPONLY


========================================================================
|                        Import Symbol Map    6                         |
========================================================================

*TYPE    FILE ID  MODULE ID  NAME

   D      00001     00001    year

*TYPE:  D=imported data  C=imported code
```

Figure 60. Prelinker Map (Part 2 of 3)

```
========================================================================
|                         Export Symbol Map    7                         |
========================================================================

*TYPE    FILE ID  NAME

   C      00001   get_year()
   C      00001   next_year()
   D      00001   this_int_is_in_writable_static
   C      00001   Name_Collision_In_First_Eight()
   C      00001   Name_Collision_In_First8()

*TYPE:  D=exported data  C=exported code


========================================================================
|               ESD Map of Defined and Long Names    8                 |
========================================================================

                   OUTPUT
*REASON  FILE ID  ESD NAME   INPUT NAME

   P              CEESTART   CEESTART
   D      00001   THIS@INT   this_int_not_in_writable_static
   D      00001   GET@YEAR   get_year()
   D      00001   NEXT@YEA   next_year()
   D      00001   @ST00003   Name_Collision_In_First8()
   D      00001   @ST00002   Name_Collision_In_First_Eight()
   P              CEESG003   CEESG003
   P      00002   CBCSG003   CBCSG003
   P              @@TRGLOR   @@TRGLOR

*REASON: P=#pragma or reserved    S=matches short name    R=RENAME card
         L=C Library              U=UPCASE option         D=Default


============  E N D   O F   P R E - L I N K A G E   M A P  =============
```

Figure 60. Prelinker Map (Part 3 of 3)

The numbers in the following text correspond to the numbers that are shown in the map.

1 Heading
: The heading is always generated. It contains the product number, the library release number, the library version number, and the date and the time the prelink step began. A list of the prelinker options that are in effect for the step follow.

2 Object Resolution Warnings
: This section is generated if objects remained undefined at the end of the prelink step, or the IPA Link step, or if duplicate objects were detected during the step. The names of the applicable objects are listed.

3 File Map
: This section lists the object modules that were included in input. An object module consisting only of `RENAME` control statements, for example, is *not* shown. Also provided in this section are source origin (`FILE NAME`), and identifier (`FILE ID`) information. The object module came from primary input because of:
- an `INCLUDE` control statement in primary or secondary input
- a `RENAME` control statement
- the resolution of long name library references

- the object module was internal and self-generated by the prelink step.

The `FILE ID` may appear in other sections, and is used as a cross reference to the object module. The `FILE NAME` can be one of:
- The data set name and, if applicable, the member name
- The ddname and, if applicable, the member name
- The HFS file name and directory

If you are prelinking an application that imports variables or functions from a DLL, the variable descriptors and function descriptors are defined in a file called `*** DESCRIPTORS ***`. This file has an origin of internal.

4 Writable Static Map

This section is generated if an object module was encountered that contains defined static external data. This area also contains variable descriptors for any imported variables and, if required, function descriptors. This section lists the names of such objects, their lengths, their relative offset within the writable static area, and a `FILE ID` for the file containing the object's definition.

5 Load Module Map

This section is generated if the application imports symbols from other load modules. This section lists the names of the load modules.

6 Import Symbol Map

This section is generated if symbols are imported from other load modules. These otherwise unresolved DLL references are resolved through `IMPORT` control statements. This section lists those symbols. It describes the type of symbol; that is, `D` (variable) or `C` (function). It also lists the file id of the object module containing the corresponding `IMPORT` control statements, the module id of the load module on that control statement, and the symbol name.

A DLL application would generate this section.

7 Export Symbol Map

This section is generated if an object module is encountered that exports symbols. This section lists those symbols. It describes the type of symbol; that is, `D` (variable) or `C` (function). It also lists the file id of the object where the symbol is defined and the symbol name. Only externally defined data objects in writable static or externally defined functions can be exported.

Code that is compiled with the `EXPORTALL` compiler option or code that contains the `#pragma export` directive would generate an object module that exports symbols.

8 ESD Map of Defined and Long Names

This section lists the names of external symbols that are not in writable static. It also shows a mapping of input long names to output short names.

If the object is defined, the `FILE ID` indicates the file that contains the definition. Otherwise, this field is left blank. For any name, the input name and output short name are listed. If the input name is indeed an long name, the rule that is used to map the long name to the short name is applied. If the name is not an long name, this field is left blank.

Note: Although mangled names exist in the object modules, the prelinker's map and messages emit the demangled equivalent, which is like the names seen in the C++ source code.

Processing the Prelinker Automatic Library Call

The following hierarchy is used to resolve a referenced and currently undefined symbol.

- The undefined name is an short name, for example `SNAME`.
  - If the `NONCAL` command option is in effect, the partitioned data sets that are concatenated to `SYSLIB` are searched in order as follows:
    - If the data set contains a `C370LIB`-directory created using the z/OS C/C++ Object Library Utility, and the `C370LIB`-directory shows that a defined symbol by that name exists, the member of the PDS containing that symbol is read.
    - If the data set does not contain a `C370LIB`-directory created using the z/OS C/C++ Object Library Utility and the reference is not to static external data, the member or alias, with the same name as `SNAME` is read.
- The undefined name is an long name.
  - If the `NONCAL` command option is in effect, the partitioned data sets that are concatenated to `SYSLIB` are searched. If the data set contains a `C370LIB`-directory created using the z/OS C/C++ Object Library Utility, and the `C370LIB`-directory shows that a defined symbol by that name exists, the member of the PDS indicated as containing that symbol is read.

For more information about the z/OS C/C++ Object Library Utility, see “Chapter 15. Object Library Utility” on page 411.

References to Currently Undefined Symbols (External References)

If the symbol is undefined after the prelink step, and is not a writable static symbol, it may be subsequently defined during the link step. However, the definition must be exactly the same as the output ESD name. For more information, see the Figure 60 on page 476.

If you are writing a C application, and the symbol is an long name that was not resolved by automatic library call and for which a `RENAME` statement with the `SEARCH` option exists, the symbol is resolved under the short name on the `RENAME` statement by automatic library call.

See “RENAME Control Statement” on page 497 for a complete description of the `RENAME` control statement.

Unresolved requests generate error or warning messages to the prelinker map.

Prelinking and Linking Under z/OS Batch

Using IBM-Supplied Cataloged Procedures

The IBM-supplied catalog procedures and REXX EXECs use the DLL versions of the IBM-supplied class libraries by default. That is, the IBM-supplied Class Libraries definition side-deck data set, `SCLBSID`, is included in the `SYSIN` concatenation.

If you are *statically* linking the relevant class library object code, you must override the `PLKED.SYSLIB` concatenation to include the `SCLBCPP` data set. You must also override the `PLKED.SYSIN` concatenation to exclude the `SCLBSID` data set.

Note: Your application cannot use multiple copies of an IBM Open Class library. If your application consists of multiple modules (for example, a main module and a DLL) that use the same class library, make sure that all your modules link dynamically to the class library. Otherwise, the class library will be linked in multiple times, and there will be multiple copies in use by your application. You cannot use multiple copies of a class library within a single application. If you do, you can have unexpected results.

You can use one of the following IBM-supplied cataloged procedures that include a link-edit step to link-edit your z/OS C program:

`EDCCL`	Compile and link-edit
`EDCCLG`	Compile, link-edit, and run
`EDCCPL`	Compile, prelink, and link-edit
`EDCCPLG`	Compile, prelink, link-edit, and run.

Note: By default, the procedures `EDCCL`, `EDCCLG`, and `EDCCPLG` do not save the compiled object. `EDCCLG` and `EDCCPLG` do not save load modules.

See "Appendix D. Cataloged Procedures and REXX EXECs" on page 529 for more information on REXX EXECs and their uses.

The following example shows the general job control procedure for link-editing a program under z/OS batch using the Language Environment Library.

```
// jobcard
//*
//* THE FOLLOWING STEP LINKS THE MEMBERS TESTFILE AND DECODE FROM
//* THE LIBRARIES USERID.WORK.OBJECT AND USERID.LIBRARY.OBJECT AND
//* PLACES THE LOAD MODULE IN USERID.WORK.LOAD(TEST)
//*
//LKED   EXEC  PGM=IEWL,REGION=1024K,PARM='AMODE=31,RMODE=ANY,MAP'
//SYSLIB   DD  DSNAME=CEE.SCEELKED,DISP=SHR
//SYSLIN   DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=USERID.WORK.LOAD(TEST),DISP=SHR
//OBJECT   DD  DSNAME=USERID.WORK.OBJECT,DISP=SHR
//LIBRARY  DD  DSNAME=USERID.LIBRARY.OBJECT,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  UNIT=VIO,SPACE=(32000,(30,30))
//SYSIN    DD  DATA,DLM=@@
     INCLUDE   OBJECT(TESTFILE)
     INCLUDE   LIBRARY(DECODE)
@@
```

Figure 61. Link-Editing a Program under z/OS Batch

You can use one of the following IBM-supplied cataloged procedures that include a prelink and link step to link your C++ program:

`CBCCL`	Compile, prelink, and link
`CBCL`	Prelink and link
`CBCCLG`	Compile, prelink, link, and run
`CBCLG`	Prelink, link, and run.

Specifying Prelinker and Link-Edit Options using Cataloged Procedures

In the cataloged procedures use the `PPARM` statement to specify prelinker options and the `LPARM` statement to specify link-edit options as follows:

```
PPARM='"prelinker-options"'
LPARM='"link-edit-options"'
```

where *prelinker-options* is a list of prelinker options and *link-edit-options* is a list of link-edit options. Separate link-edit options and prelinker options with commas.

Writing JCL for the Prelinker and Linkage Editor

You can use cataloged procedures rather than supply all of the job control language (JCL) required for a job step that invokes the prelinker or linkage editor. However, you should be familiar with these JCL statements. This familiarity enables you to make the best use of the prelinker and linkage editor and, if necessary, override the statements of the cataloged procedure.

For a description of the IBM-supplied cataloged procedures that include a prelink and link step, see "Appendix D. Cataloged Procedures and REXX EXECs" on page 529.

The following sections describe the basic JCL statements for prelinking and linking.

Using the EXEC Statement

Use the `EXEC` job control statement in your JCL to invoke the prelinker. The following example shows an `EXEC` statement that invokes the prelinker:

```
//PLKED EXEC PGM=EDCPRLK
```

You can also use the `EXEC` job control statement in your JCL to invoke the linkage editor. The following is a sample `EXEC` statement that invokes the linkage editor:

```
//LKED EXEC PGM=HEWL
```

Note: If you are using DLLs, you must use the `RENT` linkage editor option.

Using the PARM Parameter

By using the `PARM` parameter of the `EXEC` statement, you can select one or more of the optional facilities that the prelinker and linkage editor provide.

For example, if you want the prelinker to use the automatic call library to resolve unresolved references, specify the `NONCAL` prelinker option using the `PARM` parameter on the prelinker `EXEC` statement:

```
//PLKED EXEC PGM=EDCPRLK,PARM='NONCAL'
```

If you want a mapping of the load modules produced by the linkage editor, specify the `MAP` option with the `PARM` parameter on the linkage editor `EXEC` statement:

```
//LKED EXEC PGM=HEWL,PARM='MAP'
```

For a description of prelinker options see "Prelinker Options" on page 503, for linkage editor options see "Linkage Editor Options" on page 505.

Example of JCL to Prelink and Link

Figure 62 on page 483 shows a typical sequence of job control statements to link-edit an object module into a load module.

```
//*-------------------------------------------------------------
//* PRE-LINKEDIT STEP:
//*-------------------------------------------------------------
//PLKED  EXEC PGM=EDCPRLK,REGION=2048K,PARM='MAP'
//STEPLIB  DD  DSN=CEE.SCEERUN,DISP=SHR
//SYSMSGS  DD  DSN=CEE.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB   DD  DSN=CEE.SCEECPP,DISP=SHR
//         DD  DSN=CBC.SCLBCPP,DISP=SHR
//SYSIN    DD  DSN=USERID.TEXT(PROG1),DISP=SHR
//SYSMOD   DD  DSN=&&PLKSET,UNIT=VIO,DISP=(MOD,PASS),
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=32000)
//SYSDEFSD DD  DSN=USERID.TEXT(PROG1IMP),DISP=SHR
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*-------------------------------------------------------------
//* LINKEDIT STEP:
//*-------------------------------------------------------------
//LKED   EXEC PGM=HEWL,REGION=1024K,COND=(8,LE,PLKED),PARM='MAP'
//SYSLIB   DD  DSN=CEE.SCEELKED,DISP=SHR
//SYSLIN   DD  DSN=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//SYSLMOD  DD  DSN=USERID.LOAD(PROG1),DISP=SHR
//SYSUT1   DD  UNIT=VIO,SPACE=(32000,(30,30))
//SYSPRINT DD  SYSOUT=*
```

Figure 62. Creating a Load Module under z/OS Batch

Note: For a C++ application, this JCL uses static class libraries.

Specifying Link-Edit Options through JCL

In your JCL for link-edit processing, use the PARM statement to specify link-edit options:

```
PARM=(link-edit-options)
PARM.STEPNAME=('link-edit-options') (If a PROC is used)
```

where *link-edit-options* is a list of link-edit options. Separate the link-edit options with commas.

You can prelink and link C/C++ applications under z/OS batch by submitting your own JCL to the operating system or by using the IBM cataloged procedures. See “Appendix D. Cataloged Procedures and REXX EXECs” on page 529 for more information on the supplied procedures.

Secondary Input to the Linker

Secondary input is either all object modules or all load modules, but it cannot contain both types.

Specify the secondary input data sets with a **SYSLIB** statement and, if the data sets are object modules, add the linkage editor LIBRARY and INCLUDE control statements. If you have multiple secondary input data sets, concatenate them as follows:

```
//SYSLIB DD DSNAME=CEE.SCEELKED,DISP=SHR
//       DD DSNAME=AREA.SALESLIB,DISP=SHR
```

To specify additional object modules or libraries, code INCLUDE and LIBRARY statements after your DD statements as part of your job control procedure, such as in Figure 63 on page 484.

```
   ⋮
//SYSLIN DD    DSNAME=&&GOFILE,DISP=(SHR,DELETE)
//       DD    *
   INCLUDE ddname(member)
   LIBRARY ADDLIB(CPGM10)
/*
```

Figure 63. Linkage Editor Control Statements

A the linkage editor encounters the `INCLUDE` statement, it incorporates the data sets that the control statement specifies. In contrast, the linkage editor uses the data sets that are specified by the `LIBRARY` statement only when there are unresolved references after it all the other input is processed.

When you use cataloged procedures or your own JCL to invoke the linkage editor, external symbol resolution by automatic library call involves a search of the data set defined by the `DD` statement with the name `SYSLIB`.

Using Additional Input Object Modules under z/OS Batch

When you use cataloged procedures or your own JCL to invoke the prelinker and linkage editor, external symbol resolution by automatic library call involves a search of the `SYSLIB` data set. The prelinker and linkage editor locate the functions in which the external symbols are defined (if such functions exist), and include them in the output module.

You can use prelinker and linkage control statements `INCLUDE` and `LIBRARY` to do the following:

1. Specify additional object modules that you want included in the output module (`INCLUDE` statement).
2. Specify additional libraries to be searched for modules to be included in the output module (`LIBRARY` statement). This statement has the effect of concatenating any specified member names with the automatic call library.

Code these statements after your `DD` statements as part of your job control procedure. For example:

```
   ⋮
//SYSIN DD    DSNAME=&&GOFILE,DISP=(SHR,DELETE)
//      DD    *
   INCLUDE ddname(member)
   LIBRARY ADDLIB(CPGM10)
/*
```

Data sets specified by the `INCLUDE` statement are incorporated as the prelinker and linkage editor encounter the statement. In contrast, data sets specified by the `LIBRARY` statement are used only when there are unresolved references after all the other input is processed.

Any prelinker and linkage editor processing beyond the basic processing described above must be specified by linkage editor control statements in the primary input.

Under TSO

The z/OS Language Environment Prelinker is started under TSO through REXX EXECs. The IBM supplied REXX EXECs that invoke the prelinker and create an executable module are called `CXXMOD` and `CPLINK`. If you want to create a reentrant load module, you must use these REXX EXECs instead of the `TSO LINK` command.

It is recommended that you use CXXMOD instead of CPLINK. For a description of the CXXMOD REXX EXEC see “Prelinking and Linking under TSO”. For a description of the CPLINK command see “Other z/OS C Utilities” on page 551.

When using the TSO LINK command processor, the data set defined by the LIB operand will be used by the command processor for external symbol resolution. The linkage editor locates the functions in which the external symbols are defined (if such functions exist), and includes them in the load module.

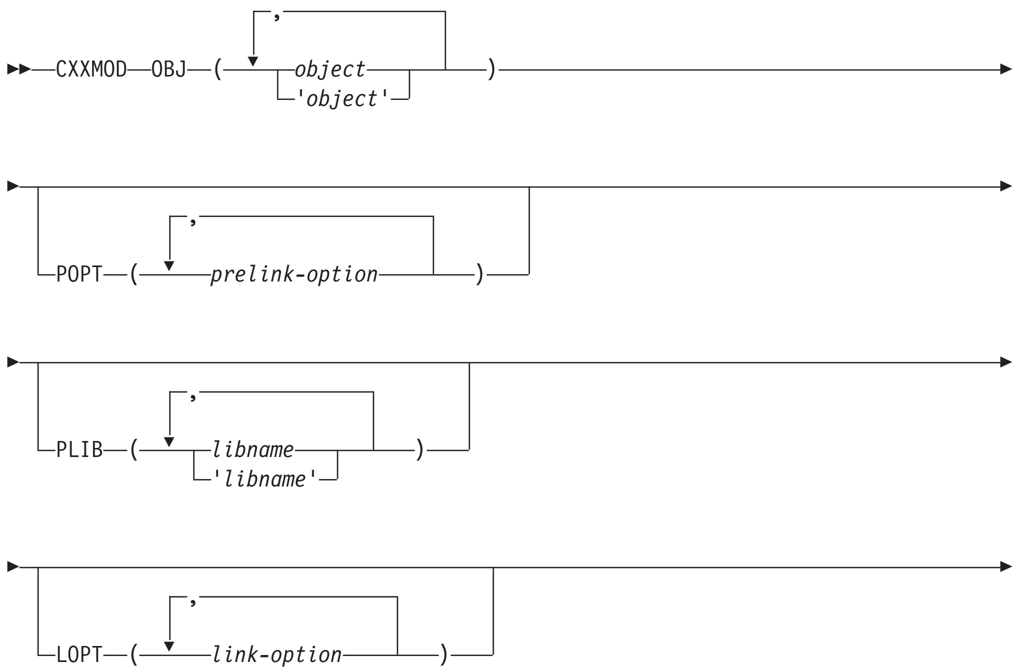
Any linkage editor processing beyond the basic processing described above must be specified by linkage editor control statements in the primary input. The IBM-supplied catalog procedures and REXX EXECs use the DLL versions of the IBM-supplied class libraries by default.

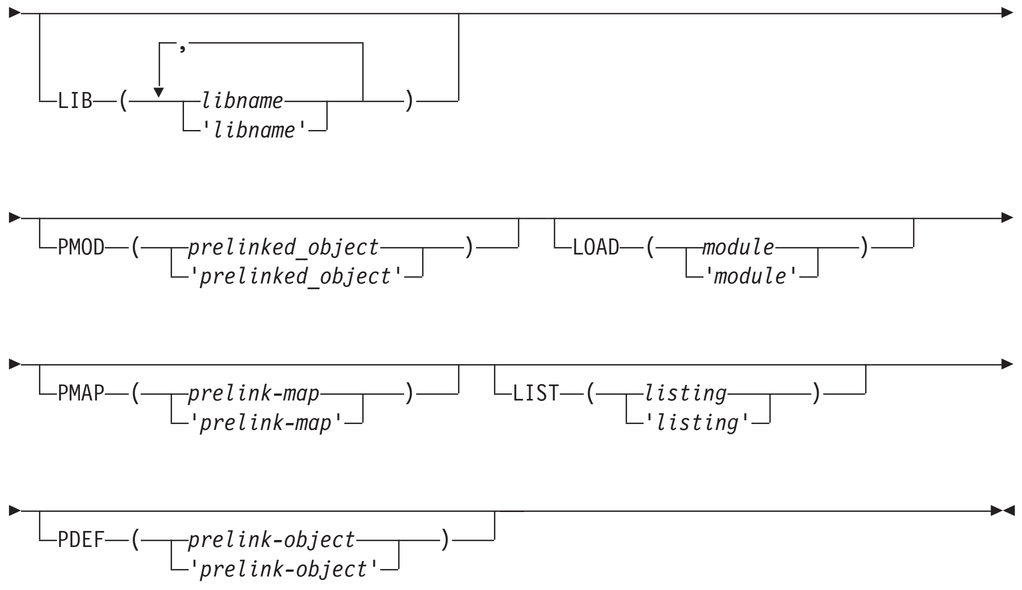
To link-edit your z/OS C program under TSO, use either the CXXMOD, CMOD, or the LINK command. It is recommended that you use CXXMOD, particularly when linking z/OS C and z/OS C++ object decks. For a description of the CXXMOD REXX EXEC see “Prelinking and Linking under TSO”. For a description of CMOD and the TSO LINK command see “Other z/OS C Utilities” on page 551.

Prelinking and Linking under TSO

This section describes how to prelink and link your z/OS C++ or z/OS C program by invoking the CXXMOD REXX EXEC. This REXX EXEC creates an executable module.

The syntax for the CXXMOD REXX EXEC is:





CXXMOD

OBJ You must **always** specify the input file names on the OBJ keyword parameter. Each input file must be a C, C++ or assembler object module. Note that the file can be either a PDS member, a sequential file or an HFS file.

If the high-level qualifier of a file is not the same as your user prefix, you must use the fully qualified name of the file and place single quotation marks around the entire name.

For HFS file names: Neither commas nor special characters need to be escaped. But you must place file names containing special characters or commas between single quotes. If a single quote is part of the file name, the quote must be specified twice. HFS filenames must be absolute names, that is they must begin with a slash (/).

POPT Prelinker options can be specified using the POPT keyword parameter. If the MAP prelink option is specified, a prelink map will be written to the file specified under the PMAP keyword parameter. For more details on generating a prelink map, see the information on the PMAP option below.

LOPT Linkage editor options can be specified using the LOPT keyword parameter. For details on how to generate a linkage editor listing, see the option LIST.

PLIB The library names that are to be used by the automatic call library facility of the prelinker must be specified on the PLIB keyword parameter. The default library used is the C++ base library, CEE.SCEECPP.

If the high-level qualifier of a library data set is not the same as your user prefix, you must use the fully qualified name of the data set and place single quotation marks around the entire name.

LIB If you want to specify libraries for the link step to resolve external references, use the LIB keyword parameter. The default library used is CEE.SCEELKED.

If the high-level qualifier of a library data set is not the same as your user prefix, you must use the fully qualified name of the data set and place single quotation marks around the entire name.

PMOD
: If you want to keep the output prelinked object module, specify the file that it should be placed in by using the PMOD keyword parameter. The default action is to create a file and erase it after the link is complete. The file can be either a data set or an HFS file.

  If the high-level qualifier of the output prelinked object module is not the same as your user prefix, you must use the fully qualified name of the file and place single quotation marks around the entire name.

LOAD
: To specify where the resultant load module should be placed, use the LOAD keyword parameter. The file can be either a data set or an HFS file.

  If the high-level qualifier of the load module is not the same as your user prefix, you must use the fully qualified name of the file and place single quotation marks around the entire name.

LIST
: To specify where the linkage editor listing should be placed, use the LIST keyword parameter. The file can be either a data set or an HFS file. If you specify *, the listing will be directed to your console.

  If the high-level qualifier of the linkage editor listing is not the same as your user prefix, you must use the fully qualified name of the file and place single quotation marks around the entire name.

PMAP
: To specify where the prelinker map should be placed, use the PMAP keyword parameter. The file can be either a data set or an HFS file. If you specify *, the prelinker map will be directed to your console.

  If the high-level qualifier of the prelinker map is not the same as your user prefix, you must use the fully qualified name of the file and place single quotation marks around the entire name.

PDEF
: To specify where the generated IMPORT control statements should be placed by the prelinker. The file can be either a data set or an HFS file.

  If the high-level qualifier of the IMPORT control statement listing is not the same as your user prefix, you must use the fully qualified name of the file and place single quotation marks around the entire name.

Example of Prelinking and Linking under TSO

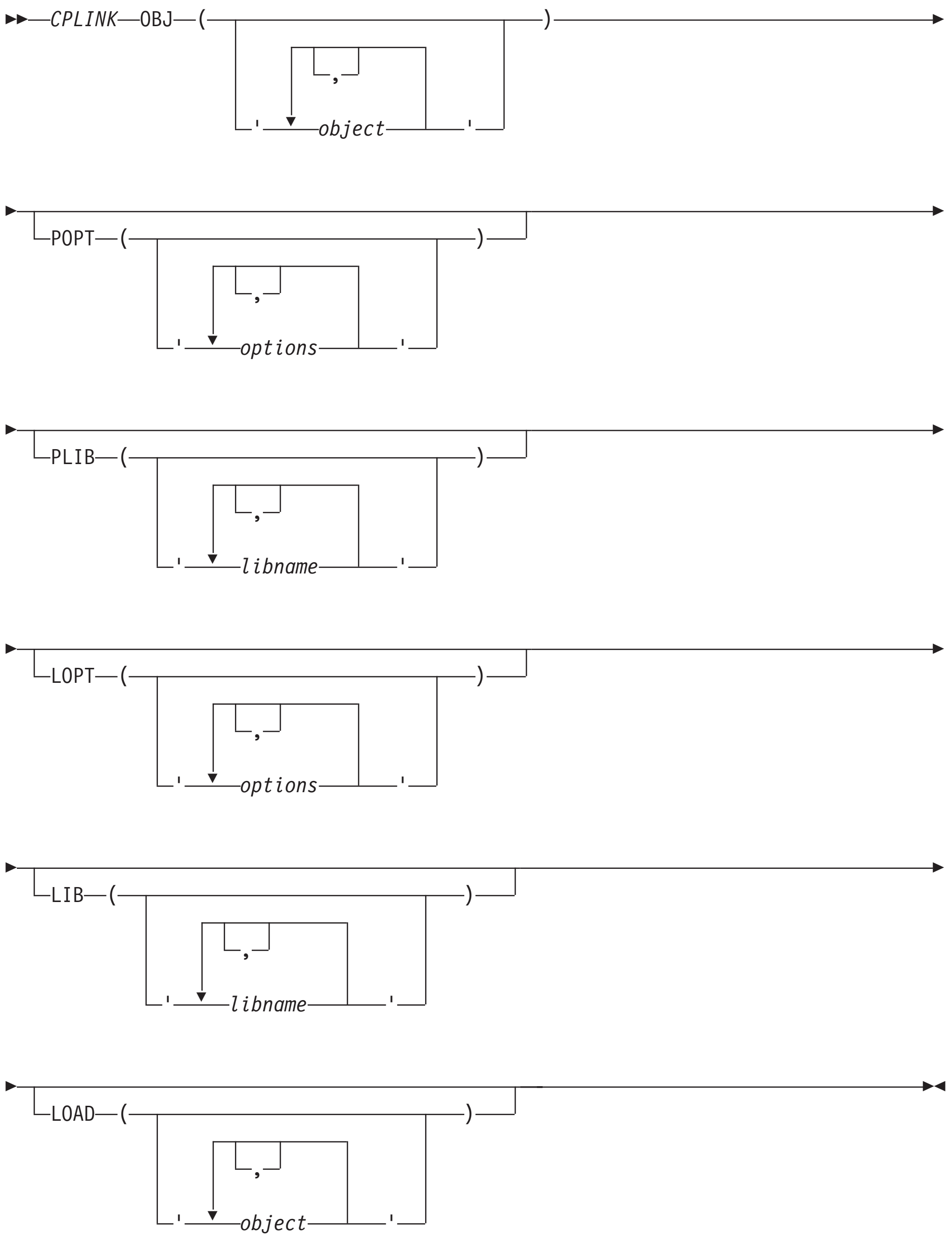
In the following example, the user prefix is RYAN and the input object module members MAIN and FN are in the PDS called 'RYAN.ACCOUNT.OBJ'. A prelink map is to be generated and placed in 'RYAN.ACCOUNT.MAP(SALES)'. The load module will be placed in a PDS member called 'GROUP.ACCOUNT.LOAD(SALES)'. The linkage editor listing will be written to 'RYAN.ACCOUNT.LIST(SALES)'.

```
CXXMOD OBJ(ACCOUNT.OBJ(MAIN), ACCOUNT.OBJ(FN))
       POPT(MAP) LOPT(XREF, MAP)
       LOAD('GROUP.ACCOUNT.LOAD(SALES)') MAP(ACCOUNT.MAP(SALES))
       LIST(ACCOUNT.LIST(SALES))
```

In this instance, both the z/OS Language Environment stub library and the partitioned data set (library) SALESLIB are available as the automatic call libraries. The linkage editor LIBRARY control statement has the effect of concatenating any specified member names with the automatic call library.

Using CPLINK

The CPLINK command has the following syntax:



OBJ specifies an input data set name.

This is a required parameter. Each input data set must be a C object module compiled with the RENT or LONGNAME compiler options, or a compiled program (C or otherwise) having no static external data.

POPT specifies a string of prelink options.

The prelinker options available for CPLINK are the same as for z/OS batch. For example, if you want the prelinker to use the MAP option, specify the following:

```
CPLINK file name POPT('MAP')..
```

When you specify the prelink MAP option (as opposed to the link MAP option), the prelinker produces a file that shows the mapping of static external data. This map shows name, length, and address information. If there are any unresolved references or duplicate symbols during the prelink step, the map displays them.

PLIB
: specifies the library names that the prelinker uses for the automatic library call facility.

LOPT
: specifies a string of linkage editor options.

For example, if you want the prelink utility to use the MAP option, and the linkage editor to use the NOMAP option, use the following CLIST command:

```
CPLINK file name POPT('MAP') LOPT('NOMAP...')
```

LIB
: specifies any additional library or libraries that the TSO LINK command uses to resolve external references. These libraries are appended to the default C library functions.

LOAD
: specifies an output data set name.

If you do not specify an output data set name, a name is generated for you. The name that the CLIST generates consists of your user prefix, followed by CPOBJ.LOAD(TEMPNAME). For more information on the file format for output data, refer to *z/OS DFSMS Program Management*.

Examples

In the following example, your user prefix is RYAN, and the data set that contains the input object module is the partitioned data set RYAN.C.OBJ(INCCOMM). This example will generate a prelink listing without using the automatic call library. After the call, the load module is placed in the partitioned data set RYAN.CPOBJ.LOAD(TEMPNAME), and the prelink listing is placed in the sequential data set RYAN.CPOBJ.RMAP.

```
CPLINK OBJ('C.OBJ(INCCOMM)')
```

In the following examples, assume that your user prefix is PAUL, and the data set that contains the input object module is the partitioned data set PAUL.C.OBJ(INCPYRL). This example will not generate a prelink listing, and the automatic call facility will use the library RAINBOW.LIB.SUB. The load module is placed in the partitioned data set PAUL.TBD.LOAD(MOD).

```
//*----------------------------------------------------------
//* Prelink and link 'PAUL.C.OBJ(INCPYRL)'
//*----------------------------------------------------------
//P0014001  EXEC EDCPL,
//          INFILE='PAUL.C.OBJ(INCPYRL)',
//          OUTFILE='PAUL.TBD.LOAD(MOD),DISP=SHR',
//          PPARM='NOMAP,NONCAL',
//          LPARM='AMODE(31),RMODE(ANY) '
//*---------------------------------------------------------
```

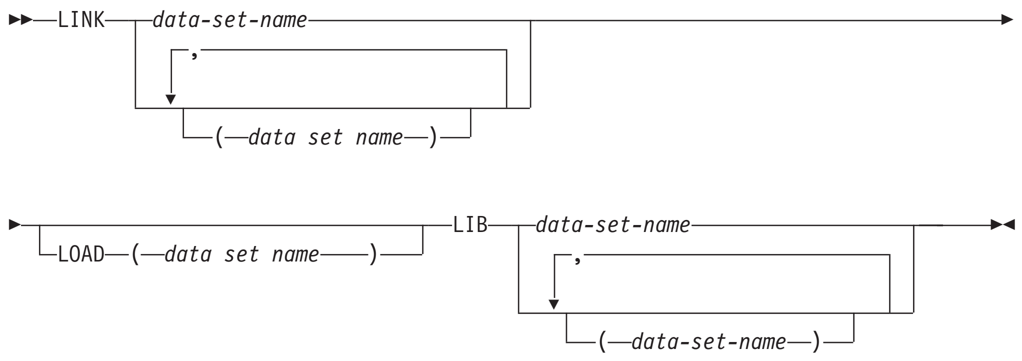
Figure 64. Example of Prelinking under z/OS Batch

```
CPLINK OBJ('''PAUL.C.OBJ(INCPYRL)''')
      POPT('NOMAP,NONCAL')
      PLIB('''RAINBOW.LIB.SUB''')
      LOAD('TBD.LOAD(MOD)')
```

Figure 65. Example of Prelinking under TSO

Using LINK

The general form of the TSO `LINK` command is:



Input to the LINK Command

You must specify one or more object module names, or load module names, after the `LINK` keyword. For example, to link-edit *program2.obj*, using the Language Environment Library, you would issue the following:

```
LINK program2.obj LIB('CEE.SCEELKED')
```

Notes:

1. You must always specify `'CEE.SCEELKED'` in the `LIB` operand. It is not required during the execution of a z/OS C/C++ program.

LIB Operand of the LINK Command

The `LIB` operand specifies the names of data sets that are to be used to resolve external references by the automatic library call facility. Language Environment Library is made available to your program in this manner and must always be specified on the `LIB` operand. In the following example, *SALESLIB.LIB.SBRT2* is used to resolve external references used in *program2*.

```
LINK program2.obj LIB('CEE.SCEELKED.', 'SALESLIB.LIB.SBRT2')
```

A request coded this way searches `CEE.SCEELKED` and `SALESLIB.LIB.SBRT2` to resolve external references.

LOAD Operand of the LINK Command

In the `LOAD` operand, you can specify the name of the data set that is to hold the load module as follows:

```
LINK LOAD(load-mod-name(member)) LIB('CEE.SCEELKED')
```

The load module produced by the linkage editor must be a member in a partitioned data set.

If you do not specify a data set name for the load module, the system constructs a name by using the first data set name that appears after the keyword `LINK`, and it will be placed in a member of the *user-prefix.program-name*.`LOAD` data set. If the input data set is sequential and you do not specify a member name, `TEMPNAME` is used.

The following example shows how to link-edit two object modules and place the resulting load module in *member* `TEMPNAME` of the *userid*.`LM.LOAD` data set.

`LINK` *program1,program2* `LOAD(`*lm*`)`

You can also specify link-edit options in the link statement:

`LINK` *program1* `LOAD(`*lm*`) LET`

Options for the linkage editor are discussed in “Output from the Linkage Editor” on page 468.

For more information about using the TSO command LINK, see *z/OS TSO/E Command Reference* .

Specifying Link-Edit Options through the TSO LINK Command

TSO users specify link-edit options through the `LINK` command. For example, to use the `MAP`, `LET`, and `NCAL` options when the object module in `SMITH.PROGRAM1.OBJ` is placed in `SMITH.PROGRAM1.LOAD(LM)`, enter:

```
LINK SMITH.PROGRAM1 'LOAD(LM) MAP LET NCAL'
```

You can use *link-edit-options* to display a map listing at your terminal:

```
LINK PROGRAM1 MAP PRINT(*)
```

Storing Load Modules in a Load Library

If you want to link C functions, to store them in a load library, and to `INCLUDE` them later with main procedures, use the `NCAL` and `LET` linkage editor options.

Prelinking and Link-Editing under the z/OS Shell

You can prelink and link your application under the shell by using the the `OMVS` prelinker option. The `OMVS` option causes the prelinker to change its processing of `INCLUDE` and `LIBRARY` control statements. The search library is pointed to immediately for any currently unresolved symbols. If the processing of subsequent `INCLUDE` or `LIBRARY` statements results in new or unresolved symbols, a previously encountered library will not be searched again. You may need another `LIBRARY` statement that points to the same library to search it again. For more information on the `OMVS` prelinker option, see “Appendix B. Prelinker and Linkage Editor Options” on page 503.

Using your JCL

The example JCL in Figure 66 on page 492 links to an archive library and to z/OS data sets. Include files may be PDS members, sequential files, or HFS files. Libraries may be partitioned data sets, or archive libraries.

```
//jobcard information...
//*-----------------------------------------------------------------
//*----- prelink ----------------------------------------------------
//RAWPLINK  EXEC PGM=EDCPRLK,
//          PARM='OMVS,MEMORY,MAP,NONCAL'
//STEPLIB   DD DISP=SHR,DSN=CEE.SCEERUN
//SYSMSGS   DD DISP=SHR,DSN=CEE.SCEEMSGP(EDCPMSGE)
//SYSLIB    DD DUMMY
//* object file
//DDOBJ1    DD PATH='/u/myuserid/callfoogoohoo.o'
//* PDS member
//DDOBJ2    DD DISP=SHR,DSN=MYUSERID.QAPARTNR.OBJ(MEM1)
//* archive library
//DDLIB3    DD PATH='/u/myuserid/mylibrary.a'
//* PDS Library
//DDLIB4    DD DISP=SHR,DSN=MYUSERID.QAPARTNR.OBJ
//SYSIN DD DATA,DLM=@@
 INCLUDE DDOBJ1
 INCLUDE DDOBJ2
 LIBRARY DDLIB3
 LIBRARY DDLIB4
@@
//SYSMOD    DD DISP=SHR,DSN=MYUSERID.TEMP.OBJ(MEM1)
//SYSOUT    DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSDEFSD  DD DUMMY
```

Figure 66. Using OMVS to prelink and link

The JCL in Figure 66 produces the following prelinker map:

```
========================================================================
|                           Prelinker Map                              |
|                                                                      |
| CPLINK:5645001 V1 R7 M00 IBM Language Environment 1997/01/20 16:28:55|
========================================================================

Command Options. . . . . : NONCAL   MEMORY   ER       DUP      MAP
                         : OMVS     NOUPCASE


========================================================================
|                     Object Resolution Warnings                       |
========================================================================

WARNING EDC4015: Unresolved references are detected:
CEEBETBL CEEROOTA goo      CEESG003 EDCINPL


========================================================================
|                              File Map                                |
========================================================================

*ORIGIN  FILE ID  FILE NAME

   PI     00001   /u/myusrd/callfoogoohoo.o
   PI     00002   MYUSRID.QAPARTNR.OBJ(MEM1)
   A      00003   /u/myusrd/mylibrary.a(foo.o)
   A      00004   MYUSRID.QAPARTNR.OBJ(MEMHOO)

*ORIGIN:  P=primary input     PI=primary INCLUDE    SI=secondary INCLUDE
          A=automatic call     R=RENAME card         L=C Library
         IN=internal


========================================================================
|                         Writable Static Map                          |
========================================================================

INFORMATIONAL EDC4013: No map displayed as no writable static was found.


========================================================================
|                  ESD Map of Defined and Long Names                   |
========================================================================

                   OUTPUT
*REASON  FILE ID  ESD NAME   INPUT NAME

   P      00001   CEESTART   CEESTART
   P      00001   CEEMAIN    CEEMAIN
   D      00001   MAIN       main
   D      00003   FOO        foo
   D              GOO        goo
   D      00004   HOO        hoo
   P              CEESG003   CEESG003
   P              EDCINPL    EDCINPL
   D      00002   FUNC@IN@   func_in_MEM1

*REASON: P=#pragma or reserved    S=matches short name    R=RENAME card
         L=C Library              U=UPCASE option         D=Default


=========== E N D   O F   P R E - L I N K A G E   M A P ============
```

Figure 67. Prelinker Map produced when prelinking using OMVS

Setting c89 to Invoke the Prelinker

The `c89`, `c++`, and cc utilities invoke the binder by default, unless the output file of the link-editing phase (-o option) is a PDS, in which case they use the Prelinker.

You can set the {_STEPS} environment for each of these utilities to use the Prelinker for link-edit output files that are PDSEs or HFS files.

Once you set the {_STEPS} environment variable for a utility so that the Prelinker bit is turned on, that utility will always use the Prelinker. If you want to use the binder, you must unset the {_STEPS} environment variable.

For a complete description of `c89`, `c++`, and cc, see "Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file" on page 555. For a description of the {_STEPS} environment variable, see *Z/OS UNIX System Services Command Reference*.

Using the c89 Utility

The `c89` utility specifies default values for some prelinker and linkage editor options. It also passes prelinker options and linkage editor options by using the `-W` option.

`c89` specifies prelinker and linkage editor options in order for it to provide the user with correct and consistent behavior. In order to determine exactly the prelinker and linkage editor options that `c89` specifies, you should use the `c89 -V` option.

Some `c89` options, such as `-V`, will change the settings of the prelinker options and the linkage editor options that `c89` specifies. For example, when you do not specify `-V`, `c89` specifies the Prelinker option `NOMAP`, and when you specify `-V`, `c89` specifies the Prelinker option `MAP`.

To explicitly override the options that `c89` specifies, use the `c89 -W` option. For example, to use the Prelinker option `MAP` even when the `c89 -V` option is not specified, invoke

```
c89 -Wl,p,map ...
```

For a list of prelinker options and their uses, see "Prelinker Options" on page 503.

Prelinker Control Statement Processing

The only control statements that the prelinker processes are `IMPORT`, `INCLUDE`, `LIBRARY`, and `RENAME` statements. The remaining control statements remain unchanged until the link step.

You can place the control statements in the input stream, or store them in a permanent data set. If you cannot fit all of the information on one control statement, you can use one or more continuations. The long name, for example, can be split across more than one statement. You can enable continuations in one of two ways:

- Place a nonblank character in column 72 of the statement that is to be continued. The continuation must begin in column 16 of the next statement.
- Enclose the name in single quotation marks. When such a name is continued across statements, it extends up to and includes column 71. Although column 72 is not considered part of the name, it must be nonblank for the name to be

continued. On the following statement, column 1 must be blank (containing the X'40' character); the name then continues in column 2.

If you have a name that contains a single quotation mark, and you want to enclose the whole name in single quotation marks, put two single quotation marks next to each other where you want the single one to appear in the name. For example, if you want the name

```
SymbolNameWithAQuote'InTheMiddle
```

specify it as follows:

```
'SymbolNameWithAQuote''InTheMiddle'
```

If you mix the two style of continuation in one control statement, after you continue a statement in column 2 due to a quote in the name, all subsequent statements will continue in column two.

IMPORT Control Statement

The `IMPORT` control statement has the following syntax:

```
►►──IMPORT──┬─CODE──┬─dll-name─────────┬──┬─function─────────┬─┬──────────────►◄
            │       └─'──dll-name──'───┘  └─'──function──'───┘ │
            └─DATA──┬─dll-name─────────┬──┬─variable─────────┬─┘
                    └─'──dll-name──'───┘  └─'──variable──'───┘
```

`dll-name`
: The name or alias of the load module for the DLL. The maximum length of an alias is 8 characters. However, the name itself can be a longname. The *`dll-name`* comes from the value specified on the `DLLNAME` prelinker option. For more information, see "Prelinker Options" on page 503.

`variable`
: An exported variable name. It is a mixed case longname. To indicate a continuation across statements, either use a non-blank character in column 72 of the card and begin the next line in column 16, or enclose the name in single quotation marks, end the first line in column 71, and put a blank character in column 1 of the next line.

`function`
: An exported function name. It is a mixed case longname. You can indicate a continuation the same way you would for a variable.

The prelinker processes `IMPORT` statements, but does not pass them on to the link step.

INCLUDE Control Statement

The `INCLUDE` control statement has the following syntax:

```
►►──INCLUDE──┬─ddname─────────┬──(──┬─member──────────────┬──)──────────────►◄
             └─'──ddname──'───┘     │  ┌─,──────────────┐ │
                                    │  ▼                │ │
                                    └───'──member──'────┴─┘
```

ddname A ddname associated with a file to be included. You can use the same kinds of continuations that you can for the *variable* on the IMPORT control statement.

member The member of the DD to be included. You can use the same kinds of continuations that you can for the *variable* on the IMPORT control statement.

The prelinker processes INCLUDE statements like the z/OS linkage editor with the following exceptions:

An attempt is made to read the DD or member of the DD (whichever is specified). This request is resolved if the read is successful.
- INCLUDEs of identical member names are not allowed.
- INCLUDEs of both a ddname and a member from the same ddname are not allowed. The prelinker ignores the second INCLUDE.

Note: The INCLUDE control statement is removed and not placed in the prelinker output object module; the system linkage editor does not see the INCLUDE control statement.

For more information on the linkage editor, refer to *z/OS DFSMS Program Management* .

LIBRARY Control Statement

The LIBRARY control statement has the following syntax:

NOOMVS

```
                                         ┌──────────────────────┐
                                         ▼                      │
►►──LIBRARY──┬──┬─name─────────┬──(──────┬─member─────────┬─────┴──)──┬──────────────►◄
             │  └─'──name──'───┘         └─'──member──'───┘           │
             │                        ┌───────────────────────┐       │
             │                        ▼                       │       │
             └──┬─────┬──(────────────┬─external─────────┬────┴──)────┘
                └─*───┘               └─'──external──'───┘
```

OMVS

```
►►──LIBRARY───name────────────────────────────────────────────────────────────────────►◄
```

name
: the name of a DD that defines a library, under z/OS. This could be a concatenation of one or more libraries that are created with or without the Object Library Utility. You can use the same kinds of continuations that you can for the *variable* on the IMPORT control statement.

member
: the name or alias of a member of the specified library. Because both short names and long names can be specified, case distinction is significant. If you use an long name, you can use the same kinds of continuations that you can for the *variable* on the IMPORT control statement.

Under z/OS, automatic library calls search the library and each subsequent library in the concatenation, if necessary, for the name instead of searching the primary input. If you specify the `OMVS` option, the only form of the `LIBRARY` card the prelinker accepts is `LIBRARY` *ddname* statement in `SYSLIB`.

external
: an external reference that may be unresolved after primary input processing. An Automatic Library call will not resolve this external reference. Because both short names and long names can be specified, case distinction is significant. If you use an long name, you can use the same kinds of continuations that you can for the *`variable`* on the `IMPORT` control statement.

Note: The `LIBRARY` control statement is removed and not placed in the prelinker output object module; the system linkage editor does not see the `LIBRARY` control statement.

RENAME Control Statement

The `RENAME` control statement has the following syntax:

NOOMVS

```
►►──RENAME──┬─long name─────────┬─short name──┬────────┬─────────────────────────►◄
            └─'──long name──'───┘             └─SEARCH─┘
```

OMVS

```
►►──RENAME──┬─long name─────────┬─short name─────────────────────────────────────►◄
            └─'──long name──'───┘
```

long name
: the name of the long name to be renamed on output. All occurrences of this long name are renamed. You can use the same kinds of continuations that you can for the *`variable`* on the `IMPORT` control statement.

short name
: the name of the short name to which the long name will be changed. This name can be at most 8 characters, and case is respected.

SEARCH
: an optional parameter specifying that if the short name is undefined, the prelinker searches by an automatic library call for the definition of the short name. This is not available with the `OMVS` option.

The `RENAME` control statement is processed by the prelinker. You can use this statement to do the following:

- Explicitly override the default name that is given to an long name when an long name is mapped to an short name.

  You can explicitly control the names that are presented to the system linkage editor so that external variable and function names are consistent from one linkage editor run to the next. This consistency makes it easier to recognize control section and label names that appear in system dumps and linkage editor listings. Another mapping rule can provide the suitable name, but if you need to replace the linkage editor control section, you need to maintain consistent names. See “Mapping Long Names to Short Names” on page 464 for a description of this rule.

- Explicitly bind an long name to an short name. This binding may be necessary when linking with other languages that use a different name for the same object.

A RENAME control statement cannot be used to rename a writable static object because its name is not contained in the output from the prelinker.

You can place RENAME control statements before, between, or after other control statements or object modules. An object module can contain only RENAME statements. RENAME statements can also be placed in input that is included because of other RENAME statements.

Usage Notes

- A RENAME statement is ignored if the long name is not encountered in the input.
- A RENAME statement for an long name is valid provided **all** of the following are true:
  - The long name was not already mapped because of a rule that preceded the RENAME statement rule in the hierarchy described in “Mapping Long Names to Short Names” on page 464.
  - The long name was not already mapped because of a previous valid RENAME statement for the long name.
  - The short name is not itself an long name. This rule holds true even if the short name has its own RENAME statement.
  - A previous valid RENAME statement did not rename another long name to the same short name.
  - Either the long name or the short name is not defined. Either the long name or the short name can be defined, but not both. This rule holds true even if the short name has its own RENAME statement.

Reentrancy

This section discusses how to use the prelinker to make your program reentrant. For detailed information on reentrancy, see the *z/OS C/C++ Programming Guide*.

Reentrant programs are structured to allow more than one user to share a single copy of a load module or to use a load module repeatedly without reloading it.

Natural or Constructed Reentrancy

Reentrant programs can be categorized as having natural or constructed reentrancy. Programs that contain no references to the writable static objects that are listed above have natural reentrancy. Programs that refer to writable static objects must be processed with the IBM Language Environment Prelinker to make them reentrant; such programs have constructed reentrancy.

If you are using C, you do not need to use the ″RENT″ compiler option if your program is naturally reentrant.

Because all C++ programs are categorized as having constructed reentrancy, C++ code must be bound by the binder using the DYNAM(DLL) option. Alternatively, the C++ code must be processed by the prelinker before being processed by the linkage editor.

Using the Prelinker to Make Your Program Reentrant

The prelinker concatenates compile-time initialization information (for writable static) from one or more object modules into a single initialization unit. In the process, the writable static part is mapped.

If your C program contains writable static, you can use the prelinker to make your program reentrant. If your C program does not contain writable static, you do not need to use the prelinker to ensure reentrancy. The prelinker is called automatically for C++ programs.

If you compile your code and wish to link it using the z/OS system link procedures such as `IEWL`, you must first call the prelinker.

The z/OS UNIX System Services features require that all z/OS UNIX System Services C/C++ application programs be reentrant. If you are using the `c89` utility, it automatically invokes the z/OS C/C++ compiler with the `RENT` option and also invokes the prelinker.

The prelinker is not a post-compiler. That is, you do not prelink the object modules individually into separate prelinked object modules as if running the prelinker was an extension of the compile step. Instead, you prelink all the object modules together in the same job into one output prelinked object module. This is because the prelinker cannot process each object deck one at a time: it assigns offsets to each data item in the writable static area for the program, and thus needs all of the object decks that refer to data items in writable static input in a single step.

The prelinker does all of the following:

- It maps input long names from the object modules to output short names (8 characters maximum)
- It collects compile-time initialization information on static objects
- It collects constructor calls and destructor calls for static objects in C++
- It collects DLL information
- It collects objects that exist in writable static into one area by assigning an offset within the writable static area to each object
- It removes all relocation and name information of objects in the writable static area

The output of the prelinker is a single prelinked object module. You can link this object module only on the same platform where you prelinked it.

Because the prelinker maps names and removes the relocation information, you cannot use the resulting object module as input for another prelink. Also, you cannot use the linkage editor to replace a control section (CSECT) that either defines or references writable static objects.

The prelinker can handle object modules from languages other than C or C++. However, only C or assembler code using the macros `EDCDXD` and `EDCLA` may refer to writable static objects.

Generating a Reentrant Load Module in C

To use the prelinker to generate a reentrant load module in C, you must follow these steps:

1. Determine whether or not your program contains writable static. If you are unsure about whether your program contains writable static, compile it with the RENT option. Invoking the prelinker with the MAP option and the object module as input produces a prelinker map. Any writable static data in the object module appears in the writable static section of the map. Unresolved writable static references may also appear in the map as errors.

   If you see the symbol @STATIC defined in the writable static section, your code contains unnamed writable static such as modifiable literal strings, or variables with the static qualifier. To ensure that literal strings stay in the code area, recompile with `#pragma strings(readonly)`, and prelink again.
2. If your program contains no writable static, compile your program as you would normally (without any special compiler options), and then go directly to step 4.
3. If your program contains writable static, you must compile your C source files with the RENT compiler option.
4. Use the z/OS Language Environment prelinker to combine *all* input object modules into a single output object module.

   Notes:

   a. The prelinker can handle compiled programs in languages other than C or C++. However, only C, C++, OO COBOL, or assembler code using the macros EDCDXD and EDCLA may refer to writable static.
   b. You cannot use the output object module as further input to the z/OS Language Environment prelinker.
5. Optionally, you can use the output object module to link the program in the LPA or ELPA area of the system.
6. Under the z/OS shell, you can run the installed program by invoking it from the HFS. To do so you must install the program in the HFS, and, from a superuser ID, enter a `chmod` Shell command to turn on the sticky bit for the program. See *z/OS UNIX System Services Planning* for more information.

Generating a Reentrant Load Module in C++

To generate a reentrant load module in C++, you must follow these steps:

1. Compile your source code.
2. Use the supplied prelink and link utilities on the module. Under TSO, you can use the CXXMOD REXX EXEC to both prelink and link your module. Under z/OS batch, use these JCL procedures:
   - CBCCL: compile and link
   - CBCL: link
   - CBCCLG: compile, link, and go
   - CBCLG: link and go

   For all of these, linking involves two steps: invocation of the prelinker, and then a call to the system linker.

Resolving Multiple Definitions of the Same Template Function

Note: For complete information on using C++ templates, see the *z/OS C/C++ Programming Guide*

When the prelinker generates template functions, it resolves multiple function definitions as follows:

- If a function has both a specialization and a generalization, the specialization takes precedence.
- If there is more than one specialization, the prelinker issues a warning message.

Because the link step does not remove unused instantiations from the executable program, instantiating the same functions in multiple compilation units may generate very large executable programs.

External Variables

See *z/OS C/C++ Programming Guide* for more information on external variables.

The POSIX 1003.1 and X/Open CAE Specification 4.2 (XPG4.2) require that the C system header files declare certain external variables. Additional variables are defined for use with POSIX or XPG4.2 functions. If you define one of the POSIX or XPG4 feature test macros and include one of these headers, the external variables will be declared in your program. These external variables are treated differently than other global variables in a multithreaded environment (values are thread-specific) and across a call to a fetched module (values are propagated). To access the global variable values (not thread specific), you must use either C with the RENT compiler option or C++ code. Also, you must specify the SCEEOBJ autocall library during the prelink. Functions to access the thread-specific values of these variables are provided for use in a multithreaded environment. The *z/OS C/C++ Language Reference* documents these functions.

For a dynamically called DLL module to share access to the POSIX external variables, with its caller, the DLL module must define the _SHARE_EXT_VARS feature test macro. You must install APAR PQ03847 in order to use this functionality. For more information, see the section on feature test macros in the *z/OS C/C++ Run-Time Library Reference*.

Appendix B. Prelinker and Linkage Editor Options

This chapter contains the prelink options and link options for your programs under z/OS Language Environment. For more information on using the z/OS Language Environment Prelinker, see “Appendix A. Prelinking and Linking z/OS C/C++ Programs” on page 461.

Prelinker Options

The following section describes the prelink options available in z/OS C/C++ by using z/OS Language Environment.

DLLNAME(dll-name)

`DLLNAME` specifies the DLL name that appears on generated `IMPORT` control statements, described in “IMPORT Control Statement” on page 495. If you specify the `DLLNAME` option, the prelinker sets the DLL name to the value that you listed on the option.

If you do not specify `DLLNAME`, the prelinker sets the DLL name to the name that appeared on the last `NAME` control statement that it processed. If there are no `NAME` control statements, and the output object module of the prelinker is a PDS member, it sets the DLL name to the name of that member. Otherwise, the prelinker sets the DLL name to the value `TEMPNAME`, and issues a warning.

DUP | NODUP

DEFAULT: `DUP`

`DUP` specifies that if duplicate symbols are detected, their names should be directed to the console, and the return code minimally set to a warning level of 4. `NODUP` does not affect the return code setting when the prelinker detects duplicates.

ER | NOER

DEFAULT: `ER`

If there are unresolved symbols, `ER` instructs the prelinker to write a messages and a list of unresolved symbols to the console. If there are unresolved references, the prelinker sets the return code to a minimum warning level of 4. If there are unresolved writable static references, the prelinker sets the return code to a minimum error level of 8. If you use `NOER`, the prelinker does not write the list of unresolved symbols to the console. If there are unresolved references, the return code is not affected. If there are unresolved writable static references, prelinker sets the return code to a minimum warning level of 4.

MAP | NOMAP

DEFAULT: `MAP`

In the z/OS UNIX System Services environment, the `c89`, `cc`, and `c++` utilities specify `MAP` when you use the `-V` flag, and `NOMAP` when you do not.

The `MAP` option specifies that the prelinker should generate a prelink listing. See “z/OS Language Environment Prelinker Map” on page 475 for a description of the map.

MEMORY | NOMEMORY

DEFAULT: NOMEMORY

The MEMORY option instructs the prelinker to retain in storage, for the duration of the prelink step, those object modules that it reads and processes.

You can use the MEMORY option to increase prelinker speed. However, you may require additional memory to use this option. If you use MEMORY and the prelink fails because of a storage error, you must increase your storage size or use the prelinker without the MEMORY option.

NCAL | NONCAL

DEFAULT: NONCAL

The NCAL option specifies that the prelinker should not use the automatic library call to resolve unresolved references.

The prelinker performs an automatic library call when you specify the NONCAL option. An automatic library call applies to a library of user routines. For NOOMVS, the data set must be partitioned, but for OMVS the data set that the prelinker searches can be either a PDS or an archive library. Automatic library call cannot apply to a library that contains load modules.

Note: If you are prelinking C++ object modules, you must use the NONCAL option and include the C++ base library in the CEE.SCEECPP data set in your SYSLIB concatenation.

OMVS | NOOMVS

DEFAULT: NOOMVS

The OMVS option causes the prelinker to change the way that it processes INCLUDE and LIBRARY control statements. The c89 utility turns on the OE option (which maps to the OMVS option) by default. Object files and object libraries from c89 are passed as primary input to the prelinker. Object files are passed via INCLUDE control statements, and object libraries via LIBRARY control statements. Only those LIBRARY control statements that are included in primary input are accepted by the prelinker. Their syntax is:

```
LIBRARY libname
```

where *libname* is the name of a DD that defines a library. The library may be either an archive file created through the ar utility or a partitioned data set (PDS) with object modules as members. The prelinker uses LIBRARY control statements like SYSLIBs, to resolve symbols through autocalls.

When you specify the OMVS option, the prelinker accepts INCLUDE and LIBRARY statements which refer to HFS files (PATH=) and data set name (DSNAME=) allocations.

When you use the OMVS option, the order in which object files and object libraries are passed is significant. The prelinker processes its primary input sequentially. It searches the library that you specified on the LIBRARY statement only at the point where it encounters the LIBRARY statement. It does not refer to that library or processs it again. For example, if you pass your object files and object libraries as follows:

```
c89 file1.o lib1.a file2.o lib2.a
```

The prelinker processes the `INCLUDE` control statement for `file1.o`, and incorporates new symbol definitions and unresolved references from the object file into the output file. The prelinker then processes the `LIBRARY` control statement for `lib1.a`, and searches the library for currently unresolved symbols. It then processes `file2.o` followed by `lib2.a`. If the processing of `file2.o` results in unresolved symbols, the prelinker will not search the library `lib1.a` again, because it has already processed it. If you have unresolved symbols that may be defined in a library that has already been processed, you must specify a new `LIBRARY` statement after your `INCLUDE` statement to resolve those symbols. You can do this on a `c89` command line as follows:

```
c89 file1.o lib1.a file2.o lib1.a lib2.a
```

`RENAME` control statements are processed on output from the prelinker, after all of its input has been processed. Because a library can be processed once only, the `SEARCH` option on the `RENAME` control statement has no effect.

Note: The `OE` prelinker option maps to the `OMVS` prelinker option.

UPCASE | NOUPCASE

DEFAULT: `NOUPCASE`

The `UPCASE` option enforces the uppercase mapping of long names that are 8 characters or fewer and have not been explicitly mapped by another mechanism. These long names are uppercased (with _ mapped to @), and names that begin with `IBM` or `CEE` are changed to `IB$` and `CE$`, respectively.

The `UPCASE` option is useful when calling routines that are written in languages other than z/OS C/C++. For example, in COBOL and assembler, all external names are in uppercase. So, if the names are coded in lowercase in the z/OS C/C++ program and you use the `LONGNAME` option, the names will not match by default. You can use the `UPCASE` option to enforce this matching. You can also use the `RENAME` control statement for this purpose.

Note: Use of this option can be dangerous, since names with a length of 8 characters or less will lose their case sensitivity. A better way to get the linkage and names correct is through the use of the appropriate pragmas.

Linkage Editor Options

You can specify Link-edit options in either of two ways:
- Through JCL
- Through the TSO `LINK` command

For a description of link-edit options, see the *z/OS DFSMS Program Management* manuals.

Appendix C. Diagnosing Problems and the PMR/APAR Process

This appendix tells you how to diagnose failures in the z/OS C/C++ compiler. If you discover that the problem is a valid compiler problem, refer to "PMR/APAR Process" on page 514.

Problem Checklist

The following list contains suggestions to help you rule out some common sources of problems.

1. Check that the program has not changed since you last compiled or executed it successfully. If it has, examine the changes. If the error occurs in the changed code and you cannot correct it, note the change that caused the error. Whenever possible, you should retain copies of both the original and the changed source programs.
2. Be sure to correct all problems that are diagnosed by error messages, and ensure that the messages that were previously generated have no correlation to the current problem. Be sure to pay attention to warning messages.
3. The message prefix can identify the system or subsystem that issued the message. This can help you determine the cause of the problem. Following are some of the prefixes and their origins.
   - CBC - indicates messages from the z/OS C/C++ compiler, its utility components, or the z/OS C/C++ IPA Link step.
   - EDC - a numeric portion between 0090 and 0096 indicates a *severe error*, and the solution should be self-evident from the accompanying text. If it is not, contact your Service Representative. If the numeric portion is in the 4000 series, this specifically relates to the prelinker and *alias* utility. Otherwise, the message relates to the z/OS C/C++-specific messages from the runtime environment.
   - CEE - for language-independent messages from the common execution environment (CEE) library component of z/OS Language Environment.
   - IBM, PLI, IGZ - for language-specific messages from z/OS Language Environment.
   - EQA - for Debug Tool messages.
   - CLB - for messages that relate to class libraries. See the *OS/390 C/C++ IBM Open Class Library Reference* for more information.
   - BPX - messages that relate to z/OS UNIX System Services.

   You can cross reference the prefix to the message manual in most cases by using the table at the beginning of the *z/OS MVS System Messages* volumes which accompany the z/OS operating system. For example, *z/OS MVS System Messages, Vol 1*.
4. Ensure that you are compiling the correct version of the source code. It is possible that you have incorrectly indicated the location of your source file. For example, check your high-level qualifiers.
5. In any program failure, keep a record of the conditions and options in effect at the time the problem occurred. The listing file shows the options. To get the listing, compile with the `SOURCE` option. The listing only contains options that appear after the command line is processed, hence #pragma options do not appear.

Information about some of the options appears as a comment at the end of the object file. For both C or C++ compiles, there is always a comment showing the `OPTIMIZE` level. For C compiles, information about the `ALIAS`, `GONUMBER`, `INLINE`, `RENT`, or `UPCONV` options is included only if you specify the option when you compile. Note any changes from the previous compilation.

6. Your installation may have received an IBM Program Temporary Fix (PTF) for the problem. Verify that you have received all issued PTFs and have installed them, so that your installation is at the current maintenance level.
7. The preventive service planning (PSP) bucket, which is an online database available to IBM customers through IBM service channels. It gives information about product installation problems and other problems. See the z/OS Program Directory for more details.
8. Use the Debug Tool, `dbx` (for z/OS UNIX System Services) or some other debugging aid to determine the statement where the program fails and possible causes of the failure.
9. If a failing application is communicating with other IBM products, make sure that it uses the correct interface procedure as documented in the *z/OS C/C++ Programming Guide*. In many cases, you can localize the failing condition by taking out the function calls or making them no-ops.
10. If your application has been developed on a different platform (such as a microcomputer or workstation) and you try to compile and run using the z/OS C/C++ compiler, the following may cause problems:
    - The source code does not support the applicable following standards:
      - *American National Standards Institute (ANSI/ISO) C Standard* (X3.159-1989)
      - ANSI/ISO draft standard
    - The source code includes dependencies on the ASCII character set or uses the long double data type in the IEEE floating-point format.
    - The source code is system dependent
11. If your application was prelinked, make sure that the prelinking was successful as indicated in “Appendix A. Prelinking and Linking z/OS C/C++ Programs” on page 461.

When Does the Error Occur?

Determine when the problem is occurring (at compile time, bind time, prelink time, link time or run time), and use the procedures in the appropriate list on the following pages. If the problem occurs when using z/OS Language Environment, for prelink-time and run time diagnosis and debugging errors you should use *z/OS Language Environment Customization* and *z/OS Language Environment Debugging Guide*. For bind time and link-time diagnosis, refer to *z/OS DFSMS Program Management*.

After you identify the failure, you can write a small test case that re-creates the problem. A test case helps you to isolate the problem and to report problems to IBM.

To create a small test case from a large program that appears to be failing, try the suggestions listed below, after you have either backed up or made a copy of your original source code. Begin with the suggestion that seems most appropriate for the problem that you are having. If the problem persists after you have tried one of the steps below, try another in the list. Continue to break your program down until you obtain the smallest possible segment of code that still contains the error. Compile with the ″PPONLY″ option and send the expanded file as your source code. This is to ensure that all embedded header files are included. Save this last failing test case because you might need it if you have to contact an IBM Support Center.

If your program uses ″#include″ directives, put all the relevant code from the ″#include″ file directly in the main file. Use the ″PPONLY″ option and then use the expanded file as your source for submission to your IBM representative.

Remove unrelated code if it is not necessary for syntactic or semantic validity.

Remove unreferenced variables. You can find them by using the ″XREF″, the ″CHECKOUT″(C only), or the ″INFO″ (C++ only) option.

Remove any code that has not been processed at the time of failure (except for code necessary to ensure the syntactic and semantic validity of the program).

Remove all code and declarations from the body of any other functions.

If your program uses structure variables, try replacing them with scalar variables.

Complexity of Optimization

For diagnostic purposes, you should always begin by using the simplest optimization level on your program. Once you address all problems at your current level, progress toward the more complex levels of optimization.

1. Begin with a non-IPA compile and link using progressively higher levels of optimization:
   - `OPT(0)`
   - `OPT(2)`
2. Next, use `IPA(OBJONLY)` and `OPT(2)`.
   - This adds the IPA compile-time optimizations.
   - This often locates the problematic source file before you invest a lot of time and effort diagnosing problems in your code at IPA Link time.
3. Use the full IPA Compile and IPA(Level(1)) Link path
   - IPA Compile-time optimizations are performed on the IPA object
   - IPA Link-time optimizations are performed on the entire application
4. Finally, use the full IPA Compile and IPA(Level(2)) Link path
   - IPA Level 2 performs additional link-time optimizations

The Error Occurs at Compile Time

1. If your program uses any of the library routines, insert an `#include` directive for the appropriate header files. Also insert an `#include` directive for any of your own header files. The compiler uses function prototypes, when present, to help detect type mismatches on function calls. You can use the `CHECKOUT` option to find missing prototyping. Note that z/OS C++ does not allow missing prototypes.
2. Compile your program with either the `CHECKOUT` (C-only) or the `INFO` (C++ only) option. These options specify that the compiler is to give informational messages that indicate possible programming errors. These options will give messages about such things as variables that are never used, and the tracing of `#include` files.
3. Compile your program with the `PPONLY` option to see the results of all `#define` and `#include` statements. This option also expands all macros; a macro may have a different result from the one you intended.

4. If your program was originally compiled using the `OPT(1)` or `OPT(2)` options, try to recompile it using the `NOOPTIMIZE` option, and run it. If you can successfully compile and run the program with `NOOPTIMIZE`, you have bypassed the problem, but not solved it. This does not however, exclude the possibility of an error in your program. You can run the program as a temporary measure, until you find a permanent solution.
5. If you compiled your program with either the `SEQUENCE` or the `MARGINS` option, the error may be due to a loss of code. If you compiled the source code with the `NOSEQUENCE` option, the compiler will try to parse the sequence numbers as code, often with surprising results. This can happen in a source file that was meant to be compiled with margins but was actually compiled without margins or different margins (available in z/OS C only).

   Either oversight could result in syntax errors or unexpected results when your program runs. Try recompiling the program with either the `NOSEQUENCE` or the `NOMARGINS` option.
6. Your source file may contain characters that are not supported by your terminal. You have two options at this point:
   a. Replace any characters that cannot be displayed in literals with the corresponding trigraph representation, or the corresponding escape sequence. Verify that the error did not result from using one of these incorrectly.
   b. You can use the `#pragma` filetag support and the `LOCALE` option to allow the compiler to work with non-standard code pages. See the *z/OS C/C++ Programming Guide* for more details.
7. Check for duplicate static constructors and destructors in your C++ source. Entries for constructors are created in the object and in a table. When a static constructor is removed, the entry in the object is removed, but the table entry stays. This will cause the static constructor and destructor to be called multiple times. If the destructor deletes (or frees) dynamically allocated storage that is associated with a pointer, it will tend to fail on subsequent invocations.
8. A compile-time abend can indicate an error in the compiler. An unsuccessful compilation due to an error in the source code or an error from the operating system should result in error messages, not an abend. However, the cause of the compiler's failure may be a syntax error or an error from the operating system.

The Error Occurs at IPA Link Time

1. Ensure that the region that is used for the IPA Link step is sufficient. In a number of instances where `OPT(2)` has been used with IPA Link, more than 256MB was required.
2. Ensure that the object module which defines main() contains IPA object.
3. Ensure that all application program parts (object modules, load modules) and all necessary interface libraries (Language Environment object modules and load module, SQL, CICS, etc) are made available to the IPA Link step.
4. Ensure that the IPA Compile step has processed all object modules for which source is available.
5. Use the IPA(LINK,MAP) option to obtain an IPA Link listing.
6. Do not attempt to IPA Link unsupported file formats, such as GOFF object modules or Program Objects.
7. Verify that there are no unresolved symbol references.

All user symbols must be resolved before invoking the binder (or prelinker and linkage editor). Any runtime symbol references generated by IPA Link must be resolved by the subsequent step to that no unresolved symbols remain.

8. If you have unresolved symbols, make sure that the definition of an object and all its references are used consistently in both the code area and the writable static area. Also, make sure that symbol references appear consistently in the same case.
9. If problems occur during IPA Link processing of DLL code, note that a symbol can only be imported if all of the following conditions hold true:
   - The symbol remains unresolved after `autocall`.
   - Only `DLL` references were seen for the symbol.
   - An `IMPORT` control statement was encountered for the symbol.
10. If you have unresolved symbols after using autocall, and you are searching for longnamed or writable static objects, make sure that each object module library has a current directory generated by the C370LIB utility. Without this directory, autocall can only be done on the member name of the object module and not on what is actually defined within the member.
11. A compiler ABEND during IPA Link step processing can indicate an error in the compiler. An unsuccessful IPA Link due to an error in the program source code, an invalid object module, an invalid load module, or an error from the operating system should result in error messages, not an ABEND.

    If the compiler ABEND during IPA Link step processing is related to an invalid IPA object module, it will require further diagnosis:
    - Save and recompile any IPA object modules created by a previous release of OS/390 C/C++ or z/OS C/C++. If the problem is corrected, contact IBM service and be prepared to supply the relevant source (PPONLY) and IPA object modules.
    - Try compiling at `OPT(2)`, and then `OPT(2)` plus `IPA(OBJONLY)`. If you are linking with IPA Level 2, try linking with Level 1. Ensure that you have first tried lower optimization levels.
    - Perform a binary search for the invalid IPA object module. To do this, compile one half of your source files with NOIPA (with or without `OBJONLY`), and the other half with IPA. When the IPA Link succeeds, reduce the set of NOIPA objects until you identify the compilation unit which produced the invalid IPA objects.

      Note that the object module which defines main() must always contain IPA object. It may be necessary to break the source file with main() into multiple pieces to determine the point of failure.

The Error Occurs at Bind Time

For information on bind time errors, see “Error recovery” on page 394.

The Error Occurs at Prelink Time

1. Do not prelink the object modules separately.
2. Use the prelinker option `MAP` to obtain a full map of input data sets and symbols.
3. Use the prelinker options `DUP` and `ER` to obtain a full list of duplicate and unresolved symbols.
4. If you have unresolved symbols, make sure that the definition of an object and all references to that object are used consistently in both the code area and the writable static area. Also, make sure that symbol references appear consistently in the same case.
5. A symbol can only be imported if all of the following conditions hold true:

- The symbol remains unresolved after `autocall`.
- Only `DLL` references were seen for the symbol.
- An `IMPORT` control statement was encountered for the symbol.

For more information on using `DLL`, see “Using DLLs” on page 470, or the *z/OS C/C++ Programming Guide*.

6. If you have unresolved symbols after using `autocall`, make sure that the libraries that are searched contain only object modules and no load modules. If you are searching for longnamed or writable static objects, make sure that each library has a current directory member generated by the `C370LIB` utility. Without this directory, `autocall` can only be done on the member name of the object module and not on what is actually defined within the member.
7. Only naturally reentrant code can be linked with the output of the prelinker. For more information, see the *z/OS C/C++ Programming Guide*.

The Error Occurs at Link Time

1. If you have a link-time error while working with the C/C++ component of z/OS Language Environment, you can find diagnostics and debugging information in *z/OS DFSMS Program Management* .
2. If you have a link time error while working with common execution environment (CEE) library component of z/OS Language Environment, you can find diagnostics and debugging information for link-time errors in *z/OS Language Environment Customization* and *z/OS Language Environment Debugging Guide*.

The Error Occurs at Run Time

1. If the problem occurs during execution, specify one or more of the following compiler options, in addition to the options originally specified, to produce the most diagnostic information:

Option	Information produced
`AGGREGATE`	(C only). Aggregate layout.
`ATTRIBUTE`	For C++ compile, cross reference listing with attribute information.
`CHECKOUT`	(C only). Indication of possible programming errors.
`EXPMAC`	Macro expansions with the original source.
`FLAG`	Specifies the minimum message severity level that you want returned from the compiler.
`GONUMBER`	Generates line number information that corresponds to input source files.
`INFO`	(C++ only). Indication of possible programming errors.
`INLINE`	(C only). Inline Summary and Detailed Call Structure Reports. (Specify with the REPORT suboption.)
`INLRPT`	Generates a report on status of functions that were inlined. The `OPTIMIZE` option must also be specified.
`LIST`	Listing of the pseudo-assembly listing produced by the compiler.
`OFFSET`	Offset addresses of functions in the listing.
`PPONLY`	Completely expanded C or C++ source code, by activating the preprocessor (PP) only. The output shows, for example, all the `#include` and `#define` directives.
`SHOWINC`	All included text in the listing.
`SOURCE`	Listing of the source file.
`SRCMSG`	Adds the corresponding source code lines to the diagnostic messages that are written to *stderr*.
`TEST`	To get information about the contents of variables at the point of the error, and to enable the use of the Debug Tool.

XREF	Cross reference listing with reference, definition, and modification information. If you specify ATTRIBUTE, the listing also contains attribute information. For C and C++ compile, and IPA Link, external symbol cross reference listing.

2. If the failure is in a statement that can be isolated, for example, an `if`, `switch`, `for`, `while`, or `do-while` statement, try placing the failing statement in the mainline code. If the problem is occurring as a result of a `switch` statement, make sure that you have “breaks” on all appropriate statements.
3. If you have used the compiler options RENT or NORENT in #pragma options or #pragma variable statements, and compiled your program at OPT(2), you can detect a possible pointer initialization error by compiling your program at OPT(0).
4. *z/OS Language Environment Customization* and *z/OS Language Environment Debugging Guide* describe diagnostics and debugging information for runtime errors when executing with z/OS Language Environment.
5. Check if you are running IBM C/370 Version 1 or Version 2 modules. Some IBM C/370 Version 1 and Version 2 modules may not be compatible with z/OS Language Environment. In some cases, old and new modules that run separately may not run together. You may need to recompile or relink the old modules, or change their source.*z/OS C/C++ Compiler and Run-Time Migration Guide* documents these solutions.
6. If IPA Link processed the program:
   a. Ensure that the program functions correctly when compiled NOIPA at the same OPT level.
   b. Subprograms (functions and C++ methods) which are not referenced will be removed unless appropriate ″retain″ directives are present in the IPA Link control file.
   c. IPA Link may expose existing problems in the program:
      - Ensure that any coalesced global variables which are character strings have sufficient space to contain all characters plus an additional byte for the terminating null.
      - Ensure that there are no dependencies on the order in which data items or subprograms (functions, C++ methods) are generated.
   d. Do the following to check for a code generation problem:
      - Specify a different OPT level during IPA Link processing. If the program executes correctly, contact IBM service and be prepared to supply the relevant source (PPONLY) and object modules.
      - Specify the option NOOPT during IPA Link processing. If the program executes correctly, contact IBM service and be prepared to supply the relevant source (PPONLY) and object modules.

      If the program executes correctly at a different OPT level or NOOPT, perform a binary search for the IPA object file which contains the function for which code is incorrectly generated. Contact IBM service and be prepared to supply the relevant source (PPONLY) and object modules.
   e. Do the following to check for an IPA optimization problem:
      - Specify NOINLINE IPA(LEVEL(1)) during IPA Link processing.

        If the program executes correctly, perform a binary search using INLINE IPA(LEVEL(1)) for the IPA object file which contains the function which is incorrectly optimized. Once you have located the IPA object file with the problem, use ″noinline″ directives within the IPA Link control file to determine the functions that are not correctly inlined. Contact IBM service and be prepared to supply the relevant source (PPONLY) and object modules and the IPA Link control file.

Functions that are inconsistently prototyped may cause problems of this type. Verify that all interfaces are consistent and complete.

- Specify IPA(LEVEL(0)) during IPA Link processing.

  If the program executes correctly, perform a binary search using INLINE IPA(LEVEL(1)) for the IPA object file which contains the function which is incorrectly optimized. Contact IBM service and be prepared to supply the relevant source (PPONLY) and object modules.
- Specify IPA(LEVEL(1)) instead of IPA(LEVEL(2))

  If you are linking with IPA Level 2, try linking with Level 1.

Installation Problems

You can avoid or solve most installation problems if you follow these steps:

1. Review the step-by-step installation procedure that is documented in the z/OS Program Directory that is applicable to your environment.
2. Consult the PSP bucket as described on page 7 on page 508.

If you still cannot solve the problem, develop a keyword string and contact your IBM Support Center.

You may need to reinstall the z/OS C/C++ product by using the procedure that is documented in the z/OS Program Directory. This procedure is tested for each product release and successfully installs the product.

PMR/APAR Process

If you have already attempted to diagnose your problem using the process described in the previous sections of this appendix then this section describes your next step in the diagnosis of failures in the z/OS C/C++ compiler. (The z/OS C/C++ compiler may be referred to as the z/OS C/C++ program in the rest of this section.) The z/OS C/C++ uses z/OS Language Environment as a runtime environment, but to diagnose product failures that you encounter in that runtime environment, you should consult *z/OS Language Environment Customization*.

This file assumes that you have already determined that the suspected failure is not a user error; that is, it was not caused by incorrect use of the z/OS C or z/OS C++ compilers or by an error in the logic of the application program. If a user error, is the cause of the problem, and the Problem Checklist found at the beginning of this appendix does not address your case, consult the following books for more information:

- *Debug Tool User’s Guide and Reference*
- *z/OS Language Environment Debugging Guide*

This process should help you to determine if a correction for a product failure similar to yours has been previously documented. If the problem has not been previously reported, “Preparing an Authorized Program Analysis Report (APAR)” on page 525 explains how to open a Problem Management Record (PMR) to report the problem to IBM, and if the problem is with an IBM product, how to prepare an Authorized Program Analysis Report (APAR).

If you are a customer who has an electronic link to one of the IBM databases, such as ServiceLink, you should have some knowledge of the databases before using them for searches. ServiceLink is the part of IBMLink that lets you access IBM service information online. Instead of calling the IBM support centre, you can use ServiceLink to search for service and support information, view product installation information and maintenance information, electronically report problems and receive answers, and monitor the status of APARs and Program Temporary Fixes (PTFs). Basically, running a search in ServiceLink consists of accessing Service Information Search (SIS), specifying what you want to search for and where, and then selecting the items to display, print or order from the lists of items found during your search. SIS provides access to a wide variety of service and support information about IBM products.

Isolating Reportable Problems

Failures in the z/OS C/C++ compiler can be described through the use of keywords. A keyword is a descriptive word or abbreviation assigned to describe one aspect of a product failure. A set of keywords, called a keyword string, describes the failure in detail. The procedures in this section will help you construct a keyword string that describes what you currently know about the compiler failure. This section is designed to help you specifically with failures in the z/OS C/C++ compiler, but it also contains generic procedures to follow when reporting any problem to IBM. For more information about failures that occur in the z/OS Language Environment runtime product, or for user runtime errors, see *z/OS Language Environment Debugging Guide*.

After you construct the keyword string, you can use it as a search argument against an IBM software support database, such as the Service Information Search (SIS). The database contains keyword and text information describing all current problems reported through Authorized Program Analysis Reports (APARs) and associated Program Temporary Fixes (PTFs). IBM Support Center personnel have access to the software support database and are responsible for storing and retrieving the information. Using the keyword string, they will search the database to retrieve records that describe similar known problems.

If you have IBMLink or some other connection to the IBM databases, you can do your own search for previously recorded product failures before calling the IBM Support Center.

If the keyword string matches an entry in the software support database, the search may yield a fuller description of the problem and possibly identify a correction or circumvention. Such a search may yield several matches to previously reported problems. Review each error description carefully to determine if the problem description in the database matches the failure.

If a match is not found, use the keyword string you have constructed to describe the failure when you contact the IBM support center for assistance and when you submit an APAR. Keywords are intended to ensure that identical program errors will be described with identical keyword strings. Spelling the keywords exactly as they are presented in this file is especially important for a successful match.

After you have stepped through each of the items in the Problem Checklist, to see if they apply to your problem, you can then develop a keyword string to use when you search the SIS software support database. It describes options which, if specified when you search the database, will supply you with all available diagnostic information. You will need this information to discuss the problem with your IBM support representative if your search fails to locate a fix for your problem.

If a problem occurs while you are using the z/OS C/C++ compiler, its cause may not be obvious; it might be an error in your application or in the z/OS C/C++ compiler itself. After you identify the failure, you may want to write a small test case that re-creates the problem. It will help you to:

- Pinpoint the problem
- Determine whether the error is in the application program or in the z/OS C/C++ compile
- Choose keywords that best describe the error if it is in the z/OS C/C++ compiler

If the problem appears to be the result of an error in the application program, change your source code accordingly and then compile, prelink (if required), and link the new code. If the problem appears to be the result of an error in the z/OS C/C++ compiler, develop a set of keywords using the procedure in “Keyword Usage”. Depending on your particular situation, then you can either search the SIS or contact your Service Representative.

Use the following procedures to diagnose the problem. As you go through the procedures, always note the sequence of events that leads to the error.

Keyword Usage

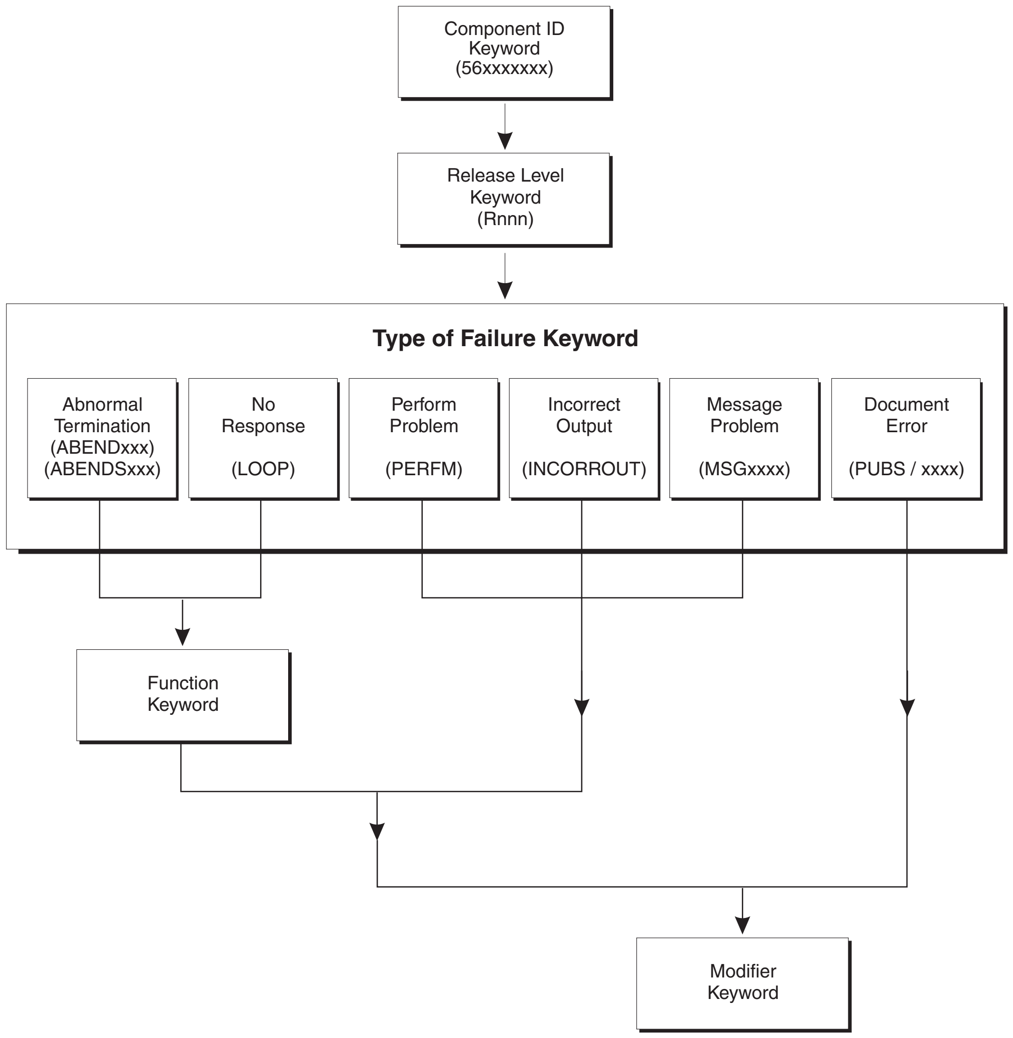
The first keyword of a keyword string is called the component identification, or ID. The component identification for the z/OS C/C++ compiler should be the component identifier, 565512101. A search of the software support database with this single keyword would locate all problems reported for the compiler. A list of component identifiers that you may need appears in “Component Identification Keyword” on page 518.

Each additional keyword added to the keyword string narrows the scope of the search and helps to eliminate unnecessary examination of problem descriptions that have similar, but not matching, characteristics. In some cases, a correction for a product failure might be located with less than a full string of keywords. If it is unclear how to select a particular keyword to describe your problem, omit that keyword to avoid incorrectly identifying the problem.

A full set of keywords for the z/OS C/C++ compiler contains:

- The component identification
- The release level
- The type of failure
- The name of the module that failed, if applicable and available
- One or more modifier keywords, depending on the type of failure, if applicable

Follow the steps in the keyword procedures until you are directed to use the keyword string in a search argument. The following figure illustrates the process of creating a keyword string.



Component ID Keyword (56xxxxxxx)
Release Level Keyword (Rnnn)
Type of Failure Keyword
Abnormal Termination (ABENDxxx) (ABENDSxxx)
No Response (LOOP)
Perform Problem (PERFM)
Incorrect Output (INCORROUT)
Message Problem (MSGxxxx)
Document Error (PUBS / xxxx)
Function Keyword
Modifier Keyword

Using the Problem Identification Worksheet

You can use the ″Problem Identification Worksheet″ to help you construct and record a keyword string. As you identify the keywords associated with your software problem, record them in the spaces provided.

```
PROBLEM IDENTIFICATION WORKSHEET

  COMPONENT IDENTIFICATION:     _________________________

  RELEASE LEVEL:                ___

  TYPE OF FAILURE:              _________________________

  MODULE:                       ___________

  SYSTEM TYPE:                  _________________________

  MODIFIERS:                    _________________________

                                _________________________

                                _________________________

  NOTE:  Some keywords may not be applicable to all problems.
```

Component Identification Keyword

This procedure shows you what to specify in the component identification keyword. It is always the first keyword placed in the search argument string. It comes from the z/OS C/C++compiler program number and limits the search to the area within a software support database such as SIS, which contains items for the z/OS C/C++ compiler.

Component identifiers you may need are:

- 565512101 z/OS C/C++ compiler
- 565512102 IBM z/OS C/C++ class library
- 568819421 Debug Tool
- 5655A4501 z/OS C/C++ Performance Analyzer
- 568819801 Common Execution Environment (CEE) library component of z/OS Language Environment
- 568819805 z/OS C/C++ Language component of z/OS Language Environment

The component identification keyword should be used with at least a type-of-failure keyword to search the software support database. If it is used without additional keywords, a full listing of all items affecting the z/OS C/C++ compiler will be produced.

1. Use 565512101 as the component identification keyword for the z/OS C/C++ compiler. (The component identification for the z/OS C/C++ language component of z/OS Language Environment is 568819805. Diagnostics for z/OS Language Environment are explained in *z/OS Language Environment Customization* and *z/OS Language Environment Debugging Guide*.)
2. If service tapes have been applied to the licensed program, note the tape level of the last service tape applied. See your system programmer for the current service level of your z/OS C/C++ compiler and z/OS Language Environment. Although the service tape level is not used as a keyword search argument, it will be useful when reviewing APARs selected during the keyword search.
3. Continue the diagnostic procedure with “Release Level Keyword” on page 519.

Release Level Keyword

The release level keyword identifies the specific release level of the z/OS C/C++ compiler, z/OS Language Environment, and the operating system you were using when the failure occurred.

1. If you do not know the release level code for your product level, you can find it in the appropriate program directory. The release level keyword is the release level code for your system prefixed with an R. If the release code for your product is 609, the release level keyword is R609. Some release level keywords you may use are:

R609	z/OS C compiler and utilities
R619	z/OS C++ compiler and utilities
R629	Kanji compiler messages
R609	Standard class libraries
R619	Collection class and application support class libraries
R629	Kanji class library messages
R639	HFS class library

2. Continue the diagnostic procedure with "Type-of-Failure Keyword".

Type-of-Failure Keyword

A specific failure may occur in the z/OS C/C++ licensed program. For product failures that occur in the z/OS Language Environment runtime library, consult *z/OS Language Environment Customization* or *z/OS Language Environment Debugging Guide*.

Read the following table and select the type of failure that best describes your problem. Refer to the associated keyword procedure for instructions on how to complete the keywords for that type of failure. If more than one keyword describes your problem, use the one that appears first in the table.

Type of Failure	**Symptom**	**Procedure**
Abnormal Termination	The compiler or program terminated abnormally with a completion code that indicates that an abend has occurred.	See "Abnormal Termination" on page 520
Message Problems	The compiler or program issued a message that is inappropriate or not valid.	See "Message Problems" on page 520
No Response from the Compiler	An unexpected compiler or program suspension occurred; no response has been received in interactive mode.	See "No Response Problems" on page 521
Documentation Problems	Information in one of the z/OS C/C++ publications is incorrect or missing.	See "Documentation Problems" on page 521
Output Problems	The output from the compiler or program is missing or not valid.	See "Output Problems" on page 522

Type of Failure	Symptom	Procedure
Performance Problems	The performance of a z/OS C/C++ compilation or program is degraded.	See “Performance Problems” on page 523

Abnormal Termination

If the z/OS C/C++ compiler terminates abnormally you will get one or more of the following:

- A message from z/OS Language Environment
- A traceback
- A system abend, such as an 0C4

If you get a z/OS Language Environment message but no traceback after an abnormal compiler termination, use z/OS Language Environment Debugging Guide and Run-Time Messages to diagnose the problem.

If you get a message from the operating system indicating a system abend but no traceback, follow the procedure described in the following information as the ″ABENDxxx Procedure.″

If you get a traceback, use the procedure described in “Function Keyword” on page 523. Record any other messages you get. You will need the other messages in order to do your keyword search or to report the problem.

Note: Do not use the abnormal termination procedures if termination was forced because too much time was spent in a wait state or an endless loop. Under MVS, this is usually accompanied by a system abend code x22. For example, an abend code of 322 indicates time exceeded and an abend code of 722 indicates lines exceeded. In this situation, refer to the procedure for “No Response Problems” on page 521.

Use the ABENDxxx procedure if the z/OS C/C++ compiler terminates abnormally with a system abend code. The following instructions help identify the ″ABENDxxx″ keyword needed for your keyword string.

1. Replace the ″xxx″ of ″ABENDxxx″ with the system abend code. For example, if the failure occurred during z/OS C compilation with a system abend ″0C4″, you would specify ″ABEND0C4″ as your keyword. Your keyword string at this point would appear as follows:

   565512101 R609 ABEND0C4

   The compiler phase is shown in the listing. Use the name of the current compiler phase as a modifier keyword. For example, if the compiler phase was ″CBC3P″, your keyword string would appear as follows:

   565512101 R609 ABEND0C4 CBC3P
2. Continue with “Modifier Keywords” on page 524.

Message Problems

The z/OS C/C++ compiler issues messages prefixed with ″EDC″ or ″CBC″. Messages with other prefixes are issued by the operating systems, subsystems, access methods or the common execution environment (CEE) library component of z/OS Language Environment runtime environment. These are not z/OS C/C++ product problems. See the message listings in the appropriate component manuals.

Use the MSGx keyword in your keyword string for any one of the following conditions:

- A message is issued when it should not have been issued
- A message contains data that is not valid or data is missing.

To construct the MSGx keyword:

1. Replace the x of MSGx with the message identifier in the format _ MSGaaannnn. For example, if the compiler message received is ″EDC0041 30″, the MSGx keyword would be MSGEDC0041. The severity code, ″30″ in this case, is not included in the MSGx keyword. Your set of keywords, so far, would look like this: 565512101 R609 MSGEDC0041
2. Proceed with “Modifier Keywords” on page 524.

No Response Problems

Use the LOOP keyword procedure for any of the following conditions:

- The z/OS C/C++ compiler seems to be doing nothing or appears to be in a loop.
- A terminal response is not received in interactive mode after the compiler is invoked.
- The compiler does not reach completion in batch mode.
- The system abend code is 322.

If the problem looks like a ″WAIT″ state and the compiler is not waiting for input from the terminal or console, it is probably a system problem. In this case, follow your local procedures for resolution, otherwise try the following suggestions:

- If you are running in batch mode and the error is a system abend with abend code 322, indicating not enough time, increase the time allotment and re-compile. If the problem is still unresolved, your set of keywords would now look like this: 565512101 R609 LOOP
- Continue with “Function Keyword” on page 523.

Documentation Problems

Follow this PUBS keyword procedure when you notice a problem caused by incorrect or missing information in one of the z/OS C/C++ hardcopy or softcopy documents.

1. If the existing information is wrong, locate the page or pages in the document or the online panel where the problem occurs, and prepare a description of the error and the problem it caused. If it is missing, locate the place where you think it should appear. This information will be required for APAR preparation if no similar problem is found in the software support database.
2. Decide whether this documentation problem is severe enough to cause lost time for other users. If the problem is not severe, fill out the Readers’ Comments Form (RCF) attached to the back of the publication in question. If the RCF is missing, send a note to the address shown on the edition notice for this book. Include the problem description you have developed, along with your name and return address, so that IBM can respond to your comments. If the problem is severe enough to cause lost time for other users, continue creating your keyword string to determine whether IBM has a record of the problem. If this is a new problem, a severity 3 or 4 documentation (PUBS) APAR will be created.
3. In the case of hardcopy or softcopy books, use the order number on the cover of the document along with the PUBS keyword as your type-of-failure keyword, but omit the hyphens. For example, if the number on the cover is SC09-2360-04 (the *z/OS C/C++ Language Reference*) then use SC09236004. Place a forward

slash '/' between PUBS and the document number. Your set of keywords would now consist of: 565512101 R609 PUBS/SC09236004

4. To determine if this documentation problem has already been reported, see "Using the Keyword String as a Search Argument" on page 525. If, after searching the IBM software support database, you do not find a matching description, return here to continue.
5. Before discontinuing your database search, you may want to search again, using the wildcard character '*' to replace a single character in the search string, as in the following format: 565512101 R609 PUBS/SC092360** In case several levels of the document exist, you can use two asterisks appended to the document number to search for all problems reported for the document rather than only those for a specific release of the document.
6. Go to "Using the Keyword String as a Search Argument" on page 525.

Output Problems

Use this procedure when the output appears to be incorrect or missing, but the z/OS C/C++ compiler otherwise terminated normally. If the data or records were repeated endlessly, follow the steps under "No Response Problems" on page 521 instead of using the ″INCORROUT″ procedure to create your keyword string.

1. Use ″INCORROUT″ as your type-of-failure keyword.
2. Select a modifier keyword from the following table to describe the type of error in the output. For more information on the use of modifier keywords, see the section "Modifier Keywords" on page 524.

Modifier Keyword	Type of Incorrect Output
MISSING	Some expected output was missing
DUPLICATE	Some data or records were duplicated, but were not repeated endlessly
INVALID	The output that appeared was not as expected; that is, the output was bad or incorrect

3. Select another modifier keyword from the following table to describe the portion of the output where the error occurred.

Modifier Keyword	Portion of Output in Error
AGGREGATE (C only)	Structure maps
ATTRIBUTE (C++ only)	List of identifiers
SOURCE	Source listing
OBJECT	Machine-language object program
XREF	Cross-reference listing
OFFSET	Storage offset listing
INLINE (C only)	Inline report
PPONLY	Preprocessor output (for z/OS C, this goes to SYSUT10 DD)
LIST	Assembler language expansion of source listing
MESSAGE	Diagnostic messages
TERM	Progress and diagnostic messages on the SYSTERM data set

Modifier Keyword	Portion of Output in Error
OFFSET	Offset listing
EXPMAC	Macro expansions
INLRPT	Inline report

For example, if you think that the compiler has given an incorrect cross-reference listing, your keyword string so far would contain the following:
565512101 R609 INCORROUT INVALID XREF

4. Continue the diagnostic procedure with "Modifier Keywords" on page 524.

Performance Problems

Most performance problems can be related to system tuning and should be handled by system engineers and system programmers. Use this keyword procedure when the performance problem could not be corrected by system tuning and performance is significantly below explicitly stated expectations.

1. Use PERFM as your type-of-failure keyword. For example, your keyword string for performance problems of the z/OS C compiler might look like this:
   565512101 R609 PERFM
2. Continue the diagnostic procedure with "Modifier Keywords" on page 524.

Function Keyword

For product failures that occur at z/OS Language Environment runtime, see *z/OS Language Environment Customization* and for user runtime problems, see *z/OS Language Environment Debugging Guide*.

Should the compiler terminate abnormally, the following procedure will help you determine if there are function or language element modifier keywords that you should use in your keyword string. Use this procedure to locate the point of compiler failure and the language element you were using when the failure occurred.

The z/OS C/C++ compiler is itself a z/OS C/C++ program, and runs under z/OS Language Environment. If an error occurs in the compiler, it is handled by z/OS Language Environment which produces a traceback with information about the compiler error. The output is directed to ″DD CEEDUMP″.

How to Find the Function Name in the Traceback

```
Status
Call
Exception
Call
Call
Call
52BA121C
CEE3DMP: Condition processing resulted in the Unhandled condition.

Information for enclave main

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E
Offset   Statement  Status
    00D63018  CEEHDSP       00CD15B8  +00001FB0  CEEHDSP       00CD15B8
+00001FB0            Call
    00D1CA40                0250DCC8  +0000008C  Tokenizer     0250DCC8
```

```
+0000008C             Exception
    00D1C950               0239D8B8  +00000474  CBCP          0239D8B8
+00000474             Call
    00D1C8A0               029108A0  +0009940C  @@FECBCBC130  029108A0
+0009940C             Call
    00D1C7B8               02D73470  +000000E4  InvokeFetch   02D73470
+000000E4             Call
    00D1C1E0               02D784E0  +000002BA  main          02D784E0
+000002BA             Call
    00D1C0C8               02ABCE46  +000000B0  @@MNINV       02ABCE46
+000000B0             Call
    00D1C018  CEEBBEXT     00CBF4B0  +00000138  CEEBBEXT      00CBF4B0
+00000138             Call

  Condition Information for Active Routines
    Condition Information for  (DSA address 00D1CA40)
      CIB Address: 00D633C8
      Current Condition:
        CEE3204S The system detected a Protection exception.
      Location:
        Program Unit:  Entry: Tokenizer Statement:  Offset: +0000008C
      Machine State:
        ILC..... 0004    Interruption Code..... 0004
        PSW..... 03EC0400 8250DD58
        GPR0..... 00D1CC08  GPR1..... 00D1CB08  GPR2..... 00000000
        GPR3..... 00000000
⋮
```

1. Look in the text that follows the word ″Traceback:″ in the sample traceback above.
2. Find the row that contains the word ″Exception″ in the ″Status″ column.
3. Use the function name found in the ″Entry″ column in that row. (Enter it in SIS in capital letters).

If the compiler failed and you received a traceback like in the previous figure, you would use ″TOKENIZER″ as your function keyword. At this point your keyword string would look like this: 565512101 R609 ″TOKENIZER″

If the failure is peculiar to a compiler statement or library function, use the statement or function name as a modifier keyword. For example, if the source of failure was the z/OS C/C++ library function ″ctime()″, your keyword string would look like this: 565512101 R609 CTIME

See “Modifier Keywords” for information about using modifier keywords in your keyword search.

Modifier Keywords

You can use one or more modifier keywords in the same keyword string to define the compiler problem. They help to make the search argument more specific. Capitalize all of the letters of the modifier in the keyword string. The various types of modifier keywords include:

- z/OS C/C++ compiler
  - Language Elements
  - Listing Control Statements
  - Compiler options (Select from your compiler listing those compiler options that you consider significant to the type of failure. The option name itself is the keyword.)
  - Message IDs
- z/OS Language Environment Library

- Library functions
- Link-edit options
- Run-time options

If the failure appears to be associated with any particular compiler option or options, such as SHOWINC, use those options as modifier keywords, as in this example: 565512101 R609 SHOWINC

Continue the diagnostic procedure with "Using the Keyword String as a Search Argument".

Using the Keyword String as a Search Argument

Searches using the SIS in ServiceLink will be most successful if you follow these rules:

- Spell keywords the way they are spelled in this file. Any variation in spelling may result in an unsuccessful search.
- Include all the appropriate keywords in any discussion with IBM support personnel or in an APAR.

The following explains how to use the keyword string as a search argument against a software support database,such as SIS.

1. Search the software support database, using the full set of keywords you have developed. Here is an example: 565512101 R609 ABEND0C4 CBC3JPRO FUNCTION
2. If the search produces a list of APARs, continue with step 3; otherwise, go to step 6.
3. When your search is complete, eliminate from the list of possible PTFs those that have already been applied to your system.
4. Compare each of the remaining closed APAR descriptions and PTF descriptions with the current failure symptoms.
5. If a match is found, apply the program temporary fix (PTF) to your system and exit from this procedure.
6. If the search did not produce a list of APARs, or you did not find an APAR description matching the current failure, broaden the search, using the following techniques:
   a. Omit the release level keyword (for example, R609) from the search argument, thereby broadening the search to include similar failures on other release levels.
   b. Drop one keyword from the right end of the search argument string.

      By dropping a keyword from the right, you broaden your search while maintaining the relevancy of your search argument string. Perform the search against the software support database, using your shortened search argument string. Repeat this step as necessary, dropping keywords from the right of the string until only the COMPID is left.
7. If a match is not found after you have used the preceding methods, go to "Preparing an Authorized Program Analysis Report (APAR)".

Preparing an Authorized Program Analysis Report (APAR)

A Problem Management Record (PMR) can be opened after you have done the following:

1. Eliminated user errors as a possible cause of the problem.

2. Followed the diagnostic procedures.
3. You or your local IBM Support Center has been unsuccessful with the keyword search.

If you have IBMLink or some other connection to IBM databases, you can open a PMR yourself. Otherwise, the IBM Support Center may open the PMR after consulting with you on the phone. You may be asked for any of the documentation in the list below. The PMR is used to document your problem and to record the work that the Support Center does on the problem. If IBM concludes that the problem described in the PMR is a problem with the z/OS C/C++ product, they will work with you to open an Authorized Program Analysis Report (APAR) so the problem can be fixed. After the APAR is opened and the fix is produced, the description of the problem and the fix will be in the software support database in SIS, accessible through ServiceLink.

Before you initiate an APAR:

1. Contact the IBM Support Center for assistance. You will have been in contact with the IBM Support Center while they were working on the PMR. Be prepared to supply the following information:
   - Customer number and security code
   - PMR number
   - Operating system
   - Operating system release level
   - Current z/OS C/C++ compiler maintenance level (PTF list and list of APAR fixes applied), which can be determined by specifying the PHASEID compiler option
   - The various keyword strings used to search the software support database
   - Processor number (model and serial)
2. From the following list, you may be asked to include the applicable z/OS C/C++ environmental information with your APAR:
   - Job control statements
   - Compiler listings, including:
     - Source listing
     - Object listing
     - Storage map
     - Traceback
     - Cross-reference listing

     Use LIST, MAP, SOURCE, XREF and any other options pertinent to the problem.
   - A machine-readable copy of the program causing the problem, including all ″#include″ files required by the program
   - A dump on tape if available
   - The compiler on tape if requested by an IBM representative
   - A hard copy of the job control language for unloading the submitted machine-readable tape
   - Any other data that may help in re-creating the problem

In addition, a description of the application, the data set organization, and the operating instructions or console log may be helpful in reproducing the error. Any listings you supply must be from the z/OS C/C++ compilation version that failed.

The following table describes how to produce documentation required for submission with the APAR. Additional requirements are explained after the table. Many of these materials may already have been produced in their required format during the formulation of the keyword string. (See "Isolating Reportable Problems" on page 515.)

Item	Materials Required	How to Obtain Materials
1	Machine-readable source program	The source program must be supplied in machine-readable form, generated by an IBM-supplied system utility program. The source program should be reduced to the smallest, least complex form that still produces the error.
2	Compiler listings: Source listing Cross-reference listing Assembler—language expansion	SOURCE option XREF option LIST option
3	Compiler termination dump	SYSMDUMP DD statement (as directed by IBM support personnel)
4	Partition/region size/virtual storage size	JCL or system programmer
5	List of applied PTFs	System programmer
6	Operating instructions or console log	Application programmer
7	Job control statements with MSGLEVEL(1,1), TSO ALLOCATE statements	The JCL that was used to invoke and run the compiler should be supplied in machine-readable form.

Preparation of Material

When submitting material for an APAR to IBM, be sure that the media containing source programs, job stream data, or interactive environment information are carefully packed and clearly identified. Each magnetic tape submitted must have the following information attached and visible:

1. The PMR number assigned by IBM.
2. A list of data sets on the tape (such as source program, JCL or interactive environment information).
3. A description of how the tape was made, including the following information:
   a. A full listing of JCL or interactive environment information used to produce the machine-readable source. Include the block size, LRECL, and format of each file. If the file was unloaded from a partitioned data set, include the block size, LRECL and number of directory blocks in the original data set.
   b. Labeling information used for the volume and its data sets.
   c. The recording mode and density.
   d. The name of the utility program that created each data set.
   e. The record format and block size used for each data set.

Maintenance

IBM Software Manufacturing Solutions (ISMS) provides corrective and preventive service for product defects, as well as support for resolving program problems. This is done through Central Service, including the IBM Support Center. For details of how these facilities work and a list of all the products supported, refer to *Field Engineering Programming System General Information*, G229-2228.

IBM distributes, when necessary, service updates in response to problems found in IBM licensed programs. When problems are identified and solutions established, a PTF is created and made available.

Corrective Service — Program Temporary Fixes (PTFS)

A program temporary fix (PTF) is a correction for a known problem or problems in a particular product. When a problem is identified and its solution established, a PTF is made available on tape. You can order this tape from IBM. You can also order a cumulative tape containing all the PTFs for your site.

A PTF arrives as a System Modification Program (SMP/E) data set accompanied by all the materials necessary for its installation and use. If you receive the PTF on tape, you must copy the PTF from the tape into a data set.

You should create a backup copy of the current compiler before the PTF tape is applied.

Preventive Service — Extended Service Offering Tapes (ESOS)

Extended service offering tapes (ESOS) can be ordered from IBM Software Manufacturing Solutions (ISMS). These tapes contain the PTFs to problems in IBM licensed programs under your operating system. Note that each ESO tape contains only one month's PTFs. Therefore, to bring z/OS C/C++ up to the latest level, you must install all prior ESO tapes sequentially, starting with the oldest.

Installing Corrective or Preventive Service

For specific procedures, always refer to the installation materials provided with the service. For product-specific information, refer to the appropriate program directory for service maintenance. The *SMP/E User's Guide*, SC28-1302 will provide detailed procedures for corrective and preventative service.

Appendix D. Cataloged Procedures and REXX EXECs

This appendix describes the REXX EXECs (TSO) and cataloged procedures that the z/OS C/C++ compiler provides in conjunction with z/OS Language Environment, to call the various z/OS C/C++ utilities.

When you specify a data set name without enclosing it in single quotation marks ('), your user prefix will be added to the beginning of the data set name. If you enclose the data set name in quotation marks, it is treated as a fully qualified name.

For more information on the REXX EXECs and EXECs that z/OS Language Environment provides, and on the cataloged procedures that do not contain a compile step, see *z/OS Language Environment Programming Guide*.

For a description of `CXXBIND` see "Chapter 12. Binding z/OS C/C++ Programs" on page 353. For a description of `CXXMOD` see "Prelinking and Linking under TSO" on page 485. For a list of the old syntax REXX EXECs, see "Other z/OS C Utilities" on page 551.

	Name	**Task Description**
REXX EXECs for z/OS C and z/OS C++	C370LIB	Maintain an object library under TSO
	CXXBIND	Generate an executable module under TSO
	CXXMOD	Generate an executable module under TSO
	DLLRNAME	Run the DLLRNAME utility
Cataloged Procedures for z/OS C and z/OS C++	EDCDLLRN	Rename DLLs with the DLLRNAME utility
	EDCLIB	Maintain an object library
REXX EXECs for z/OS C	CC	Compile (new syntax - recommended approach)
	CDSECT	Run DSECT utility
	CPLINK	Interactively prelink and link a C program
	GENXLT	Generate a translate table
	ICONV	Run the character conversion utility
	LOCALEDEF	Produce a locale object

	Name	Task Description
Cataloged Procedures for z/OS C	CEEWG	Run
	CEEWL	Link
	CEEWLG	Link and run
	CEEXL	Bind an XPLINK z/OS C program
	CEEXLR	Bind and run an XPLINK z/OS C program
	CEEXR	Run an XPLINK z/OS C program
	EDCC	Compile
	EDCCB	Compile and Bind
	EDCCBG	Compile, bind, and run
	EDCCL	Compile and link-edit
	EDCCLG	Compile, link-edit, and run
	EDCCLIB	Compile and maintain an object library
	EDCI	Run IPA Link step
	EDCPL	Prelink and link-edit
	EDCCPLG	Compile, prelink, link-edit, and run.
	EDCDSECT	Run the DSECT Conversion Utility
	EDCGNXLT	Generate a translate table
	EDCICONV	Run the character conversion utility
	EDCLDEF	Produce a locale object
	EDCXCB	Compile and bind an XPLINK z/OS C program
	EDCXCBG	Compile, bind, and run an XPLINK z/OS C program
	EDCXLDEF	Create z/OS C source from a locale, compile, and bind the XPLINK program to produce an XPLINK locale object
REXX EXECs for z/OS C++	CXX	Compile under TSO

	Name	Task Description
Cataloged procedures for z/OS C++	CBCC	Compile
	CBCCB	Compile and bind
	CBCCBG	Compile, bind and run
	CBCB	Bind
	CBCBG	Bind and run
	CBCCL	Compile, prelink and link
	CBCCLG	Compile, prelink, link and run
	CBCG	Run
	CBCI	Run IPA Link step
	CBCL	Prelink and link
	CBCLG	Prelink, link and run
	CBCXB	Bind an XPLINK z/OS C++ Program
	CBCXBG	Bind and run an XPLINK z/OS C++ Program
	CBCXCB	Compile and bind an XPLINK z/OS C++ program
	CBCXCBG	Compile, bind, and run an XPLINK z/OS C++ program
	CBCXG	Run an XPLINK z/OS C++ program

Tailoring PROCs, REXX EXECs, and EXECs

Your system programmer must modify the PROCs, and REXX EXECs before they are used. For example, the prefix symbolic parameters `LIBPRFX` and `LNGPRFX` should be changed from the defaults supplied by IBM to the high-level qualifier that you chose to install the z/OS C/C++ compiler and z/OS Language Environment.

The following data sets contain the PROCs and REXX EXECs that are to be modified:

- `CBC.SCBCPRC`
- `CBC.SCBCUTL`
- `CEE.SCEEPROC`
- `CEE.SCEECLST`

The IBM-supplied cataloged procedures provide many parameters to allow each site to customize them easily. The table below describes the commonly used parameters. Use only those parameters that apply to the cataloged procedure you are using. For example, if you are only compiling (`EDCC`), do not specify any binder parameters.

Parameter	Description
INFILE	For compile procedures, the input z/OS C/C++ source file name, PDS name of source files, or directory name of source files. For IPA link procedures (EDCI and CBCI), the input IPA object. For prelink, link and bind procedures, the input object. If you do not specify the input data set name, you must use JCL statements to override the appropriate SYSIN DD statement in the cataloged procedure.
OUTFILE	Output module name and file characteristics for procedures that do not have an execution step (EDCC, EDCCL, and EDCPL). For the cataloged procedures containing a link-edit step, specify the name of the file where the load module is to be stored. For cataloged procedures without a link-edit step, specify the name of the file where the object module is to be stored. If you do not specify an OUTFILE name, a temporary data set will be generated.
CPARM	Compiler options: If two contradictory options are specified, the last is accepted and the first ignored.
BPARM	Bind utility options: If two contradictory options are specified, the last is accepted and the first ignored.
IPARM	IPA Link step options: If two contradictory options are specified, the last is accepted and the first ignored.
PPARM	Prelink utility options: If two contradictory options are specified, the last is accepted and the first ignored.
LPARM	Linkage-editor options: If two contradictory options are specified, the last is accepted and the first ignored.
GPARM	Language Environment runtime (Go step) options and parameters: If two contradictory Language Environment runtime options are specified, the last is accepted and the first ignored.
CRUN	Compile step execution runtime parameters for the z/OS C/C++ compiler.
IRUN	IPA Link step runtime parameters: for the z/OS C/C++ compiler.
OPARM	Object Library Utility parameters. Required for EDCLIB.
OBJECT	Object module to be added to the library. The data-set name (DSN=...) and any applicable keyword parameters (such as, DCB, DISP,) can be specified using this parameter. The default is OBJECT=DUMMY. OBJECT is required for EDCLIB if the ADD function is selected.
LIBRARY	Data-set name for the library for the requested function (ADD, DEL, MAP, or DIR). An example is LIBRARY='FRED.LIB.OBJ'. LIBRARY is required for EDCLIB and EDCCLIB.
MEMBER	Member of the library to contain the object module. An example is MEMBER='MYPROG'. In z/OS C, MEMBER is required for EDCCLIB.

Data Sets Used

The following table gives a cross-reference of the data sets that each job step requires, and a description of how the data set is used. Refer to the input/output section of the *z/OS C/C++ Programming Guide* for more information about the attributes that are used when opening different types of files.

Table 52. Cross Reference of Data Set Used and Job Step

DD Statement	COMPILE	IPA Link	BIND	PLKED (Prelink)	LKED (Link-Edit)	GO (Run)	EDCALIAS (Object Library)
STEPLIB[1]	X	X	X	X		X	X
SYSCPRT	X	X					
SYSIN	X	X	X	X	X		X
SYSLIB	X	X	X	X	X		X
SYSLIN	X	X	X		X		
SYSLMOD			X		X		
SYSMOD				X			
SYSMSGS				X			X
SYSOUT	X	X		X			X
SYSPRINT	X	X		X	X	X	X
SYSUTx	X	X			X (SYSUT1)		
IPACNTL		X					

Note: [1] Optional data sets, if the compiler and runtime library are installed in the LPA or ELPA. To save resources (especially in z/OS UNIX System Services), do not unnecessarily specify data sets on the STEPLIB ddname.

Description of Data Sets Used

The following table lists the data sets that the IBM-supplied cataloged procedures use. It describes the uses of the data set, and the attributes that it supports. You require compiler work data sets only if you specified `NOMEM` at compile time.

Note: You should check the defaults at your site for `SYSOUT=*`

Table 53. Data Set Descriptions for Cataloged Procedures

In Job Step	DD Statement	Description and Supported Attributes (You should check the defaults at your site for `SYSOUT=*`)
COMPILE	SYSIN	For a C++, C, or IPA compilation, the input data set containing the source program. `RECFM=VS, V, VB, VBS, F, FB, FBS, or FS, LRECL≤32760`. It can be a PDS.
COMPILE	SYSLIB	For a C++, C, or IPA compilation, the data set for z/OS C/C++ system header files for a source program. SYSLIB must be a PDS (`DSORG=PO`) *and* `RECFM=VS, V, VB, VBS, F, FB LRECL≤32760`. For more information on searching system header files, see “SEARCH | NOSEARCH” on page 167.

Table 53. Data Set Descriptions for Cataloged Procedures *(continued)*

In Job Step	DD Statement	Description and Supported Attributes (You should check the defaults at your site for SYSOUT=*)
COMPILE	SYSLIN	Data set for object module. One of the following: • RECFM=F or FS • RECFM=FB or FBS. It can be a PDS.
COMPILE	SYSOUT	Data set for displaying compiler error messages. LRECL=137, RECFM=VBA, BLKSIZE=882. (Defaults for SYSOUT=*).
COMPILE	STEPLIB	Data set for z/OS C/C++ compiler runtime library modules. STEPLIB must be a PDS (DSORG=PO) with RECFM=U, BLKSIZE≤32760.
COMPILE	SYSCPRT	Output data set for compiler listing. LRECL=137, RECFM=VBA, BLKSIZE=882 (default for SYSOUT=*)
COMPILE	SYSUT1 and SYSUT4	Work data sets. LRECL=80 and RECFM=F or FB or FBS.
COMPILE	SYSUT5, SYSUT6, SYSUT7, SYSUT8, and SYSUT14	Work data sets. LRECL=3200, RECFM=FB, and BLKSIZE=3200**n* (where *n* is an integer value).
COMPILE	SYSUT9	Work data set. LRECL=137, RECFM=VB, and BLKSIZE=137**n* (where *n* is an integer value) in z/OS C, or 882 in z/OS C++.
COMPILE	SYSUT10	PPONLY output data set. 72≤LRECL≤32760, RECFM=VS, V, VB, VBS, F, FB, FBS or FS (if not pre-allocated, V is the default). It can be a PDS.
COMPILE	SYSEVENT	Events output file. Must be allocated by the user.
COMPILE	TEMPINC C++ only	Template instantiation file. Must be a PDS. 72≤LRECL≤32760, RECFM=VS, V, VB, VBS, F or FB (default is V).

Table 53. Data Set Descriptions for Cataloged Procedures (continued)

In Job Step	**DD Statement**	**Description and Supported Attributes (You should check the defaults at your site for SYSOUT=*)**
COMPILE	USERLIB	User header files. Must be a PDS. LRECL≤32760, and RECFM=VS, V, VB, VBS, F or FB. For more information on searching user header files, see "SEARCH | NOSEARCH" on page 167.
IPA Link	SYSIN	Data set containing object module for the IPA Link step. LRECL=80 and RECFM=F or FB.
IPA Link	IPACNTL	IPA Link control file directives. RECFM=VS, V, VB, VBS, F, FB, FBS, or FS, LRECL≤32760. It can be a PDS.
IPA Link	SYSLIB	IPA Link step secondary input. SYSLIB can be a mix of two types of libraries: • Object module libraries. These can be PDSs (DSORG=PO) or PDSEs, with attributes RECFM=F or RECFM=FB, and LRECL=80. • Load module libraries. These must be PDSs (DSORG=PO) with attributes RECFM=U and BLKSIZE≤32760. SYSLIB must be cataloged.
IPA Link	SYSLIN	Data set for object module. One of the following: • RECFM=F or FS • RECFM=FB or FBS It can be a PDS.
IPA Link	SYSOUT	Data set for displaying compiler error messages. LRECL=137, RECFM=VBA, BLKSIZE=882. (Defaults for SYSOUT=*).
IPA Link	STEPLIB	Data set for z/OS C/C++ compiler/runtime library modules. STEPLIB must be a PDS (DSORG=PO) with RECFM=U, BLKSIZE≤32760.
IPA Link	SYSCPRT	Output data set for IPA Link step listings. LRECL=137, RECFM=VBA, BLKSIZE=882 (default for SYSOUT=*).
IPA Link	SYSUT1 and SYSUT4	Work data sets. LRECL=80 and RECFM=F or FB or FBS.

Table 53. Data Set Descriptions for Cataloged Procedures (continued)

In Job Step	DD Statement	Description and Supported Attributes (You should check the defaults at your site for SYSOUT=*)
IPA Link	SYSUT5, SYSUT6, SYSUT7, SYSUT8, and SYSUT14	Work data sets. LRECL=3200, RECFM=FB, and BLKSIZE=3200**n* (where *n* is an integer value).
IPA Link	SYSUT9	Work data set. LRECL=137, RECFM=VB, and BLKSIZE=137**n* (where *n* is an integer value).
BIND	SYSDEFSD	Output from binding a DLL (an application that exports symbols). LRECL=80 and RECFM=F or FB or FBS
BIND	SYSIN	Data set containing object module for the binder. LRECL=80 and RECFM=F, FB or FBS.
BIND	SYSLIB	Data set for binder automatic call library.
BIND	SYSLIN	Primary input data set for the binder One of the following: RECFM=F or FS RECFM=FB or FBS BIND SYSLMOD Output Program Object Library. PDSE with RECFM=U and BLKSIZE<=32760.
BIND	SYSLMOD	Output Program Object Library. PDSE with RECFM=U and BLKSIZE<=32760.
BIND	SYSPRINT	Data set for listing of binder diagnostic messages. LRECL=137, RECFM=VBA, BLKSIZE=882. (Default attributes for SYSOUT=*).
PLKED	STEPLIB	Data set containing prelink utility modules. STEPLIB must be a PDS (DSORG=PO) *and* RECFM=U and BLKSIZE≤32760.
PLKED	SYSDEFSD	Output from prelinking a DLL (an application that exports symbols). LRECL=80 and RECFM=F or FB or FBS
PLKED	SYSIN	Data set containing object module for the prelink utility. LRECL=80 and RECFM=F, FB or FBS.
PLKED	SYSLIB	Data set for prelinkage automatic call library. SYSLIB must be cataloged *and* LRECL=80 and RECFM=F or FB or FBS. DSORG=PO

Table 53. Data Set Descriptions for Cataloged Procedures (continued)

In Job Step	DD Statement	Description and Supported Attributes (You should check the defaults at your site for SYSOUT=*)
PLKED	SYSMOD	Data set for output of the prelink utility LRECL=80 and RECFM=F or FB or FBS.
PLKED	SYSMSGS	Data set containing prelink utility messages. LRECL=150, RECFM=F or FB or FBS and BLKSIZE=6150.
PLKED	SYSOUT	Data set for the prelinker map. LRECL=80 and RECFM=F or FB or FBS
PLKED	SYSPRINT	Data set for listing of prelink utility diagnostic messages. LRECL=137, RECFM=VBA, BLKSIZE=882. (Default attributes for SYSOUT=*).
LKED	SYSLIB	Data set for z/OS C/C++ autocall library. SYSLIB must be a PDS (DSORG=PO) *and* have the attributes RECFM=U and BLKSIZE≤32760.
LKED	SYSLIN	Primary input data set for linkage editor One of the following: • RECFM=F or FS • RECFM=FB or FBS
LKED	SYSLMOD	Output load module library. RECFM=U and BLKSIZE≤32760.
LKED	SYSPRINT	Data set for listings and diagnostics produced by the linkage editor. One of the following: • LRECL=121, and RECFM=FA • LRECL=121, RECFM=FBA, and BLKSIZE=121**n* (where *n* is less than or equal to 40).
LKED	SYSUT1	Work data set. The data set attributes will be supplied by the linkage editor.
GO	STEPLIB	Runtime libraries. STEPLIB must be a PDS (DSORG=PO) *and* have the attributes RECFM=U and BLKSIZE≤32760.

Table 53. Data Set Descriptions for Cataloged Procedures (continued)

In Job Step	DD Statement	Description and Supported Attributes (You should check the defaults at your site for SYSOUT=*)
GO	CEEDUMP	Data set for error messages generated by Language Environment Dump Services. CEEDUMP must be a sequential data set *and* it must be allocated to SYSOUT, a terminal, or a unit record device, or the data set must have the attributes RECFM=VBA, LRECL=125, and BLKSIZE=882.
GO	SYSPRINT	Data set for listings and diagnostics from user program. LRECL=137, RECFM=VBA, BLKSIZE=882. (default attributes for SYSOUT=*).
OUTILITY	SYSIN	Input data set for object module to be added to the library. It can be sequential or partitioned (with a member name specified). LREL=80, RECFM=F or FB or FBS.
OUTILITY	SYSLIB	Library for which the member name is to be added (ADD); for which the member name is to deleted (DEL); which is to be listed (MAP); for which the C370LIB-directory is to be built. It must be partitioned and not concatenated and member names must not be specified. LREL=80, RECFM=F or FB or FBS.
OUTILITY	SYSOUT	Output data set for the C370LIB-directory map. It can be sequential or partitioned (with a member name specified). LREL=80, RECFM=F or FB or FBS.
OUTILITY	SYSMSGS	Data set containing the input messages. LRECL=150, RECFM=F or FB or FBS.
OUTILITY	SYSPRINT	Data set for target error and warning messages. The default is to SYSOUT=*. LRECL=137, RECFM=VBA, BLKSIZE=882

Examples Using Cataloged Procedures

```
//*-------------------------------------------------------------------
//* Compile a Partitioned Data Set program with various options
//*-------------------------------------------------------------------
//EXAMPLE1 EXEC EDCC,
//          INFILE='PATRICK.TEST.PDSSRC(CPROG1)',
//          OUTFILE='PATRICK.TEST.OBJECT(CPROG1),DISP=SHR',
//          CPARM='OPT NOSEQ NOMAR LIST'
//COMPILE.USERLIB DD DSNAME=PATRICK.HDR.FILES,DISP=SHR
//*
//*-------------------------------------------------
//* Compile a Sequential program with various options
//*-------------------------------------------------
//EXAMPLE2 EXEC EDCC,
//          INFILE='PATRICK.TEST.SEQSRC.CPROG2',
//          OUTFILE='PATRICK.TEST.OBJECT(CPROG2),DISP=SHR',
//          CPARM='OPT SOURCE XREF FLAG(E)'
//COMPILE.USERLIB DD DSNAME=PATRICK.HDR.FILES,DISP=SHR
```

Figure 68. Example Compilation for z/OS C Using EDCC

```
//*
//CCMEM   EXEC CBCC,          * Compile C++ source member
//        INFILE='MIKE.CPP(ONLYONE)',
//        OUTFILE='MIKE.SAMPLE.OBJ(ONLYONE),DISP=SHR ',
//        CPARM='OPT SOURCE SHOWINC LIST'
//*
//CCPDS   EXEC CBCC,          * Compile C++ source PDS
//        INFILE='MIKE.CPP',
//        OUTFILE='MIKE.PROJECT.OBJ,DISP=SHR ',
//        CPARM='NOOPT'
```

Figure 69. Example Compilation for z/OS C++ Using CBCC

Appendix E. Using Assembler Macros

To compile your C/C++ source program dynamically under z/OS, you can use macro instructions such as `ATTACH`, `LINK`, or `CALL` in an assembler language program. For complete information on these macro instructions, refer to the list of manuals in *z/OS Information Roadmap*.

The following is the syntax of each macro instruction:
where:

```
►►──┬───────┬──ATTACH──EP=CBCDRVR──,──PARAM=──(────────────────────────────►
    └─label─┘

►─option_list──┬────────────────┬──────────────────────────────────────────►
               └─,──ddname_list─┘

►─)──,──VL=1──,──DCB=dcb_addr──,──TASKLIB=dcb_addr─────────────────────────►◄


►►──┬───────┬──LINK──EP=CBCDRVR──,──PARAM=──(──────────────────────────────►
    └─label─┘

►─option_list──┬────────────────┬──)──,──VL=1──────────────────────────────►◄
               └─,──ddname_list─┘


►►──┬───────┬──CALL──EP=CBCDRVR──,──(──option_list──┬────────────────┬─────►
    └─label─┘                                       └─,──ddname_list─┘

►─)──,──VL─────────────────────────────────────────────────────────────────►◄
```

EP	Specifies the symbolic name of the z/OS C/C++ compiler CBCDRVR. The control program determines the entry point at which execution is to begin.
PARAM	Specifies a list that contains the addresses of the parameters to be passed to the z/OS C/C++ compiler
option_list	Specifies the address of a list that contains the options that you want to use for the compilation. The option list must begin on a halfword boundary. The first 2 bytes must contain a count of the number of bytes in the remainder of the list. You specify the options in the same manner as you would on a JCL job, with spaces between options. If you do not want to specify any options, the count must be zero. For C++ compiler invocation, you must include the characters CXX, and a blank before the list of compiler options. The number of bytes therefore should be 4 bytes longer.
ddname_list	Specifies the address of a list that contains alternative ddnames for the data sets that are used during the compiler processing. If you use standard ddnames, you can omit this parameter. The ddname list must begin on a halfword boundary. The first two bytes must contain a count of the number of bytes in the remainder of the list. You must left-justify each name in the list, and pad it with blanks to a length of 8 bytes.

The sequence of ddnames in the list is:

- SYSIN
- SYSLIN
- SYSMSGS - this ddname is no longer used, but is kept in the list for compatibility with old assembler macros.
- SYSLIB
- USERLIB
- SYSPRINT
- SYSCPRT
- SYSPUNCH
- SYSUT1
- SYSUT4
- SYSUT5
- SYSUT6
- SYSUT7
- SYSUT8
- SYSUT9
- SYSUT10
- SYSUT14
- SYSUT15
- SYSEVENT
- TEMPINC

You can omit an alternative ddname from the list by entering binary zeros in its 8-byte entry, or if it is at the end of the list, by shortening the list. If you omit the ddname, the compiler assumes the standard ddname.

VL **or VL=1**	Specifies that the sign bit is to be set to 1 in the last fullword of the address parameter.
DCB	Specifies the address of the control block for the partitioned data set that contains the compiler.
TASKLIB	Specifies the address of the DCB for the library that is to be used as the attached tasks library.

The return code from the compiler is returned in register 15.

If you code the macro instructions incorrectly, the compiler is not invoked, and the return code is 32. This error could be caused if the count of bytes in the alternative ddnames list is not a multiple of 8, or is not between 0 to 128.

If you specify an alternative ddname for SYSPRINT, the stdout stream is redirected to refer to the alternate ddname.

The following examples show the use of three assembler macros that rename ddnames completely or partially. Following each macro is the JCL that is used to invoke it.

CBC3UAAP

```
***********************************************************************
*                                                                     *
*  This assembler routine demonstrates DD Name renaming               *
*  (Dynamic compilation) using the Assembler ATTACH macro.            *
*                                                                     *
*  In this specific scenario all the DDNAMES are renamed.             *
*                                                                     *
*  The TASKLIB option of the ATTACH macro is used                     *
*  to specify the steplib for the ATTACHed command (ie. the compiler) *
*                                                                     *
*  The Compiler and Library should be specified on the DD             *
*  referred to in the DCB for the TASKLIB if one or both              *
*  are not already defined in LPA.  The compiler and library do not   *
*  need to be part of the steplib concatenation.                      *
*                                                                     *
***********************************************************************
ATTACH   CSECT
         STM   14,12,12(13)
         BALR  3,0
         USING *,3
         LR    12,15
         ST    13,SAVE+4
         LA    15,SAVE
         ST    15,8(,13)
         LR    13,15
*
*   Invoke the compiler using ATTACH macro
*
         OPEN  (COMPILER)
         ATTACH EP=CBCDRVR,PARAM=(OPTIONS,DDNAMES),VL=1,DCB=COMPILER, X
                ECB=ECBADDR,TASKLIB=COMPILER
         ST    1,TCBADDR
         WAIT  1,ECB=ECBADDR
         DETACH TCBADDR
         CLOSE (COMPILER)
         L     13,4(,13)
         LM    14,12,12(13)
         SR    15,15
         BR    14
*
*   Constant and save area
*
 SAVE DC 18F'0'
ECBADDR DC F'0'
TCBADDR DC F'0'
OPTIONS DC H'12',C'SOURCE EVENT'
```

Figure 70. Using the Assembler ATTACH Macro (Part 1 of 2)

```
*   For C++, substitute the above line with
*   OPTIONS  DC    H'10',C'CXX SOURCE'

DDNAMES  DC    H'152'
         DC    CL8'NEWIN'
         DC    CL8'NEWLIN'
         DC    CL8'DUMMY'   PLACEHOLDER - NO LONGER USED
         DC    CL8'NEWLIB'
         DC    CL8'NEWRLIB'
         DC    CL8'NEWPRINT'
         DC    CL8'NEWCPRT'
         DC    CL8'NEWPUNCH'
         DC    CL8'NEWUT1'
         DC    CL8'NEWUT4'
         DC    CL8'NEWUT5'
         DC    CL8'NEWUT6'
         DC    CL8'NEWUT7'
         DC    CL8'NEWUT8'
         DC    CL8'NEWUT9'
         DC    CL8'NEWUT10'
         DC    CL8'NEWUT14'
         DC    CL8'NEWUT15'
         DC    CL8'NEWEVENT'
COMPILER DCB   DDNAME=MYCOMP,DSORG=PO,MACRF=R
         END
```

Figure 70. Using the Assembler ATTACH Macro (Part 2 of 2)

CBC3UAAQ

```
//*-------------------------------------------------------------------
//* Standard DDname Renaming  (ASM ATTACH from driver program)
//*    compiles                  MYID.MYPROG.SOURCE(HELLO)
//*    and places the object in   MYID.MYPROG.OBJECT(HELLO)
//*
//*    User header files come from MYID.MYHDR.FILES
//*    using MYCOMP as the compile time steplib.
//*
//*    Compilation is controlled by the assembler module named
//*    CBC3UAAP which is stored in MYID.ATTACHDD.LOAD
//*
//*    This example uses the Language Environment Library
//*-------------------------------------------------------------------
//G001001B  EXEC PGM=CBC3UAAP
//STEPLIB   DD DSN=MYID.ATTACHDD.LOAD,DISP=SHR
//MYCOMP    DD DSN=CBC.SCBCCMP,DISP=SHR
//          DD DSN=CEE.SCEERUN,DISP=SHR
//NEWIN     DD DSN=MYID.MYPROG.SOURCE(HELLO),DISP=SHR
//NEWLIB    DD DSN=CEE.SCEEH.H,DISP=SHR
//NEWLIN    DD DSN=MYID.MYPROG.OBJECT(HELLO),DISP=SHR
//NEWPRINT  DD SYSOUT=*
//NEWCPRT   DD SYSOUT=*,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=882)
//NEWPUNCH  DD DSN=...
//SYSTERM   DD DUMMY
//NEWUT1    DD DSN=...
//NEWUT4    DD DSN=...
//NEWUT5    DD DSN=...
//NEWUT6    DD DSN=...
//NEWUT7    DD DSN=...
//NEWUT8    DD DSN=...
//NEWUT9    DD DSN=...
//NEWUT10   DD SYSOUT=*
//NEWUT14   DD DSN=...
//NEWUT15   DD DSN=...
//NEWEVENT  DD DSN=...
//NEWRLIB   DD DSN=MYID.MYHDR.FILES,DISP=SHR
//*------------------------------------------------------------------
```

Figure 71. JCL for the Assembler ATTACH Macro

Note that the sharing of resources between attached programs is not supported.

```
***********************************************************************
*                                                                     *
*  This assembler routine demonstrates DD Name renaming               *
*  (Dynamic compilation) using the assembler LINK macro.              *
*                                                                     *
*  In this specific scenario a subset of all the DDNAMES are          *
*  renamed.  The DDNAMES you do not want to rename are set to zero.   *
*                                                                     *
*  The Compiler and the Library should be in the LPA, or should       *
*  be specified on the STEPLIB DD in your JCL                         *
*                                                                     *
***********************************************************************
*
LINK     CSECT
         STM   14,12,12(13)
         BALR  3,0
         USING *,3
         LR    12,15
         ST    13,SAVE+4
         LA    15,SAVE
         ST    15,8(,13)
         LR    13,15
*
*   Invoke the compiler using LINK macro
*
         LINK  EP=CBCDRVR,PARAM=(OPTIONS,DDNAMES),VL=1
         L     13,4(,13)
         LM    14,12,12(13)
         SR    15,15
         BR    14
```

Figure 72. Using the Assembler LINK Macro (Part 1 of 2)

```
*
*   Constant and save area
*
*   This macro will compile for the Language Environment Library
*
SAVE     DC    18F'0'
OPTIONS  DC    H'8',C'SO EVENT'
*   For C++, substitute the above line with
*   OPTIONS  DC    H'6',C'CXX SO'
DDNAMES  DC    H'152'
         DC    CL8'NEWIN'
         DC    XL8'0000000000000000'
         DC    XL8'0000000000000000'
         DC    XL8'0000000000000000'
         DC    CL8'NEWRLIB'
         DC    XL8'0000000000000000'
         DC    CL8'NEWCPRT'
         DC    XL8'0000000000000000'
         DC    2XL8'0000000000000000'
         DC    2XL8'0000000000000000'
         DC    2XL8'0000000000000000'
         DC    XL8'0000000000000000'
         DC    XL8'0000000000000000'
         DC    XL8'0000000000000000'
         DC    XL8'0000000000000000'
         DC    XL8'0000000000000000'
         END
```

Figure 72. Using the Assembler LINK Macro (Part 2 of 2)

CBC3UAAS

```
//*-------------------------------------------------------------------------
//* Standard DDname Renaming  using the assembler LINK macro
//*    compiles                    MYID.MYPROG.SOURCE(HELLO)
//*    and places the object in    MYID.MYPROG.OBJECT(HELLO)
//*
//*    User header files come from MYID.MYHDR.FILES
//*
//*    Compilation is controlled by the assembler module named
//*    CBC3UAAR that is stored in MYID.LINKDD.LOAD
//*
//*    This JCL uses the Language Environment Library.
//*
//*-------------------------------------------------------------------------
//G001003A  EXEC PGM=CBC3UAAR
//STEPLIB   DD DSN=CBC.SCBCCMP,DISP=SHR
//          DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=MYID.LINKDD.LOAD,DISP=SHR
//NEWIN     DD DSN=MYID.MYPROG.SOURCE(HELLO),DISP=SHR
//SYSLIB    DD DSN=CEE.SCEEH.H,DISP=SHR
//SYSLIN    DD DSN=MYID.MYPROG.OBJECT(HELLO),DISP=SHR
//SYSPRINT  DD SYSOUT=*
//NEWCPRT   DD SYSOUT=*,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=882)
//SYSPUNCH  DD SYSOUT=*
//SYSTERM   DD DUMMY
//SYSUT1    DD DSN=...
//SYSUT4    DD DSN=...
//SYSUT5    DD DSN=...
//SYSUT6    DD DSN=...
//SYSUT7    DD DSN=...
//SYSUT8    DD DSN=...
//SYSUT9    DD DSN=...
//SYSUT10   DD SYSOUT=*
//SYSUT14   DD DSN=...
//SYSUT15   DD DSN=...
//SYSEVENT  DD DSN=...
//NEWRLIB   DD DSN=MYID.MYHDR.FILES,DISP=SHR
//*------------------------------------------------------------------------
```

Figure 73. JCL for the Assembler LINK Macro

CBC3UAAT

```
***********************************************************************
*                                                                     *
*  This assembler routine demonstrates DD Name renaming               *
*  (Dynamic compilation) using the Assembler CALL macro.              *
*                                                                     *
*  In this specific scenario, a subset of all the DDNAMES are         *
*  renamed.  This renaming is accomplished by shortening              *
*  the list of ddnames.                                               *
*                                                                     *
*  The Compiler and the Library should be either be in the LPA or     *
*  be specified on the STEPLIB DD in your JCL                         *
*                                                                     *
***********************************************************************
*
LINK     CSECT
         STM   14,12,12(13)
         USING LINK,15
         LA    3,MODE31
         O     3,=X'80000000'
         DC    X'0B03'
MODE31   DS    0H
         USING *,3
         LR    12,15
         ST    13,SAVE+4
         LA    15,SAVE
         ST    15,8(,13)
         LR    13,15
*
*   Invoke the compiler using CALL macro
*
         LOAD  EP=CBCDRVR
         LR    15,0
         CALL  (15),(OPTIONS,DDNAMES),VL
         L     13,4(,13)
         LM    14,12,12(13)
         SR    15,15
         BR    14
```

Figure 74. Using the Assembler CALL Macro (Part 1 of 2)

```
*
*   Constant and save area
*
SAVE     DC    18F'0'
OPTIONS  DC    H'2',C'SO'
*   For C++, substitute the above line with
*   OPTIONS  DC    H'6',C'CXX SO'
DDNAMES  DC    H'96'
         DC    CL8'NEWIN'
         DC    CL8'NEWLIN'
         DC    CL8'DUMMY'       PLACEHOLDER - NO LONGER USED
         DC    CL8'NEWLIB'
         DC    CL8'NEWRLIB'
         DC    CL8'NEWPRINT'
         DC    CL8'NEWCPRT'
         DC    CL8'NEWPUNCH'
         DC    CL8'NEWUT1'
         DC    CL8'NEWUT4'
         DC    CL8'NEWUT5'
         DC    CL8'NEWUT6'
         END
```

Figure 74. Using the Assembler CALL Macro (Part 2 of 2)

CBC3UAAU

```
//*-------------------------------------------------------------------
//* Standard DDname Renaming  using the assembler CALL macro
//*    compiles                     MYID.MYPROG.SOURCE(HELLO)
//*    and places the object in     MYID.MYPROG.OBJECT(HELLO)
//*
//*    User Header files come from  MYID.MYHDR.FILES
//*
//*    Compilation is controlled by the assembler module named
//*    CBC3UAAT which is stored in MYID.CALLDD.LOAD
//*
//*    This JCL uses the Language Environment Library.
//*
//*-------------------------------------------------------------------
//G001004C  EXEC PGM=CBC3UAAT
//STEPLIB   DD DSN=CBC.SCBCCMP,DISP=SHR
//          DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=MYID.CALLDD.LOAD,DISP=SHR
//NEWIN     DD DSN=MYID.MYPROG.SOURCE(HELLO),DISP=SHR
//NEWLIB    DD DSN=CEE.SCEEH.H,DISP=SHR
//NEWLIN    DD DSN=MYID.MYPROG.OBJECT(HELLO),DISP=SHR
//NEWPRINT  DD SYSOUT=*
//NEWCPRT   DD SYSOUT=*,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=882)
//NEWPUNCH  DD DSN=...
//SYSTERM   DD DUMMY
//NEWUT1    DD DSN=...
//NEWUT4    DD DSN=...
//NEWUT5    DD DSN=...
//NEWUT6    DD DSN=...
//SYSUT7    DD DSN=...
//SYSUT8    DD DSN=...
//SYSUT9    DD DSN=...
//SYSUT10   DD SYSOUT=*
//SYSUT14   DD DSN=...
//SYSUT15   DD SYSOUT=*
//NEWRLIB   DD DSN=MYID.MYHDR.FILES,DISP=SHR
//*------------------------------------------------------------------
```

Figure 75. JCL for the Assembler CALL Macro

Other z/OS C Utilities

Starting with C/C++ for MVS/ESA V3R2, several improvements were made to the REXX EXECs provided with the C/C++ compiler. The improved REXX EXECs use a different syntax, which we refer to as the *new syntax*. The *old syntax* is the syntax of the REXX EXECs prior to the C/C++ for MVS/ESA V3R2 release of the compiler. This section describes the old syntax for these REXX EXECs, which is still supported. In the following table we indicate the corresponding updated REXX EXECs which will provide new features and greater flexibility.

Name	Task Description	Substitute
CC (old syntax)	Compile	CC (new syntax)
CMOD	Generate an executable module	CXXMOD

Figure 76. Utilities for z/OS C

For a description of `CXXMOD` see “Prelinking and Linking under TSO” on page 485.

Using the Old Syntax for CC

The CC command can now be invoked using a new syntax. At installation time, your system programmer can customize the `CC EXEC` to accept:

- Only the old syntax (the one supported by compilers prior to C/MVS Version 3 Release 2)
- Only the new syntax
- Both syntaxes

The `CC EXEC` should be customized to accept only the new syntax. If you customize the `CC EXEC` to accept only the old syntax, keep in mind that it does not support Hierarchical File System (HFS) files. If you customize the `CC EXEC` to accept both the old and new syntaxes, you must invoke it using either the old syntax *or* the new syntax, but not a mixture of both. If you invoke this EXEC with the old syntax, it will not support HFS files.

For information on the new syntax, see "Using the CC and CXX REXX EXECs" on page 293. Refer to the z/OS C/C++ Program Directory for more information about installation and customization.

The old syntax for the `CC REXX EXEC` is:

```
►►──CC──source──┬──────────────────────────┬──────────────────────────────────►
                └─OBJ──(──object───)───────┘

►──┬──────────────────────────────┬───────────────────────────────────────────►
   │          ┌─,──────────┐      │
   │          ▼            │      │
   └─COPT──(──┴─┬────────┬─┴──)───┘
                └─option─┘

►──┬────────────────────────────────────┬──┬────────────┬─────────────────────►
   │              ┌─,──────────┐        │  └─C370LIB────┘
   │              ▼            │        │
   └─USERLIB──(───┴─┬─────────┬┴───)────┘
                    └─libname─┘

►──┬────────────────────────────┬────────────────────────────────────────────►◄
   └─LISTING──(──listing───)────┘
```

You can override the default compiler options by specifying the options:

- In the `COPT` keyword parameter
- In a `#pragma OPTIONS` directive in your source file
- By specifying them directly on the invocation line

However, any options specified on `#pragma options` directives are overridden by options specified on the invocation line.

The following rules apply when you use the old syntax for the `CC REXX EXEC`:

- When you are specifying a data set name, if the name is not enclosed in single quotation mark ('), your user prefix will be added to the beginning of the data set name. If the data set name is enclosed in quotation marks, it will be treated as a fully qualified name.

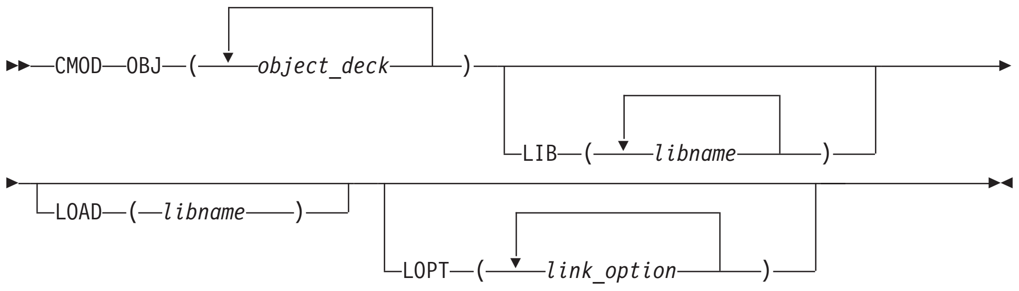
- When you need to use spaces, commas, single quotation marks, or parentheses within a REXX EXEC option, the text must be placed inside a string using single quotation marks.
- If you want to use a single quotation mark inside a string, you must use two quotation marks in place of each quotation mark.

The following example demonstrates these rules:

```
CC TEST.C(STOCK) COPT ('SEARCH(CLOTHES.H ''MARK.SUPPLY.C(ORDER)'')')
```

Using CMOD

The `CMOD REXX EXEC` makes a call to `LINK` with the appropriate library. The syntax of the `CMOD REXX EXEC` is:



OBJ	Specifies the object decks that you want to link.
LIB	Specifies the libraries that are to be used to resolve external entries.
LOAD	Specifies the output library in which the load module is to be stored.
LOPT	Specifies the options that you want to pass to the linkage editor. All options are passed to the TSO LINK command.

A non-zero return code indicates that an error has occurred. For diagnostic information, refer to “Appendix C. Diagnosing Problems and the PMR/APAR Process” on page 507. `CMOD` can also return the return code from `LINK`. See the appropriate book in your TSO library for more information on `LINK`.

Appendix F. c89 — Compile, link-edit and assemble an z/OS C program and create an executable file

Format

c89 | **cc** | **c++** | **cxx**   [**–+CcEFfgOpqrsVv012**]
    [**–D** *name*[*=value*]]... [**–U** *name*]...
    [**–e** *function*] [**–u** *function*]...
    [**–W** *phase,option*[*,option*]...]...
    [**–o** *outfile*]
    [**–I** *directory*]... [**–L** *directory*]...
    [*file.C*]... [*file.i*]... [*file.c*]... [*file.s*]...
    [*file.o*]... [*file.x*]... [*file.p*]... [*file.I*]... [*file.a*]... [**–l** *libname*]...

Note: The **I** option signifies an uppercase **i**, not a lowercase **L**.

Description

c89 and **cc** compile, assemble, and link-edit z/OS C programs; **c++** does the same for z/OS C++ programs.

- **c89** should be used when compiling C programs that are written according to *Standard C*.
- **cc** should be used when compiling C programs that are written according to *Common Usage C*.
- **c++** must be used when compiling C++ programs, which are written according to *Draft Proposal International Standard for Information Systems — Programming Language C++ (X3J16)*. **c++** can compile both C++ and C programs, and can also be invoked by the name **cxx** (all references to **c++** throughout this document apply to both names).

| **c89**, **cc**, and **c++** call other programs for each step of the compilation, assemble and link-editing phases. The list below contains the following: the step name, the documention which describes the program you use for that step and the book which describes any messages issued by that program, and prefixes to those messages:

Table 54. c89, cc, and c++ Programs and Reference Documentation

Step Name	Book Describing Options and How to Call Program	Book Containing Messages Issued by Program	Prefix of Messages Issued by Program
ASSEMBLE	*HLASM Programmer's Guide*	*HLASM Programmer's Guide*	ASMA
COMPILE, IPACOMP, TEMPINC, IPATEMP, IPALINK	*z/OS C/C++ User's Guide*	*z/OS C/C++ User's Guide*	CBC
PRELINK	*z/OS Language Environment Programming Guide*	*z/OS Language Environment Debugging Guide*	EDC

Table 54. c89, cc, and c++ Programs and Reference Documentation (continued)

Step Name	Book Describing Options and How to Call Program	Book Containing Messages Issued by Program	Prefix of Messages Issued by Program
LINKEDIT	*z/OS DFSMS Program Management*	*z/OS MVS System Messages, Vol 4*	IEW2

Execution of any Language Environment program (including **c89** and the C/z/OS C++ compiler) can result in run-time messages. These messages are described in *z/OS Language Environment Debugging Guide* and have an EDC prefix. In some cases **c89** issues messages with Language Environment messages appended to them. Messages issued by **c89** have an FSUM3 prefix.

In order for **c89**, **cc**, and **c++** to perform C and C++ compiles, the C/z/OS C++ Optional Feature must be installed on the system. The C/z/OS C++ Optional Feature provides a C compiler, a C++ compiler, C++ Class Libraries, and some utilities. See the *z/OS Introduction and Release Guide* for further details. Also see {_CLIB_PREFIX} in "Environment Variables" on page 567 for information about the names of the C/z/OS C++ Optional Feature data sets that must be made available to **c89/cc/c++**.

First, **c89**, **cc**, and **c++** perform the compilation phase (including preprocessing) by compiling all source file operands (*file.C*, *file.i*, and *file.c*, as appropriate). For **c++**, if automatic template generation is being used (which is the default), then z/OS C++ source files may be created or updated in the **tempinc** subdirectory of the working directory during the compilation phase (the **tempinc** subdirectory will be created if it does not already exist). Then, **c89**, **cc**, and **c++** perform the assemble phase by assembling all operands of the *file.s* form. The result of each compile step and each assemble step is a *file.o* file. If all compilations and assemblies are successful, or if only *file.o* and/or *file.a* files are specified, **c89**, **cc**, and **c++** proceed to the link-editing phase. For **c++**, the link-editing phase begins with an automatic template generation step when applicable. For IPA (Interprocedural Analysis) optimization an additional IPA link step comes next. The link-edit step is last. See the environment variable **{_STEPS}** under "Environment Variables" on page 567 for more information about the link-editing phase steps.

In the link-editing phase, **c89**, **cc**, and **c++** combine all *file.o* files from the compilation phase along with any *file.o* files that were specified on the command line. For **c++**, this is preceded by compiling all z/OS C++ source files in the **tempinc** subdirectory of the working directory (possibly creating and updating additional z/OS C++ source files during the automatic template generation step). After compiling all the z/OS C++ source files, the resulting object files are combined along with the *file.o* files from the compilation phase and the command line. Any *file.a* files, *file.x* files and **–l** *libname* operands that were specified are also used.

The usual output of the link-editing phase is an executable file. For **c89**, **cc**, and **c++** to produce an executable file, you must specify at least one operand which is of other than **–l** *libname* form. If **–r** is used, the output file is not executable.

For more information about automatic template generation, see *z/OS C/C++ User's Guide* and *z/OS C/C++ Programming Guide* . Note that the **c++** command only supports using the **tempinc** subdirectory of the working directory for automatic template generation.

IPA is further described under the **–W** option on page 562.

Options

–+
Specifies that all source files are to be recognized as C++ source files. All *file.s*, *file.o*, and *file.a* files will continue to be recognized as assembler source, object, and archive files respectively. However, any C *file.c* or *file.i* files will be processed as corresponding C++ *file.C* or *file.i* files, and any other file suffix which would otherwise be unrecognized will be processed as a *file.C* file.

This option effectively overrides the environment variable **{_EXTRA_ARGS}**. This option is only supported by the **c++** command.

–C
Specifies that C and C++ source comments should be retained by the preprocessor. By default all comments are removed by the preprocessor. This option is ignored except when used with the **–E** option.

–c
Specifies that only compilations and assemblies be done. Link-edit is not done.

–D *name***[**=*value***]**
Defines a C or C++ macro for use in compilation. If only *name* is provided, a value of 1 is used for the macro it specifies. For information about macros that **c89/cc/c++** automatically define, see Usage Note 5 on page 585. Also see Usage Note 13 on page 586.

–E
Specifies that output of the compiler preprocessor phase be copied to **stdout**. Compilation into object and link-edit are not done.

–e *function*
Specifies the name of the function to be used as the entry point of the program. This can be useful when creating a fetchable program, or a non–C or non–C++ main, such as a COBOL program. Non–C++ linkage symbols of up to 1024 characters in length may be specified. You can specify an S-name by preceding the function name with double slash (//). (For more information about S-names, see Usage Note 23 on page 589.)

Specify a null S-name ("-e //") so that no function name is identified by **c89/cc/c++** as the entry point of the program. In that case, the Program Management Binder (link editor) default rules will determine the entry point of the program. For more information about the Program Management Binder and the ENTRY control statement, see *z/OS DFSMS Program Management*.

The function **//ceestart** is the default. When the default function entry point is used, a binder ORDER control statement is generated by **c89/cc/c++** to cause the CEESTART code section to be ordered to the beginning of the program. Specify the name with a trailing blank to disable this behavior, as in "**//ceestart** ".

–F
Ignored by **cc**, provided for compatibility with historical implementations of **cc**. Flagged as an error by **c89** and **c++**.

–f
Ignored by **cc**, provided for compatibility with historical implementations of **cc**. Flagged as an error by **c89** and **c++**.

–g
Specifies that the output file (executable) is to contain symbolic information and is to be loaded into read/write storage, which is required for source-level debugging with **dbx**, the debugger.

When specified for the compilation phase, the object file contains symbolic information for source-level debugging.

When specified for the link-editing phase, the executable file is marked as being serially reusable and will always be loaded into read/write storage.

dbx requires that all the executables comprising the process be loaded into read/write storage so that it can set break points in these executables. When **dbx** is attached to a running process, this cannot be guaranteed because the process was already running and some executables were already loaded. There are two techniques that will guarantee that all the executables comprising the process is loaded into read-write storage:

1. Specify the **–g** option for the link-editing phase of each executable. After this is done, the executable is always loaded into read/write storage.

   Because the executable is marked as being serially reusable, this technique works except in cases where the executable must be marked as being reentrant. For example:

   - If the executable is to be used by multiple processes in the same user space.
   - If the executable is a DLL that is used on more than one thread in a multithreaded program.

   In these cases, use the following technique instead:
2. Do not specify the **–g** option during the link-editing phase so that the executable will be marked as being reentrant. Before invoking the program, export the environment variable **_BPX_PTRACE_ATTACH** with a value of `YES`. After you do this, then executables will be loaded into read/write storage regardless of their reusability attribute.

If you compile an MVS data set source using the **–g** option, you can use **dbx** to perform source-level debugging for the executable file. You must first issue the **dbx use** subcommand to specify a path of double slash (//), causing dbx to recognize that the symbolic name of the primary source file is an MVS data set. For information on the dbx command and its use subcommand, see *Z/OS UNIX System Services Command Reference*.

For more information on using dbx, see *z/OS UNIX System Services Programming Tools*.

The z/OS UNIX System Services web page also has more information about **dbx**. Go to

`http://www.s390.ibm.com/products/oe/index.html`

For more information on the **_BPX_PTRACE_ATTACH** environment variable, see *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

–I *directory*

Note: The **I** option signifies an uppercase **i**, not a lowercase **L**.
–I specifies the directories to be used during compilation in searching for *include files* (also called *header files*).

Absolute pathnames specified on **#include** directives are searched exactly as specified. The directories specified using the **–I** option or from the usual places are not searched.

If absolute pathnames are not specified on **#include** directives, then the search order is as follows:

1. Include files enclosed in double quotes (") are first searched for in the directory of the file containing the **#include** directive. Include files enclosed in angle-brackets (< >) skip this initial search.
2. The include files are then searched for in all directories specified by the **–I** option, in the order specified.
3. Finally, the include files are searched for in the usual places. (See Usage Note 4 on page 584 for a description of the usual places.)

You can specify an MVS data set name as an include file search directory. Also, MVS data set names can explicitly be specified on **#include** directives. You can indicate both by specifying a leading double slash (//). For example, to include the include file DEF that is a member of the MVS PDS ABC.HDRS, code your C or C++ source as follows:

```
#include <//'abc.hdrs(def)'>
```

MVS data set include files are handled according to z/OS C/C++ compiler conversion rules (see Usage Note 4 on page 584). When specifying an **#include** directive with a leading double slash (in a format other than**#include**<**//'***dsname***'**> and **#include**<**//dd:***ddname*>), the specified name is paired only with MVS data set names specified on the **–I** option. That is, when you explicitly specify an MVS data set name, any hierarchical file system (HFS) directory names specified on the **–I** option are ignored.

–L *directory*
: Specifies the directories to be used to search for archive libraries specified by the **–l** operand. The directories are searched in the order specified, followed by the usual places. You cannot specify an MVS data set as an archive library directory.

  For information on specifying C370LIB libraries, see the description of the **–l** *libname* operand. Also see Usage Note 7 on page 585 for a description of the usual places.

–0, –O (–1), –2
: Specifies the level of compiler optimization (including inlining) to be used. The level **–1** (number one) is equivalent to **–O** (letter capital O). The level **–2** gives the highest level of optimization. The default is **–0** (level 0), no optimization and no inlining, when not using IPA (Interprocedural Analysis).

  When using IPA, the default is **–0** (level 1) optimization and inlining. IPA optimization is independent from and can be specified in addition to this optimization level. IPA is further described under the **–W** option on page 562.

  If you compile your program to take advantage of **dbx** source-level debugging (see the **–g** option on page 557), you will always get **–0** (level zero) optimization regardless of which of these compiler optimization levels you specify.

  In addition to using optimization techniques, you may want to control writable strings by using the **#pragma strings(readonly)** directive or the `ROSTRING` compiler option. For more information on this topic, refer to the chapter on reentrancy in z/OS C/C++ in *z/OS C/C++ Programming Guide* or the description of the `ROSTRING` option in the *z/OS C/C++ User's Guide*.

–o *outfile*
: Specifies the name of the **c89/cc/c++** output file.

If the **–o** option is specified in addition to the **–c** option, and only one source file is specified, then this option specifies the name of the output file associated with the one source file. See *file.o* under “Operands” on page 565 for information on the default name of the output file.

Otherwise the **–o** option specifies the name of the executable file produced during the link-editing phase. The default output file is **a.out**. For related information, see Usage Note 3 on page 584.

–p Ignored by **cc**, provided for compatibility with historical implementations of **cc**. Flagged as an error by **c89** and **c++**.

–q Ignored by **cc**, provided for compatibility with historical implementations of **cc**. Flagged as an error by **c89** and **c++**.

–r Specifies that **c89/cc/c++** is to save relocation information about the object files which are processed. When the output file (as specified on **–o**) is created, it is not made an executable file. Instead, this output file can later be used as input to **c89/cc/c++**. This can be used as an alternative to an archive library.

IPA Usage Note:

When using **-r** and link-editing IPA compiled object files, you must link-edit with IPA (see the description of IPA under the **-W** option). However, the **-r** option is typically not useful when creating an IPA optimized program. This is because link-editing with IPA requires that all of the program information is available to the link editor (that is, all of the object files). It is not acceptable to have unresolved symbols, especially the program entry point symbol (which is usually *main*). The **-r** option is normally used when you wish to combine object files incrementally. You would specify some object files during the initial link-edit that uses **-r**. Later, you would specify the output of the initial link-edit, along with the remaining object files in a final link-edit that is done without using **-r**. In such situations where you wish to combine IPA compiled object files, there is an alternative which does not involve the link editor. That alternative is to concatenate the object files into one larger file. This larger file can then later be used in a final link-edit, when the remainder of the object files are also made available. (This concatenation can easily be done using the **cp** or **cat** utilities.)

–s Specifies that the compilation phase is to produce a *file.o* file that does *not* include symbolic information, and that the link-editing phase produce an executable that is marked reentrant. This is the default behavior for **c89/cc/c++**.

–U *name*
Undefines a C or C++ macro specified with *name*. This option affects only macros defined by the **–D** option, including those automatically specified by **c89/cc/c++**. For information about macros that **c89/cc/c++** automatically define, see Usage Note 5 on page 585. Also see Usage Note 13 on page 586.

–u *function*
Specifies the name of the function to be added to the list of symbols which are not yet defined. This can be useful if the only input to **c89/cc/c++** is archive libraries. Non–C++ linkage symbols of up to 255 characters in length may be specified. You can specify an S-name by preceding the function name with double slash (//). (For more information about S-names, see Usage Note 23 on page 589.) The function **//ceemain** is the default for non-IPA link-editing, and the function **main** is the default for IPA

link-editing. However if this **-u** option is used, or the **DLL** link editor option is used, then the default function is not added to the list.

–V
: This verbose option produces and directs output to **stdout** as compiler, assembler, IPA linker, prelinker, and link editor listings. If the **–O** or **–2** options are specified and cause **c89/cc/c++** to use the compiler **INLINE** option, then the inline report is also produced with the compiler listing. Error output continues to be directed to **stderr**. Because this option causes **c89/cc/c++** to change the options passed to the steps producing these listings so that they produce more information, it may also result in additional messages being directed to **stderr**. In the case of the compile step, it may also result in the return code of the compiler changing from 0 to 4.

–v
: This verbose option causes pseudo-JCL to be written to **stdout** before the compiler, assembler, IPA linker, prelinker, and link editor programs are run. It provides information about exactly which compiler, prelinker, and link editor options are being passed, and also which data sets are being used. If you want to obtain this information without actually invoking the underlying programs, specify the **–v** option more than once on the **c89/cc/c++** command string. For more information about the programs which are executed, see Usage Note 14 on page 587.

–W *phase, option[,option]...*
: Specifies options to be passed to the steps associated with the compile, assemble, or link-editing phases of **c89/cc/c++**. The valid phase codes are:

0
: Specifies the compile phase (used for both non-IPA and IPA compilation).

a
: Specifies the assemble phase.

c
: Same as phase code **0**.

I
: Enables IPA (Interprocedural Analysis) optimization.

Unlike other phase codes, the IPA phase code **I** does not require that any additional options be specified, but it does allow them. In order to pass IPA suboptions, specify those suboptions using the IPA phase code. For example, to specify that an IPA compile should save source line number information, without writing a listing file, specify:

```
c89 -W I,list file.c
```

To specify that an IPA link-edit should write the map file to **stdout**, specify:

```
c89 -W I,map file.o
```

l
: Specifies the link-editing phase.

- To pass options to the prelinker, the first link-editing phase option must be **p** or **P**. All the following options are then prelink options. For example, to write the prelink map to stdout, specify:

  ```
  c89 –W l,p,map file.c
  ```

  Note: The prelinker is no longer used in the link-editing phase in most circumstances. If it is not used, any options passed are accepted but ignored. See the environment variable **{_STEPS}** under “Environment Variables” on page 567 for more information about the link-editing phase prelink step.

- To pass options to the IPA linker, the first link-editing phase option must be **i** or **I**. All the following options are then IPA link options. For example, to specify the size of the SPILL area to be used during an IPA link-edit, you could specify:

```
c89 –W l,I,"spill(256)" file.o
```

- To link-edit a DLL (Dynamic Link Library) the link-editing phase option DLL must be specified. For example:

```
c89 –o outdll –W l,dll file.o
```

Most z/OS C/C++ extensions can be enabled by using this option. Those which do not directly pass options through to the underlying steps, or involve files which are extensions to the compile and link-edit model, are described here:

DLL (Dynamic Link Library)

A DLL is a part of a program that is not statically bound to the program. Instead, linkage to symbols (variables and functions) is completed dynamically at execution time. DLLs can improve storage utilization, because the program can be broken into smaller parts, and some parts may not always need to be loaded. DLLs can improve maintainability, because the individual parts can be managed and serviced separately.

In order to create a DLL, some symbols must be identified as being exported for use by other parts of the program. This can be done with the z/OS C/C++ **#pragma export** compiler directive, or by using the z/OS C/C++ **EXPORTALL** compiler option. If during the link-editing phase some of the parts have exported symbols, the executable which is created is a DLL. In addition to the DLL, a definition side-deck is created, containing link-editing phase **IMPORT** control statements which name those symbols which were exported by the DLL. In order for the definition side-deck to be created, the **DLL** link editor option must be specified. This definition side-deck is subsequently used during the link-editing phase of a program which is to use the DLL. See the *file.x* operand under Operands on page 566 for information on where the definition side-deck is written. In order for the program to refer to symbols exported by the DLL, it must be compiled with the **DLL** compiler option. For example, to compile and link a program into a DLL, you could specify:

```
c89 –o outdll –W c,dll,expo –W l,dll file.c
```

To subsequently use *file.x* definition side-decks, specify them along with any other *file.o* object files specified for **c89/cc/c++** link-editing phase. For example:

```
c89 –o myappl –W c,dll myappl.c outdll.x
```

In order to run an application which is link-edited with a definition side-deck, the DLL must be made available (the definition side-deck created along with the DLL is not needed at execution time). When the DLL resides in the HFS, it must be in either the working directory or in a directory named on the **LIBPATH** environment variable. Otherwise it must be a member of a data set in the search order used for MVS programs.

IPA (interprocedural analysis)

IPA optimization is independent from and can be used in addition to the **c89/cc/c++** optimization level options (such as **–O**). IPA optimization can also improve the execution time of your application. IPA is a mechanism for performing optimizations across function boundaries, even across compilation units. It also performs optimizations not otherwise available with the C/C++ compiler.

Note: IPA does not support XPLINK (Extra Performance Linkage).

When phase code **I** is specified for the compilation phase, then IPA compilation steps are performed. When phase code **I** is specified for the link-editing phase, or when the first link-editing phase (code **l**) option is **i** or **I**, then an additional IPA link step is performed prior to the prelink and link-edit steps.

With conventional compilation and link-editing, the object code generation takes place during the compilation phase. With IPA compilation and link-editing, the object code generation takes place during the link-editing phase. Therefore, you might need to request listing information about the program (such as with the **–V** option) during the link-editing phase.

Unlike the other phase codes, phase code **I** does not require that any additional options be specified. If they are, they should be specified for both the compilation and link-editing phases.

No additional preparation needs to be done in order to use IPA. So for example to create the executable **myIPApgm** using **c89** with some existing source program **mypgm.c**, you could specify:

```
c89 —W I —o myIPApgm mypgm.c
```

When IPA is used with **c++**, and automatic template generation is being used, phase code **I** will control whether the automatic template generation compiles are done using IPA. If you do not specify phase code **I**, then regular compiles will be done. Specifying **i** as the first option of the link-editing phase option (that is, -W i,l), will cause the IPA linker to be used, but will not cause the IPA compiler to be used for automatic template generation unless phase code **I** (that is, -W I) is also specified.

XPLINK (Extra Performance Linkage)

z/OS XPLINK provides improved performance for many C/C++ programs. The C/C++ **XPLINK** compiler option instructs the C/C++ compiler to generate high performance linkage for subroutine calls. It does so primarily by making subroutine calls as fast and efficient as possible, by reducing linkage overhead, and by passing function call parameters in registers. Furthermore, it reduces the data size by eliminating unused information from function control blocks.

An XPLINK-compiled program is implicitly a DLL-compiled program (the C/C++ **DLL** compiler option need not be specified along with the **XPLINK** option). XPLINK improves performance when crossing function boundaries, even across compilation units, since XPLINK uses a more efficient linkage mechanism.

For more information about the z/OS C/C++ **XPLINK** compiler option, refer to *z/OS C/C++ User's Guide*. For more information about Extra Performance Linkage, refer to *z/OS Language Environment Programming Guide*.

To use XPLINK, you must both compile and link-edit the program for XPLINK. All C and C++ source files must be compiled XPLINK, as you cannot statically link together XPLINK and non-XPLINK C and C++ object files (with the exception of non-**XPLINK** ″OS″ linkage). You can however mix XPLINK and non-XPLINK executables across DLL and fetch() boundaries.

To compile a program as XPLINK, specify the z/OS C/C++ **XPLINK** compiler option. If there are any exported symbols in the executable and you want to produce a definition side-deck, specify the **DLL** link editor option. To indicate that different libraries should be concatenated, specify the **XPLINK** on **c89**. Here is an example of compiling and link-editing an XPLINK application in one command:

```
c89 -o outxpl -W c,XPLINK -W l,XPLINK,dll file.c
```

In order to execute an XPLINK program, the SCEERUN2 as well as the SCEERUN data set must be in the MVS program search order (see the {_PLIB_PREFIX} environment variable).

You cannot use **–W** to override the compiler options that correspond to **c89/cc/c++** options, with the following exceptions:

- Listing options (corresponding to **–V**)
- Inlining options (corresponding to **–O** and **–2**)
- Symbolic options (corresponding to **–s** and **–g**); symbolic options can be overridden only when neither **–s** nor **–g** is specified.

Notes:

1. Most compiler, prelinker, and IPA linker options have a positive and negative form. The negative form is the positive with a prepended NO (as in XREF and NOXREF).
2. The compiler **#pragma options** directives as well as any other **#pragma** directives which are overridden by compiler options, will have no effect in source code compiled by **c89/cc/c++**.
3. Link editor options must be specified in the *name=value* format. Both the option *name* and *value* must be spelled out in full. If you do not specify a value, a default value of YES is used, except for the following options, which if specified without a value, have the default values shown here:

   ALIASES
   ALIASES=ALL

   COMPAT
   COMPAT=CURRENT

   DYNAM
   DYNAM=DLL

   LET LET=8

   LIST LIST=NOIMPORT

   Note: References throughout this document to the link editor are generic references. **c89/cc/c++** specifically uses the Program Management binder for this function.
4. The z/OS C/C++ compiler is described in *z/OS C/C++ User’s Guide*. Related information about the z/OS C/C++ runtime library, including information about DLL and IPA support, is described in *z/OS C/C++*

Programming Guide. Related information about the z/OS C and z/OS C++ languages, including information about compiler directives, is described in *z/OS C/C++ Language Reference* .

5. Since some compiler options are z/OS C–only and some compiler options are z/OS C++–only, you may get warning messages and a compiler return code of 4, if you use this option and compile both C and C++ source programs in the same **c++** command invocation.
6. The prelinker is described in *z/OS C/C++ User's Guide*.
7. The *z/OS C/C++ User's Guide* also describes C/C++ compiler options and documents any messages produced by it (CBC messages).
8. You may see runtime messages (CEE or EDC) in executing your applications. These messages are described in *z/OS Language Environment Debugging Guide*.
9. The link editor (the Program Management binder) is described in *z/OS DFSMS Program Management*. The Program Management binder messages are described in *z/OS MVS System Messages, Vol 4*.

Operands

c89/cc/c++ generally recognize their file operand types by file suffixes. The suffixes shown here represent the default values used by **c89/cc/c++**. See "Environment Variables" on page 567 for information on changing the suffixes to be used.

Unlike **c89** and **c++**, which report an error if given an operand with an unrecognized suffix, **cc** determines that it is either an object file or a library based on the file itself. This behavior is in accordance with the environment variable **{_EXTRA_ARGS}**. Also see Usage Note 3 on page 584 for related information.

file.a
Specifies the name of an archive file, as produced by the **ar** command, to be used during the link-editing phase. You can specify an MVS data set name, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *LIB*. The data set specified must be a C370LIB object library or a load library. See the description of the **–l** *libname* operand for more information about using data sets as libraries.

file.C
Specifies the name of a C++ source file to be compiled. You can specify an MVS data set name by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *CXX*. This operand is only supported by the **c++** command.

file.c
Specifies the name of a C source file to be compiled. You can specify an MVS data set name by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *C*. (The conventions formerly used by **c89** for specifying data set names are still supported. See the environment variables **{_OSUFFIX_HOSTRULE}** and **{_OSUFFIX_HOSTQUAL}** for more information.)

file.I
Specifies the name of a IPA linker output file produced during the **c89/cc/c++** link-editing phase, when the **–W** option is specified with phase code **I**. IPA is further described under the **–W** option on page 562. By default the IPA linker output file is written to a temporary file. To have the IPA linker output file written to a permanent file, see the environment variable **{_TMPS}** under Environment Variables.

When an IPA linker output file is produced by **c89/cc/c++**, the default name is based upon the output file name. See the **–o** option under Options on page 559, for information on the name of the output file.

If the output file is named *a.out*, then the IPA linker output file is named *a.I*, and is always in the working directory. If the output file is named *//a.load*, then the IPA linker output file is named *//a.IPA*. If the output file specified already has a suffix, that suffix is replaced. Otherwise the suffix is appended. This file may also be specified on the command line, in which case it is used as a file to be link-edited.

file.i
Specifies the name of a preprocessed C or C++ source file to be compiled. You can specify an MVS data set name, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *CEX*.

When using the **c++** command, this source file is recognized as a C++ source file, otherwise it is recognized as a C source file. **c++** can be made to distinguish between the two. For more information see the environment variables **{_IXXSUFFIX}** and **{_IXXSUFFIX_HOST}**.

file.o
Specifies the name of a C, C++, or assembler object file, produced by **c89/cc/c++**, to be link-edited.

When an object file is produced by **c89/cc/c++**, the default name is based upon the source file. If the source file is named *file.c*, then the object file is named *file.o*, and is always in the working directory. If the source file were a data set named *//file.C*, then the object file is named *//file.OBJ*.

If the data set specified as an object file has undefined (U) record format, then it is assumed to be a load module. Load modules are not processed by the prelinker.

You can specify an MVS data set name to be link-edited, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *OBJ*. If a partitioned data set is specified, more than one member name may be specified by separating each with a comma (,), for example:

```
c89 //file.OBJ(mem1,mem2,mem3)
```

file.p
Specifies the name of a prelinker composite object file produced during the **c89/cc/c++** link-editing phase. By default the composite object file is written to a temporary file. To have the composite object file written to a permanent file, see the environment variable **{_TMPS}** under Environment Variables.

When a composite object file is produced by **c89/cc/c++**, the default name is based upon the output file name. See the **–o** option under Options on page 559, for information on the name of the output file.

If the output file is named *a.out*, then the composite object file is named *a.p*, and is always in the working directory. If the output file is named *//a.load*, then the composite object file is named *//a.CPOBJ*. If the output file specified already has a suffix, that suffix is replaced. Otherwise the suffix is appended. This file may also be specified on the command line, in which case it is used as a file to be link-edited.

file.s
Specifies the name of an assembler source file to be assembled. You can specify an MVS data set name, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *ASM*.

file.x
Specifies the name of a definition side-deck produced during the **c89/cc/c++** link-editing phase when creating a DLL (Dynamic Link Library), and used during the link-editing phase of an application using the DLL. DLLs are further described under the **–W** option.

When a definition side-deck is produced by **c89/cc/c++**, the default name is based upon the output file name. See the **–o** option under Options on page 559, for information on the name of the output file.

If the output file is named *a.dll*, then the definition side-deck is named *a.x*, and is always in the working directory. If the output file is named *//a.DLL*, then the definition side-deck is named *//a.EXP*. If the output file specified already has a suffix, that suffix is replaced. Otherwise the suffix is appended.

You can specify an MVS data set name to be link-edited, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *EXP*. If a partitioned data set is specified, more than one member name may be specified by separating each with a comma (,), for example:

```
c89 //file.EXP(mem1,mem2,mem3)
```

–l *libname*

Specifies the name of an archive library. **c89/cc/c++** searches for the file **lib***libname***.a** in the directories specified on the **–L** option and then in the usual places. The first occurrence of the archive library is used. For a description of the usual places, see Usage Note 7 on page 585.

You can also specify an MVS data set; you must specify the full data set name, because there are no rules for searching library directories.

The data set specified must be a C370LIB object library or a load library. If a data set specified as a library has undefined (U) record format, then it is assumed to be a load library. For more information about the z/OS C/C++ Object Library Utility, *z/OS C/C++ Programming Guide*. For more information about how load libraries are searched, see Usage Note 7 on page 585.

Environment Variables

You can use environment variables to specify necessary system and operational information to **c89/cc/c++**. When a particular environment variable is not set, **c89/cc/c++** uses the default shown. For information about the JCL parameters used in these environment variables, see *z/OS MVS JCL User's Guide*.

At the beginning of each environment variable description below, the name of the variable is shown in a *symbolic* notation. At the end of the description, the *actual* variable names used by the utilities are listed. The symbolic name is the same as the actual variable names, but omits the prefix and is enclosed in curly braces ({_variable_name}) to indicate that it is a symbolic name. Throughout the remainder of this command description, only the symbolic names are shown, but you must use the actual name when setting these variables. This means to specify **cc** environment variables, the name shown must be prefixed with **_CC** (for example, **_CC_ACCEPTABLE_RC**). To specify **c89** environment variables, the name shown must be prefixed with **_C89** (for example, **_C89_ACCEPTABLE_RC**). To specify **c++** environment variables, the name shown must be prefixed with **_CXX** (for example, **_CXX_ACCEPTABLE_RC**).

Note: **c89/cc/c++** can accept parameters only in the syntax indicated here. A null values indicate that **c89/cc/c++** is to omit the corresponding parameters during dynamic allocation. Numbers in parentheses following the environment variable name correspond to usage notes, which begin on Page 584, and indicate specific usage information for the environment variable.

{_ACCEPTABLE_RC}

The maximum allowed return code (result) of any step (compile, assemble, IPA link, prelink, or link-edit). If the result is between zero and this value (inclusive), then it is treated internally by **c89/cc/c++** exactly as if it were a zero result, except that message FSUM3065 is also issued. The default value is:

```
"4"
```

When used under **c89/cc/c++**, the prelinker by default returns at least a `4` when there are duplicate symbols or unresolved writable static symbols (but not for other unresolved references). The link editor returns at least a `4` when there are duplicate symbols, and at least an `8` when there are unresolved references and automatic library call was used.

Actual Variable Names: `_C89_ACCEPTABLE_RC`, `_CC_ACCEPTABLE_RC`, `_CXX_ACCEPTABLE_RC`

{_ASUFFIX} (15)

The suffix by which **c89/cc/c++** recognizes an archive file. This environment variable does not affect the treatment of archive libraries specified as **–l** operands, which are always prefixed with *lib* and suffixed with *·a*. The default value is:

```
"a"
```

Actual Variable Names: `_C89_ASUFFIX`, `_CC_ASUFFIX`, `_CXX_ASUFFIX`

{_ASUFFIX_HOST} (15)

The suffix by which **c89/cc/c++**recognizes a library data set. This environment variable does not affect the treatment of data set libraries specified as **–l** operands, which are always used exactly as specified. The default value is:

```
"LIB"
```

Actual Variable Names: `_C89_ASUFFIX_HOST`, `_CC_ASUFFIX_HOST`, `_CXX_ASUFFIX_HOST`

{_CCMODE}

Controls how **c89/cc/c++** does parsing. The default behavior of **c89/cc/c++** is to expect all options to precede all operands. Setting this variable allows compatibility with historical implementations (other **cc** commands). When set to 1, **c89/cc/c++** operates as follows:
- Options and operands can be interspersed.
- The double dash (—) is ignored.

Setting this variable to `0` results in the default behavior. The default value is:

```
"0"
```

Actual Variable Names: `_C89_CCMODE`, `_CC_CCMODE`, `_CXX_CCMODE`

{_CLIB_PREFIX} (14,17)

The prefix for the following named data sets used during the compilation phase and execution of your application.

To be used, the following data sets must be cataloged:
- The data sets {_CLIB_PREFIX}.SCLBH.+ contain the z/OS C/C++ Class Library include (header) files.
- The data set {_CLIB_PREFIX}.SCLBSID contains the z/OS C/C++ Class Library definition side-decks.

The following data sets are also used:

- The data set {_CLIB_PREFIX}.SCBCCMP contains the compiler programs called by **c89/cc/c++**.
- The data set {_CLIB_PREFIX}.SCLBDLL contains the z/OS C/C++ Class Library DLLs and messages.

The preceding data sets contain MVS programs that are invoked during the execution of **c89/cc/c++** and during the execution of a C/C++ application built by **c89/cc/c++**. To be executed correctly, these data sets must be made part of the MVS search order. Regardless of the setting of this or any other **c89/cc/c++** environment variable, **c89/cc/c++** does not affect the MVS search order. These data sets are listed here for information only, to assist in identifying the correct data sets to be added to the MVS program search order.

The default value is:

```
"CBC"
```

Actual Variable Names: `_C89_CLIB_PREFIX`, `_CC_CLIB_PREFIX`, `_CXX_CLIB_PREFIX`

{_CMEMORY}

A suggestion as to the use of compiler C/C++ Runtime Library memory files. When set to 0, **c89/cc/c++** will prefer to use the compiler **NOMEMORY** option. When set to 1, **c89/cc/c++** will prefer to use the compiler **MEMORY** option. When set to 1, and if the compiler **MEMORY** option can be used, **c89/cc/c++** need not allocate data sets for the corresponding work files. In this case it is the responsibility of the user to not override the compiler options (using the **–W** option) with the **NOMEMORY** option or any other compiler option which implies the **NOMEMORY** option.

The default value is:

```
"1"
```

Actual Variable Names: `_C89_CMEMORY`, `_CC_CMEMORY`, `_CXX_CMEMORY`

{_CMSGS} (14)

The Language Environment national language name used by the compiler program. A null value will cause the default Language Environment **NATLANG** runtime name to be used, and a non-null value must be a valid Language Environment **NATLANG** runtime option name (Language Environment runtime options are described in *z/OS Language Environment Programming Guide* . The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_CMSGS`, `_CC_CMSGS_RC`, `_CXX_CMSGS`

{_CNAME} (14)

The name of the compiler program called by **c89/cc/c++**. It must be a member of a data set in the search order used for MVS programs. The default value is:

```
"CBCDRVR"
```

Actual Variable Names: `_C89_CNAME`, `_CC_CNAME`, `_CXX_CNAME`

{_CSUFFIX} (15)
The suffix by which **c89/cc/c++** recognizes a C source file. The default value is:

```
"c"
```

Actual Variable Names: `_C89_CSUFFIX`, `_CC_CSUFFIX`, `_CXX_CSUFFIX`

{_CSUFFIX_HOST} (15)
The suffix by which **c89/cc/c++** recognizes a C source data set. The default value is:

```
"C"
```

Actual Variable Names: `_C89_CSUFFIX_HOST`, `_CC_CSUFFIX_HOST`, `_CXX_CSUFFIX_HOST`

{_CXXSUFFIX} (15)
The suffix by which **c++** recognizes a C++ source file. The default value is:

```
"C"
```

This environment variable is only supported by the **c++** command.

Actual Variable Names: `_C89_CXXSUFFIX`, `_CC_CXXSUFFIX`, `_CXX_CXXSUFFIX`

{_CXXSUFFIX_HOST} (15)
The suffix by which **c++** recognizes a C++ source data set. The default value is:

```
"CXX"
```

This environment variable is only supported by the **c++** command.

Actual Variable Names:`_C89_CXXSUFFIX_HOST`, `_CC_CXXSUFFIX_HOST`, `_CXX_CXXSUFFIX_HOST`

{_CSYSLIB} (4, 16)
The system library data set concatenation to be used to resolve *#include* directives during compilation.

Normally *#include* directives are resolved using all the information specified including the directory name. When **c89/cc/c++** can determine that the directory information can be used, such as when the include (header) files provided by Language Environment are installed in the default location (in accordance with **{_INCDIRS}**), then the default concatenation is:

```
"" (null)
```

When **c89/cc/c++** cannot determine that the directory information can be used, then the default concatenation is:

```
"{_PLIB_PREFIX}.SCEEH.H"
"{_PLIB_PREFIX}.SCEEH.SYS.H"
"{_PLIB_PREFIX}.SCEEH.ARPA.H"
"{_PLIB_PREFIX}.SCEEH.NET.H"
"{_PLIB_PREFIX}.SCEEH.NETINET.H"
```

When this variable is a null value, then no allocation is done for compiler system library data sets. In this case, the use of //DD:SYSLIB on the **–I** option and the *#include* directive will be unsuccessful. Unless there is a dependency on the use of //DD:SYSLIB, it is recommended that for improved performance this variable be allowed to default to a null value.

Actual Variable Names: `_C89_CSYSLIB`, `_CC_CSYSLIB`, `_CXX_CSYSLIB`

{_DAMPLEVEL}
The minimum severity level of dynamic allocation messages returned by dynamic allocation message processing. Messages with severity greater than or equal to this number are written to **stderr.** However, if the number is out of the range shown here (that is, less than 0 or greater than 8), then **c89/cc/c++** dynamic allocation message processing is disabled. The default value is:

```
"4"
```

Following are the values:

0	Informational
1–4	Warning
5–8	Severe

Actual Variable Names: `_C89_DAMPLEVEL`, `_CC_DAMPLEVEL`, `_CXX_DAMPLEVEL`

{_DAMPNAME} (14)
The name of the dynamic allocation message processing program called by **c89/cc/c++**. It must be a member of a data set in the search order used for MVS programs. The default dynamic allocation message processing program is described in *z/OS MVS Programming: Authorized Assembler Services Guide*. The default value is:

```
"IEFDB476"
```

Actual Variable Names: `_C89_DAMPNAME`, `_CC_DAMPNAME`, `_CXX_DAMPNAME`

{_DCBF2008} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format fixed unblocked and minimum block size of 2008. The block size must be in multiples of 8, and the maximum depends on the phase in which it is used but can be at least 5100. The default value is:

```
"(RECFM=F,LRECL=4088,BLKSIZE=4088)"
```

Actual Variable Names: `_C89_DCBF2008`, `_CC_DCBF2008`, `_CXX_DCBF2008`

{_DCBU} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format undefined and data set organization partitioned. This DCB is used by **c89/cc/c++** for the output file when it is to be written to a data set. The default value is:

```
"(RECFM=U,LRECL=0,BLKSIZE=6144,DSORG=PO)"
```

Actual Variable Names: `_C89_DCBU`, `_CC_DCBU`, `_CXX_DCBU`

{_DCB121M} (21)
The DCB parameters used by **c89/cc/c++**for data sets with the attributes of record format fixed blocked and logical record length 121, for data sets whose records may contain machine control characters. The default value is:

```
"(RECFM=FBM,LRECL=121,BLKSIZE=3630)"
```

Actual Variable Names: `_C89_DCB121M`, `_CC_DCB121M`, `_CXX_DCB121M`

{_DCB133M} (21)
The DCB parameters used by **c89/cc/c++**for data sets with the attributes of

record format fixed blocked and logical record length 133, for data sets whose records may contain machine control characters. The default value is:

```
"(RECFM=FBM,LRECL=133,BLKSIZE=3990)"
```

Actual Variable Names: `_C89_DCB133M`, `_CC_DCB133M`, `_CXX_DCB133M`

{_DCB137} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format variable blocked and logical record length 137. The default value is:

```
"(RECFM=VB,LRECL=137,BLKSIZE=882)"
```

Actual Variable Names: `_C89_DCB137`, `_CC_DCB137`, `_CXX_DCB137`

{_DCB137A} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format variable blocked and logical record length 137, for data sets whose records may contain ISO/ANSI control characters. The default value is:

```
"(RECFM=VB,LRECL=137,BLKSIZE=882)"
```

Actual Variable Names: `_C89_DCB137A`, `_CC_DCB137A`, `_CXX_DCB137A`

{_DCB3200} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format fixed blocked and logical record length 3200. The default value is:

```
"(RECFM=FB,LRECL=3200,BLKSIZE=12800)"
```

Actual Variable Names: `_C89_DCB3200`, `_CC_DCB3200`, `_CXX_DCB3200`

{_DCB80} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format fixed blocked and logical record length 80. This value is also used when **c89/cc/c++** allocates a new data set for an object file. The default value is:

```
"(RECFM=FB,LRECL=80,BLKSIZE=3200)"
```

Actual Variable Names: `_C89_DCB80`, `_CC_DCB80`, `_CXX_DCB80`

{_ELINES}
This variable controls whether the output of the **-E** option will include **#line** directives. **#line** directives provide information about the source file names and line numbers from which the preprocessed source came. The preprocessor only inserts **#line** directives where it is necessary. When set to 1, the output of the **c89/cc/c++ -E** option will include **#line** directives where necessary. When set to `0`, the output will not include any **#line** directives. The default value is:

```
"0"
```

Actual Variable Names: `_C89_ELINES`, `_CC_ELINES`, `_CXX_ELINES`

{_EXTRA_ARGS}
The setting of this variable controls whether **c89/cc/c++** treats a file operand with an unrecognized suffix as an error, or attempts to process it. When the **c++** command **–+** option is specified, all suffixes which otherwise

would be unrecognized are instead recognized as C++ source, effectively disabling this environment variable. See page 557 for information about the **–+** option.

When set to 0, **c89/cc/c++** treats such a file as an error and the command will be unsuccessful, because the suffix will not be recognized.

When set to 1, **c89/cc/c++** treats such a file as either an object file or a library, depending on the file itself. If it is neither an object file nor a library then the command will be unsuccessful, because the link-editing phase will be unable to process it. The default value for **c89** and **c++** is:

```
"0"
```

The default value for **cc** is:

```
"1"
```

Actual Variable Names: `_C89_EXTRA_ARGS`, `_CC_EXTRA_ARGS`, `_CXX_EXTRA_ARGS`

{_ILCTL} (14)
The name of the control file used by the IPA linker program. By default the control file is not used, so the **–W** option must be specified to enable its use, as in:

```
c89 -WI,control ...
```

The default value is:

```
"ipa.ctl"
```

Actual Variable Names: `_C89_ILCTL`, `_CC_ILCTL`, `_CXX_ILCTL`

{_ILMSGS} (14)
The name of the message data set member, or the Language Environment national language name, used by the IPA linker program. The default value is whatever **{_CMSGS}** is. So if **{_CMSGS}** is set or defaults to "" (null), the default value is:

```
"" (null)
```

Actual Variable Names: `_C89_ILMSGS`, `_CC_ILMSGS`, `_CXX_ILMSGS`

{_ILNAME} (14)
The name of the IPA linker program called by **c89/cc**. It must be a member of a data set in the search order used for MVS programs. The default value is whatever **{_CNAME}** is. So if **{_CNAME}** is set or defaults to "CBCDRVR" the default value is:

```
"CBCDRVR"
```

Actual Variable Names: `_C89_ILNAME`, `_CC_ILNAME`, `_CXX_ILNAME`

{_ILSUFFIX} (15)
The suffix **c89/cc** uses when creating an IPA linker output file. The default value is:

```
"I"
```

Actual Variable Names: `_C89_ILSUFFIX`, `_CC_ILSUFFIX`, `_CXX_ILSUFFIX`

{_ILSUFFIX_HOST} (15)
The suffix **c89/cc** uses when creating an IPA linker output data set. The default value is:

```
"IPA"
```

Actual Variable Names: `_C89_ILSUFFIX_HOST`, `_CC_ILSUFFIX_HOST`, `_CXX_ILSUFFIX_HOST`

{_ILSYSLIB} (7, 16)
The system library data set list to be used to resolve symbols during the IPA link step of the link-editing phase of non-XPLINK programs. The default value is whatever {_PSYSLIB} is set or defaults to, followed by whatever {_LSYSLIB} is set or defaults to.

Actual Variable Names: `_C89_ILSYSLIB`, `_CC_ILSYSLIB`, `_CXX_ILSYSLIB`

{_ILSYSIX} (7, 16)
The system definition side-deck list to be used to resolve symbols during the IPA link step of the link-editing phase in non-XPLINK programs. The default value is whatever {_PSYSIX} is set or defaults to.

Actual Variable Names: `_C89_ILSYSIX`, `_CC_ILSYSIX`, `_CXX_ILSYSIX`

{_ILXSYSLIB} (7, 16)
The system library data set list to be used to resolve symbols during the IPA link step of the link-editing phase when using XPLINK (see XPLINK (Extra Performance Linkage) in “Options” on page 557). The default value is whatever {_LXSYSLIB} is set or defaults to.

Actual Variable Names: `_C89_ILXSYSLIB`, `_CC_ILXSYSLIB`, `_CXX_ILXSYSLIB`

{_ILXSYSIX} (7, 16)
The system definition side-deck list to be used to resolve symbols during the IPA link step of the link-editing phase when using XPLINK (see XPLINK (Extra Performance Linkage) in “Options” on page 557). The default value is whatever {_LXSYSIX}} is set or defaults to.

Actual Variable Names: `_C89_ILXSYSIX`, `_CC_ILXSYSIX`, `_CXX_ILXSYSIX`

{_INCDIRS} (22)
The directories used by **c89/cc/c++** as a default place to search for include files during compilation (before searching **{_INCLIBS}** and **{_CSYSLIB}**). If **c++** is not being used the default value is:

```
"/usr/include"
```

If **c++** is being used the default value is:

```
/usr/include /usr/lpp/ioclib/include
```

Actual Variable Names: `_C89_INCDIRS`, `_CC_INCDIRS`, `_CXX_INCDIRS`

{_INCLIBS} (22)
The directories used by **c89/cc/c++** as a default place to search for include files during compilation (after searching **{_INCDIRS}** and before searching **{_CSYSLIB}**). The default value depends on whether or not **c++** is being used. If **c++** is not being used the default value is:

```
"//'{_PLIB_PREFIX}.SCEEH.+'"
```

If **c++** is being used, the default value is:

```
"//'{_PLIB_PREFIX}.SCEEH.+' //'{_CLIB_PREFIX}.SCLBH.+'"
```

Actual Variable Names: `_C89_INCLIBS`, `_CC_INCLIBS`, `_CXX_INCLIBS`

{_ISUFFIX} (15)
: The suffix by which **c89/cc/c++** recognizes a preprocessed C source file. The default value is:

```
"i"
```

Actual Variable Names: `_C89_ISUFFIX`, `_CC_ISUFFIX`, `_CXX_ISUFFIX`

{_ISUFFIX_HOST} (15)
: The suffix by which **c89/cc/c++** recognizes a preprocessed (expanded) C source data set. The default value is:

```
"CEX"
```

Actual Variable Names: `_C89_ISUFFIX_HOST`, `_CC_ISUFFIX_HOST`, `_CXX_ISUFFIX_HOST`

{_IXXSUFFIX} (15)
: The suffix by which **c++** recognizes a preprocessed C++ source file. The default value is:

```
"i"
```

This environment variable is only supported by the **c++** command.

Actual Variable Names: `_C89_IXXSUFFIX`, `_CC_IXXSUFFIX`, `_CXX_IXXSUFFIX`

{_IXXSUFFIX_HOST} (15)
: The suffix by which **c++** recognizes a preprocessed (expanded) C++ source data set. The default value is:

```
"CEX"
```

This environment variable is only supported by the **c++** command.

Actual Variable Names: `_C89_IXXSUFFIX_HOST`, `_CC_IXXSUFFIX_HOST`, `_CXX_IXXSUFFIX_HOST`

{_LIBDIRS} (22)
: The directories used by **c89/cc/c++** as the default place to search for archive libraries which are specified using the **–l** operand. The default value is:

```
"/lib /usr/lib"
```

Actual Variable Names: `_C89_LIBDIRS`, `_CC_LIBDIRS`, `_CXX_LIBDIRS`

{_LSYSLIB} (7, 16)
: The system library data set concatenation to be used to resolve symbols during the IPA link step and the link-edit step of the non-XPLINK link-editing phase. The **{_PSYSLIB}** libraries always precede the **{_LSYSLIB}** libraries when resolving symbols in the link-editing phase. The default value is the concatenation:

```
"{_PLIB_PREFIX}.SCEELKEX"
"{_PLIB_PREFIX}.SCEELKED"
"{_SLIB_PREFIX}.CSSLIB"
```

Actual Variable Names: `_C89_LSYSLIB`, `_CC_LSYSLIB`, `_CXX_LSYSLIB`

{_LXSYSLIB} (7, 16)
: The system library data set concatenation to be used to resolve symbols

during the IPA link step and the link-editing phase when using XPLINK (see XPLINK (Extra Performance Linkage) in “Options” on page 557). The default value is the concatenation:

```
"{_PLIB_PREFIX}.SCEEBIND"
"{_SLIB_PREFIX}.CSSLIB"
```

Actual Variable Names: `_C89_LXSYSLIB`, `_CC_LXSYSLIB`, `_CXX_LXSYSLIB`

{_LXSYSIX} (7, 16)
The system definition side-deck list to be used to resolve symbols during the link-editing phase when using XPLINK (see XPLINK (Extra Performance Linkage) in “Options” on page 557). A definition side-deck contains link-editing phase IMPORT control statements naming symbols which are exported by a DLL. The default value depends on whether or not **c++** is being used. If **c++** is not being used, the default value is the list containing the single entry:

```
"{_PLIB_PREFIX}.SCEELLIB(CELHS003,CELHS001)"
```

If **c++** is being used, the default value is the list:

```
"{_PLIB_PREFIX}.SCEELIB(CELHS003,CELHS001,CELHSCPP)"
"{_CLIB_PREFIX}.SCLBSID(ASCCOLL,COMPLEX,IOSTREAM)"
```

Actual Variable Names: `_C89_LSYSLIB`, `_CC_LSYSLIB`, `_CXX_LSYSLIB`

{_MEMORY}
A suggestion as to the use of C/C++ Runtime Library memory files by **c89/cc/c++**. When set to 0, **c89/cc/c++** uses temporary data sets for all work files. When set to 1, **c89/cc/c++** uses memory files for all work files that it can. The default value is:

```
"1"
```

Actual Variable Names: `_C89_MEMORY`, `_CC_MEMORY`, `_CXX_MEMORY`

{_NEW_DATACLAS} (18)
The DATACLAS parameter used by **c89/cc/c++** for any new datasets it creates. The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_NEW_DATACLAS`, `_CC_NEW_DATACLAS`, `_CXX_NEW_DATACLAS`

{_NEW_DSNTYPE} (18, 20)
The DSNTYPE parameter used by **c89/cc/c++** for any new data sets it creates. The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_NEW_DSNTYPE`, `_CC_NEW_DSNTYPE`, `_CXX_NEW_DSNTYPE`

{_NEW_MGMTCLAS} (18)
The MGMTCLAS parameter used by **c89/cc/c++** for any new datasets it creates. The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_NEW_MGMTCLAS`, `_CC_NEW_MGMTCLAS`, `_CXX_NEW_MGMTCLAS`

{_NEW_SPACE} (18, 19)
The SPACE parameters used by **c89/cc/c++**for any new data sets it creates. A value for the number of directory blocks should always be specified. When allocating a sequential data set, **c89/cc/c++** automatically ignores the specification. The default value is:
`"(,(10,10,10))"`

Actual Variable Names: `_C89_NEW_SPACE`, `_CC_NEW_SPACE`, `_CXX_NEW_SPACE`

{_NEW_STORCLAS} (18)
The STORCLAS parameter used by **c89/cc/c++**for any new data sets it creates. The default value is:
`"" (null)`

Actual Variable Names: `_C89_NEW_STORCLAS`, `_CC_NEW_STORCLAS`, `_CXX_NEW_STORACLAS`

{_NEW_UNIT} (18)
The UNIT parameter used by **c89/cc/c++** for any new data sets it creates. The default value is:
`"" (null)`

Actual Variable Names: `_C89_NEW_UNIT`, `_CC_NEW_UNIT`, `_CXX_NEW_UNIT`

{_OPERANDS} (22)
These operands are parsed as if they were specified after all other operands on the **c89/cc/c++** command line. The default value is:
`"" (null)`

Actual Variable Names: `_C89_OPERANDS`, `_CC_OPERANDS`, `_CXX_OPERANDS`

{_OPTIONS} (22)
These options are parsed as if they were specified before all other options on the **c89/cc/c++** command line. The default value is:
`"" (null)`

Actual Variable Names: `_C89_OPTIONS`, `_CC_OPTIONS`, `_CXX_OPTIONS`

{_OSUFFIX} (15)
The suffix by which **c89/cc/c++** recognizes an object file. The default value is:
`"o"`

Actual Variable Names: `_C89_OSUFFIX`, `_CC_OSUFFIX`, `_CXX_OSUFFIX`

{_OSUFFIX_HOST} (15)
The suffix by which **c89/cc/c++** recognizes an object data set. The default value is:
`"OBJ"`

Actual Variable Names: `_C89_OSUFFIX_HOST`, `_CC_OSUFFIX_HOST`, `_CXX_OSUFFIX_HOST`

{_OSUFFIX_HOSTQUAL}
The data set name of an object data set is determined by the setting of this option. If it is set to 0, then the suffix **{_OSUFFIX_HOST}** is appended to the source data set name to produce the object data set name. If it is set to 1, then the suffix **{_OSUFFIX_HOST}** replaces the last qualifier of the

source data set name to produce the object data set name (unless there is only a single qualifier, in which case the suffix is appended). The default value is:

```
"1"
```

Note: Earlier versions of **c89** always appended the suffix, which was inconsistent with the treatment of files in the hierarchical file system. It is recommended that any existing data sets be converted to use the new convention.

Actual Variable Names: `_C89_OSUFFIX_HOSTQUAL`, `_CC_OSUFFIX_HOSTQUAL`, `_CXX_OSUFFIX_HOSTQUAL`

{_OSUFFIX_HOSTRULE}
The way in which suffixes are used for host data sets is determined by the setting of this option. If it is set to 0, then data set types are determined by the rule described in the note which follows. If it is set to 1, then the data set types are determined by last qualifier of the data set (just as a suffix is used to determine the type of hierarchical file system file). Each host file type has an environment variable by which the default suffix can be modified. The default value is:

```
"1"
```

Notes:

1. Earlier versions of **c89** scanned the data set name to determine if it was an object data set. It searched for the string **OBJ** in the data set name, exclusive of the first qualifier and the member name. If it was found, the data set was determined to be an object data set, and otherwise it was determined to be a C source data set. It is recommended that any existing data sets be converted to use the new convention. Also, because the earlier convention only provided for recognition of C source files, assembler source cannot be processed if it is used.
2. The **c++** command does not support this environment variable, as the earlier convention would not provide for recognition of both C++ and C source files. Therefore regardless of its setting, **c++** always behaves as if it is set to "1".

Actual Variable Names: `_C89_OSUFFIX_HOSTRULE`, `_CC_OSUFFIX_HOSTRULE`, `_CXX_OSUFFIX_HOSTRULE`

{_PLIB_PREFIX} (14,17)
The prefix for the following named data sets used during the compilation, assemble, and link-editing phases, and during the execution of your application.

To be used, the following data sets must be cataloged:

- The data sets {_PLIB_PREFIX}.SCEEH.+ contain the include (header) files for use with the runtime library functions (where + can be any of H, SYS.H, ARPA.H, NET.H, and NETINET.H).
- The data set {_PLIB_PREFIX}.SCEEMAC contains COPY and MACRO files to be used during assembly.
- The data sets {_PLIB_PREFIX}.SCEEOBJ and {_PLIB_PREFIX}.SCEECPP contain runtime library bindings which exploit constructed reentrancy, used during the link-editing phase of non-XPLINK programs.

- The data set {_PLIB_PREFIX}.SCEELKEX contains C runtime library bindings which exploit L-names used during the link-editing phase of non-XPLINK programs. For more information about L-names, see the usage note 23 on page 589.
- The data set {_PLIB_PREFIX}.SCEELKED contains all other Language Environment runtime library bindings, used during the link-editing phase of non-XPLINK programs.
- The data set {_PLIB_PREFIX}.SCEEBIND contains all static Language Environment runtime library bindings, used during the link-editing phase of XPLINK programs.
- The data set {_PLIB_PREFIX}.SCEELIB contains the definition side-decks for the runtime library bindings (CELHS003 and CELHSCPP), and the Language Environment Callable Services (member CELHS001), used during the link-editing phase of XPLINK programs.

The following data sets are also used:

- The data sets {_PLIB_PREFIX}.SCEERUN and {_PLIB_PREFIX}.SCEERUN2 contains the runtime library programs.

The above data sets contain MVS programs that are invoked during the execution of **c89/cc/c++** and during the execution of a C/C++ application built by **c89/cc/c++**. To be executed correctly, these data sets must be made part of the MVS search order. Regardless of the setting of this or any other **c89/cc/c++** environment variable, **c89/cc/c++** does not affect the MVS program search order. These data sets are listed here for information only, to assist in identifying the correct data sets to be added to the MVS program search order. The default value is:

```
"CEE"
```

Actual Variable Names: `_C89_PLIB_PREFIX`, `_CC_PLIB_PREFIX`, `_CXX_PLIB_PREFIX`

{_PMEMORY}
A suggestion as to the use of prelinker C/C++ Runtime Library memory files. When set to 0, **c89/cc/c++** uses the prelinker **NOMEMORY** option. When set to 1, **c89/cc/c++** uses the prelinker **MEMORY** option. The default value is:

```
"1"
```

`_C89_PMEMORY`, `_CC_PMEMORY`, `_CXX_PMEMORY`

{_PMSGS} (14)
The name of the message data set used by the prelinker program. It must be a member of the cataloged data set {_PLIB_PREFIX}.SCEEMSGP. The default value is:

```
 "EDCPMSGE"
```

Actual Variable Names: `_C89_PMSGS`, `_CC_PMSGS`, `_CXX_PMSGS`

{_PNAME} (14)
The name of the prelinker program called by **c89/cc/c++**. It must be a member of a data set in the search order used for MVS programs. The prelinker program is shipped as a member of the {_PLIB_PREFIX}.SCEERUN data set. The default value is:

```
"EDCPRLK"
```

Actual Variable Names: `_C89_PNAME`, `_CC_PNAME`, `_CXX_PNAME`

{_PSUFFIX} (15)
The suffix **c89/cc/c++** uses when creating a prelinker (composite object) output file. The default value is:

```
"p"
```

Actual Variable Names: `_C89_PSUFFIX`, `_CC_PSUFFIX`, `_CXX_PSUFFIX`

{_PSUFFIX_HOST} (15)
The suffix **c89/cc/c++** uses when creating a prelinker (composite object) output data set. The default value is:

```
"CPOBJ"
```

Actual Variable Names: `_C89_PSUFFIX_HOST`, `_CC_PSUFFIX_HOST`, `_CXX_PSUFFIX_HOST`

{_PSYSIX} (16)
The system definition side-deck list to be used to resolve symbols during the non-XPLINK link-editing phase. A definition side-deck contains link-editing phase **IMPORT** control statements naming symbols which are exported by a DLL. The default value when **c++** is being used is the list containing the following single entry. Otherwise, the default value is null:

```
"{_CLIB_PREFIX}.SCLBSID(ASCCOLL,COMPLEX,IOSTREAM)"
```

Actual Variable Names: `_C89_PSYSIX`, `_CC_PSYSIX`, `_CXX_PSYSIX`

{_PSYSLIB} (16)
The system library data set list to be used to resolve symbols during the non-XPLINK link-editing phase. The **{_PSYSLIB}** libraries always precede the **{_LSYSLIB}** libraries when resolving symbols in the link-editing phase. The default value depends on whether or not **c++** is being used. If **c++** is not being used, the default value is the list containing the single entry:

```
"{_PLIB_PREFIX}.SCEEOBJ"
```

If **c++** is being used, the default value is the list:

```
"{_PLIB_PREFIX}.SCEEOBJ"
"{_PLIB_PREFIX}.SCEECPP"
```

Actual Variable Names: `_C89_PSYSLIB`, `_CC_PSYSLIB`, `_CXX_PSYSLIB`

{_SLIB_PREFIX} (17)
The prefix for the named data sets used by the link editor (CSSLIB) and the assembler system library data sets (MACLIB and MODGEN). The data set {_SLIB_PREFIX}.CSSLIB contains the z/OS UNIX assembler callable services bindings. The data sets {_SLIB_PREFIX}.MACLIB and {_SLIB_PREFIX}.MODGEN contain COPY and MACRO files to be used during assembly. These data sets must be cataloged to be used. The default value is:

```
"SYS1"
```

Actual Variable Names: `_C89_SLIB_PREFIX`, `_CC_SLIB_PREFIX`, `_CXX_SLIB_PREFIX`

{_SNAME} (14)
The name of the assembler program called by **c89/cc/c++**. It must be a member of a data set in the search order used for MVS programs. The default value is:

```
"ASMA90"
```

Actual Variable Names: `_C89_SNAME`, `_CC_SNAME`, `_CXX_SNAME`

{_SSUFFIX} (15)
: The suffix by which **c89/cc/c++** recognizes an assembler source file. The default value is:

```
"s"
```

Actual Variable Names: `_C89_SSUFFIX`, `_CC_SSUFFIX`, `_CXX_SSUFFIX`

{_SSUFFIX_HOST} (15)
: The suffix by which **c89/cc/c++** recognizes an assembler source data set. The default value is:

```
"ASM"
```

Actual Variable Names: `_C89_SSUFFIX_HOST`, `_CC_SSUFFIX_HOST`, `_CXX_SSUFFIX_HOST`

{_SSYSLIB} (16)
: The system library data set concatenation to be used to find COPY and MACRO files during assembly. The default concatenation is:

```
"{_PLIB_PREFIX}.SCEEMAC"
"{_SLIB_PREFIX}.MACLIB"
"{_SLIB_PREFIX}.MODGEN"
```

Actual Variable Names: `_C89_SSYSLIB`, `_CC_SSYSLIB`, `_CXX_SSYSLIB`

{_STEPS}
: The steps that are executed for the link-editing phase can be controlled with this variable. For example, the prelinker step can be enabled, so that the inputs normally destined for the link editor instead go into the prelinker, and then the output of the prelinker becomes the input to the link editor.

This variable allows the prelinker to be used in order to produce output which is compatible with previous releases of **c89/cc/c++**. The prelinker is normally used by **c89/cc/c++** when the output file is a data set which is not a PDSE (partitioned data set extended).

Note: The Prelinker and **XPLINK** are incompatible. When using the link editor **XPLINK** option, the Prelinker cannot be used. Thus, specifying the Prelinker on this variable will have no effect.

The format of this variable is a set of binary switches which either enable (when turned on) or disable (when turned off) the corresponding step. Turning a switch on will not cause a step to be enabled if it was not already determined by **c89/cc/c++** that any other conditions necessary for its use are satisfied. For example, the IPA link step will not be executed unless the **–W** option is specified to enable the IPA linker. Enabling the IPA linker is described under the **–W** option on page 562.

Considering this variable to be a set of 32 switches, numbered left-to-right from 0 to 31, the steps corresponding to each of the switches are as follows:

0-27	Reserved
28	TEMPINC/IPATEMP
29	IPALINK
30	PRELINK

31 LINKEDIT

For example, to override the default behavior of **c89/cc/c++** and cause the prelinker step to be run (this is also the default when the output file is a data set which is not a PDSE), set this variable to:

```
"0xffffffff" or the equivalent, -1
```

The default value when the output file is an HFS file or a PDSE data set is:

```
"0xfffffffD" or the equivalent, -3
```

Note: The IPATEMP step is the IPA equivalent of the TEMPINC (automatic template generation) step, just as the IPACOMP step is the IPA equivalent of the COMPILE step. See the description of IPA under the **-W** option for more information.

Actual Variable Names: `_C89_STEPS`, `_CC_STEPS`, `_CXX_STEPS`

{_SUSRLIB} (16)
The user library data set concatenation to be used to find COPY and MACRO files during assembly (before searching **{_SSYSLIB}**). The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_SUSRLIB`, `_CC_SUSRLIB`, `_CXX_SUSRLIB`

{_TMPS}
The use of temporary files by **c89/cc/c++** can be controlled with this variable.

The format of this variable is a set of binary switches which either cause a temporary file to be used (when turned on) or a permanent file to be used (when turned off) in the corresponding step.

The correspondence of these switches to steps is the same as for the variable **{_STEPS}**. Only the prelinker and IPA linker output can be captured using this variable.

For example, to capture the prelinker output, set this variable to:

```
"0xfffffffD" or the equivalent, -3
```

The default value is :

```
"0xffffffff" or the equivalent, -1
```

Actual Variable Names: `_C89_TMPS`, `_CC_TMPS`, `_CXX_TMPS`

{_WORK_DATACLAS} (18)
The DATACLAS parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_WORK_DATACLAS`, `_CC_WORK_DATACLAS`, `_CXX_WORK_DATACLAS`

{_WORK_DSNTYPE} (18, 20)
The DSNTYPE parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_WORK_DSNTYPE`, `_CC_WORK_DSNTYPE`, `_CXX_WORK_DSNTYPE`

{_WORK_MGMTCLAS} (18)
The MGMTCLAS parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_WORK_MGMTCLAS`, `_CC_WORK_MGMTCLAS`, `_CXX_WORK_MGMTCLAS`

{_WORK_SPACE} (18, 19)
The SPACE parameters used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"(32000,(30,30))"
```

Actual Variable Names: `_C89_WORK_SPACE`, `_CC_WORK_SPACE`, `_CXX_WORK_SPACE`

{_WORK_STORCLAS} (18)
The STORCLAS parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"" (null)
```

Actual Variable Names: `_C89_WORK_STORCLAS`, `_CC_WORK_STORCLAS`, `_CXX_WORK_STORCLAS`

{_WORK_UNIT} (18)
The UNIT parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"SYSDA"
```

Actual Variable Names: `_C89_WORK_UNIT`, `_CC_WORK_UNIT`, `_CXX_WORK_UNIT`

{_XSUFFIX} (15)
The suffix by which **c89/cc/c++** recognizes a definition side-deck file of e*x*ported symbols. The default value is:

```
"x"
```

Actual Variable Names: `_C89_XSUFFIX`, `_CC_XSUFFIX`, `_CXX_XSUFFIX`

{_XSUFFIX_HOST} (15)
The suffix by which **c89/cc/c++** recognizes a definition side-deck data set of e*x*ported symbols. The default value is:

```
"EXP"
```

Actual Variable Names: `_C89_XSUFFIX_HOST`, `_CC_XSUFFIX_HOST`, `_CXX_XSUFFIX_HOST`

Files

libc.a	C/C++ Runtime Library function library (see Usage Note 7 on page 585).
libm.a	C/C++ Runtime Library math function library (see Usage Note 7 on page 585).
libl.a	**lex** function library.
liby.a	**yacc** function library.

/dev/fd0, **/dev/fd1**, **...**
: Character special files required by **c89/cc/c++**. For installation information, see *z/OS UNIX System Services Planning*.

/usr/include
: The usual place to search for include files (see Usage Note 4 on page 584).

/lib
: The usual place to search for runtime library bindings (see Usage Note 7 on page 585).

/usr/lib
: The usual place to search for runtime library bindings (see Usage Note 7 on page 585).

Usage Notes

1. To be able to specify an operand that begins with a dash (–), before specifying any other operands that do not, you must use the double dash (—) end-of-options delimiter. This also applies to the specification of the **–l** operand. (See the description of environment variable {_CCMODE} for an alternate style of argument parsing.)
2. When invoking **c89/cc/c++** from the shell, any option-arguments or operands specified that contain characters with special meaning to the shell must be escaped. For example, some **–W** option-arguments contain parentheses. Source files specified as PDS member names contain parentheses; if they are specified as fully qualified names, they contain single quotes.

   To escape these special characters, either enclose the option-argument or operand in double quotes, or precede each character with a backslash.
3. Some **c89/cc/c++** behavior applies only to files (and not to data sets).
   - The **–o** option does not allow a source file (*file.C*, *file.c*, *file.i*, *file.s*) to be specified.
   - If the compile or assemble is not successful, the corresponding object file (*file.o*) is always removed.
   - If the DLL option is passed to the link-editing phase, and afterwards the *file.x* file exists but has a size of zero, then that file is removed.
4. MVS data sets may be used as the usual place to resolve C and C++ **#include** directives during compilation.

   Such data sets are installed with Language Environment. When it is allocated, searching for these include files can be specified on the **–I** option as *//DD:SYSLIB*. (See the description of environment variable **{_CSYSLIB}** for information.

   When include files are MVS PDS members, z/OS C/C++ uses conversion rules to transform the include (header) file name on a **#include** preprocessor directive into a member name. If the "//'*dataset_prefix*.+'" syntax is not used for the MVS data set which is being searched for the include file, then this transformation strips any directory name on the **#include** directive, and then takes the first 8 or fewer characters up to the first dot `(.)`.

   If the "//'*dataset_prefix*.+'" syntax is used for the MVS data set which is being searched for the include file, then this transformation uses any directory name on the **#include** directive, and the characters following the first dot `(.)`, and substitutes the "+" of the dataset being searched with these qualifiers.

   In both cases the data set name and member name are converted to uppercase and underscores `(_)` are changed to at signs `(@)`.

If the include (header) files provided by Language Environment are installed into the hierarchical file system in the default location (in accordance with the **{_INCDIRS}** environment variable), then the compiler will use those files to resolve **#include** directives during compilation. **c89/cc/c++** by default searches the directory **/usr/include** as the usual place, just before searching the data sets just described. See the description of environment variables **{_CSYSLIB}**, **{_INCDIRS}**, and **{_INCLIBS}** for information on customizing the default directories to search.

5. Feature test macros control which symbols are made visible in a source file (typically a header file). **c89/cc/c++** automatically defines the following feature test macros along with the errno macro, according to whether or not **cc** was invoked.
   - Other than **cc**
     –D "errno=(*__errno())"
     –D _OPEN_DEFAULT=1
   - **cc**
     –D "errno=(*__errno())"
     –D _OPEN_DEFAULT=0
     –D _NO_PROTO=1

   c89/cc/c++add these macro definitions only after processing the command string. Therefore, you can override these macros by specifying **–D** or **–U** options for them on the command string.
6. The default LANGLVL and UPCONV compiler options are set according to whether or not **cc** was invoked. The LANGLVL compiler option affects z/OS C/C++ predefined macros, which are used like feature test macros to control which symbols are made visible in a source file (typically a header file), but are not defined or undefined except by this compiler option. Together the LANGLVL and UPCONV options affect the language rules used by the compiler. For more information about these compiler options, see *z/OS C/C++ User's Guide* . For more information about z/OS C/C++ predefined macros, see *z/OS C/C++ Language Reference*. The options are shown here in a syntax that the user can specify on the **c89/cc/c++** command line to override them:
   - Other than **cc**
     –W "0,langlvl(ansi),noupconv"
   - **cc**
     –W "0,langlvl(commonc),upconv"
7. By default the usual place for the **–L** option search is the **/lib** directory followed by the **/usr/lib** directory. See the description of environment variable **{_LIBDIRS}** for information on customizing the default directories to search.

   The archive libraries **libc.a** and **libm.a** exist as files in the usual place for consistency with other implementations. However, the runtime library bindings are not contained in them.

   Instead, MVS data sets installed with the Language Environment runtime library are used as the usual place to resolve runtime library bindings. In the final step of the link-editing phase, any MVS load libraries specified on the **–l** operand are searched in the order specified, followed by searching these data sets. See the **{_PLIB_PREFIX}** description, as well as descriptions of the following environment variables:

   This list of environment variables affects the link-editing phase of **c89**, but only for non-XPLINK link-editing. See XPLINK (Extra Performance Linkage) in "Options" on page 557.

   {_ILSYSLIB}
   {_ILSYSIX}

{_LSYSLIB}
{_PSYSIX}
{_PSYSLIB}

This list of environment variables affects the link-editing phase of **c89**, but only for XPLINK link-editing. See XPLINK (Extra Performance Linkage) in "Options" on page 557.

{_ILXSYSLIB}
{_ILXSYSIX}
{_LXSYSLIB}
{_LXSYSIX}

8. Because archive library files are searched when their names are encountered, the placement of **–l** operands and *file.a* operands is significant. You may have to specify a library multiple times on the command string, if subsequent specification of *file.o* files requires that additional symbols be resolved from that library.
9. When the prelinker is used during the link-editing phase, you cannot use as input to **c89/cc/c++** an executable file produced as output from a previous use of **c89/cc/c++**. The output of **c89/cc/c++** when the **–r** option is specified (which is not an executable file) may be used as input.
10. All MVS data sets used by **c89/cc/c++** must be cataloged (including the system data sets installed with the z/OS C/C++ compiler and the Language Environment runtime library).
11. **c89/cc/c++** operation depends on the correct setting of their installation and configuration environment variables (see "Environment Variables" on page 567). Also, they require that certain character special files are in the **/dev** directory. For additional installation and configuration information, see *z/OS UNIX System Services Planning*.
12. Normally, options and operands are processed in the order read (from left to right). Where there are conflicts, the last specification is used (such as with **–g** and **–s**). However, some **c89/cc/c++** options will override others, regardless of the order in which they are specified. The option priorities, in order of highest to lowest, are as follows:

    –v specified twice
    The pseudo-JCL is printed only, but the effect of all the other options and operands as specified is reflected in the pseudo-JCL.

    –E Overrides **–0**, **–O**, **–1**, **–2**, **–V**, **–c**, **–g** and **–s** (also ignores any *file.s* files).

    –g Overrides **–0**, **–O**, **–1**, **–2**, and **–s**.

    –s Overrides **–g** (the last one specified is honored).

    –0, –O, –1, –2, –V, –c
    All are honored if not overridden. **–0**, **–O**, **–1**, and **–2**, override each other (the last one specified is honored).
13. For options that have option-arguments, the meaning of multiple specifications of the options is as follows:

    –D All specifications are used. If the same name is specified on more than one **–D** option, only the first definition is used.

    –e The entry function used will be the one specified on the last **–e** option.

    –I All specifications are used. If the same directory is specified on more than one **–I** option, the directory is searched only the first time.

–L All specifications are used. If the same directory is specified on more than one **–L** option, the directory is searched only the first time.

–o The output file used will be the one specified on the last **–o** option.

–U All specifications are used. The name is *not* defined, regardless of the position of this option relative to any **–D** option specifying the same name.

–u All specifications are used. If a definition cannot be found for any of the functions specified, the link-editing phase will be unsuccessful.

–W All specifications are used. All options specified for a phase are passed to it, as if they were concatenated together in the order specified.

14. The following environment variables can be at most eight characters in length. For those whose values specify the names of MVS programs to be executed, you can dynamically alter the search order used to find those programs by using the **STEPLIB** environment variable.

    c89/cc/c++ environment variables do not affect the MVS program search order. Also, for **c89/cc/c++** to work correctly, the setting of the **STEPLIB** environment variable should reflect the Language Environment library in use at the time that **c89/cc/c++** is invoked.

    For more information on the **STEPLIB** environment variable, see *z/OS UNIX System Services Planning*. It is also described under the **sh** command. Note that the STEPLIB allocation in the pseudo-JCL produced by the **–v** verbose option is shown as a comment, and has no effect on the MVS program search order. Its appearance in the pseudo-JCL is strictly informational.

    {_CMSGS}
    {_CNAME}
    {_DAMPNAME}
    {_ILNAME}
    {_ILMSGS}
    {_PMSGS}
    {_PNAME}
    {_SNAME}

15. The following environment variables can be at most 15 characters in length. You should not specify any dots (·) when setting these environment variables since they would then never match their corresponding operands:

    {_ASUFFIX}
    {_ASUFFIX_HOST}
    {_CSUFFIX}
    {_CSUFFIX_HOST}
    {_CXXSUFFIX}
    {_CXXSUFFIX_HOST}
    {_ISUFFIX}
    {_ISUFFIX_HOST}
    {_ILSUFFIX}
    {_ILSUFFIX_HOST}
    {_IXXSUFFIX}
    {_IXXSUFFIX_HOST}
    {_OSUFFIX}
    {_OSUFFIX_HOST}
    {_PSUFFIX}
    {_PSUFFIX_HOST}
    {_SSUFFIX}
    {_SSUFFIX_HOST}

{_XSUFFIX}
{_XSUFFIX_HOST}

16. The following environment variables are parsed as colon-delimited data set names, and represent a data set concatenation or a data set list. The maximum length of each specification is 1024 characters:
    {_CSYSLIB}
    {_ILSYSLIB}
    {_ILSYSIX}
    {_ILXSYSLIB}
    {_ILXSYSIX}
    {_LSYSLIB}
    {_LXSYSLIB}
    {_LXSYSIX}
    {_PSYSIX}
    {_PSYSLIB}
    {_SSYSLIB}
    {_SUSRLIB}
17. The following environment variables can be at most 44 characters in length:
    {_CLIB_PREFIX}
    {_PLIB_PREFIX}
    {_SLIB_PREFIX}
18. The following environment variables can be at most 63 characters in length:
    {_NEW_DATACLAS}
    {_NEW_DSNTYPE}
    {_NEW_MGMTCLAS}
    {_NEW_SPACE}
    {_NEW_STORCLAS}
    {_NEW_UNIT}
    {_WORK_DATACLAS}
    {_WORK_DSNTYPE}
    {_WORK_MGMTCLAS}
    {_WORK_SPACE}
    {_WORK_STORCLAS}
    {_WORK_UNIT}
19. The following environment variables are for specification of the SPACE parameter, and support only the syntax as shown with their default values (including all commas and parentheses). Also as shown with their default values, individual subparameters can be omitted, in which case the system defaults are used.
    {_NEW_SPACE}
    {_WORK_SPACE}
20. The following environment variables are for specification of the DSNTYPE parameter, and support only the subparameters LIBRARY or PDS (or null for no DSNTYPE):
    {_NEW_DSNTYPE}
    {_WORK_DSNTYPE}
21. The following environment variables can be at most 127 characters in length:
    {_DCBF2008}
    {_DCBU}
    {_DCB121M}
    {_DCB133M}
    {_DCB137}
    {_DCB137A}
    {_DCB3200}

{_DCB80}

These environment variables are for specification of DCB information, and support only the following DCB subparameters, with the noted restrictions:

RECFM
: Incorrect values are ignored.

LRECL
: None

BLKSIZE
: None

DSORG
: Incorrect values are treated as if no value had been specified.

22. The following environment variables are parsed as blank-delimited words, and therefore no embedded blanks or other white-space is allowed in the value specified. The maximum length of each word is 1024 characters:
    {_INCDIRS}
    {_INCLIBS}
    {_LIBDIRS}
    {_OPTIONS}
    {_OPERANDS}
23. An S-name is a *short* external symbol name, such as produced by the z/OS C/C++ compiler when compiling z/OS C programs with the `NOLONGNAME` option. An L-name is a *long* external symbol name, such as produced by the z/OS C/C++ compiler when compiling z/OS C programs with the `LONGNAME` option.
24. The C/C++ Runtime Library supports a file naming convention of // (the filename can begin with exactly two slashes). **c89/cc/c++** indicate that the file naming convention of // can be used.

    However, the Shell and Utilities feature *does not* support this convention. Do not use this convention (//) unless it is specifically indicated (as here in **c89/cc/c++**). The z/OS Shell and Utilities feature does support the POSIX file naming convention where the filename can be selected from the set of character values excluding the slash and the null character.
25. When coding in C and C++, **c89**, **cc**, and **c++**, by default, produce reentrant executables. For more information, see *z/OS C/C++ Programming Guide*. When coding in assembler language, the code must not violate reentrancy. If it does, the resulting executable may not be reentrant.
26. When shell variable **_MAKE_BI** is set to YES, **sh** will use the built-in **c89**, **cc**, **c++**, and **make** commands instead of **/bin/make, /bin/c89, /bin/cc**, and **/bin/c++**. **make** will also call the built-in **c89**, **cc**, and **c++** commands, instead of **/bin/c89, /bin/cc**, and **/bin/c++**. For more information, see *z/OS UNIX System Services Planning*.

Localization

c89/cc/c++ use the following localization environment variables:
- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

Exit Values

`0`	Successful completion.
`1`	Failure due to incorrect specification of the arguments.

2	Failure processing archive libraries: • Archive library was not in any of the library directories specified. • Archive library was incorrectly specified, or was not specified, following the **–l** operand.
3	Step of compilation, assemble, or link-editing phase was unsuccessful.
4	Dynamic allocation error, when preparing to call the compiler, assembler, IPA linker, prelinker, or link editor, for one of the following reasons: • The file or data set name specified is incorrect. • The file or data set name cannot be opened.
5	Dynamic allocation error, when preparing to call the compiler, assembler, prelinker, IPA linker, or link editor, due to an error being detected in the allocation information.
6	Error copying the file between a temporary data set and a hierarchical file system file (applies to the **–2** option, when processing assembler source files, and **–r** option processing).
7	Error creating a temporary control input data set for the link-editing phase.
8	Error creating a temporary system input data set for the compile or link-editing phase.

Portability

For **c89**, X/Open Portability Guide, POSIX.2 C-Language Development Utilities Option.

For **cc**, POSIX.2 C-Language Development Utilities Option, UNIX systems.

The following are extensions to the POSIX standard:
- The **—v**, **—V**, **—0**, **—1**, and **—2** options
- DLL support
- IPA optimization support
- The behavior of the **—o** option in combination with the **—c** option and a single source file.

Related Information

See the information on the following utilities in *Z/OS UNIX System Services Command Reference*: **ar**, **dbx**, **file**, **lex**, **make**, **nm**, **strings**, **strip**, **yacc**

Appendix G. z/OS C/C++ Compiler Return Codes and Messages

This appendix 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 55. 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 V1R1 was used.	Compilation terminated abnormally. Successful execution not possible.

The return code indicates the highest possible error severity that the compiler may be detect. 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 may have issued a Severe, Error, Warning, or Informational message. But it does not necessarily mean that all these error types are present in that particular compile.

Compiler Messages

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

nnnn error message number

text message which appears on the screen

&*n* compiler substitution variable

CBC1001 **″private″ assumed for base class ″&1″.**

Where: &1 is the name of the base class.

Explanation: No access specifier has been provided for a base class. A base class can be declared with the access specifier ″public″ or ″private″. The C++ language specification requires that ″private″ becomes the default when no access specifier is present. It is good coding practice to explicitly provide the access specifier for the base class.

User Response: Provide an access specifier or accept the default.

CBC1002 ″&1″ is not used in function ″&2″.

Where: &1 is a C++ symbol name &2 is a function name

Explanation: The specified symbol has been declared within a function but it has not been set or used. This is only an informational message because you can declare symbols that are not unused, but it is probably undesirable.

User Response: Ignore the message, use the symbol, or remove the symbol.

CBC1003 Ambiguous conversion between ″&1″ and ″&2″.

Where: &1 is a C++ type &2 is a C++ type

Explanation: The compiler was not able to find a single type common to the two specified types and was therefore unable to convert from one to the other.

User Response: Explicitly cast the type to an intermediate type and then convert to requested type.

CBC1004 ″&1″ statement is not allowed in this scope.

Explanation: The specified statement was found outside the valid scope for such a statement. This typically means that it is outside any function.

User Response: Place the statement in the correct scope or remove it.

CBC1005 Duplicate ″default″ statement in switch.

Explanation: Only one ″default″ label is allowed in a ″switch″ statement. This ″default″ label is not the first in the switch statement.

User Response: If you have nested switch statements, check that the braces match correctly. If they do not match, remove one of the ″default″ labels.

CBC1006 Duplicate definition of label ″&1″.

Where: &1 is a C++ label name

Explanation: The specified label has already been defined in the current function. A label can only be declared once within a function.

User Response: Remove or rename one of the label definitions.

CBC1008 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.

CBC1009 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 the correct file is being read. If the file resides on a LAN drive, ensure that the LAN is working correctly.

CBC1010 Source file name is missing.

Explanation: The name of the source file to be compiled was missing from the compiler invocation.

User Response: Ensure that you specify the source file name. Ensure the compiler options are specified correctly as well; the compiler may misinterpret the command line if the options are specified incorrectly.

CBC1011 ″&1″ is unmatched at end of file.

Where: &1 is a start-comment token or a left brace.

Explanation: The end of the source file was reached and the comment or block was not closed. It is also possible that there was a typographical error earlier in the source file.

User Response: Check the source file for typographical errors. End the comment or block before the end of the file.

CBC1012 A return value is not allowed for this function.

Explanation: A function with a return type of ″void″ cannot return a value.

User Response: Remove the value or expression from the return statement, remove the return statement, or change the return type of the function.

CBC1013 Identifier ″&1″ is undefined.

Where: &1 is a C++ name

Explanation: The specified identifier is used but has not been defined.

User Response: Define the identifier before using it. Check its spelling. If the identifier has been defined in a header file, check that any required macros have been defined.

CBC1014 Wrong number of arguments for macro ″&1″.

Explanation: The specified macro was defined with a different number of arguments than are used in this macro call.

User Response: Ensure that the macro call corresponds to the macro definition. Check the number and levels of corresponding braces in the macro.

CBC1015 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.

CBC1016 &1 member ″&2″ cannot be accessed.

Where: &1 is the keyword ″private″ or ″protected″ &2 is a class member name

Explanation: The specified member is private, protected, or is a member of a private base class and cannot be accessed from the current scope.

User Response: Check the access specification rules for the member function and change the access specifier if necessary. If the member function belongs to a base class, check the access specifier of the base class where the current class is defined.

CBC1017 Return value of type ″&1″ is expected.

Where: &1 is a C++ type

Explanation: No value is returned from the current function, but the function is expecting a non-void return value. The function was declared with a return type but the compiler did not detect a return statement. Only functions with a void return type may have no return statement or have a return statement with no return value.

User Response: Return a value from the function or change the functions's return type to void.

CBC1018 ″&1″ cannot be made a &2 member.

Where: &1 is a class member name &2 is the keyword ″public″, ″protected″ or ″private″

Explanation: An attempt is made to give private access to a base class member or to give an access that is different from the access the member was declared with. A derived class can only change the access of a base class member to public or protected if the access of that member was not private in the base class.

User Response: Remove the invalid access statement or change the access specifier in the base class.

CBC1019 Case expression is not an integral constant expression.

Explanation: The expression in a ″case″ statement must be an integral constant expression followed by a colon (:). A constant expression has a value that can be determined during compilation and does not change during execution.

User Response: Use an integral constant expression.

CBC1020 Inline assembly code is ignored.

Explanation: The compiler does not emit executable code from inlined assembly language instructions.

User Response: Ignore the message or replace the assembly code with C++.

CBC1021 Expected ″end of line″ and found ″&1″ in preprocessor directive.

Where: &1 is the unexpected token found by the compiler

Explanation: The compiler detected a preprocessor directive at the beginning of this line, then found an error in the directive. The rest of the line in the preprocessor directive is ignored.

User Response: Remove the unexpected token so that only the preprocessor directive appears on the line.

CBC1022 ″&1″ was previously declared as ″&2″.

Where: &1 is the name being declared &2 is the conflicting attribute in the previous declaration

Explanation: The declaration conflicts with a previous declaration of the same name.

User Response: Change one of the names or eliminate one of the declarations.

CBC1023 ″&1″ has already been defined.

Where: &1 is a C++ name

Explanation: An attempt is being made to define a name that has already been defined.

User Response: Change one of the names or remove one of the definitions. Check the spelling and the scope of the two variables.

CBC1024 Declaration of ″&1″ must be a function definition.

Where: &1 is a member function name

Explanation: A declaration of a member function outside its member list must be a function definition. Once you declare a member function inside a class declaration, you cannot redeclare it outside the class.

User Response: Either remove the member function declaration outside the member list or change it to a definition.

CBC1025 ″&1″ conflicts with ″&2″.

Where: &1 is keyword &2 is keyword

Explanation: These two attributes cannot both be specified in the same declaration.

User Response: Remove one of the specified attributes.

CBC1026 Keyword ″&1″ is not allowed.

Where: &1 is a keyword

Explanation: The specified keyword is not allowed in this context.

User Response: Remove the keyword.

CBC1027 Preprocessor directive ″#&1″ is not recognized.

Where: &1 is a preprocessor directive

Explanation: The compiler identified a # character at the start of a line and did not recognize the preprocessor directive following it.

User Response: Check the spelling of the preprocessor directive.

CBC1028 The syntax of the file name in the ″#include″ directive is not valid.

Explanation: The compiler detected an #include preprocessor directive but could not parse the file name. The file name in the #include directive must be within double quotation marks (″″) or angle brackets (<>).

User Response: Correct the syntax of the file name.

CBC1029 Expected integer line number and found ″&1″.

Where: &1 is a C++ token

Explanation: The operand of the ″#line″ directive must be an integer line number.

User Response: Ensure that the ″#line″ directive contains an integer line number operand.

CBC1030 The macro ″&1″ has already been defined.

Where: &1 is a macro name

Explanation: An active definition already exists for the macro name being defined. The last definition will be used.

User Response: Remove or rename one of the macro definitions.

CBC1032 Unexpected preprocessor directive ″#&1″.

Where: &1 is a preprocessor directive

Explanation: An ″#else″, ″#elif″ or ″#endif″ preprocessor directive was found out of context.

User Response: Remove or move the preprocessor directive. Check nesting of #if, #else, #elif, and #endif.

CBC1033 The for-init-statement must be a declaration or expression.

Explanation: The initializer statement within a ″for″ statement must be a declaration or expression.

User Response: Change the for-init-statement to a declaration or an expression.

CBC1034 ″&1″ has a function body but is not a function.

Where: &1 is a C++ name

Explanation: The name is not declared as a function; there may be parentheses missing after the function name.

User Response: Correct the declaration.

CBC1035 The array boundary in ″&1″ is missing.

Where: &1 is a C++ type

Explanation: An array must be defined with at least one element. Use a pointer if you want to dynamically allocate memory for the array.

User Response: Add an array bound.

CBC1036 The bit-field length must be an integral constant expression.

Explanation: The bit-field length, which is the value to the right of the colon, must be an integer. A constant expression has a value that can be determined during compilation and does not change during execution.

User Response: Change the bit-field length to an integral constant expression.

CBC1037 ″&1″ is not a base class of ″&2″.

Where: &1 is a class name &2 is a class name

Explanation: A derived class attempted to access elements of a class it did not inherit from. A derived class can only access elements of its base class or base classes.

User Response: Ensure the class names are correct and the classes are derived properly.

CBC1038 The array bound must be a positive integral constant expression.

Explanation: The compiler detected an array declaration that did not have a constant that is greater than 0 for the array bounds. Use pointers if you want to dynamically allocate storage for arrays.

User Response: Change the array bound to an integral constant expression or change it to a pointer. A constant expression has a value that can be determined during compilation and does not change during execution.

CBC1039 ″&1″ has the same name as its containing class.

Where: &1 is a C++ name

Explanation: The compiler has detected conflicting names for objects within a class declaration. Nested class declarations must have different names.

User Response: Change the name of the conflicting class.

CBC1040 A destructor can only be used in a function declaration or in a function call.

Explanation: The compiler has detected an incorrect destructor call.

User Response: Check the call to the destructor to ensure no braces are missing. If the braces are correct, remove the destructor call.

CBC1041 An initializer is not allowed for ″&1″.

Where: &1 is a C++ name or keyword

Explanation: The compiler detected an initializer where one is not permitted. For example, a class member declarator cannot contain an initializer.

User Response: Remove the initializer.

CBC1042 Function ″&1″ is nested within another function.

Where: &1 is a function name

Explanation: You cannot nest functions in C++.

User Response: Ensure that a ″}″ is not missing before the start of the function. Remove the nested function.

CBC1043 The string must be terminated before the end of the line.

Explanation: The compiler detected a string that was not terminated before an end-of-line character was found.

User Response: End the string before the end of the line, or use ″\″ to continue the string on the next line. The ″\″ must be the last character on the line.

CBC1044 extern ″&1″ is not a recognized linkage; extern ″C″ is assumed.

Where: &1 is a string

Explanation: The linkage string in a linkage declaration is not one of the linkages supported by this compiler.

User Response: Change the linkage string to a valid value.

CBC1045 Preprocessor error - expected ″&1″ and found ″&2″.

Where: &1 is a C++ token &2 is a C++ token

Explanation: A syntax error was found during preprocessing. The message identifies what the compiler expected and what it actually found.

User Response: Correct the syntax.

CBC1047 An expression of type ″&1″ cannot be followed by the function call operator ().

Explanation: The compiler detected an expression followed by the function call operator. The expression must be of type function, pointer to function, or reference to function.

User Response: Change the type of expression or remove the function call operator.

CBC1048 The ″this″ keyword is only valid in class scope.

Explanation: An attempt to use the C++ keyword ″this″ was detected outside class scope. The keyword ″this″ cannot be used outside a class member function body.

User Response: Remove or move the ″this″ keyword.

CBC1049 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.

CBC1050 A destructor cannot have arguments.

User Response: Remove the arguments from the destructor.

CBC1051 A declaration has been made without a type specification.

Explanation: The compiler detected a typedef specification that did not have a type associated with it.

User Response: Add a type specification to the declaration.

CBC1052 Return type cannot be specified for ″&1″.

Where: &1 is a function name

Explanation: The compiler detected a return type where one is not permitted. For example, putting a return type on a constructor is not permitted.

User Response: Remove the return type specification for the function.

CBC1053 Class qualification for ″&1″ is not allowed.

Where: &1 is a C++ name

Explanation: Explicit class qualification is not allowed in this context.

User Response: Remove the class qualification.

CBC1054 The ″&1″ operator is not allowed between ″&2″ and ″&3″.

Where: &1 is a C++ operator &2 is a C++ type &3 is a C++ type

Explanation: The compiler detected an illegal operator between two operands. For user-defined types, you must overload the operator to accept the user-defined types.

User Response: Change the operator or change the operands.

CBC1055 ″&1″ cannot be converted to ″&2″.

Where: &1 is a C++ type &2 is a C++ type

Explanation: The type conversion cannot be performed because there is no conversion between the types. This can occur in an initialization, assignment, or expression statement.

User Response: Change one of the types or overload the operator.

CBC1056 Operand for ″&1″ must be a pointer or an array.

Where: &1 is a C++ operator

Explanation: The specified operator must have an operand which is a pointer or an array.

User Response: Change the operand to either a pointer or an array.

CBC1057 Syntax error - ″&1″ is not a class name.

Where: &1 is a C++ name

Explanation: A class name must be specified in this context.

User Response: Specify a class name. Check the spelling.

CBC1058 Operand of ″&1″ operator must be an lvalue.

Where: &1 is a C++ operator

Explanation: The compiler detected an operand that is not an lvalue. An lvalue is an expression that represents an object. For example, the left hand side of an assignment statement must be an lvalue.

User Response: Change the operand to an lvalue.

CBC1059 const expression cannot be modified.

Explanation: You can initialize a const object, but its value cannot change afterwards.

User Response: Eliminate the const type qualifier from the expression or do not use it with the increment/decrement operators.

CBC1060 An expression of type ″&1″ is not allowed on the left side of ″&2&3″.

Where: &1 is a C++ type &2 is a C++ operator &3 is a C++ name

Explanation: The compiler detected a mismatch between the operands of an operator.

User Response: Change the operand type or use a different operator.

CBC1061 ″&1″ is neither an immediate base class nor a non-static data member of class ″&2″.

Where: &1 is a C++ name

Explanation: The compiler has detected an element of the initializer list that is not an element of the member list. In the constructor initializer list, you can only initialize immediate base classes and data members not inherited from a base class.

User Response: Change the constructor initializer list.

CBC1062 Constructor initializer list is not allowed for non-constructor function.

Explanation: An attempt is being made to give a constructor initializer list to a non-constructor function. A constructor initializer list is only allowed for a constructor function.

User Response: Remove the constructor initializer list.

CBC1063 Variable ″&1″ is not allowed in an argument initializer.

Where: &1 is a C++ name

Explanation: The compiler has detected a default argument initialized by a parameter.

User Response: Remove the parameter from the default argument initialization.

CBC1064 There are too many initializers in the initializer list.

Explanation: The compiler detected more initializers than were present in the function declaration.

User Response: Remove one or more initializers from the initializer list. Make sure the number of initializers in the initializer list corresponds to the number of arguments in the function declaration.

CBC1065 An initializer is not allowed for an array allocated by ″new″.

User Response: Remove the initializer or remove the ″new″ allocation.

CBC1066 The bit-field length must not be more than &1.

Where: &1 is a number

Explanation: The bit-field length must not exceed the maximum bit size of the bit-field type.

User Response: Reduce the bit-field length.

CBC1067 The type of ″&1″ cannot be ″&2″.

Where: &1 is a C++ construct &2 is a C++ type

Explanation: The compiler detected a conflict in a type declaration.

User Response: Change the type.

CBC1068 Function overloading conflict between ″&1″ and ″&2″.

Where: &1 is a function type &2 is a function type

Explanation: The compiler detected function argument types that did not match.

User Response: Change the argument declarations of the functions.

CBC1069 Declarations of the same &1 must not specify default initializers for the same argument.

Where: &1 is the word ″function″ or the keyword ″template″

Explanation: The compiler has detected a duplicate default initializer value for the same argument in both overloaded functions or in both templates.

User Response: Ensure that you wanted to declare the same function or template. If that is the case, remove one of the default initializers. Otherwise, remove one of the declarations or overload the function.

CBC1070 Call does not match any argument list for ″&1″.

Where: &1 is a function name

Explanation: No variant of the overloaded function matches the argument list. The argument mismatch could be by type or number of arguments.

User Response: Change the argument list on the call to the overloaded function or change the argument list on one of the overloaded function variants so that a match is found.

CBC1071 Call to ″&1″ matches more than one function.

Where: &1 is a function name

Explanation: More than one variant of the overloaded function matches equally well with the argument list specified on the call.

User Response: Change the argument list on the call to the overloaded function or change the argument list on one of the overloaded function variants so that only one match is found.

CBC1072 Linkage for ″&1″ cannot be redefined.

Where: &1 is a function name

Explanation: The specified name has already been declared with a different linkage than the current declaration.

User Response: Remove the redefinition or change one of the names.

CBC1073 The ″operator″ declaration must declare a function.

Explanation: The keyword ″operator″ can only be used to declare an operator function.

User Response: Check the declaration of the operator and make sure the function declarator () appears after it. Use the ″operator″ keyword to declare an operator function or remove it.

CBC1074 Operand for ″&1″ is of type ″&2″ which is not of type pointer to member.

Where: &2 is a C++ type

Explanation: The specified operator must have an operand which is of type pointer to member.

User Response: Change the operand to type pointer to member.

CBC1075 ″&1″ is not allowed as a function return type.

Where: &1 is a C++ type

Explanation: You cannot declare a function with a function or an array as its return type.

User Response: Declare the function to return a pointer to the function or the array element type.

CBC1076 ″&1″ is not allowed as an array element type.

Where: &1 is a C++ type

Explanation: The C++ language does not allow the declaration of an array of functions or references, or an array of type void.

User Response: Remove the declaration or change the declaration so that it is an array of pointer to functions, pointers to references, or pointers to void.

CBC1077 const variable ″&1″ does not have an initializer.

Where: &1 is a variable name

Explanation: You can only assign a value to a const variable using an initializer. This variable has no initializer, so it can never be given a value.

User Response: Initialize the variable or remove the ″const″ keyword.

CBC1078 Non-static member ″&1″ must be associated with an object or a pointer to an object.

Where: &1 is a class member name

Explanation: The compiler detected a non-static member making a reference to an object that has not been instantiated. You can reference only static members without associating them with an instance of the containing class.

User Response: Check the spelling and the class definition. Change the name of the class or function, or define the function as static in that class.

CBC1079 ″&1″ is not a member of ″&2″.

Where: &1 is a C++ name &2 is a class name

Explanation: The class is used explicitly as the scope qualifier of the member name, but the class does not contain a member of that name.

User Response: Check the spelling of the scope qualifier. Change the scope qualifier to the class containing that member, or remove it.

CBC1080 Wrong number of arguments for ″&1″.

Where: &1 is a function or type name

Explanation: A function or an explicit cast has been specified with the wrong number of arguments.

User Response: Use the correct number of arguments. Ensure that overloaded functions have the correct number and type of arguments.

CBC1081 ″&1″ must be a class member.

Where: &1 is a C++ name

Explanation: Conversion functions and certain operator functions must be class members. They cannot be defined globally.

User Response: Remove the global definition or make the function a class member.

CBC1082 An argument type of ″&1″ is not allowed for ″&2″.

Where: &1 is a C++ type &2 is a function name

Explanation: The function being declared has restrictions on what types its arguments can have. The specified type is not allowed for this argument.

User Response: Change the argument type.

CBC1083 ″&2″ cannot have a return type of ″&1″.

Where: &1 is a C++ type &2 is an operator function

Explanation: The specified operator function has the wrong return type.

User Response: Change the return type.

CBC1084 The array operator must have one operand of pointer type and one of integral type.

Explanation: This error may result from the incorrect use of the array operator.

User Response: Change the operands of the array operator.

CBC1085 Wrong number of arguments specified in the function call.

Explanation: The number of arguments in the function call does not match the number of arguments in the function declaration.

User Response: Ensure the function declaration and function call specify the same number of arguments.

CBC1086 &1

Where: &1 is an error message

Explanation: This message has been generated by the ″#error″ preprocessor directive, which is a user-defined error message placed in the source code.

CBC1087 ″&1″ operator is not allowed for type ″&2″.

Where: &1 is a C++ operator &2 is a C++ type

Explanation: The specified operator cannot be used with operands of this type.

User Response: Change either the operator or the operands.

CBC1088 Insufficient memory.

Explanation: The compiler ran out of memory during compilation.

User Response: Increase your storage and recompile.

CBC1089 More than one function ″&1″ has non-C++ linkage.

Where: &1 is a function name

Explanation: If a function is overloaded, at most one of its variants can have non-C++ linkage.

User Response: Remove one of the non-C++ linkages or do not overload the function.

CBC1090 Syntax error - expected ″&1″ and found ″&2″.

Where: &1 is a C++ token &2 is a C++ token

Explanation: A syntax error was found while parsing the program. The message identifies what the compiler expected and what it actually found. Often the source of the error is an unmatched parenthesis or a missing semicolon.

User Response: Correct the syntax.

CBC1091 ″&1″ is not allowed for &2.

Where: &1 is a keyword &2 is a C++ construct

Explanation: The attribute or name cannot be specified in the given context. The compiler detected incompatible names that conflict with the language definition.

User Response: Remove the attribute or name.

CBC1092 ″&1″ conflicts with previous ″&2″ declaration.

Where: &1 is a keyword &2 is a keyword

Explanation: The declaration conflicts with a previous declaration of the same symbol.

User Response: Remove one of the declarations or make them identical.

CBC1093 Initializer is too long.

Explanation: The string initializer for a character or wide-character array has more characters than the array. Note that the trailing null character is treated as part of the initializer.

User Response: Increase the size of the array or reduce the size of the initializer.

CBC1094 The ″operator->″ function must return a class type that contains an ″operator->″ function.

Explanation: The ″operator->″ function must return either a class type, a reference to a class type, or a pointer to class type, and the class type must itself have an ″operator->″ function.

User Response: Change the return value of the ″operator->″ function.

CBC1095 Unused ″&1″ definition.

Where: &1 is the keyword struct or class

Explanation: An unnamed class or struct definition was found that has no object associated with it. The definition can never be referenced. A class can be unnamed, but it cannot be passed as an argument or returned as a value. An unnamed class cannot have any constructors or destructors.

User Response: Create an object for the class or struct, or remove the definition.

CBC1096 Internal compiler error at line &1 in module ″&2″: &3.

Explanation: The compiler detected an error within itself from which it cannot recover. The error was found within the compiler itself.

User Response: Note the line and module references in this message. Contact your IBM Representative or C SET ++ support.

CBC1097 Reference to member ″&1″ of undefined class ″&2″.

Where: &1 is a member name &2 is a class name

Explanation: The member has been explicitly given the specified class as a scope qualifier but the class (and hence the member) has not been defined.

User Response: Check for a missing #include file. Define the class and member.

CBC1098 Pointer conversion may be wrong if the classes are related in a multiple inheritance hierarchy.

Explanation: The relationship between the classes in a pointer conversion is not known. If the target class is later defined as a base class of the source class in a multiple inheritance, this conversion will be wrong if the value of the pointer should have been modified by the conversion.

User Response: Change the ambiguous reference in the conversion.

CBC1099 ″&1″ is used but not set in function ″&2″.

Where: &1 is a variable name &2 is a function name

Explanation: The specified symbol is being used but has not been assigned a valid value. Its value will be undefined.

User Response: Define or initialize the symbol before using it.

CBC1100 ″&1″ is set but not used in function ″&2″.

Where: &1 is a variable name &2 is a function name

Explanation: The specified symbol was given a value but was never used.

User Response: Use the symbol or remove it.

CBC1101 ″&1″ is used before it is set.

Where: &1 is a variable name

Explanation: The specified symbol is being used before it has been assigned a value. The value of the symbol is undefined.

User Response: Define or initialize the symbol before using it.

CBC1102 The reference variable ″&1″ is uninitialized.

Where: &1 is a variable name

Explanation: Reference variables must be initialized.

User Response: Initialize the reference variable or remove it.

CBC1103 ″&1″ must already be declared.

Where: &1 is a class or enum name

Explanation: The specified class or enum name must have been declared before this use of the name.

User Response: Declare the class or enum name before you use it. Check the correct spelling of the name.

CBC1104 Unrecognized source character ″&1″, code point &2.

Where: &1 is a character &2 is an integer

Explanation: The specified character is not a valid character in a C++ program. The code point displayed represents its hexadecimal value.

User Response: Remove the character.

CBC1105 A local class cannot have a non-inline member function ″&1″.

Where: &1 is a function name

Explanation: A class declared within a function must have all of its member functions defined inline, because the class will be out of scope before non-inline functions can be defined.

User Response: Define the functions inline, or move the class definition out of the scope of the function.

CBC1106 The size of ″&1″ is unknown in ″&2″ expression.

Where: &1 is a C++ type

Explanation: The operation cannot be performed because the size of the specified type is not known.

User Response: Ensure the size of the type is known before this expression.

CBC1107 Assignment in logical expression.

Explanation: The logical expression contains an assignment (=). An equality comparison (==) may have been intended.

User Response: Change the operator or the expression.

CBC1108 Conversion from ″&1″ to ″&2″ may cause truncation.

Where: &1 is a C++ type &2 is a C++ type

Explanation: The specified conversion from a wider to a narrower type may cause the loss of significant data.

User Response: Remove the conversion from a wider to a narrower type.

CBC1109 ″goto &1″ bypasses initialization of ″&2″.

Where: &1 is the C++ label used with the goto keyword &2 is the variable being initialized

Explanation: Jumping past a declaration with an explicit or implicit initializer is not valid unless the declaration is in an inner block or unless the jump is from a point where the variable has already been initialized.

User Response: Enclose the initialization in a block statement.

CBC1110 References to ″&1″ may be ambiguous. The name is declared in base classes ″&2″ and ″&3″.

Where: &3 is a C++ class name

Explanation: The compiler detected the base classes of a derived class have members with the same names. This will cause ambiguity when the member name is used. This is only an informational message because the declaration of a member with an ambiguous name in a derived class is not an error. The ambiguity is only flagged as an error if you use the ambiguous member name.

User Response: Change one of the names, or always fully qualify the name.

CBC1111 Ambiguous reference to ″&1″, declared in base classes ″&2″ and ″&3″.

Where: &3 is a C++ class name

Explanation: The derived class made a reference to a member that is declared in more than one of its base classes and the compiler cannot determine which base class member to choose.

User Response: Change one of the names, or always fully qualify the name.

CBC1112 Conversion from ″&1″ to ″&2″ is ambiguous.

Where: &1 is a C++ type &2 is a C++ type

Explanation: There is more than one way to perform the specified conversion. This ambiguity may be caused by an overloaded function.

User Response: Change or remove the conversion.

CBC1113 ″&1″ is only valid for non-static member functions.

Where: &1 is the keyword const or volatile

Explanation: const and volatile are only significant for non-static member functions, since they are applied to the ″this″ pointer.

User Response: Remove const and volatile from all static members.

CBC1114 Duplicate case value.

Explanation: Case values must be unique within each ″switch″ statement.

User Response: Change or remove one of the duplicate case values. Check the braces if you have nested case statements.

CBC1115 Character literal is null.

Explanation: An empty character literal has been specified. A string literal may have been intended.

User Response: Remove the character literal, change it to a string literal, or give it a value.

CBC1116 ″&1″ is given wider scope for compatibility reasons.

Explanation: A type defined in class scope has been given the scope of the enclosing function or file because of a compiler option.

User Response: Ensure this is correct scope.

CBC1117 ″&1″ has more than one base class ″&2″.

Where: &1 is a class name &2 is a class name

Explanation: A derived class has inherited the same base class in more than one path and the compiler cannot determine which one to choose.

User Response: Remove one of the inheritances.

CBC1118 ″&1″ is a &2 base class of ″&3″.

Where: &1 is a class name &2 is the keyword ″private″ or ″protected″ &3 is a class name

Explanation: An attempt is being made to convert a pointer to a derived class into a pointer to a private or protected base class.

User Response: Remove the pointer conversion.

CBC1119 The statement is unreachable.

Explanation: The flow of control in the program never allows the statement to be be reached.

User Response: Ensure that the statement is accessible to the flow of control, or remove the statement.

CBC1120 &1 ″&2″ is not allowed in a union.

Where: &1 is a C++ construct &2 is a C++ name

Explanation: Unions must not be declared with base classes, virtual functions, static data members, members with constructors, members with destructors, or members with class copying assignment operators.

User Response: Remove any such members from the union declaration.

CBC1121 union ″&1″ cannot be used as a base class.

Where: &1 is a union name

Explanation: Unions cannot be used as base classes for other class declarations.

User Response: Remove the union as a base class for other class declarations.

CBC1122 Local variable ″&1″ is inaccessible from ″&2″.

Where: &1 is a variable name &2 is a class name

Explanation: An automatic variable within a function is not accessible from local classes declared within the function.

User Response: Remove the reference to the local variable, or move the variable to a different scope.

CBC1123 Value of enumerator ″&1″ is too large.

Where: &1 is an enumerator name

Explanation: The value of an enumerator must be a constant expression that is promotable to a signed integer value.

User Response: Reduce the value of the enumerator.

CBC1124 No path specified for -I option.

Explanation: The option requires a path name to search but no path was found.

User Response: Supply the name of a directory containing include files after the option.

CBC1125 Missing macro name after -D or -U command line option.

Explanation: The option requires the name of a macro to be defined or undefined, and no name was found.

User Response: Add a macro name after the option.

CBC1126 Argument ″&1″ is not used in function ″&2″.

Where: &1 is an argument name &2 is a function name

Explanation: The argument has been declared in a function but has not been set or used.

User Response: Use the argument or remove it.

CBC1127 Global symbol ″&1″ is not used.

Where: &1 is a C++ name

Explanation: The specified symbol has been declared as a global symbol but has not been set or used.

User Response: Use the symbol or remove it.

CBC1129 Default initializers are not allowed in local friend functions.

Explanation: You cannot use default arguments in the friend functions of the local class.

User Response: Remove the default initializers from the local friend function.

CBC1130 A constant is being used as a conditional expression.

Explanation: The condition to an if, for, or switch is constant and therefore, that condition will always hold.

User Response: No response is necessary.

CBC1131 The argument to a not (!) operator is constant.

Explanation: The compiler has detected a constant after the ! operator which may be a coding error.

User Response: Remove the constant or ignore this message.

CBC1132 There is more than one character in a character constant.

Explanation: Using more than one character in a character constant (for example, 'ab') may not be portable across machines.

User Response: Remove the extra character(s) or change the character constant to a string constant.

CBC1133 Possible pointer alignment problem with the ″&1″ operator.

Where: &1 is a C++ operator

Explanation: A pointer that points to a type with less strict alignment requirements is being assigned, cast, returned or passed as a parameter to a pointer that is a more strictly aligned type. This is a potential portability problem.

User Response: Remove the pointer reference or change the alignment.

CBC1134 A constant expression is being cast to a pointer.

Explanation: Casting a constant value to a pointer is not portable to other platforms.

User Response: Remove the constant expression from the cast expression.

CBC1135 Precision will be lost in assignment to (possibly sign-extended) bit-field ″&1″.

Explanation: A constant is being assigned to a signed bit field that cannot represent the constant. Precision may be lost and the stored value will be incorrect.

User Response: Increase the size of the bit field.

CBC1136 Precision will be lost in assignment to bit-field ″&1″.

Explanation: A constant is being assigned to a bit field, and because the bit field has a smaller size, the precision will be lost.

User Response: Change the assignment expression.

CBC1137 Enumeration type clash with the ″&1″ operator.

Where: &1 is a C++ operator

Explanation: Operands from two different enumerations are used in an operation.

User Response: Ensure both operands are from the same enumeration.

CBC1138 Comparison of an unsigned value with a negative constant.

Explanation: An unsigned value is being compared to a negative number. The unsigned value will always compare greater than the negative number. This may be a programming error.

User Response: Remove the comparison or change the type.

CBC1139 Unsigned comparison is always true or always false.

Explanation: The comparison is either ″unsigned >= 0″, which is always true, or ″unsigned < 0″, which is always false.

User Response: Remove or change the comparison.

CBC1140 Comparison is equivalent to ″unsigned value &1 0″.

Explanation: The comparison is either ″unsigned > 0″ or ″unsigned <= 0″, and could be written as ″unsigned != 0″ or ″unsigned == 0″.

User Response: Change the comparison.

CBC1141 Argument &1 for ″&2″ must be of type ″&3″.

Where: &1 is an argument number &2 is a function name &3 is a C++ type

Explanation: The indicated function requires an argument of a particular type. However, the argument specified is of a different type than the type required.

User Response: Ensure that the argument is of the correct type.

CBC1142 The operand for the ″#line″ directive must be an integer in the range 1 to 32767.

Explanation: The operand of the ″#line″ directive must be an integer in the specified range.

User Response: Ensure that the operand is in the specified range.

CBC1143 Definition of ″&1″ is not allowed.

Where: &1 is the keyword class, struct, union or enum.

Explanation: You cannot define a type in a type cast or a conversion function declaration.

User Response: Move the definition to a new location, or remove it.

CBC1144 Reference to ″&1″ is not allowed.

Where: &1 is a C++ name

Explanation: The name has a special meaning in a C++ program and cannot be referenced in this way.

User Response: Remove the reference.

CBC1145 Escape sequence &1 is out of the range 0-&2. Value is truncated.

Where: &2 is the maximum allowed value of the escape sequence

User Response: Make the escape sequence small enough to fit the specified range.

CBC1146 A wide character constant is larger than the size of a ″wchar_t″. Only the last character is used.

Explanation: A wide character constant can only contain one character. This error may be caused by a literal containing a multibyte character if the multibyte character compile option is not used.

User Response: Make the wide character constant smaller.

CBC1147 A character constant is larger than the size of an ″int″. Only the rightmost &1 characters are used.

Where: &1 is an integer number

User Response: Make the character constant smaller.

CBC1148 Linkage specification must be at file scope.

Explanation: A linkage specification may only be defined at file scope, that is, outside all functions and classes.

User Response: Move the linkage specification or remove it.

CBC1149 Default initializers cannot be followed by uninitialized arguments.

Explanation: If a default initializer is specified in an argument list, all following arguments must also have default initializers.

User Response: Remove the default initializers, or provide them for the following arguments, or move the arguments to the end of the list.

CBC1150 Cannot take the address of ″&1″.

Where: &1 is a C++ name

Explanation: You cannot take the address of a constructor, a destructor or a reference member.

User Response: Remove the address operator (&) from the expression or remove the expression.

CBC1151 &1 compiler temporary of type ″&2″ has been generated.

Where: &1 is a storage class &2 is a C++ type

Explanation: The compiler has generated a temporary variable. This variable will be destroyed automatically when it goes out of scope. This messages is generated for your information only, it does not necessarily indicate a problem with your program.

User Response: Ensure that your program does not attempt to reference the temporary variable outside of its scope.

CBC1152 An error was detected 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 the file name is correct.

CBC1153 Duplicate qualifier ″&1″ ignored.

Where: &1 is a keyword

Explanation: The keyword has been specified more than once. Extra occurrences are ignored.

User Response: Remove one of the duplicate qualifiers.

CBC1154 ″&1″ operator cannot be overloaded.

Where: &1 is an operator name

Explanation: The specified operator cannot be overloaded using an operator function. The following operators cannot be overloaded: . .* :: ?:

User Response: Remove the overloading declaration or definition.

CBC1155 At least one argument of ″&1″ must be of class or enum type.

Where: &1 is an operator function name

Explanation: The non-member operator function must have at least one argument which is of class or enum type.

User Response: Add an argument of class or enum type.

CBC1156 Call matches built-in operator.

Explanation: The compiler detected an operator that is similar to the built-in one, and is providing additional information.

User Response: Ensure this is the desired match.

CBC1157 The divisor for the modulus or division operator cannot be zero.

User Response: Change the expression used in the divisor.

CBC1158 The address of the bit-field ″&1″ cannot be taken.

Where: &1 is a member name

Explanation: An expression attempts to take the address of a bit-field, or to use the bit-field to initialize a reference variable or argument.

User Response: Remove the expression causing the error.

CBC1159 ″&1″ must not have default initializers.

Where: &1 is an operator function name or ″template function″

Explanation: Default initializers are not allowed within the declaration of an operator function or a template function.

User Response: Remove the default initializers.

CBC1160 The &1 ″&2″ cannot be initialized because it does not have a default constructor.

Where: &1 is ’base class’ or ’class member’ &2 is a C++ name

Explanation: The specified base class or member cannot be constructed since it is not initialized in the constructor initializer list and its class has no default constructor.

User Response: Specify a default constructor for the class or initialize it in the constructor initializer list.

CBC1163 Template class ″&1″ has the wrong number of arguments.

Where: &1 is a template class name

Explanation: A template class instantiation has a different number of template arguments than the template declaration.

User Response: Ensure that the template class has the same number of declarations as the template declaration.

CBC1164 Non-&1 member function ″&2″ cannot be called for a &1 object.

Where: &2 is a function name with arguments

Explanation: The member function is being called for a const or volatile object but the member function has not been declared with the const or volatile qualifier.

User Response: Supply a version of the member function with the correct set of ″const″ and ″volatile″ qualifiers.

CBC1165 Null statement.

Explanation: Possible extraneous semi-colon has been specified.

User Response: Check for extra semi-colons in statement.

CBC1166 Bit-field ″&1″ cannot be used in a conditional expression that is to be modified.

Explanation: The bit-field is part of a conditional expression that is to be modified. Only objects that can have their address taken are allowed as part of such an expression, and you cannot take the address of a bit field.

User Response: Remove the bit-field from the conditional expression.

CBC1167 The ″&1″ qualifier cannot be applied to ″&2″.

Where: &2 is a name or a type

Explanation: The qualifier is being applied to a name or a type for which it is not valid.

User Response: Remove the qualifier.

CBC1168 Local type ″&1″ cannot be used as a &2 argument.

Where: &2 is either the keyword template or the keyword function

Explanation: The type cannot be used as a function argument or in the instantiation of a template because the scope of the type is limited to the current function.

User Response: Remove the local type.

CBC1169 Exception specification for function ″&1″ does not match previous declaration.

Where: &1 is a function name

Explanation: If an exception specification is given in more than one declaration of a function, it must be the same in all such declarations.

User Response: Ensure that all exception specifications match.

CBC1170 Default initializers for non-type template arguments are only allowed for class templates.

Explanation: Default initializers have been given for non-type template arguments, but the template is not declaring a class.

User Response: Remove the default initializers.

CBC1171 A function argument must not have type ″void″.

Explanation: A function argument may be an expression of any object type. However, ″void″ is not the type of any object, and cannot be used as an argument type.

User Response: Change the type of the function argument.

CBC1172 Insufficient memory in line &1 of file ″&2″.

Where: &1 is a line number &2 is a file name

Explanation: The compiler ran out of memory during compilation.

User Response: Increase your storage and recompile.

CBC1173 Unable to initialize source conversion from codepage &1 to codepage &2.

Where: &1 is a codepage name i.e. IBM-1047 &2 is a codepage name i.e. IBM-1047

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.

CBC1174 An object of abstract class ″&1″ cannot be created.

Where: &1 is a class name

Explanation: You cannot create instances of abstract classes. An abstract class is a class that has or inherits at least one pure virtual function.

User Response: Derive another object from the abstract class.

CBC1175 Invalid use of an abstract class.

Explanation: An abstract class must not be used as an argument type, as a function return type, or as the type of an explicit conversion.

User Response: Derive another class from the abstract, instantiate it so it becomes a concrete object, and then use it instead.

CBC1176 ″&1″ has been used more than once in the same base class list.

Where: &1 is base class name

Explanation: A base class may only be specified once in the base class list for a derived class.

User Response: Remove one of the specifications.

CBC1177 Template argument &1 of type ″&2″ does not match declared type ″&3″.

Where: &1 is an integer number &2 is a C++ type &3 is a C++ type

Explanation: A non-type template argument must have a type that exactly matches the type of the corresponding argument in the template declaration.

User Response: Ensure that the types match.

CBC1178 Template argument &1 of type ″&2″ is not an allowable constant value or address.

Where: &1 is an integer number &2 is a C++ type

Explanation: A non-type template argument must be a constant value or the address of an object, function, or static data member that has external linkage. String literals cannot be used as template arguments because they have no name, and therefore no linkage.

User Response: Change the template argument.

CBC1179 Template argument list is empty.

Explanation: At least one template argument must be specified in a template declaration.

User Response: Specify a template argument in the declaration.

CBC1180 Formal template argument &1 is of type ″&2″ which is not an allowable integral, enumeration, or pointer type.

Where: &1 is an integer number &2 is a C++ type

Explanation: A non-type template argument must be of integral, or enumeration, or pointer type, so that it can be matched with a constant integral value.

User Response: Change the template argument.

CBC1181 ″&1″ is defined in a template declaration but it is not a static member.

Where: &1 is a C++ name

Explanation: A member of a template class defined in a template declaration must be a static member.

User Response: Make the member static or remove it from the template declaration.

CBC1182 Template argument ″&1″ is not used in the declaration of the name or the argument list of ″&2″.

Where: &1 is a template argument name &2 is a C++ name

Explanation: All template arguments for a non-class template must be used in the declaration of the name or the function argument list.

User Response: Ensure all template arguments are used in the declaration of the name or the function argument list.

CBC1183 Template declaration does not declare a class, a function, or a template class member.

Explanation: Following the template argument, a template declaration must declare a class, a function, or a static data member of a template class.

User Response: Change the template declaration to declare a class, a function, or a template class member.

CBC1184 Return type ″&1″ for function ″&2″ differs from previous return type of ″&3″.

Where: &1 is a C++ type &2 is a function name &3 is a C++ type

Explanation: The declaration of the function differs from a previous declaration in only the return type.

User Response: Change the return type so that it matches the previous return type.

CBC1185 ″&1″ is a member of ″&2″ and cannot be used without qualification.

Where: &2 is a possibly qualified class name

Explanation: The specified name is a class member, but no class qualification has been used to reference it.

User Response: Add a class qualification to the class member.

CBC1186 The expression is not a valid preprocessor constant expression.

Explanation: The expression in an ″#if″ or ″#elif″ preprocessor directive is either not a valid expression or not a constant expression. No keywords are recognized in such an expression and non-macro identifiers are replaced by the constant 0.

User Response: Change the expression for the preprocessor directive.

CBC1187 ″&1″ cannot be initialized multiple times.

Where: &1 is a member or base class name

Explanation: An initializer was already specified in the constructor definition.

User Response: Remove the additional initializer.

CBC1188 A macro parameter is expected after the ″#″ operator.

Explanation: The ″#″ operator in a macro replacement list must be followed by a macro parameter.

User Response: Add a macro parameter after the ″#″ operator.

CBC1189 ″##″ operator is at the start or end of the replacement list.

Explanation: The ″##″ operator must be preceded and followed by valid tokens in the macro replacement list.

User Response: Move the ″##″ operator in the replacement list.

CBC1190 One or more ″#endif″ statements are missing at end of file.

Explanation: The end of file has been reached and there are still ″#if″, ″#ifdef″ or ″#ifndef″ statements without a matching ″#endif″ statement.

User Response: Ensure that all ″#if″, ″#ifdef″, and ″#ifndef″ statements have matching ″#endif″ statements.

CBC1191 No suitable copy assignment operator exists to perform the assignment.

Explanation: A copy assignment operator exists but it does not accept the type of the given parameter.

User Response: Change the copy assignment operator.

CBC1192 Identifier ″&1″ in preprocessor expression is assigned 0.

Where: &1 is an identifier name

Explanation: Identifiers are not recognized in a preprocessor expression. The specified identifier has been treated as a non-macro identifier and assigned the constant 0.

CBC1193 Explicit call to constructor ″&1″ is not allowed.

Where: &1 is a constructor name

Explanation: You cannot call a constructor explicitly. It is called implicitly when an object of the class is created.

User Response: Remove the call to the constructor.

CBC1194 ″catch(&1)″ will never be reached because of previous ″catch(&2)″.

Where: &1 is a C++ type or the token '...' &2 is a C++ type or the token '...'

Explanation: The catch clause can never be reached since any exception type that matches it will also be matched by the specified previous catch clause.

User Response: Change or remove one of the catch clauses.

CBC1195 No default constructor exists for ″&1″.

Where: &1 is a class name

Explanation: An array of class objects must be initialized by calling the default constructor, but one has not been declared.

User Response: Declare a default constructor for the array.

CBC1196 More than one default constructor exists for ″&1″.

Where: &1 is a class name

Explanation: An array of class objects must be initialized by calling the default constructor, but the call is ambiguous.

User Response: Ensure that only one default constructor exists.

CBC1197 It is invalid to have a throw expression with type ″&1″.

Where: &1 is a C++ type

Explanation: You cannot throw a function or an expression of type ″void″.

User Response: Change the type or remove the throw expression.

CBC1198 The exception specification is ignored in this declaration.

Explanation: The declaration contains a function declarator with an exception specification but is not the declaration of a function. The exception specification is ignored.

User Response: Change the function declarator so that it is the declaration of a function.

CBC1199 The compiler cannot generate a default copy constructor for ″&1″.

Explanation: The default copy constructor cannot be generated for this class because there exists a member or base class that has a private copy constructor, or there are ambiguous base classes, or this class has no name.

User Response: Ensure that a member or base class does not have a private copy constructor. If not then ensure the class is named and there are no ambiguous references to base classes.

CBC1200 The compiler cannot generate a default copy assignment operator for ″&1″.

Explanation: The default copy assignment operator cannot be generated for this class because it has a const member or a reference member or a member (or base class) with a private copy assignment operator.

User Response: Ensure there are no const members, reference members or members with a private copy assignment operator.

CBC1201 &1 too few non-option arguments.

Where: &1 is an integer number

Explanation: You can generate this message only when you are running the compiler passes manually.

User Response: Add non-option arguments.

CBC1202 ″&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″.

CBC1203 Pure virtual function called.

Explanation: A call has been made to a pure virtual function from a constructor or destructor. In such functions, the pure virtual function would not have been overridden by a derived class and a runtime error would occur.

User Response: Remove the call to the pure virtual function.

CBC1204 ″&1″ is not allowed as a conversion function type.

Where: &1 is a C++ type

Explanation: A conversion function cannot be declared with a function or an array as its conversion type, since the type cannot be returned from the function.

User Response: Declare the function as converting to a pointer to the function or the array element type.

CBC1205 Syntax error - ″&1″ is followed by ″&3″ but is not the name of a &2.

Where: &1 is a C++ name &2 is the keyword class or template &3 is the token ’::’ or ’<’

Explanation: The name is not a class or template name but the context implies that it should be.

User Response: Change the name to a class or template name.

CBC1206 The previous &1 messages apply to the definition of template ″&2″.

Where: &1 is an integer number &2 is a template name

Explanation: The instantiation of the specified template caused the messages, even though the line numbers in the messages refer to the original template declaration.

User Response: This message supplies additional information for previously emitted messages. Refer to the descriptions of those messages for recovery information.

CBC1207 The previous message applies to the definition of template ″&1″.

Where: &1 is a template name

Explanation: The instantiation of the specified template caused the message, even though the line number in the message refers to the original template declaration.

User Response: This message supplies additional information for previously emitted messages. Refer to the descriptions of those messages for recovery information.

CBC1208 No suitable constructor exists for conversion from ″&1″ to ″&2″.

Where: &1 is a class name &2 is a C++ type

Explanation: A constructor is required for the class but no user-defined constructor exists and the compiler could not generate one.

User Response: Create a suitable constructor for conversion.

CBC1209 class ″&1″ does not have a copy assignment operator.

Where: &1 is a class name

Explanation: A copy assignment operator is required for the class but no user-defined copy assignment operator exists and the compiler could not generate one.

User Response: Create a copy assignment operator.

CBC1210 ″&1″ cannot be used as a template name since it is already known in this scope.

Where: &1 is a C++ name

Explanation: A template name must not match the name of an existing template, class, function, object, value or type.

User Response: Change one of the template names.

CBC1211 ″&1″ is expected for template argument &2.

Where: &1 is either 'expression' or 'type name' &2 is an integer number

Explanation: Either the argument is a type and the template has a non-type argument, or the argument is an expression and the template has a type argument.

User Response: Ensure the argument matches the template.

CBC1212 ″&1″ cannot be defined before the template definition of which it is an instance.

Where: &1 is a class template name

Explanation: An explicit definition of a template class cannot be given before the corresponding template definition.

User Response: Move the template definition so that it occurs before any template class definitions.

CBC1213 An ellipsis (...) cannot be used in the argument list of a template function.

Explanation: Since an exact match is needed for template functions, an ellipsis cannot be used in the function argument list.

User Response: Remove the ellipsis from the argument list.

CBC1214 The suffix for the floating point constant is not valid.

Explanation: You have provided an incorrect suffix for the floating point constant. Valid suffixes for floating point constants are L and F.

User Response: Change the suffix for the floating point constant.

CBC1215 Statement has no effect.

Explanation: The expression has no side effects and produces a result that is not used.

User Response: Remove the statement or use its result.

CBC1216 ″/*″ detected in comment.

Explanation: ″/*″ has been detected within a ″/*″ type comment. Nested comments are not allowed.

User Response: Remove the imbedded ″/*″ and ensure that you are not missing the end of the other comment.

CBC1217 Predefined macro name ″&1″ cannot be redefined or undefined.

Where: &1 is a predefined macro name

Explanation: The specified macro name is predefined by the compiler and cannot be redefined with #define or undefined with #undef.

User Response: Remove the definition expression or change the macro name.

CBC1218 The suffix for the integer constant is not valid.

Explanation: The integer constant is a suffix letter that is not recognized as a valid suffix.

User Response: Change the suffix to either ″u″ or ″l″.

CBC1219 The expression contains a division by zero.

User Response: Remove the division by zero from the expression.

CBC1220 The expression contains a modulus by zero.

User Response: Remove the modulus by zero from the expression.

CBC1221 Static member ″&1″ can only be defined at file scope.

User Response: Move the static member so that it is defined at file scope.

CBC1222 ″&1″ needs a constructor because &2 ″&3″ needs a constructor initializer.

Where: &1 is a class name &2 is 'class member' or 'base class' &3 is the member or base class name.

Explanation: You have not provided a constructor for the class, because the member or base class does not have a default constructor.

User Response: Add a constructor.

CBC1223 ″&1″ cannot be redeclared since it has already been used in this scope.

Where: &1 is a C++ name

Explanation: The name is being declared in a member list but was previously declared outside the member list and then used in the member list.

User Response: Change or remove one of the occurrences.

CBC1224 Conversion from ″&1″ to a reference to a non-const type ″&2″ requires a temporary.

Where: &1 is a C++ type &2 is a C++ type

Explanation: A temporary may only be used for conversion to a reference type when the reference is to a const type.

User Response: Change to a const type.

CBC1225 ″&2″ is too small to hold a value of type ″&1″.

Where: &1 is a C++ type &2 is a C++ type

Explanation: A conversion from a pointer type to an integral type is only valid if the integral type is large enough to hold the pointer value.

User Response: Remove the conversion from a pointer type to an integral type or use a larger integral type.

CBC1226 Object of type ″&1″ cannot be constructed from ″&2″ expression.

Where: &1 is a C++ type &2 is a C++ type

Explanation: There is no constructor taking a single argument that can be called using the given expression.

User Response: Change the expression.

CBC1227 The compiler cannot generate a copy constructor for conversion to ″&1″.

Where: &1 is a C++ type

Explanation: A copy constructor is required for the conversion. No suitable user-defined copy constructor exists and the compiler could not generate one.

User Response: Create a copy constructor for the conversion.

CBC1228 No suitable constructor or conversion function exists for conversion from ″&1″ to ″&2″.

Where: &1 is a C++ type &2 is a C++ type

Explanation: A constructor or conversion function is required for the conversion but no such constructor or function exists.

User Response: Create a constructor or conversion function for the conversion.

CBC1229 The file is empty.

Explanation: An empty source or include file has been encountered while reading source. The source file name or include file name may not be spelled correctly.

User Response: Check the file name.

CBC1230 Syntax error - ″&1″ has been inserted before ″&2″.

Where: &1 is a token &2 is a token

Explanation: A syntax error was found while parsing the program. The message identifies what the compiler expected and what it actually found. The compiler inserts the expected value and compilation continues.

User Response: Correct the syntax.

CBC1231 Call to ″&1″ matches some functions best in some arguments, but no function is a best match for all arguments.

Where: &1 is a function name

Explanation: No function matches each call argument as well as or better than all other functions.

User Response: Change the function call so that it matches only one function.

CBC1232 Call matches ″&1″.

Where: &1 is a function name and type

Explanation: The compiler detected an overloaded function or operator that is similar to another and is providing additional information.

User Response: Ensure this is the desired match.

CBC1233 Cannot adjust access of ″&1::&2″ because a member in ″&3″ hides it.

Where: &1 is a class name &2 is a member name &3 is the name of the derived class.

Explanation: You cannot modify the access of the specified member because a member of the same name in the specified class hides it.

User Response: Remove the access adjustment expression or unhide the member.

CBC1234 ″&1″ cannot be redeclared.

Where: &1 is a C++ name

Explanation: The specified name cannot be redeclared because it has already been used.

User Response: Change or remove one of the declarations.

CBC1235 Syntax error - ″&1″ is not allowed; ″&2″ has already been specified.

Where: &1 is a keyword &2 is a keyword

Explanation: You cannot use both of the specified attributes in the same declaration.

User Response: Remove the attributes.

CBC1236 Missing option to ″#pragma &1″; the directive is ignored.

Where: &1 is a pragma name

Explanation: A required option of the specified pragma directive is missing.

User Response: Ensure all options for the pragma are present.

CBC1238 Invalid or out of range pragma parameter; parameter is ignored.

Explanation: The pragma parameter specified is either not a valid parameter, or is out of range.

User Response: Remove the parameter or replace it with one within the range.

CBC1239 Function ″&1″ has internal linkage but is undefined.

Where: &1 is the invalid option

Explanation: If a static function or inline member function is referenced in this compilation unit, it must be defined in the same compilation unit.

User Response: Define the function in the same compilation unit it is referenced in.

CBC1240 Call to ″&1″ matches more than one template function.

Where: &1 is a function name and type

Explanation: More than one template for the function matches equally well with the argument list specified on the call.

User Response: Change the call so that it matches only one template function.

CBC1241 ″&1″ is declared inline, but is undefined.

Where: &1 is a function name and type

Explanation: An inline function must be defined in every compilation unit in which it is used.

User Response: Define the inline function in this compilation unit.

CBC1242 Non-&1 member function called for a &1 object via pointer of type ″&2″.

Where: &2 is a pointer or member-pointer type

Explanation: The member function is being called indirectly for a const or volatile object but it has not been declared with the corresponding const or volatile attribute.

User Response: Ensure that the function call and the function declaration match.

CBC1243 ″&1″ cannot be a base of ″&2″ because ″&3″ contains the type name ″&2″.

Where: &1 is a class name &2 is both the derived class name and a type name &3 is the class containing &2

Explanation: A class cannot inherit a type name that is the same as the class name.

User Response: Change the name of either the derived class or the inherited class.

CBC1244 ″&1″ cannot be a base of ″&2″ because ″&3″ contains the enumerator ″&2″.

Where: &1 is a class name &2 is both the derived class name and the enumerator name &3 is the class containing &2

Explanation: A class cannot inherit an enumerator with the same name as the class name.

User Response: Change the name of either the derived class or the inherited enumerator.

CBC1245 compiler doesn't generate this message any more

Explanation: n/a

CBC1246 Symbol length of &1 exceeds limit of &2 bytes.

Where: &1 is an integer number &2 is an integer number

Explanation: The compiler limit for the length of a symbol has been exceeded.

User Response: Shorten the symbol length.

CBC1247 The result of this pointer to member operator can be used only as the operand of the function call operator ().

Explanation: If the result of the .* or ->* is a function, then that result can be used only as the operand for the function call operator ().

User Response: Make the result the operand of the function call operator ().

CBC1248 When ″&1″ is used as an operand to the arrow or dot operator, the result must be used with the function call operator ().

Where: &1 is a member name

Explanation: If the result of the dot or arrow operator is a function, then that result can be used only as the operand for the function call operator ().

User Response: Make the result the operand of the function call operator ().

CBC1249 A class with a reference or const member needs a constructor.

Explanation: const and reference members must be initialized in a constructor initializer list.

User Response: Add a constructor to the class.

CBC1250 Base class initializers cannot contain virtual function calls.

Explanation: The virtual function table pointers are not set up until after the base classes are initialized.

User Response: Remove the call to a virtual function in the base class initializer.

CBC1251 The previous declaration of ″&1″ did not have a linkage specification.

Explanation: If you want to declare a linkage specification for a function, it must appear in the first declaration of the function.

User Response: Add a linkage specification to the first declaration of the function

CBC1252 The destructor for ″&1″ does not exist. The call is ignored.

Where: &1 is a C++ type

Explanation: The destructor call is for a type that does not have a destructor. The call is ignored.

User Response: Add a destructor to the type.

CBC1253 ″&1″ has been added to the scope of ″&2″.

Where: &1 is the name on a friend declaration &2 is a class name

Explanation: Because the friend class has not been declared yet, its name has been added to the scope of the class containing the friend declaration.

User Response: If this is not intended, move the declaration of the friend class so that it appears before it is declared as a friend.

CBC1254 The body of friend member function ″&1″ cannot be defined in the member list of ″&2″.

Where: &1 is the friend member function &2 is a class name

Explanation: A friend function that is a member of another class cannot be defined inline in the member list.

User Response: Define the body of the friend function at file scope.

CBC1255 The initializer list must be complete because ″&1″ does not have a default constructor.

Where: &1 is a class without a default constructor.

Explanation: An array of objects of a class with constructors uses the constructors in initialization. If there are fewer initializers in the list than elements in the array, the default constructor is used. If there is no default constructor the initializer list must be complete.

User Response: Complete the initializer list or add a default constructor to the class.

CBC1256 ″&1″ cannot be opened. The nested include file limit of &2 has been exceeded.

Where: &1 is a file name &2 is an integer number

Explanation: The compiler limit for nested include files has been reached.

User Response: Remove the nesting of one or more of the include files.

CBC1257 An &1 at file scope must have a storage class of static.

Where: &1 is one of TXanonymousclass, TXanonymousstruct, or TXanonymousunion

User Response: Change the storage class of the anonymous class/struct/union to static.

CBC1258 A pure virtual destructor needs an out-of-line definition in order for its class to be a base of another class.

User Response: Move the definition of the pure virtual destructor so that it is not inline.

CBC1259 The braces in the initializer are incorrect.

User Response: Correct the braces on the initializer.

CBC1260 Invalid octal integer constant.

Explanation: The octal integer constant contains an ’8’ or a ’9’. Octal numbers include 0 through 7.

User Response: Ensure that the octal integer constant is valid.

CBC1261 All the arguments must be specified for ″&1″ because its default arguments have not been checked yet.

Where: &1 is a function name and type

Explanation: For member functions, names in default argument expressions are bound at the end of the class declaration. Calling a member function as part of a second member function’s default argument is an error if the first member function’s default arguments have not been checked and the call does not specify all of the arguments.

User Response: Specify all the arguments for the function.

CBC1262 Ellipsis (...) cannot be used for ″&1″.

Where: &1 is an operator name

Explanation: An operator function has been specified with an ellipsis (...), but since the number of operands of an operator are fixed, an ellipsis is not allowed.

User Response: Remove the ellipsis, and specify the correct number of operands.

CBC1263 Syntax error - expected ″&1″ or ″&2″ and found ″&3″.

Where: &1 is a token &2 is a token &3 is a token

Explanation: A syntax error was found while parsing the program. The message identifies what the compiler expected and what it actually found.

User Response: Correct the syntax error.

CBC1264 A character constant must end before the end of the line.

Explanation: The compiler detected a character constant that was not terminated before an end-of-line character was found.

User Response: End the character constant or use ″\″ to continue it on the next line. The ″\″ must be the last character on the line.

CBC1265 A pure virtual function initializer must be 0.

Explanation: To declare a pure virtual function use an initializer of 0.

User Response: Set the virtual function initializer to 0.

CBC1266 ″&1″ is given ″&2″ access.

Where: &1 is a member name &1 is the keyword public, protected or private

Explanation: Access of the class has changed.

User Response: Ensure this change is as intended.

CBC1267 ″&1″ has been qualified with the ″this″ pointer.

Where: &1 is a member name

User Response: Ensure this qualification is intended.

CBC1268 Invalid escape sequence; the backslash is ignored.

Explanation: You have provided invalid character(s) after the backslash that does not represent an escape sequence. Therefore, the backslash is ignored and the rest of the escape sequence is read as is.

User Response: Ensure the escape sequence is valid.

CBC1269 The result of an address expression is being deleted.

User Response: Ensure this action is intended.

CBC1270 Conversion from ″&1″ to ″&2″ matches more than one conversion function.

Explanation: More than one conversion function could be used to perform the specified conversion.

User Response: Create a new conversion function for this conversion or change one of the types.

CBC1271 Conversion matches ″&1″.

Where: &1 is a function name and type

User Response: Ensure this is the intended match.

CBC1272 ″&1″ cannot be initialized with an initializer list.

Where: &1 is a class name

Explanation: Only an object of a class with no constructors, no private or protected members, no virtual functions and no base classes can be initialized with an initializer list.

User Response: Remove the class from the initializer list.

CBC1273 A pointer to a virtual base ″&1″ cannot be converted to a pointer to a derived class ″&2″.

Where: &1 is a C++ type &2 is a C++ type

Explanation: A pointer to a class B may be explicitly converted to a pointer to a class D that has B as a direct or indirect base class, only if an unambiguous conversion from D to B exists, and B is not a virtual base class.

User Response: Remove the conversion of the pointer.

CBC1274 The arguments passed using the ellipsis may not be accessible.

Explanation: Arguments passed using an ellipsis are only accessible if there is an argument preceding the ellipsis and the preceding argument is not passed by reference.

User Response: Ensure that there is an argument preceding the ellipsis and that the preceding argument is not passed by reference.

CBC1275 Member function ″&1″ has already been declared.

Where: &1 is the member function name

Explanation: A member function cannot be redeclared in the class definition.

User Response: Remove one of the declarations.

CBC1276 Assignment to a constant expression is not allowed.

Explanation: The left hand side of the assignment operator is an expression referring to a ″const″ location. For example, in ″a.b″, either ″b″ is a ″const″ member or ″a″ is a ″const″ variable.

User Response: Remove the assignment.

CBC1277 Assignment to const variable ″&1″ is not allowed.

Where: &1 is the variable name

Explanation: The left hand side of the assignment operator is a variable with the ″const″ attribute. ″const″ variables may be initialized once at the point where they are declared, but may not be subsequently assigned new values.

User Response: Remove the assignment to the const variable.

CBC1278 Syntax error found while parsing the bit-field declarator.

Explanation: The part of this member declaration up to the colon ″:″ appears to be a declaration of a bit-field, but the constant expression expected after the colon was either not found or incorrectly formed.

User Response: Correct the syntax error.

CBC1279 The return type for the ″operator->″ cannot be the containing class.

Explanation: The return type for the ″operator->″ must be a pointer to a class type, a class type, or a reference to a class type. If it is a class or reference, the class must be previously defined and must contain an ″operator->″ function.

User Response: Change the return type for the ″operator->″.

CBC1280 The virtual function table for ″&1″ is defined with ″&2″ linkage.

Where: &1 is class name &2 is a the keyword extern or static

Explanation: The class has one or more virtual function tables. A definition of each table will be generated in the current compilation.

User Response: Ensure this is the desired result.

CBC1281 The virtual function table for ″&1″ will be defined where ″&2″ is defined.

Where: &1 is class name &2 is a member function name

Explanation: The class has one or more virtual function tables. None will be defined in the current compilation, but will be defined in the compilation containing the definition of the specified member function.

User Response: Ensure this is the desired result.

CBC1282 The virtual function table for ″&1″ will be defined in a file specified by the user.

Where: &1 is class name

User Response: Ensure this is the desired result.

CBC1283 The previous message applies to function argument &1.

Where: &1 is an integer corresponding to the function argument number

Explanation: The previous message applies to the specified argument number. This message does not indicate another error or warning, it indicates which argument of the function call is the subject of the previous message.

CBC1284 Conversion from ″&1″ to a reference to a non-const type ″&2″ requires a temporary.

Where: &1 is a C++ type &2 is a C++ type

Explanation: A temporary may only be used for conversion to a reference type when the reference is to a const type. This is a warning rather than an error message because the ″compat″ language level is active.

User Response: Change the reference so that it is to a const type.

CBC1285 The address of a local variable or compiler temporary is being used in a return expression.

Explanation: The address of a local variable may not be valid once control is passed out of the function.

User Response: Declare the variable in the calling function or as a global variable, or change the return expression to not use the variable.

CBC1286 Keyword ″&1″ cannot be used with a function definition.

Where: &1 is a keyword.

User Response: Remove the keyword.

CBC1287 The #pragma directive must occur before the first C++ statement in program; The directive is ignored.

User Response: Remove the directive or place it before the first C++ statement in the program.

CBC1288 The pointer to member function must be bound to an object when it is used with the function call operator ().

Explanation: The pointer to member function must be associated with an object or a pointer to an object when it is used with the function call operator ().

User Response: Remove the pointer or associate it with an object.

CBC1289 The static data member ″&1″ has already been declared.

Where: &1 is the static data member

User Response: Remove or change one of the declarations.

CBC1290 Option ″&1″ must be specified on the command line or before the first C++ statement in the program.

Where: &1 is the option specified with the #pragma options directive

User Response: Remove the option or place it before the first statement in the C++ program.

CBC1291 The direct base ″&1″ of class ″&2″ is ignored because ″&1″ is also an indirect base of ″&2″.

Where: &1 is a base class name

Explanation: A reference to a member of ″&1″ will be ambiguous because it is inherited from two different paths.

User Response: Remove the indirect inheritance.

CBC1292 The ″&1″ operator cannot be applied to undefined class ″&2″.

Where: &1 is a class type

Explanation: A class is undefined until the definition of its tag has been completed. A class tag is undefined when the list describing the name and type of its members has not been specified. The definition of the tag must be given before the operator is applied to the class.

User Response: Complete the definition of the class before applying an operator to it.

CBC1293 ″&1″ hides the &2 ″&3″.

Where: &1 is the name of the derived class’s member &2 is ″pure virtual″ or ″virtual″ &3 is the name of the hidden virtual function

Explanation: A member in the derived class hides a virtual function member in a base class.

User Response: Ensure the hiding of the virtual function member is intended.

CBC1294 ″&1″ is not the name of a function.

Where: &1 is a C++ name

Explanation: A function name is required in this context. The specified name has been declared but it is not the name of a function.

User Response: Check the spelling. If necessary, change to a function name.

CBC1296 The virtual functions ″&1″ and ″&2″ are ambiguous since they override the same function in virtual base class ″&3″.

Where: &1 is a function name and type &2 is a function name and type

Explanation: The two functions are ambiguous and the virtual function call mechanism will not be able to choose the correct one at runtime.

User Response: Remove one of the virtual functions.

CBC1297 The ″this″ address for ″&1″ is ambiguous because there are multiple instances of ″&2″.

Where: &1 is a function name and type &2 is a class name

Explanation: Two or more ″this″ addresses are possible for this virtual function. The virtual function call mechanism will not be able to determine the correct address at runtime.

User Response: Remove the ″this″ expression or change the function name.

CBC1298 Conversion from ″&1″ matches more than one conversion function.

Where: &1 is a function name and type

Explanation: More than one conversion function could be applied to perform the conversion from the specified type.

User Response: Create a new conversion function or remove the conversion.

CBC1299 Function ″&1″ must not be declared as ″&2″.

Where: &2 is a keyword

Explanation: The specified function has a storage class that is not allowed in the context that the function is declared in.

User Response: Remove the declaration or change the storage class of the function.

CBC1300 The declaration of ″&1″ must initialize the const member ″&2″.

Where: &1 is a variable name &2 is the member name

User Response: Initialize the member in the declaration.

CBC1301 The declaration of ″&1″ must initialize the reference member ″&2″.

Where: &1 is a variable name &2 is the member name

User Response: Initialize the member in the declaration.

CBC1302 ″&1″ is not allowed as a function return type. There may be a ″;″ missing after a ″}″.

Where: &1 is the function return type

Explanation: A class or enum definition must not be specified as a function return type. A semicolon may be missing after the definition.

User Response: Ensure that a semicolon is not missing after the definition or change the return type.

CBC1303 ″&1″ cannot be a base of ″&2″ because ″&3″ contains a member function called ″&2″.

Where: &1 is a class name &2 is both the derived class name and the member function &3 is the class containing &2

Explanation: A class cannot inherit a function that has the same as the class.

User Response: Change the name of either the base class or the inherited function.

CBC1304 Forward declaration of the enumeration ″&1″ is not allowed.

Explanation: The declaration of an enumeration must contain its member list.

User Response: Fully declare the enumeration.

CBC1305 Unrecognized value ″&1″ specified with option ″&2″.

Where: &1 is the value specified with the option &2 is the option name

User Response: Remove the unrecognized value.

CBC1306 The previous message applies to argument &1 of function ″&2″.

Where: &1 is the argument number &2 is the function name and type

Explanation: The previous message applies to the specified argument number. This message does not indicate another error or warning, it indicates which argument of the function call is the subject of the previous message.

CBC1307 Unrecognized pragma ″&1″.

Explanation: The pragma is not supported by this compiler.

User Response: Change or remove the #pragma directive.

CBC1308 The nested class object ″&1″ needs a constructor so that its &2 members can be initialized.

Where: &1 is the nested class name &2 is the word const or reference

User Response: Create a constructor for the nested class object.

CBC1309 The integer constant is out of range.

Explanation: You have provided an integer constant that is out of range. For the range of integer constants check limits.h.

User Response: Ensure the integer constant is in range.

CBC1310 The floating point constant is out of range.

Explanation: You have provided a floating point constant that is out of range. For the range of floating point constants check float.h.

User Response: Ensure the floating point constant is in range.

CBC1311 The &1 member ″&2″ must be initialized in the constructor's initializer list.

Where: &1 is the word const or reference &2 is the member name

Explanation: Using the constructor's member initializer list is the only way to initialize nonstatic const and reference members.

User Response: Initialize the member in the constructor's initializer list.

CBC1312 Unexpected end of file: newline expected.

Explanation: The file did not end with a new-line character.

User Response: Ensure the file ends with a new-line character.

CBC1313 Constructors and conversion functions are not considered when resolving an explicit cast to a reference type.

Explanation: You cannot resolve an explicit cast to a reference type using constructors or conversion functions.

User Response: Cast the type to a temporary type and then take the reference to it.

CBC1314 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.

User Response: Ensure both strings have the same prefix, or no prefix at all.

CBC1315 All members of type ″&1″ must be explicitly initialized with all default arguments specified.

Where: &1 is a class name &2 is the member name

Explanation: Default arguments for member functions are not checked until the end of the class definition. Default arguments for member functions of nested classes are not semantically checked until the containing class is defined. A call to a member function must specify all of the arguments before the default arguments have been checked.

User Response: Specify all default arguments with all members of the type.

CBC1316 The nested class ″&1″ is undefined and cannot be defined later.

Where: &1 is the nested class name

Explanation: A class must be defined in the scope that it was introduced.

User Response: Define the class in the scope in which it was introduced.

CBC1317 The address of an overloaded function can be taken only in an initialization or an assignment.

User Response: Ensure the address of an overloaded function is used on an initialization or an assignment, or remove the expression.

CBC1319 **The mangled name for ″&1″ contains a compiler-generated name. It will not be visible from other compilation units.**

Explanation: One of the arguments to the function was given a compiler-generated name. This name could be different in offer compilation units.

User Response: Provide a type name for the argument that the compiler generated a name for.

CBC1320 **Syntax error - found ″&1 &2″ : ″&1″ is not a type name.**

Where: &1 is a token &2 is a token

Explanation: The compiler detected a non-type symbol where a type is required. A type must be used to declare an object.

User Response: Change to a type name or remove the expression.

CBC1321 **A temporary of type ″&1″ is needed: ″&2″ is an abstract class.**

Explanation: The compiler has determined that it must use a temporary to store the result of the expression, but the result is an abstract base type. An abstract base type cannot be used to create an object.

User Response: Change the type of the result.

CBC1322 **Nesting level of template class definitions may cause the compiler to fail.**

Explanation: Template class definitions are nested in such a way that the compiler may not be able to continue.

User Response: Reduce the number of nesting levels of template class definitions.

CBC1323 **″&1″ hides pure virtual function ″&2″ in the nonvirtual base ″&3″.**

Where: &1 is the derived member's name &2 is the name of the pure virtual function &3 is the name of the class that contains the pure virtual

Explanation: The pure virtual function in a nonvirtual base cannot be overridden once it has been hidden.

User Response: Make the pure virtual function visible, or make the base it is derived from virtual.

CBC1324 **The class qualifier ″&1″ for ″&2″ must be a template class that uses the template arguments.**

Where: &1 is a (possibly qualified) class name. &2 is a C++ name.

Explanation: A non-class template can only declare a global function or a member of a template class. If it declares a member of a template class, the template class arguments must include at least one of the non-class template arguments.

User Response: Change the template declaration so that it either declares a global function or a member of a template class that uses the non-class template arguments.

CBC1325 **The class ″&1″ cannot be passed by value because it does not have a copy constructor.**

Where: &1 is a class name

Explanation: The compiler needs to generate a temporary to hold the return value of the function. To generate the temporary object, a copy constructor is needed to copy the contents of the object being returned into the temporary object.

User Response: Create a copy constructor for the class or change the argument to pass by value.

CBC1326 **The previous &1 messages show situations that could arise if the corresponding template definitions were instantiated.**

Where: &1 is an integer number

Explanation: During the processing of a class template, possible errors were found in the class declaration. These errors may occur when the template is instantiated.

User Response: Ensure that the errors will not occur when the template is instantiated.

CBC1327 **The previous message shows a situation that could arise if the corresponding template definition was instantiated.**

Explanation: During the processing of a class template, a possible error was found in the class declaration. This error may occur when the template is instantiated.

User Response: Ensure that the error will not occur when the template is instantiated.

CBC1328 **The output file name ″&1″ cannot be the same as the input file name.**

Explanation: The compiler detected a condition where the name of the input source file is the same as an output file being generated by the compiler.

User Response: Change either the input file name or the output file name.

CBC1329 The external variable ″&1″ cannot be defined at block scope.

Explanation: The compiler has detected the declaration of an automatic variable that was previously defined as having external linkage.

User Response: Move, remove, or change the external variable definition.

CBC1330 ″&1″ cannot have an initializer list.

Where: &1 is a function name

Explanation: A member function that is not a constructor is defined with an initializer list.

User Response: Remove the initializer list.

CBC1331 Return value of type ″&1″ is expected.

Where: &1 is a C++ type

Explanation: No return value is returned from the current function but the function is expecting a non-void return value.

User Response: Ensure a value is returned, or change the return type of the function to void.

CBC1332 ″&1″ bypasses initialization of ″&2″.

Where: &1 is one of the keywords default, case &2 is the variable being initialized

Explanation: It is invalid to jump past a declaration with an explicit or implicit initializer unless the declaration is in an inner block that is also jumped past.

User Response: Enclose the initialization in a block statement.

CBC1333 ″&1″ is being redeclared as a member function. It was originally declared as a data member.

Where: &1 is a variable name

Explanation: The template redeclares a data member of a class template as a member function.

User Response: Change the original declaration of the variable to a member function, or change the redeclaration of the variable to a data member.

CBC1334 ″&1″ is being redeclared as a non-function member or has syntax errors in its argument list.

Where: &1 is a variable name

Explanation: The template redeclares a member function of a class template as a data member. There may be syntax errors in the declaration.

User Response: Change one of the declarations if necessary.

CBC1335 A string literal cannot be longer than &1 characters.

Where: &1 is a number. This number is system dependent.

Explanation: The compiler limit for the length of a string literal has been exceeded. The string literal is too long for the compiler to handle.

User Response: Specify a shorter string literal.

CBC1336 A wide string literal cannot be longer than &1 characters.

Where: &1 is a number. This number is system dependent.

Explanation: The compiler limit for the length of a wide string literal has been exceeded. The wide string literal is too long for the compiler to handle.

User Response: Specify a shorter string literal.

CBC1337 The definition of ″&1″ is not contained in an include file. It may be needed for automatic generation of template functions.

Where: &1 is a class name with a class keyword, e.g. ″struct S″.

Explanation: The definition of the class can only be used during automatic generation of template functions if it is contained in an include file.

User Response: Add the definition to an include file.

CBC1338 Invalid ″multibyte character sequence character″ (MBCS) character.

Explanation: The compiler has detected a multibyte character sequence that it does not recognize.

User Response: Replace the ″multibyte character sequence character″ (MBCS) character.

CBC1339 ″&1″ is an undefined pure virtual function.

Explanation: The user tried to call a member function that was declared to be a pure virtual function.

User Response: Remove or define the function as pure virtual.

CBC1341 Missing value for option ″&1″.

Where: &1 is an option name

Explanation: The option was messing a required parameter. See the ″Users Guide″ for details on the option.

User Response: Add a value for the option.

CBC1342 Template ″&1″ cannot be instantiated because the actual argument for formal argument ″&2″ has more than one variant.

Where: &1 is the name of a function template. &2 is the name of a formal template argument.

Explanation: The argument is a function template or an overloaded function with two or more variants. The compiler cannot decide which variant to choose to bind to the argument type.

User Response: Change the formal template argument or remove the extra variants.

CBC1343 More than 32760 files in a compilation unit.

Explanation: The compiler limit has been exceeded for the number of include files allowed in a compilation unit.

User Response: Reduce the number of files.

CBC1345 Pointer to a built-in function not allowed.

Explanation: Because you cannot take the address of a built-in function, you cannot declare a pointer to a built-in function.

User Response: Remove the pointer.

CBC1346 Built-in function ″&1″ not recognized.

Where: &1 is the name of a function.

Explanation: The function declared as a built-in is not recognized by the compiler as being a built-in function.

User Response: Ensure the function is a built-in function or remove the built-in keyword from the declaration.

CBC1347 ″&1″ is not supported.

Where: &1 is a C++ operator

User Response: Remove the operator from the expression.

CBC1348 Function calls are not supported.

Explanation: You can only generate this message in the debugger, when you use an expression that includes a function call.

User Response: Remove function calls from the expression.

CBC1349 The expression is too complicated.

User Response: Simplify the expression.

CBC1350 Evaluation of the expression requires a temporary.

User Response: Change the expression so that a temporary object is not required.

CBC1351 ″&1″ is an overloaded function.

Where: &1 is the name of a function.

Explanation: The identifier refers to an overloaded function with two or more variants. The compiler requires a prototype argument list to decide which variant to process.

User Response: Specify a prototype argument list or remove variants of the overloaded function.

CBC1352 Identifier or function prototype expected.

Explanation: The symbol must be the name of a data object, the name of a function with no variants, or a function or operator name followed by a parenthesized argument list.

User Response: Ensure the symbol is either the name of a data object, the name of a function with no variants, or a function or operator name followed by a parenthesized argument list.

CBC1353 ″&1″ does not have external linkage.

Where: &1 is an identifier.

Explanation: The pragma directives #map, #import, and #export can only be applied to objects or functions that are external.

User Response: Add or remove the #pragma directive.

CBC1354 ″&1″ has already been mapped.

Explanation: Only one map name may be given to any object or function.

User Response: Change one of the map names.

CBC1356 Invalid option with #pragma.

Explanation: The option specified for the #pragma directive is not valid.

User Response: Remove or change the option.

CBC1358 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.

CBC1362 Compiler-generated name ″&1″ overridden, may cause link problems.

Explanation: The specified object has a special compiler-generated external name, but appears in a #pragma map directive that would override that name. Using #pragma map to replace the name may cause link errors or prevent argument type checking across compilation units.

User Response: Remove the #pragma map directive that overrides the compiler-generated external name.

CBC1363 The bit-field length must not be negative.

Explanation: The bit-field length must be a non-negative integer value.

User Response: Change the bit-field length to a non-negative integer value.

CBC1364 A zero-length bit-field must not have a name.

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: Remove the name from the zero-length bit-field.

CBC1365 The bit-field is too small; &1 bits are needed for ″&2″.

Where: &2 is a C++ name

Explanation: The bit-field length is smaller than the number of bits needed to hold all values of the enum.

User Response: Increase the bit-field length.

CBC1366 The bit-field is larger than necessary; only &1 bits are needed for ″&2″.

Where: &2 is a C++ name

Explanation: The bit-field length is larger than the number of bits needed to hold all values of the enum.

User Response: Decrease the bit-field length.

CBC1370 A template friend declaration may only declare, not define, a class or function.

Explanation: The class or function declared in the template friend declaration must be defined at file scope.

User Response: Remove the definition from the template friend declaration.

CBC1371 The function ″&1″ must not be declared ″&2″ at block scope.

Where: &2 is a C++ keyword.

Explanation: There can be no static or inline function declarations at block scope.

User Response: Move the function so that it is not defined at block scope.

CBC1372 The previous &1 messages apply to function argument &2.

Where: &1 is an integer corresponding to the function argument number

Explanation: The previous message applies to the specified argument number. This message does not indicate another error or warning, it indicates which argument of the function call is the subject of the previous message.

CBC1373 The previous &1 messages apply to argument &2 of function ″&3″.

Where: &1 is the number of messages &2 is the argument number &3 is the function name and type

Explanation: The previous message applies to the specified argument number. This message does not indicate another error or warning, it indicates which argument of the function call is the subject of the previous message.

CBC1374 ″&1″ is not a static member of ″&2″.

Where: &2 is a class name.

Explanation: Non-static data members cannot be defined outside the class definition.

User Response: Make the member a static member or move it into the class definition.

CBC1375 The initializer must be enclosed in braces.

Explanation: Array element initializers must be enclosed in braces.

User Response: Put braces around the initializer.

CBC1376 union ″&1″ has multiple initializers associated with its constructor ″&2″.

Explanation: A union can only contain one member object at any time, and therefore can be initialized to only one value.

User Response: Remove all but one of the initializers.

CBC1377 ″&1″ is declared on line &2 of ″&3″.

Where: &1 is a C++ name &2 is a line number &2 is a file name

Explanation: This is an informational message; no response is necessary.

CBC1378 ″&1″ is defined on line &2 of ″&3″.

Where: &1 is a C++ name &2 is a line number &2 is a file name

Explanation: This is an informational message; no response is necessary.

CBC1379 Maximum number of error messages exceeded.

Where: Either the default or user-defined maximum number of error messages has been exceeded.

User Response: Correct the error and recompile.

CBC1380 You cannot override virtual function ″&1″ because ″&3″ is an ambiguous base class of ″&2″.

Where: &3 is the class name of an ambiguous base of &2

Explanation: The compiler must generate code to convert the actual return type into the type that the overridden function returns (so that calls to the original overridden function is supported). However, the conversion is ambiguous.

User Response: Clarify the base class.

CBC1381 The operands have type ″&1″ and ″&2″.

Explanation: This message provides more information when the array operator was used with invalid types. The message tells the user what the types were used with the array operator.

CBC1382 ″&1″ is defined in this compilation and cannot be imported.

Where: &1 is a function name and type.

Explanation: Only externally-defined functions can be imported.

User Response: Remove the directive that imports the function or define the function externally.

CBC1383 ″&1″ is not defined in this compilation and cannot be exported.

Where: &1 is a function name and type.

Explanation: Only functions defined in this compilation can be exported.

User Response: Remove the directive that exports the function or define the function in this compilation unit.

CBC1385 Macro ″&1″ has been invoked with an incomplete argument for parameter ″&2″.

Where: &2 is a macro parameter name.

Explanation: The terminating ″,″ or ″)″ for the argument was not found.

User Response: Ensure the terminating ″,″ or ″)″ is in the argument.

CBC1386 The enum cannot be packed to the requested size of &1.

Where: &1 is 1, 2, or 4.

Explanation: The enum type is too large to fit in the storage requested with the /Su option.

User Response: Redefine the storage to a larger size by specifying a larger number for /Su option.

CBC1387 ″&1″ is not initialized until after the base class is initialized.

Where: &1 is the class member referenced in the base class initializer.

Explanation: First, the base classes are initialized in declaration order, then the members are initialized in declaration order, then the body of the constructor is executed.

User Response: Do not reference the class member in the base class initializer.

CBC1388 The expression to the left of the ″&1″ operator is a relational expression (″&2″). The ″&3″ operator may have been intended.

Where: &1 is the bitwise operator | or &. &2 is one of the relational operators. &3 is either the operator || or the operator &&.

Explanation: The compiler has detected the mixing of relational and bitwise operators in what was determined to be a conditional expression.

User Response: Ensure the correct operator is being used.

CBC1389 The expression to the left of the ″&1″ operator is a logical expression (″&2″). The ″&3″ operator may have been intended.

Where: &1 is the bitwise operator | or &. &2 is one of the relational operators. &3 is either the operator || or the operator &&.

Explanation: The compiler has detected the mixing of relational and bitwise operators in what was determined to be a conditional expression.

User Response: Ensure the correct operator is being used.

CBC1390 The expression to the left of the ″&1″ operator is an equality expression (″&2″). The ″&3″ operator may have been intended.

Where: &1 is the bitwise operator | or &. &2 is one of the relational operators. &3 is either the operator || or the operator &&.

Explanation: The compiler has detected the mixing of relational and bitwise operators in what was determined to be a conditional expression.

User Response: Ensure the correct operator is being used.

CBC1391 The expression to the right of the ″&1″ operator is a relational expression (″&2″). The ″&3″ operator may have been intended.

Where: &1 is the bitwise operator | or &. &2 is one of the relational operators. &3 is either the operator || or the operator &&.

Explanation: This message is generated by the /Wcnd option. This option warns of possible redundancies or problems in conditional expressions involving relational expressions and bitwise operators.

User Response: Ensure the correct operator is being used.

CBC1392 The expression to the right of the ″&1″ operator is a logical expression (″&2″). The ″&3″ operator may have been intended.

Where: &1 is the bitwise operator | or &. &2 is one of the relational operators. &3 is either the operator || or the operator &&.

Explanation: This message will be generated when /Wcnd option is specified, in order to warn possible redundancies or problems in conditional expressions involving logical expressions and bitwise operators.

User Response: Ensure the correct operator is being used.

CBC1393 The expression to the right of the ″&1″ operator is an equality expression (″&2″). The ″&3″ operator may have been intended.

Where: &1 is the bitwise operator | or &. &2 is one of the relational operators. &3 is either the operator || or the operator &&.

Explanation: This message will be generated when /Wcnd option is specified, in order to warn possible redundancies or problems in conditional expressions involving equality expressions and bitwise operators.

User Response: Ensure the correct operator is being used.

CBC1394 Assignment to the ″this″ pointer is not allowed.

Explanation: The ″this″ pointer is a const pointer and cannot be modified.

User Response: Remove the assignment to the ″this″ pointer.

CBC1395 ″&1″ must not have any arguments.

Where: &1 is a special member function.

User Response: Remove all arguments from the special member function.

CBC1396 The second operand to the ″offsetof″ operator is not valid.

Explanation: The second operand to the ″offsetof″ operator must consist only of ″.″ operators and ″[]″ operators with constant bounds.

User Response: Remove or change the second operand.

CBC1397 ″&1″ is a member of ″&2″ and cannot be used without qualification.

Where: &2 is a possibly qualified class name

Explanation: The specified name is a class member, but no class qualification has been used to reference it.

User Response: Use the scope operator (::) to qualify the name.

CBC1398 ″&1″ is undefined. Every variable of type ″&2″ will assume ″&1″ has no virtual bases and no multiple inheritance.

Where: &2 is a pointer to member type

Explanation: The definition of the class is not given but the compiler must implement the pointer to member. It will do so by assuming the class has at most one nonvirtual base class.

User Response: If this assumption is incorrect, define the class before declaring the member pointer.

CBC1399 ″&1″ is undefined. The delete operator will not call a destructor.

Where: &1 is a name of a class, struct, or union

Explanation: The definition of the class is not given so the compiler does not know whether the class has a destructor. No destructors will be called.

User Response: Define the class.

CBC1400 Label ″&1″ is undefined.

Where: &1 is a C++ name

Explanation: The specified label is used but is not defined.

User Response: Define the label before using it.

CBC1401 The initializer for enumerator ″&1″ must be an integral constant expression.

Where: &1 is an enumerator name

Explanation: The value of an enumerator must be a constant expression that is promotable to a signed int value. A constant expression has a value that can be determined during compilation and does not change during program execution.

User Response: Change the initializer to an integral constant expression.

CBC1403 Overriding virtual function ″&1″ may not return ″&2″ because class ″&3″ has multiple base classes or a virtual base class.

Where: &1 is the name of a virtual function &2 is an abstract declarator &3 is the class being returned

Explanation: Contravariant virtual functions are supported only for classes with single inheritance and no virtual bases.

User Response: Ensure the class has single inheritance and no virtual bases.

CBC1404 Virtual function ″&1″ is not a valid virtual function override because ″&3″ is an inaccessible base class of ″&2″.

Where: &3 is the class name of an inaccessible base of &2

Explanation: The compiler must generate code to convert the actual return type into the type that the overridden function returns (so that calls to the original overridden function is supported). However, the target type is inaccessible to the overriding function.

User Response: Make the base class accessible.

CBC1405 ″&1″ is a member of &2 classes. To reference one of these members, ″&3″ must be qualified.

Where: &1 is a C++ member name &2 is an integer greater than 1 &3 is a C++ member name

Explanation: The class member specified is defined in more than one class nested within the base class and cannot be referenced from the base class if it is not qualified. This message is generated by the /Wund option.

User Response: Use the scope operator (::) to qualify the name.

CBC1406 ″&1″ is a member of ″&2″.

Where: &2 is a C++ class name

Explanation: This message will be invoked with /Wund option when UNQUALIFIED_MEMBER message (about unqualified members) is generated. This message tells you about the member data and the class it belongs to.

CBC1407 ″&1″ is not the name of a function.

Where: &1 is a name

Explanation: A function name is required in this context. The specified name has been declared but it is not the name of a function.

User Response: Ensure the name is the correctly-spelled name of a function.

CBC1408 The value given for the ″#pragma priority″ is in range reserved for the system.

Explanation: #pragma priority values less than -2147482624 are reserved for system purposes.

User Response: Change the #pragma priority value so that it is greater than -2147482624.

CBC1409 Priority values in successive ″#pragma priority″ statements must increase.

Explanation: The current priority cannot be higher than the priority specified in the previous #pragma priority statement. As the priority value increases with each #pragma priority directive, the priority level decreases.

User Response: Ensure priority values increase with each #pragma priority statements.

CBC1410 Initialization or termination done before first ″#pragma priority″ statement.

Explanation: Static objects should not be initialized or terminated before the first #pragma priority directive. before the first #pragma priority.

User Response: Ensure initialization or termination follows the first ″#pragma priority″ statement.

CBC1416 The option ″enum″ is not allowed in the middle of a declaration of an enum. This option is ignored.

Explanation: #pragma options with the option enum (#pragma options enum=) cannot be specified within an enumeration declaration.

User Response: Remove the enum option from the declaration.

CBC1417 Enum type ″&1″ cannot contain both negative and unsigned values.

Explanation: The enumerator type values should fit into an integer. Specifying both unsigned and negative values will exceed this limit.

User Response: Remove the negative or unsigned values.

CBC1427 Cannot take the address of the machine-coded function ″&1″.

Explanation: Because the function is machine-coded, you cannot take its address.

User Response: Remove the reference to that function.

CBC1429 Incorrect #pragma ignored.

Explanation: The pragma is not supported by this compiler or the syntax of this pragma is invalid.

User Response: Correct or remove the #pragma.

CBC1431 Invalid pragma name ″&1″ ignored.

Explanation: The pragma specified is not valid. The compiler ignores it.

User Response: Remove the invalid pragma name.

CBC1433 An initializer is not allowed for the nonvirtual function ″&1″.

Where: &1 is a function name

Explanation: The declaration of a pure virtual function must include the keyword virtual.

User Response: Remove the initializer.

CBC1459 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.

CBC1460 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

CBC1461 ″&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.

CBC1462 ″&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.

CBC1463 Sub-option is not allowed in ″&1″ option.

Where: &1 is the option name.

User Response: Remove the sub-option.

CBC1464 ″&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.

CBC1465 ″&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.

CBC1466 ″&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.

CBC1467 The include path specified was more than 54 characters.

Explanation: To map cleanly to MVS, path names have to be limited to 54 chars (max PDS length).

User Response: Shorten the include path.

CBC1468 The include filename has more than 8 characters. It has been truncated to ″&1″.

Explanation: To map cleanly to MVS, file names have to be limited to 8 chars (max member length).

User Response: Shorten the include file name.

CBC1469 The include file extension was more than 8 characters. It has been truncated to ″&1″.

Explanation: To map cleanly to MVS, file extensions have to be limited to 8 chars.

User Response: Shorten the include file extension.

CBC1470 ″&1″ has extern ″C++″ linkage and can not be mapped to ″&2″.

Where: ″&1″ is the original function name, ″&2″ is the new name.

Explanation: Only functions with extern ″C″ linkage can be mapped using #pragma map.

CBC1471 The linkage specification ″&1″ is not valid.

Where: ″&1″ is the linkage the user specified in pragma linkage

Explanation: The linkage specified with #pragma linkage is not valid for this identifier. Check the allowed linkage specifications.

User Response: Remove the linkage specification.

CBC1472 The identifier ″&1″ has not been declared yet, so cannot have a ″&2″ specified.

Where: ″&1″ is the unknown identifier, ″&2″ is 'linkage', 'map', or 'noinline'.

Explanation: Linkage, map or noinline can only apply to those identifiers which have been declared.

User Response: Declare the identifier before linkage, mapping or noinline.

CBC1473 Invalid syntax for pragma ″&1″. Expected ″&2″.

Explanation: The compiler encountered a pragma with an invalid syntax. The message identifies what the compiler expected and what it actually found.

User Response: Correct the syntax.

CBC1474 Argument to va_start must be a parameter name.

Explanation: va_start initializes the argument to point to the beginning of the list.

User Response: Ensure the argument to va_start is a parameter name.

CBC1475 A local variable or compiler temporary is being used to initialize reference member ″&1″.

Explanation: The local variable is only active until the end of the function, but it is being used to initialize a member reference variable.

User Response: Ensure that no part of your program depends on the variable or temporary.

CBC1482 ″&1″ must appear inside the member list for its class.

Where: &1 is one of the SOM pragmas, e.g. #pragma SOMReleaseOrder.

Explanation: The specified pragma may only appear within the member list for the class to which it applies.

User Response: Move the pragma inside the definition of the class.

CBC1483 ″&1″ must be declared to have non-C++ linkage in order to be fetchable.

Where: Change the declaration of function &1 so that it does not have C++ linkage.

Explanation: A fetchable function cannot have C++ linkage.

CBC1485 ″&1″ is not the SOM name of a SOM class.

Explanation: A SOM name that represents a SOM class is expected, and was not found. The SOM name of a class may differ from its C++ name.

User Response: Ensure that you use the correct SOM name for the class.

CBC1486 SOM class version must be an integer greater than or equal to zero.

Explanation: The major and minor version numbers supplied in the SOMClassVersion pragma must both be integers greater than or equal to zero.

User Response: Replace the number with a valid version number.

CBC1487 ″&1″ must specify the C++ name of a SOM class.

Where: &1 is one of the SOM pragmas, e.g. #pragma SOMReleaseOrder.

Explanation: The pragma requires the name of a SOM class. Some pragmas may also permit an asterisk, if the pragma appears inside the class definition. The name you have supplied does not represent a SOM class visible in the current scope.

User Response: Ensure that you use the correct C++ name of a SOM class. correctly.

CBC1488 ″SOMObject″ method ″&1″ is missing or misplaced in the release order.

Explanation: The definition of the special SOM class ″SOMObject″ is not compatible with the use of the Direct-to-SOM feature. A valid definition is found in the standard Direct-to-SOM include header ″somobj.hh″.

User Response: Use the valid definition or disable the Direct-to-SOM feature.

CBC1489 Definition of ″&1″ is only allowed at file scope.

Where: &1 is a C++ template class type

Explanation: A template class is being defined in a scope other than file scope. Because all template class names have file scope this definition is not allowed.

User Response: Move the template class definition to file scope.

CBC1490 Class template ″&1″ cannot be used until its containing template has been instantiated.

Where: &1 is a C++ class template type

Explanation: The class template referenced cannot be used until the template that contains it has been instantiated. template cannot be used.

User Response: Declare the class template at file scope or instantiate the template that contains it.

CBC1491 The data in precompiled header &1 does not have the correct format.

Where: &1 is the name of the precompiled header file

Explanation: The precompiled header file has been corrupted or is not actually a precompiled header file.

User Response: Delete the corrupted header file or use the correct option to regenerate it.

CBC1492 Unable to open precompiled header &1. The original header will be used.

Where: &1 is the name of the precompiled header file

Explanation: The specified error occurred when the compiler attempted to open the precompiled header file.

User Response: Correct the condition that prevented the open.

CBC1493 Precompiled header &1 was created by a later release of the compiler. The original header will be used.

Where: &1 is the name of the precompiled header file

Explanation: The precompiled header cannot be used because it was created by a later version of the compiler.

User Response: Delete the header or use the -genpcomp option to regenerate it.

CBC1494 Unable to write to precompiled header &1.

Where: &1 is the name of the precompiled header file

Explanation: The specified error occurred when the compiler attempted to write to the precompiled header file.

User Response: Correct the condition which prevented the write operation.

CBC1495 Invalid wchar_t value &1.

Where: &1 is the value which is not valid

Explanation: A multibyte character or escape sequence in a literal has been converted to an invalid value for type wchar_t.

User Response: Change the character or escape sequence.

CBC1496 Macro &1 has been invoked with an empty argument for parameter &2.

Where: &2 is the name of a macro parameter

Explanation: The argument corresponding to the specified parameter has no tokens.

User Response: If necessary, specify an argument.

CBC1498 Precompiled header &1 created.

Where: &1 is the name of the precompiled header file

Explanation: The precompiled header was successfully created.

CBC1499 Cannot open precompiled header &1 for output.

Where: &1 is the name of the precompiled header file

Explanation: The specified error occurred when the compiler attempted to open the precompiled header file.

User Response: Correct the condition which prevented the compiler from opening the file.

CBC1500 Precompiled header &1 not used because the header file was modified.

Where: &1 is the name of the precompiled header file

Explanation: The precompiled header cannot be used because the header file that created it was modified after the precompiled header file was generated.

User Response: Delete the header or use the -genpcomp option to regenerate it.

CBC1501 Precompiled header &1 used.

Where: &1 is the name of the precompiled header file

Explanation: The compiler is using the precompiled header file indicated.

CBC1502 ″&1″ was introduced in class ″&2″ and can only be specified in the SOMReleaseOrder list for that class.

Where: &2 is a C++ class name.

Explanation: Only those virtual functions or operators introduced in a class may appear in its release order list. Virtual functions that override a base class virtual function should normally only appear in the release order of the base class. However, under special circumstances you can use the ″!″ notation to also put it in the derived class release order; see the description of the SOMReleaseOrder pragma for details.

User Response: Remove the member from this release order pragma.

CBC1503 SOM class ″&1″ has a non-SOM base class ″&2″.

Where: &2 is a C++ name.

Explanation: All base classes of a SOM class must themselves be SOM classes.

User Response: Change the list of base classes so they are either all SOM or all non-SOM.

CBC1504 Instances of SOM class ″&1″ will inherit more than one sub-object of base class ″&2″.

Where: &2 is a C++ name.

Explanation: All base classes of a SOM class that appear more than once in the class hierarchy must be virtual base classes.

User Response: Make the base class a virtual base class.

CBC1507 ″&1″ has already been defined as a SOM class name, metaclass name or SOM module name.

Where: &1 is a C++ name.

Explanation: All SOM names of classes, metaclasses and modules must be unique in the same scope, without regard to case sensitivity.

User Response: Change the SOM name of the class, the metaclass or the module to make it unique.

CBC1508 ″&1″ has already been defined as a SOM member name in class ″&2″.

Where: &2 is a C++ name.

Explanation: All SOM member names must be unique within the class, without regard to case sensitivity.

User Response: Change the SOM name of the member to make it unique.

CBC1509 ″&1″ already has a SOM name ″&2″.

Where: &2 is a C++ name.

Explanation: A SOM class or member is being assigned a SOM name when it already has one. It may have already been assigned a SOM name by a preceding pragma, or it may have assumed a default SOM name because the pragma occurs too late in the source. We recommended that you put the pragma inside the class definition to avoid the latter problem.

User Response: Remove redundant pragmas or move unique ones into the class definition.

CBC1511 ″&1″ already has a SOM metaclass name ″&2″.

Where: &2 is a C++ name.

Explanation: A SOM class is being assigned a SOM metaclass name when it already has one.

User Response: Remove the redundant pragma.

CBC1512 ″&1″ is not a member of a SOM class.

Where: &1 is a C++ name.

Explanation: The name is not declared as a member of a SOM class.

User Response: Replace the name with the name of a member of a SOM class.

CBC1513 ″#pragma SOMAttribute″ cannot be applied to ″&1″.

Where: &1 is a C++ name.

Explanation: The SOMAttribute pragma can only be applied to nonstatic data members. In addition, the data member cannot be a reference to an abstract class type.

User Response: Remove the SOMAttribute pragma or change the data member type.

CBC1514 Direct-to-SOM class ″SOMObject″ must have a SOMReleaseOrder pragma.

Explanation: The definition of the special SOM class SOMObject is not compatible with the use of the Direct-to-SOM feature. A valid definition is found in the standard Direct-to-SOM include header ″somobj.hh″.

User Response: Use the valid definition or disable the Direct-to-SOM feature.

CBC1515 Argument ″&1″ of ″#pragma SOMClassInit″ is not of the necessary function type.

Where: &2 is a C++ type.

Explanation: Argument ″&1″ of #pragma SOMClassInit must be a non-member or static member function taking one argument of type ″SOMClass*″ and returning void.

User Response: Replace the argument with a function of the correct type.

CBC1516 Entry ″&1″ in the release order list has either been specified twice or is not a member of the class.

Where: &1 is a C++ name.

Explanation: You can only specify a member of a class in the release order list of that class, and each member may not appear more than once.

User Response: Remove the entry from the release order list and recompile or add a corresponding member to the class and recompile.

CBC1517 Some members did not appear in the release order list for SOM class ″&1″.

Where: &1 is a C++ name.

Explanation: There are public or protected members that were not mentioned in the release order list. They are added to the end of the list.

User Response: Add the missing members to the release order list.

CBC1518 The string must be terminated before the end of the line.

Explanation: The compiler detected a string that was not terminated before an end-of-line character was found.

User Response: End the string or use ″\″ to continue the string on the next line. The ″\″ must be the last character on the line.

CBC1519 A character constant must end before the end of the line.

Explanation: The compiler detected a character constant that was not terminated before an end-of-line character was found.

User Response: End the character constant or use ″\″ to continue it on the next line. The ″\″ must be the last character on the line.

CBC1520 A matching &1 function named ″&2″ could not be found.

Where: &1 is one of ’const’, ’volatile’ or ’const volatile’. &2 is the name of the called function (without the argument list).

Explanation: The call may have failed because no member function exists that accepts the ’const/volatile’ qualifications of the object.

User Response: Ensure the type qualifier is correct and that the function name is spelled correctly.

CBC1521 ″&1″ was previously declared as a SOM class, but is not being defined as a SOM class.

Where: &1 is a C++ name.

Explanation: The class was expected to be a SOM class, probably because its name appeared in a SOMClassName or SOMMetaClass pragma, but it is being defined as a non-SOM class. It will be defined as a SOM class only if the SOMAsDefault pragma is ″on″, or if the class inherits from the SOMObject class.

User Response: Change the class declaration or definition to make them consistent.

CBC1522 The macro ″&1″ has been redefined.

Where: &1 is a macro name

Explanation: An active definition already exists for the macro name being defined. The second definition will be used.

User Response: Remove or rename one of the macro definitions if necessary.

CBC1523 ″&1″ is a type name being used where a variable name is expected.

Where: &1 is a C++ name

Explanation: The identifier must be a variable name not a type name.

User Response: Check that the identifier is a variable name and ensure the variable is not hidden by a type name.

CBC1524 Template ″&1″ has a missing or incorrect template argument list.

Where: &1 is a C++ name

Explanation: A template name was found where a variable name was expected.

User Response: Complete the template argument list or change the identifier to a variable name.

CBC1526 Template friend declaration does not declare a class or a function.

Explanation: A template friend declaration must declare a class or a function following the template arguments.

User Response: Change the template declaration to declare a class or a function.

CBC1528 The ’const’ object has been cast to a non-’const’ object.

Explanation: A cast has been used to possibly modify a ’const’ object. This may cause undefined behaviour at run-time.

User Response: Remove the cast or make the object non-const.

CBC1531 #pragma HasHome must be specified for ″&1″ before #prama IsHome may be specified.

Where: &1 is the name of a class, struct or union.

Explanation: You must specify #pragma HasHome for ″&1″ before you can specify #pragma IsHome for it.

User Response: Specify the #pragma HasHome before the #pragma IsHome or remove the #pragma IsHome.

CBC1532 &1 may only be used at file scope.

Where: &1 is the name of the pragma: eg. #pragma IsHome

Explanation: The pragma is only valid at file scope.

User Response: Move the pragma so that it is in file scope.

CBC1533 Global friend functions may not be defined in a local class.

Explanation: A local class cannot have a friend function.

User Response: Make the function a member function in the local class.

CBC1534 No matching #pragma &1&2 for #pragma &1(pop)

Where: &1 is the name of one of the on|off SOM pragmas or the ObjectModel pragma &2 is either (on|off) or (iom|som|com|native|default)

Explanation: Either a #pragma &1&2 was not specified or a #pragma &1(pop) has already been encountered.

User Response: Remove the pop or add on|off or iom|som|com|native|default to the pragma.

CBC1543 SOM class &1 must not have operator= functions and somAssign functions.

Where: &1 is a class name.

Explanation: somAssign is an obsolete SOM member function, introduced by the SOMObject base class, that performs similar to operator= function. somAssign is still supported for backward compatibility, but classes may not have both somAssign member functions and operator= members. If you define only one, the compiler will supply a compatible version of the other.

User Response: Remove either the operator= methods or the somAssign methods.

CBC1544 #pragma argument directive has already been specified for function ″&1″. This #pragma will be ignored.

Explanation: #pragma argument for a function can only be defined once. Any subsequent specification will be ignored.

CBC1545 Function ″&1″ specified within #pragma argument should not have any linkage type (except ″C″) associated with it.

Explanation: Function with linkage type such as ″OS″ or ″built-in″ are not allowed in the #pragma directive.

User Response: Remove the linkage type or make it ″C″.

CBC1546 Function specified in #pragma argument must be defined or declared before the directive.

Explanation: The function specified within the #pragma argument is either not defined or defined after the directive.

User Response: Define the function before the #pragma argument directive.

CBC1548 The initial #pragma SOMAsDefault(on) is not at file scope.

Explanation: The SOMAsDefault pragma may not appear nested inside a class or in a function. If you are declaring nested classes and wish them to be SOM classes, you must either use the pragma at file scope, or explicitly derive the classes from the SOMObject class.

User Response: Delete the pragma or move it to file scope.

CBC1549 ″&1″ cannot be converted to ″&2″ because one has SOMCallStyle(oidl) and the other has SOMCallStyle(idl).

Explanation: The callstyle of a class affects how its methods are called, and methods of a SOM class must be invoked using the call style they were constructed to expect. For this reason, pointers to members of a SOM class cannot be converted to pointers to members of a class with an incompatible call style.

User Response: Change the classes to have the same call style.

CBC1550 Unimplemented SOM feature: &1.

Explanation: You have attempted to use a feature of SOM that is not supported, or a C++ language construct that is not supported for SOM objects. This may occur if you try to use the unsupported construct directly, or if the compiler needs to use an unsupported construct in order to implement your code.

User Response: Rewrite your code using different constructs, to accomplish the same result.

CBC1551 The address of data member ″&1″ cannot be taken because the member is being referenced through a _get_ function.

Explanation: An attribute is access through a ″_get_″ method if its backing data is not accessible, or if the SOMNoDataDirect pragma is in effect for the class. Since the ″__get″ method returns the value of the member, and not its address, it isn't possible to use the address operator ″&″ on the member to create an ordinary pointer. This error may also be generated if you haven't used the ″&″ operator explicitly, but the compiler needs to use it to implement your code. You can create a a pointer-to-member that refers to an attribute.

User Response: Rewrite the expression that causes the address to be taken, or remove the SOMAttribute pragma.

CBC1552 Local class ″&1″ may not be a SOM class.

Explanation: A local class is a class defined inside a function body. Local classes that are SOM classes are not supported.

User Response: Make the class a non-SOM class or define it at file scope.

CBC1553 Class ″&1″ has multiple SOMClassVersion pragmas with differing values.

Explanation: A single pair of major and minor version numbers should be associated with the implementation of each class, and with each client, so that SOM can detect incompatibilities.

User Response: Remove all but one of the pragmas.

CBC1554 ″&1″ must occur at SOM class scope.

Explanation: Some SOM pragmas must appear within the class scope, because the compiler must have the information they supply to correctly process the class definition. Put these pragmas inside the braces of the class definition, but not inside any nested classes or function bodies.

User Response: Move the pragma inside the class definition.

CBC1555 Using '*' to represent the current class for ″&1″ is only valid at SOM class scope.

Explanation: Some SOM pragmas that take a SOM class name as a parameter can also use an asterisk '*' to identify the class, but only if the pragma appears in the definition of the class. The pragma also must not be inside a nested scope, such as a nested class or function body.

User Response: Move the pragma inside the class definition or replace '*' with the class name.

CBC1556 '*' is not valid for SOM ″&1″.

Explanation: The pragma does not accept '*' to designate the current class. You may achieve the same effect by turning the pragma 'on' before the class, and using the 'pop' parameter after the class.

User Response: Replace '*' with one of 'on', 'off', or 'pop'.

CBC1557 Expected one of 'on'/'off'/'pop' for ″&1″.

Explanation: This pragma toggles a SOM option on or off when the 'on' or 'off' parameters are supplied, and returns to the previous value when the 'pop' parameter is used. No other parameter values are valid.

User Response: Replace the invalid parameter with one of 'on', 'off', or 'pop'.

CBC1558 Expected one of 'on'/'off'/'pop'/'*' for ″&1″.

Explanation: This pragma toggles a SOM option default on or off when the 'on' or 'off' parameters are supplied, and returns to the previous value when the 'pop' parameter is used. In addition, if the parameter appears inside a SOM class scope, you can use the '*' parameter to turn the option on only for the current class. No other parameter values are valid.

User Response: Replace the invalid parameter with one of 'on', 'off', 'pop', 'or' *.

CBC1559 Expected one of 'idl'/'oidl' for ″&1″.

Explanation: The callstyle of a SOM class determines how its methods are called, and is specified using the SOMCallStyle pragma. This pragma takes a single parameter which must be either 'idl' or 'oidl'.

User Response: Replace the invalid or missing parameter with 'idl' or 'oidl'.

CBC1560 More than one ″&1″ in class &2.

Explanation: Only one instance of this pragma is permitted in any particular SOM class.

User Response: Remove the extra pragmas.

CBC1561 The address of a dereferenced pointer-to-member expression may not be taken because get/set methods are being used.

Explanation: An attribute is accessed through a ″__get″ method if its backing data is not accessible, or if the SOMNoDataDirect pragma is in effect for the class.

It is possible to create a pointer-to-member that refers to an attribute, and the ″__get″ method will be called when the pointer-to-member is dereferenced. However, since the ″__get″ method returns the value of the member and not its address, it isn't possible to use the address operator ″&″ on the dereferenced pointer-to-member to create an ordinary pointer.

User Response: Don't take the address of the attribute, or make the attribute's backing store accessible, or remove the SOMAttribute pragma so the data member will not be an attribute.

CBC1562 Alignment is determined at the left brace of the definition.

Explanation: The alignment has been changed during a class definition.

User Response: Remove the #pragma align or place it before the class definition.

CBC1563 '!' was specified for ″&1″, which was introduced in the current class.

Where: &1 is a C++ member name.

Explanation: '!' must only be used for names introduced in a base class.

User Response: Remove the '!' from the SOMReleaseOrder entry.

CBC1564 No assignment operator exists for ″&1″ of class ″&2″. Compiler-generated ″_set_&1″ will call SOMError.

Where: &1 is a C++ member name.

Explanation: The compiler cannot generate the _set function for a class member because the member is an instance of a class that does not have a suitable operator=.

User Response: Use #pragma SOMAttribute(var, noset) and write your own _set function.

CBC1565 The #include of <somobj.hh> is not at file scope.

Explanation: A line containing #pragma SOM was found nested inside a class or function, but the pragma is only valid at file scope. Since the pragma is normally found only within the standard header file somobj.hh, it is likely that somobj.hh (or another header file that includes it, such as som.hh) was included inside a class or function scope.

User Response: Move the #include line to file scope.

CBC1566 SOMMethodName for ″&1″ is not valid because ″&1″ was introduced in class ″&2″.

Where: &2 is a C++ class name.

Explanation: The SOMMethodName pragma cannot be applied to virtual functions that override a function in a base class, because the introducing function and all its overrides must have the same SOM name. The pragma can be applied to virtual functions only in the introducing class, and to any non-virtual member functions.

User Response: Remove the SOMMethodName pragma or make the function not virtual.

CBC1567 #pragma SOMCallStyle may not be changed more than once per class.

Explanation: The callstyle of a SOM class is associated with the class and affects all of its member functions. Callstyle cannot be changed once set, and cannot have different values for different member functions.

User Response: Remove the extra #pragma SOMCallStyle.

CBC1568 ″&1″ may only be used for SOM classes.

Explanation: This pragma is valid only for SOM classes. A class is a SOM class if it inherits from SOMObject, or if the SOMAsDefault pragma is 'on'.

User Response: Remove the pragma.

CBC1569 Option ″&1″ is not supported for &2.

Explanation: The option is not supported by &2 compiler.

User Response: Remove the option.

CBC1570 Syntax error while processing option ″&1″ - expected ″&2″ and found ″&3″.

Where: &1 is an option name &2 is a C++ token &3 is a C++ token

Explanation: A syntax error was found while parsing the option. The message identifies what the compiler expected and what it actually found. Often the source of the error is an unmatched parenthesis or a missing semicolon.

User Response: Correct the syntax.

CBC1571 User cast between SOM and non-SOM pointer types ″&1″ and ″&2″.

Where: &1 is a C++ type &2 is a C++ type

Explanation: Either the source or destination type is a pointer to a SOM object, and the other type is a pointer to some non-SOM object. This cast could cause problems because the layout of SOM objects is known only to SOM, and may change in later versions of the class or of its base classes.

User Response: If this is the desired cast, cast to void* first to suppress the warning.

CBC1572 &1 is not valid for &2.

Where: &1 is a SOM pragma. &2 is a C++ member name.

Explanation: This pragma is valid only for certain kinds of SOM class members, but has been applied to a different kind of member. For example, the SOMMethodName pragma is valid only for functions, and attempts to use it on data members will generate this error.

User Response: Remove the pragma.

CBC1573 ″&1″ is not a typename.

Where: &1 is a C++ type.

Explanation: Parameters to #pragma SOMIDLTypes must be type names, which are either typedef names or the names of classes, structs, unions, or enums. Variable names are not valid, nor are the predefined basic types such as 'int'.

User Response: Do not use variables or basic types as parameters to pragma SOMIDLTypes.

CBC1588 ″&1″ must have type ″Environment *″, not ″&2″.

Explanation: The __SOMEnv variable that you declared does not have the correct type.

User Response: Correct the declared type.

CBC1590 Address of ″&1″ may not be taken, because ″&1″ is an indirect attribute.

Explanation: An indirect attribute may not be converted to a pointer to data member, because the compiler cannot generate code to correctly dereference the pointer later. Indirect attributes are for compatibility with SOM 1.0.

User Response: Remove the indirect attribute from the #pragma SOMAttribute for the variable.

CBC1596 ″&1″ does not have external linkage. Pragma export ignored.

Where: &1 is an identifier.

Explanation: The pragmas map, import, and export can only be applied to objects or functions that are external.

User Response: Give the identifier external linkage.

CBC1597 ″&1″ is already exported. Duplicate directive ignored.

Where: &1 is a function name and type.

Explanation: A function may be imported or exported at most once.

User Response: Remove one of the directives.

CBC1598 The compiler could not open the output file ″&1″.

Where: &1 is a file name.

Explanation: The open command failed for file ″&1″.

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.

CBC1599 ″&1″ cannot be exported. Directive ignored.

Explanation: The function main cannot be exported.

User Response: Remove the directive.

CBC1601 _get/_set routines must not be declared by users.

Explanation: The compiler will generate _get/_set declarations for SOM attributes. You must not declare them. You may define these routines if the noget/noset/nodata attributes are on.

User Response: Use #pragma SOMAttribute to declare the _get and _set routines.

CBC1602 Only one of 'nodata', 'publicdata', 'protecteddata', 'privatedata' may be specified in a ″#pragma SOMAttribute″.

Explanation: A SOM attribute must either have no backing data, or the backing data must be public, protected or private. It is an error to specify more than one backing data modifier for a variable.

User Response: Remove the extra modifiers.

CBC1603 Access mode ″&1″ for backing data is more accessible than mode ″&2″ for member ″&3″.

Where: &1 is an access specifier. &2 is an access specifier. &3 is a variable name.

Explanation: It is an error for the backing data for a SOM attribute to be more accessible than the _get/_set routines. An example of the problem would be a private attribute with public backing data.

User Response: Change the access of the backing data or of the member.

CBC1604 #pragma SOMAttribute may only be given once for data member ″&1″.

Where: &1 is a variable name.

Explanation: The SOMAttribute pragma may only be specified once for each data member.

User Response: Combine all the attributes into one #pragma SOMAttribute.

CBC1605 Assignment to read-only variable ″&1″ is not allowed.

Where: &1 is the variable name

Explanation: A SOM readonly variable is similar to a C++ const variable. It is an error to assign to the variable using the _set_var procedure call. You can assign to a read-only attribute only if its backing data is accessible.

User Response: Remove the assignment to the SOM read-only variable.

CBC1606 ″&1″ already has a class initializer function ″&2″.

Where: &1 is a SOM class name. &2 is a function name.

Explanation: Only one class initializer function is allowed for each SOM class.

User Response: Remove the extra SOMClassInit pragmas.

CBC1607 The address of compiler-generated routine ″&1″ may not be taken unless it appears in a #pragma SOMReleaseOrder.

Where: &1 is a function name.

Explanation: A compiler-generated operator assignment function for a SOM class may not be converted to a pointer-to-function member unless the routine is mentioned in the release order. The purpose of this restriction is to ensure binary compatibility if the class is modified later.

User Response: Add the method to the release order.

CBC1608 ″&1″ is not a SOM attribute.

Where: &1 is a variable name.

Explanation: Only data members that are SOM attributes may be specified in the SOMReleaseOrder pragma.

User Response: Remove the member from the release order.

CBC1609 The return type ″&1″ is not valid for a function of ″&2″ linkage.

Explanation: For example, functions with COBOL linkage cannot return a value; they must return void.

User Response: Use a valid return type.

CBC1610 Invalid or out of range pragma parameter; pragma is ignored.

Explanation: The pragma parameter specified is invalid or out of range.

User Response: Remove the parameter or replace it with one within the range.

CBC1611 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.

CBC1612 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.

CBC1613 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.

CBC1614 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.

CBC1615 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.

CBC1616 Cannot adjust access of ″&1″ because it is not an attribute.

Where: &1 is a member name

Explanation: The access to a SOM class data member can be adjusted only if the data member is designated an attribute by use of the SOMAttribute pragma.

User Response: Remove the access adjustment expression or use the SOMAttribute pragma.

CBC1617 Precompiled header file cannot be generated because a declaration was not complete when the last header file ended.

Explanation: A declaration may not begin in a header file and end in the main program file. No precompiled header file is generated.

User Response: Complete the declaration before the end of the header file.

CBC1618 Pointer to member does not refer to an attribute when the SOMNoDataDirect pragma is in effect for the class.

Where: &1 is a member name

Explanation: A pointer to member is assigned the address of a data member &1 that is not an attribute when the SOMNoDataDirect pragma is in effect for the class. If the pointer is dereferenced, the compiler will attempt to use ″__get″ and ″__set″ methods to access the member, which will result in a run-time error, because these methods do not exist for the data member.

User Response: Rewrite the expression that causes the address to be taken, designate the data member as an attribute using the SOMAttribute pragma, or remove the SOMNoDataDirect pragma.

CBC1619 Undefined class &1 specified with &2 option.

Where: &1 is a class name

Explanation: The class name &1 was specified with the &2 command line option, but this class is not defined. A release order will not be generated for the class.

User Response: Define the class, change the name specified with the &2 option, or remove the &2 option.

CBC1620 #pragma &1 directive can be specified only once per source file.

Explanation: You can specify the #pragma directive indicated only once in each source file.

User Response: Remove one of the #pragma &1 statements.

CBC1622 Option ″&1″ is turned on because option ″&2″ is specified.

Explanation: If option &2 is on, option &1 is also required to be on to achieve a better options combination.

User Response: Also turn on &1 if &2 is specified.

CBC1623 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.

CBC1624 &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.

CBC1625 &1 does not exist.

Explanation: The dataset does not exist.

User Response: Supply an existing dataset.

CBC1626 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.

CBC1627 &1 should be a partitioned dataset.

Explanation: A partitioned dataset is expected.

User Response: Supply a partitioned dataset.

CBC1628 &1 should not be a partitioned dataset.

Explanation: A non-partitioned dataset is expected.

User Response: Supply a non-partitioned dataset.

CBC1629 &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.

CBC1630 &1 has attributes &2.

Explanation: The dataset has the attributes indicated.

User Response: None.

CBC1631 The attributes should be &1.

Explanation: The dataset should have the attributes indicated.

User Response: None.

CBC1632 The attributes should be one of the following:

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

User Response: None.

CBC1633 Unable to allocate &1.

Explanation: Unable to allocate the dataset.

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

CBC1634 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.

CBC1635 Timestamp error on &1.

Explanation: Timestamp error while compiling a partitioned dataset.

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

CBC1636 Address of readonly attribute taken when the SOMNoDataDirect pragma is in effect for the class.

Where: &1 is a member name

Explanation: The address is taken of a data member &1 that is a readonly attribute, when the SOMNoDataDirect pragma is in effect for the class. If the address is used to modify the member, the compiler will attempt to use the ″__set″ method to access the member, which will result in a run-time error, because this method does not exist for the data member.

User Response: Rewrite the expression that causes the address to be taken, remove the readonly designation for the attribute, or remove the SOMNoDataDirect pragma.

CBC1637 Compiler does not supply volatile operator= functions for SOM classes.

Where: &1 is a class name

Explanation: An assignment was attempted to a volatile SOM object of type &1 for which no matching operator= could be found. The compiler supplies four operator= functions which are not qualified with volatile. In order to operate on volatile SOM objects, you must supply volatile versions of the member functions. It is recommended that you supply volatile versions of all four assignment operators.

User Response: Rewrite the expression that causes the assignment, remove the volatile qualifier for the SOM object, or supply volatile versions of the operator= functions for the SOM class.

CBC1638 The header file name in the #include directive cannot be empty.

User Response: Specify a non-empty header file name in the #include directive.

CBC1641 Direct access to &1 is valid only through the ″this″ pointer when NoDataDirect is in effect for the class.

Where: &1 is a member name

Explanation: A pointer or reference is used to directly access the instance data for non-attribute member &1 when NoDataDirect is in effect for the class. If the object is remote, this will result in a run-time error because the data is not locally accessible. When

NoDataDirect is in effect for the class, direct access to instance data, even within a member function, is valid only through the ″this″ pointer, because then the object is guaranteed to be local.

User Response: Rewrite the expression that references the data, designate the data member as an attribute using the SOMAttribute pragma, or remove NoDataDirect for the class.

CBC1642 #&1 condition evaluates to &2.

Where: &2 is an integer value

Explanation: This message traces preprocessor expression evaluation.

User Response: No response.

CBC1643 defined(&1) evaluates to &2.

Where: &2 is an integer value

Explanation: This message traces preprocessor #ifdef and #ifndef evaluation.

User Response: No response.

CBC1644 Stop skipping tokens.

Explanation: This messages traces conditional compilation activity.

User Response: No response.

CBC1645 File &1 has already been #included.

Where: &1 is the name of a file

Explanation: This #include directive is redundant.

User Response: Remove the #include directive.

CBC1646 #include found file &1.

Where: &1 is the name of a file

Explanation: This message traces the activity of the #include directive.

User Response: No response.

CBC1647 #line directive changing line to &1 and file to &2.

Where: &2 is a file name

Explanation: Traces #line directive evaluation.

User Response: No response.

CBC1648 #line directive changing line to &1.

Where: &1 is an integer value

Explanation: Traces #line directive evaluation.

User Response: No response.

CBC1649 The macro definition will override the keyword ″&1″.

Where: &1 is an identifier name

Explanation: Overriding a C keyword with a preprocessor macro may cause unexpected results.

User Response: Change the name of the macro if necessary.

CBC1650 Some program text not scanned due to &1 option or #pragma &2.

Where: &2 is the name of the margins or sequence pragma

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.

CBC1651 Macro &1 redefined with identical definition.

Where: &1 is the name of a macro

Explanation: Identical macro redefinitions are allowed but not necessary. The amount of whitespace separating tokens have no bearing on whether macros are considered identical.

User Response: Remove the identical definition if necessary.

CBC1652 #&1 nesting level is &2.

Where: &2 is an integer value

Explanation: Traces conditional compilation activity.

User Response: No response.

CBC1653 Compiler internal name ″&1″ has been defined as a macro.

Where: &1 is the name of a macro

Explanation: Internal compiler names should not be redefined.

User Response: Delete the macro definition or change the name of the macro being defined.

CBC1654 Compiler internal name ″&1″ has been undefined as a macro.

Where: &1 is the name of a macro

Explanation: Internal compiler names should not be undefined.

User Response: Delete the undefined macro.

CBC1655 Begin skipping tokens.

Explanation: This messages traces conditional compilation activity.

User Response: No response.

CBC1656 A trigraph sequence occurred in a character literal.

Explanation: The trigraph sequence will be converted, although a literal interpretation may have been desired.

User Response: Change the value of the character literal if necessary.

CBC1657 A trigraph sequence occurred in a string literal.

Explanation: The trigraph sequence will be converted, although a literal interpretation may have been desired.

User Response: Change the value of the string literal if necessary.

CBC1658 #undef undefining macro name ″&1″.

Where: &1 is the name of a macro

Explanation: Traces #undef preprocessor directive evaluation.

User Response: No response.

CBC1659 Unknown macro name ″&1″ on #undef directive.

Where: &1 is the name of a macro.

Explanation: An attempt is being made to undefine a macro that has not been previously defined.

User Response: Remove the #undef directive.

CBC1660 Header &1 included again because it is never empty.

Where: &1 is the name of a header file.

Explanation: The referenced header file has already been #included and will be physically #included again because there is no conditional compilation path in it which results in an empty file.

User Response: If desired, at the top of the header, test a macro name which is defined by the header to prevent subsequent inclusions.

CBC1661 Header &1 not included again because it is empty.

Where: &1 is the name of a header file.

Explanation: The referenced header file has already been #included and will not be physically #included again because it is empty.

User Response: If desired, do not #include the header since it is empty.

CBC1662 Header &1 included again because conditional compilation analysis is incomplete.

Where: &1 is the name of a header file.

Explanation: The referenced header file has already been #included and will be physically #included again because the inclusion is recursive and the conditional compilation analysis of the header is therefore incomplete.

User Response: If desired, test a macro name which is defined by the header at the point of inclusion to prevent subsequent inclusions.

CBC1663 Header &1 not included again because it would have no effect due to conditional compilation.

Where: &1 is the name of a header file.

Explanation: The referenced header file has already been #included and will not be physically #included again because conditional compilation would expose no additional source to the compiler.

User Response: If desired, do not #include the header since it is redundant.

CBC1664 End of precompiled header processing.

Explanation: The compiler has finished processing a precompiled header.

User Response: No response. This message merely traces the activity of the precompiled header feature.

CBC1665 Macro ″&1″ is required by the precompiled header and is defined differently than when the precompiled header was created.

Where: &1 is the name of a macro

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

CBC1666 One or more assertions are defined which were not defined when the precompiled header was created.

Where: &1 is the name of an assertion.

Explanation: An assertion is defined which was not defined when the precompiled header was generated. Since 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.

CBC1667 One or more macros are defined which were not defined when the precompiled header was created.

Explanation: A macro is defined which was not defined when the precompiled header was generated. Since 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.

CBC1668 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.

CBC1669 Assertion ″&1″ is required by the precompiled header and is not defined.

Where: &1 is the name of an assertion.

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.

CBC1670 Macro ″&1″ is required by the precompiled header and is not defined.

Where: &1 is the name of a macro

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.

CBC1671 Unable to use precompiled header &1.

Where: &1 is the name of a header file.

Explanation: The precompiled header can not be used for this compilation. A subsequent message will explain the reason.

User Response: Correct the problem indicated by the subsequent message.

CBC1672 Expecting &1 and found &2.

Where: &2 is the name of a header file.

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.

CBC1673 Syntax error - the argument list in the new placement syntax is empty.

Explanation: At least one new placement argument must be specified.

User Response: Specify an argument in the new placement syntax or remove the new placement.

CBC1674 Pointer to member declared using non-SOM class, but accessed through SOM object.

Explanation: The pointer to member was declared for a non-SOM class, but is being applied to a SOM object. This is likely to occur if a forward declaration for the class as non-SOM occurred, followed by the pointer to member declaration, followed by the actual class definition as a SOM class.

User Response: Specify the pointer to member declaration after the SOM class is defined or use pragma SOMAsDefault with the forward declaration of the class.

CBC1675 The __unaligned type qualifier is applicable only to the types that are referenced or ″pointed to″. It is not valid here and is ignored.

Explanation: The __unaligned type qualifier specifies that the symbol accessed through a pointer or a reference is not naturally aligned.

User Response: Remove the __unaligned qualifier.

CBC1676 An expression of type ″&1″ cannot be an operand for dynamic_cast because ″&2″ is not a class, struct or union.

Where: &1 is the type of the operand. &2 is the class name that the &1 points to (or is an lvalue of).

Explanation: The expression operand of a dynamic_cast operator must be a pointer to or an lvalue of a complete class.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1677 The operand is not a pointer type.

Explanation: The expression operand of a dynamic_cast operator must be a pointer when the target type is a pointer.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1678 The type ″&1″ is not allowed as the target type of the dynamic_cast operator.

Where: &1 is the target type.

Explanation: The target type of a dynamic_cast operator must be a pointer or reference to a complete class or a pointer to void.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1679 The type ″&1″ is not allowed as the target type of the dynamic_cast operator.

Where: &1 is the target type.

Explanation: The target type of a dynamic_cast operator must be a pointer to a complete class or a pointer to void when the operand is a pointer type.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1680 The type ″&1″ is not allowed as the target type of the dynamic_cast operator.

Where: &1 is the target type.

Explanation: The target type of a dynamic_cast operator must be a reference to a complete class when the operand is an lvalue.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1681 The type of the operand is ″&1″ but ″&2″ is not a polymorphic class.

Where: &1 is the type of the operand. &2 is the class name that the &1 points (or refers) to.

Explanation: One may only dynamic cast to a non-base class from a polymorphic class. A polymorphic class is a class that has a virtual function or that has a polymorphic base class.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1682 The target type has less qualification than the source type.

Explanation: The target type must have the same type qualifiers (or more) as the source type. Note that dynamic_cast may not used to cast away ’const’.

User Response: Add qualifiers to the target type to match the source type.

CBC1683 The type ″&1″ is not allowed as the target type of the dynamic_cast operator because ″&2″ is incomplete.

Where: &1 is the type of the cast. &2 is the class name that the &1 points (or refers) to.

Explanation: The target type of a dynamic_cast operator must be a pointer or reference to a complete class.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1684 An expression of type ″&1″ cannot be an operand for dynamic_cast because ″&2″ is incomplete.

Where: &1 is the type of the operand. &2 is the class name that the &1 points (or refers) to.

Explanation: The expression operand of a dynamic_cast operator must be a pointer or reference to a complete class.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1685 The operand is not an lvalue.

Explanation: The expression operand of a dynamic_cast operator must be an lvalue if the target type is a reference.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1687 Function ″&1″ does not have any parameters before the '...' parameter. This is not legal in IDL.

Explanation: Functions in IDL must have at least one named parameter before a ... parameter.

User Response: Add a named parameter before the ... parameter.

CBC1688 ″&1″ is an IDL keyword or type defined by <somobj.idl>.

Explanation: The user has a variable or field name that conflicts with an IDL keyword or type.

User Response: Rename the variable.

CBC1689 Unable to create the type_info objects because of the improper type_info class definition.

Explanation: The user has a type_info class definition that conflicts with the standard.

User Response: Rename the user type_info class definition.

CBC1690 IDL name ″&1″ conflicts with a variable or type in the same scope.

Explanation: The name conflicts with a previous IDL name in this compilation unit.

User Response: Rename at least one name/type.

CBC1691 Expected one of 'on'/'pop'/'*' for ″&1″.

Explanation: This pragma specifies a SOM option setting when the 'on' parameter is supplied, and returns to the previous value when the 'pop' parameter is used. In addition, if the parameter appears inside a SOM class scope, you can use the '*' parameter to specify the setting only for the current class. No other parameter values are valid.

User Response: Replace the invalid parameter with one of 'on', 'pop', 'or' *.

CBC1692 Invalid abistyle parameter specified for pragma SOMAbiStyle.

Explanation: The abistyle parameter value specified with the SOMAbiStyle pragma must be one of 2, 3, ″2″, ″2+3″ or ″3″.

User Response: Replace the abistyle parameter with one of 2, 3, ″2″, ″2+3″ or ″3″.

CBC1693 ″&1″ was specified using SOMMigratedMethod, but it was introduced in the current class.

Where: &1 is a C++ function member name.

Explanation: SOMMigratedMethod must only be used for names introduced in a base class.

User Response: Remove the ″&1″ from the SOMMigratedMethod .

CBC1694 Cannot specify default function on PowerPC; #pragma weak ignored.

Explanation: On PowerPc, #pragma weak only takes one parameter, the weak function.

User Response: Before calling the weak function on PowerPC, check to be sure it's there.

CBC1695 Must specify default function on Intel; #pragma weak ignored.

Explanation: On Intel, #pragma weak takes two parameters, the weak function and the default one.

User Response: On Intel, a default function must be provided for use when the weak function is not linked in.

CBC1696 Use option -Fb* to generate browser information for symbols in system include files

Explanation: You have definition for a symbol from a System Include file. If you want browser information for all symbols, use the -Fb* option.

User Response: Ignore the message, or recompile using the -Fb* option.

CBC1697 &1 cannot be casted to &2 because &3 is a SOM class but the other type is not.

Where: &1 is the type of the source expression &2 is the cast type &3 is the name of the SOM class

Explanation: The source and target class types must both be SOM classes or they must both be non-SOM classes.

User Response: Use the static_cast or reinterpret_cast operator instead of the dynamic_cast operator.

CBC1698 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″.

CBC1699 A call to the thread object's destructor may not be invoked if the current process ends before all of its threads end.

Explanation: You have declared a thread object with a destructor. Destructor calls of thread local storage objects are not fully supported on NT.

User Response: Allow enough time to the process so that its threads can end gracefully before the process terminates.

CBC1700 ″&1″ keyword is not supported on this platform. Keyword is ignored.

Explanation: A keyword has been specified on a platform that does not support it.

User Response: Remove the keyword.

CBC1701 The ″&1″ qualifier cannot be applied to thread object ″&2″.

Where: &2 is a name of a thread object

Explanation: The qualifier is being applied to an object with __thread attribute for which the qualifier is not valid.

User Response: Remove the qualifier.

CBC1702 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.

CBC1703 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.

CBC1704 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″.

CBC1705 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.

CBC1706 The source file is a CMS file. The suboption should specify a CMS file or a BFS file in an existing directory.

CBC1707 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.

CBC1708 The source file is a BFS directory. The suboption should specify an existing BFS directory.

CBC1709 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.

CBC1710 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.

CBC1711 The source file is a PDS. The suboption should specify a PDS or an existing HFS directory.

CBC1712 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.

CBC1713 The source file is a HFS directory. The suboption should specify an existing HFS directory.

CBC1714 Detected &1 : &2

Where: &1 the LE message number, &2 is the text of the CEE message.

Explanation: An LE informational message was detected while parsing #pragma runopts options.

CBC1715 Detected &1 : &2

Where: &1 the LE message number, &2 is the text of the CEE message.

Explanation: An LE warning message was detected while parsing #pragma runopts options.

CBC1716 Cannot generate IDL reference to ″&1″ because it is nested within a non-SOM class.

Where: &1 is a class name.

Explanation: C++ classes that are nested within non-SOM classes are not generated as types in the IDL file. Such classes cannot be explicitly referenced in the generated IDL file.

User Response: Remove the reference to the class or redefine the class so that it is not nested within a non-SOM class.

CBC1717 Pragma cannot be processed due to compiler error. Pragma is ignored.

Explanation: The compiler detected an error while processing pragma directive and cannot recover. The pragma will be ignored.

User Response: Contact your IBM Representative.

CBC1718 Data members cannot follow zero-sized array.

Explanation: The zero-sized array must be the last member in the class.

User Response: Make the zero-sized array the last member of the class

CBC1719 class ″&1″ cannot be used base class because it contains a zero-sized array.

Where: &1 is a C++ class name

Explanation: Using a class with a zero-sized array as a base class will result in members being added after the array.

User Response: Either remove the zero-sized array from the base class or re-structure the class hierarchy so the base class(es) don’t include this base class.

CBC1720 Expected text ″&1″ was not encountered on option ″&2″.

Explanation: A syntax error was detected while processing the sub-option. A token was expected but not found. The suboption is ignored.

User Response: Use the correct syntax for specifying the option

CBC1721 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

CBC1722 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.

CBC1723 #pragma &1 is ignored because the locale compiler option is not specified.

Where: &1 pragma name

Explanation: The locale compiler option is required for #pragma &1

User Response: Remove all the #pragma &1 directives or specify the locale compiler option.

CBC1724 #pragma filetag is ignored because the conversion table from &1 to &2 cannot be opened.

Where: &1 locale name, &2 locale name.

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.

CBC1725 Error messages are not converted because the conversion table from &1 to &2 cannot be opened.

Where: &1 locale name, &2 locale name.

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.

CBC1726 No conversion for character &1 because it does not belong to the input code set &2.

Where: &1 a character, &2 locale name.

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.

CBC1727 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.

CBC1728 Only conversion tables that map single byte characters to single byte characters are supported.

Explanation: Compiler is expecting a 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 contains only single byte to single byte mappings.

CBC1729 Invalid conversion descriptor was encountered during the conversion of the source line.

Explanation: No conversion was performed because conversion descriptor is not valid.

CBC1730 #pragma &1 must appear as the first directive before any source code.

Where: &1 pragma name.

User Response: Place this #pragma as the first directive before any code.

CBC1731 Function linkage differs from that of overridden function ″&1″.

Explanation: The linkage of a virtual function must agree with the linkage of base class member functions that it overrides.

User Response: Change the linkage keyword to agree with the base class method.

CBC1732 ″&1″ linkage cannot be specified for a virtual function.

Explanation: Virtual functions may not have 16 bit or Pascal linkage.

User Response: Remove or replace the linkage modifier.

CBC1733 Unexpected end of line encountered.

Explanation: A statement on this line is incomplete.

User Response: Either finish the incomplete statement or remove it.

CBC1737 The pathname of SYSPATH or USERPATH compiler option specified is longer than 44 characters. ″&1″ pathname is ignored.

Explanation: The pathname of the compiler options SYSPATH or USERPATH must not be longer than 44 characters. (max member length).

User Response: Shorten the pathname.

CBC1738 The pathname of SYSPATH or USERPATH compiler option specified is invalid. ″&1″ pathname is ignored.

Explanation: The pathname of the compiler options SYSPATH or USERPATH is invalid.

User Response: See the ″Users Guide″ for the restrictions of valid pathname.

CBC1739 Attempting to pop an empty alignment stack. Current alignment may change.

Explanation: Alignment stack is empty. New packing value is set to either the alignment specified for this pragma or the default alignment for this module.

User Response: Remove 'POP' operation, or ensure alignment stack has been set up correctly.

CBC1740 Identifier does not exist in the alignment stack. Current alignment may change.

Explanation: Identifier does not exist in alignment stack. New Packing value is set to either the alignment specified for the pragma or the default alignment for the module.

User Response: Remove identifier, or ensure alignment stack has been set up correctly.

CBC1742 Csect option is ignored due to naming error.

Explanation: The compiler was unable to generate valid csect names.

User Response: Use the #pragma csect to name the code and static control sections.

CBC1743 The divisor for the division operator cannot be zero.

User Response: Change the expression used in the divisor.

CBC1744 The pragma is accepted by the compiler. The pragma will have no effect.

Explanation: The pragma is not supported by this compiler.

User Response: Change or remove the #pragma directive.

CBC1745 &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.

CBC1748 ″&1″ was declared as ″&2″, but is now declared as ″&3″. Export is assumed.

Where: &1 is the name being declared &2 is either _Import or _Export &3 is the other one.

Explanation: The declaration conflicts with a previous declaration of the same name.

User Response: Change one of the names or eliminate one of the declarations.

CBC1749 ″&1″ was previously declared as ″&2″, but is now defined. Export is assumed.

Explanation: Defined symbols cannot be imported. The compiler will assume that you want to export this symbol rather than import it.

CBC1750 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.

CBC1751 ″&1″ is not defined in this compilation and cannot be used in pragma noinline directive.

Where: &1 is a function name and type.

Explanation: Only functions defined in this compilation can be used in pragma noinline directive.

User Response: Remove the pragma noinline directve or define the function in this compilation unit.

CBC1752 The physical size of an array is too large.

Explanation: The compiler cannot handle any size which is too large to be represented internally.

User Response: Reduce the size of the array.

CBC1753 The physical size of a struct or union is too large.

Explanation: The compiler cannot handle any size which is too large to be represented internally.

User Response: Reduce the size of the struct or union members.

CBC1754 The static storage is too large.

Explanation: The compiler detected an static storage declaration that has a constant greater than 16773104.

User Response: Change the storage size to an integral constant expression less then or equal to 16773104.

CBC1755 #include searching for file &1.

Explanation: The preprocessor is searching for the specified include file.

CBC1756 The ″&1″ qualifier is not supported on the target platform.

Explanation: A qualifier has been specified on a platform that does not support it.

User Response: Remove the qualifier.

CBC1757 The main function, ″&1″, cannot be overloaded.

Explanation: The user attempted to declare or define a function that overloads the name of the main function.

User Response: Change the name of the function being declared or defined.

CBC1758 ″&1″ is an IDL keyword or type already defined by <somobj.idl>.

Explanation: The user has defined a type name that conflicts with an IDL keyword or type. Type definition is ignored for IDL generation.

User Response: Rename or remove the type definition.

CBC1759 The array bound is too large.

Explanation: The array bound should be a value less than or equal to max int.

User Response: Reduce the number of elements in the array.

CBC1760 ″&1″ was not specified in the previous declaration of ″&2″.

Where: &1 is an attribute. &2 is a name.

Explanation: An attribute has been specified that conflicts with the previous declaration of a name.

User Response: Remove the attribute.

CBC1761 Record truncated while writing to IDL file.

Explanation: A record being written to the IDL file has been truncated because the IDL file (DD:SYSUT15) has a record length too short to hold the record being written.

User Response: Redefine DD:SYSUT15 with a longer record length.

CBC1762 Pragma ″&1″ is not supported on the target platform. It is ignored.

Explanation: A pragma has been specified on a platform that does not support it.

User Response: Remove the pragma.

CBC1763 Suboption ″&1″ for option ″&2″ is not supported for C++ programs. Suboptions is ignored.

Where: &1 is a suboption &2 is an option

Explanation: The command line has an option with a suboption that is not supported in the C++ language.

User Response: Change/remove the suboption.

CBC1764 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.

CBC1765 Pragma import is not supported on the target platform, the _Import keyword should be used instead.

Explanation: #pragma import is not supported. The _Import keyword should be used in the symbol declaration.

User Response: Remove #pragma import and add the _Import keyword to the symbol declaration.

CBC1766 Variable ″&1″ needs an explicit ″__thread″ specifier if its initializer is process dependent.

Explanation: This variable was assumed to be sharable because it was declared ″const″, but it is dynamically initialized. If that initialization may yield different values in different processes, the variable should be declared with the ″__thread″ specifier.

User Response: Add ″__thread″ to the declaration if required.

CBC1767 The ″&1″ feature of OS/390 is not enabled. Contact your system programmer.

Explanation: This feature of OS/390 is not enabled at your installation. Your system programmer can contact IBM OS/390 service to have this element enabled. d

CBC1768 Compiling ″&1″.

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

CBC1769 The path &1 does not exist. Please create the directory and recompile.

Explanation: The path shown does not exist. The compiler will only perform output to existing HFS directories.

CBC1770 The name &1 is invalid. Please correct and recompile.

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

CBC1771 The memory required for precompiled header processing is not available.

Explanation: To generate (GENPCH) or use (USEPCH) a precompiled header, a sufficiently large and contiguous memory space must be available. Additionally, to use a precompiled header (USEPCH), the same memory address range that was obtained when the precompiled header was generated must be available. These conditions could not be satisfied and precompiled header processing has been stopped. Compilation will continue.

User Response: This situation can be caused by either insufficient memory or the required memory address range not being available.

CBC1772 The preprocessor macro ″&1″ was expanded inside a pragma directive.

Explanation: A preprocessor macro was expanded inside a pragma directive. Please ensure that this is the desired result.

User Response: Please ensure that the macro is intended for expansion.

CBC1773 The ″&1″ keyword is not supported on the target platform.

Explanation: A keyword has been specified on a platform that does not support it.

User Response: Remove the keyword.

CBC1774 The ″&1″ keyword is not supported on the target platform.

Explanation: A keyword has been specified on a platform that does not support it.

User Response: Remove the keyword.

CBC1775 ″&1″ must specify the name of a C++ class or struct that is not a template.

Where: &1 is the SOM pragma ″#pragma SOMModule″

Explanation: The supplied name either does not represent a C++ class or struct visible in the current scope or represents a template class.

User Response: Ensure that you use the correct C++ class name.

CBC1776 Using '*' in ″&1″ is only valid within the definition of a named C++ class or struct which is not a template.

Where: &1 is the SOM pragma ″#pragma SOMModule″

Explanation: The pragma is specified outside of a C++ class or struct definition, or the enclosing scope does not represent a C++ class or struct that is not a template.

User Response: Move the pragma inside the correct class definition.

CBC1777 ″&1″ must not be specified for ″&2″.

Where: &1 is the SOM pragma ″#pragma SOMModule″ &2 is class name.

Explanation: The pragma can only be specified inside a class definition.

User Response: Move the pragma inside the correct class definition.

CBC1778 SOM module class ″&1″ must not have base classes and it must not be used as a base class.

Where: &1 is class name.

Explanation: A SOM module class is mapped into an IDL module. Since an IDL module denotes a scope for its containing identifiers and is not a class, it must not be used to build a class hierarchy.

User Response: Change the class to have no bases and make sure that it has no derived classes.

CBC1779 SOM module class ″&1″ must contain only nested types and classes.

Where: &1 is class name.

Explanation: A SOM module class is mapped into an IDL module. An IDL module must not have member functions, data members, or C++ friend declarations.

User Response: Correct the pragma or the class.

CBC1780 SOM module class ″&1″ must not be imported or exported.

Where: &1 is class name.

Explanation: A SOM module class is mapped into an IDL module. IDL modules are significant only in SOM context. The ″_Import″ and ″_Export″ keywords do not apply.

User Response: Remove the ″_Import″ or ″_Export″ keyword.

CBC1781 SOM module class ″&1″ must be nested in another SOM module class or defined at file scope.

Where: &1 is class name.

Explanation: A SOM module class is mapped into an IDL module. An IDL module can only be nested inside another IDL module or defined at file scope.

User Response: Correct the pragma or the nesting of classes.

CBC1782 SOM module class ″&1″ must not be used as a type.

Where: &1 is class name.

Explanation: A SOM module class is mapped into an IDL module. Since an IDL module denotes a scope for its containing identifiers and is not a class, it cannot be used as a type.

User Response: Do not use the SOM module class as a type.

CBC1783 SOM module class ″&1″ must not be forward declared.

Where: &1 is class name.

Explanation: A SOM module class must not have been previously forward declared.

User Response: Remove the forward declaration of the SOM module class.

CBC1784 ″&1″ cannot derive from ″&2″ because of conflicting object models.

Where: ″&1″ and ″&2″ are the class names.

Explanation: A class is derived from another class whose object model is not the same as the derived class.

User Response: Change the object model of the base or the derived class.

CBC1785 Suboption ″&1″ for option ″&2″ is not supported on the target platform. The option is ignored.

Where: ″&1″ is a suboption ″&2″ is an 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.

CBC1786 Argument ″&1″ for pragma ″&2″ is not supported on the target platform. Pragma is ignored.

Where: ″&1″ is a suboption ″&2″ is an option

Explanation: The pragma has been specified with an argument that is not supported on the target platform.

User Response: Remove the pragma or ignore this message.

CBC1787 The ″%%″ and ″%%%%″ digraphs will be obsolete in the next release of this product. Please use ″%:″ and ″%:%:″ instead.

Explanation: The ″%%″ and ″%%%%″ digraphs will not be supported in the next release. Please use the new digraphs ″%:″ and ″%:%:″.

User Response: Replace the old digraphs with the new digraphs.

CBC1788 Ambiguous reference to ″&1″, declared in SOM base classes ″&2″ and ″&3″. In C++ this is an error.

Where: &3 is a C++ class name

Explanation: The derived SOM class made a reference to a member that is declared in more than one of its SOM base classes. SOM function call mechanism will choose which function to invoke.

User Response: Change one of the names, or always fully qualify the name.

CBC1789 The virtual functions ″&1″ and ″&2″ are ambiguous since they override the same function in virtual SOM base class ″&3″.

Where: &1 is a function name and type &2 is a function name and type

Explanation: The two functions are ambiguous. SOM function call mechanism will choose which function to invoke.

User Response: Remove one of the virtual functions.

CBC1790 Instances of SOM class ″&1″ will not inherit more than one sub-object of base class ″&2″.

Where: &2 is a C++ name.

Explanation: All non virtual base classes of a SOM class appear only once in the class hierarchy.

User Response: Make the base class a virtual base class.

CBC1791 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.

CBC1792 Timestamp information is not available for #include ″&1″.

Where: &1 is the name of an include file from the #include directive.

Explanation: Precompiled header processing requires that include files have timestamp information. Partitioned datasets can have timestamps although not all commands and utilities will create or maintain timestamps. The EDIT command can be used to add timestamp information to partitioned dataset members which do have timestamps. Sequential datasets do not have timestamps and should not be used for include files when GENPCH or USEPCH options are specified. When the GENPCH option is specified the timestamp information of include files is stored in the generated precompiled header. When the USEPCH option is specified the stored timestamps are used to determine if any include files have been modified since the precompiled header was generated.

User Response: Avoid using sequential datasets for include files and ensure partitioned dataset members have timestamp information.

CBC1793 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.

CBC1794 The external name &1 must not conflict with the name in #pragma csect or the csect name generated by the compiler.

Explanation: A external name is the same as the name defined in a pragma csect or the csect name generated by the compiler.

User Response: Change either the external name or the csect name.

CBC1795 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.

CBC1796 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.

CBC1797 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.

CBC1798 There is more than one #pragma csect statement.

Explanation: A duplicate #pragma csect is ignored.

User Response: Remove the duplicate #pragma csect statement.

CBC1799 An error occurred when attempting to open the &1 file &2.

Where: &2 a file name.

Explanation: The compiler received an error when attempting to open a file to support the indicated option. No information will be output to this file.

User Response: Correct the problem and compile again.

CBC1800 #&1 directive has no effect.

Where: &1 a preprocessor directive.

Explanation: A preprocessor directive has been specified that has no effect.

User Response: Remove the preproccessor directive.

CBC1801 Attempting to pop an empty enum stack. Pragma is ignored.

Explanation: Enum stack is empty. Size of enum is set to default value.

User Response: Remove ’pop’ or ’reset’ operation, or ensure enum stack has been set up correctly.

CBC1810 The suboption specified for the ″&1″ option is not allowed when the ″&2″ option is specified.

Where: &1 and &2 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.

CBC1820 Invalid syntax for pragma ″&1″, ″&2″ assumed.

Explanation: The compiler encountered a pragma with invalid syntax. The message identifies the compiler’s recovery action.

User Response: Correct or remove the pragma.

CBC1822 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.

CBC1823 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.

CBC1824 The dynamic_cast operator is not supported.

Explanation: The dynamic_cast operator is not supported in this version of the C++ compiler. The dynamic_cast operation is ignored.

User Response: Change the program to remove all uses of or dependencies upon the dynamic_cast operator.

CBC1825 The parameter type &1 is not valid for a function of this linkage type.

Explanation: Long long integers are not valid for COBOL and PLI linkage types.

User Response: Update the parameter to a type that may be used by this linkage type.

CBC1826 The enum is too large to fit into the requested type &1.

Explanation: The enum type is too large to fit in the storage requested with the qenum option.

User Response: Try using a different qenum setting.

CBC1827 Invalid syntax for pragma ″&1″, pragma ignored.

Explanation: The compiler encountered a pragma with invalid syntax. The pragma is ignored.

User Response: Correct or remove the pragma.

CBC1831 extern ″&1″ is not a supported linkage in this environment.

Where: &1 is a string

Explanation: The linkage string in a linkage declaration is not one of the linkages supported by this setting of the environment.

User Response: Change the linkage string to a supported value.

CBC1832 ″&1″ is not defined in this compilation and cannot be used in pragma ″&2″ directive.

Where: &1 is an identifier name. &2 is the name of the #pragma.

Explanation: Only functions defined in this compilation can be used in the pragma directive.

User Response: Define the identifier or remove the #pragma.

CBC1833 Csect name ″&1″ has been truncated to ″&2″.

Explanation: The static, data and test csect names have been truncated to 8 characters.

CBC3001 INTERNAL COMPILER ERROR: Procedure &1.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your VisualAge for C++ Service Representative.

CBC3002 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.

CBC3003 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.

CBC3004 #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.

CBC3005 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.

CBC3006 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.

CBC3007 ″&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.

CBC3008 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.

CBC3009 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.

CBC3010 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.

CBC3012 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.

CBC3013 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.

CBC3014 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.

CBC3017 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.

CBC3018 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.

CBC3019 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.

CBC3020 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.

CBC3021 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.

CBC3022 ″&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.

CBC3023 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.

CBC3025 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.

CBC3026 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.

CBC3027 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.

CBC3028 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.

CBC3029 #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.

CBC3030 Identifier &1 cannot be redeclared.

Explanation: The identifier has already been declared.

User Response: Remove one of the declarations.

CBC3031 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.

CBC3032 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.

CBC3033 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.

CBC3034 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.

CBC3035 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.

CBC3036 Range error.

Explanation: The value is outside of the valid range.

User Response: Change value to be within the required limits.

CBC3037 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.

CBC3039 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.

CBC3041 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.

CBC3043 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.

CBC3044 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.

CBC3045 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.

CBC3046 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.

CBC3047 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.

CBC3048 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.

CBC3049 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.

CBC3050 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.

CBC3051 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.

CBC3052 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.

CBC3053 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.

CBC3054 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..

CBC3055 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.

CBC3056 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.

CBC3057 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.

CBC3058 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.

CBC3059 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.

CBC3062 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.

CBC3067 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.

CBC3068 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.

CBC3070 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.

CBC3073 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.

CBC3076 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.

CBC3077 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.

CBC3078 #&1 directive has no effect.

Explanation: A preprocessor directive has been specified that has no effect.

User Response: Remove the preproccessor directive.

CBC3085 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.

CBC3095 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.

CBC3098 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.

CBC3099 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.

CBC3103 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.

CBC3104 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.

CBC3108 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.

CBC3112 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.

CBC3115 Duplicate type specifier ″&1″ ignored.

Explanation: A duplicate type specifier appears in the type declaration.

User Response: Remove one of the duplicate type specifiers.

CBC3117 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.

CBC3119 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.

CBC3120 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.

CBC3122 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.

CBC3127 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.

CBC3131 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.

CBC3134 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.

CBC3137 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.

CBC3152 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.

CBC3155 Option &1 requires suboption(s).

Explanation: The option is not completely specified; a suboption is required.

User Response: Add a suboption.

CBC3159 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.

CBC3160 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.

CBC3162 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.

CBC3164 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.

CBC3166 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.

CBC3167 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.

CBC3168 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.

CBC3169 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.

CBC3170 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.

CBC3172 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.

CBC3173 Option &1 is not recognized.

Explanation: An invalid option was specified.

User Response: Correct the spelling of the option.

CBC3174 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.

CBC3175 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.

CBC3176 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.

CBC3177 Type combination is not valid.

CBC3178 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.

CBC3180 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.

CBC3181 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.

CBC3182 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.

CBC3183 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.

CBC3184 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.

CBC3185 #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.

CBC3186 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.

CBC3188 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.

CBC3189 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.

CBC3190 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.

CBC3191 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.

CBC3192 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.

CBC3193 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.

CBC3194 Incomplete type is not allowed.

Explanation: Except for pointers, you cannot declare an object of incomplete type.

User Response: Complete the type declaration.

CBC3195 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.

CBC3196 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.

CBC3197 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.

CBC3198 #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.

CBC3199 #&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.

CBC3200 #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.

CBC3201 #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.

CBC3202 #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.

CBC3204 Unexpected end of file.

Explanation: The end of the source file has been encountered prematurely.

User Response: Check for misplaced braces.

CBC3205 &1

Explanation: The #error directive was encountered. Compilation terminated.

User Response: Recompile with correct macro definitions.

CBC3206 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.

CBC3207 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>.

CBC3208 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.

CBC3209 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.

CBC3210 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.

CBC3211 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.

CBC3212 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.

CBC3213 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.

CBC3215 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.

CBC3218 Unknown preprocessing directive #&1.

Explanation: An unrecognized preprocessing directive has been encountered.

User Response: Check the spelling and syntax or remove the directive.

CBC3219 #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.

CBC3220 #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.

CBC3221 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.

CBC3224 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.

CBC3225 Error reading file &1. (&2)

User Response: Ensure that the file exists and that the compiler can access it.

CBC3226 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.

CBC3229 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.

CBC3231 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.

CBC3232 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.

CBC3234 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.

CBC3235 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.

CBC3236 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.

CBC3238 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.

CBC3242 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.

CBC3243 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.

CBC3244 External variable &1 cannot be redefined.

Explanation: An attempt was made to redefine an external variable.

User Response: Remove the redefinition.

CBC3245 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.

CBC3246 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.

CBC3247 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.

CBC3248 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.

CBC3249 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.

CBC3250 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.

CBC3251 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.

CBC3255 #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.

CBC3258 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.

CBC3260 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.

CBC3261 Suboption &1 is not valid for option &2.

Explanation: An invalid suboption was specified for some option.

User Response: Change the suboption.

CBC3262 #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.

CBC3263 #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.

CBC3264 #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.

CBC3266 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.

CBC3267 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.

CBC3268 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.

CBC3269 A char array pointer cannot be assigned to a nonchar pointer.

CBC3270 A wide char array pointer cannot be assigned to a nonwide char pointer.

CBC3271 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.

CBC3272 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.

CBC3273 Missing type in declaration of &1.

Explanation: A declaration was made without a type specifier.

User Response: Insert a type specifier into the declaration.

CBC3274 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.

CBC3275 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.

CBC3276 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.

CBC3277 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.

CBC3278 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.

CBC3279 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.

CBC3280 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.

CBC3281 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.

CBC3282 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.

CBC3283 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.

CBC3285 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.

CBC3286 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.

CBC3287 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.

CBC3288 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.

CBC3289 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.

CBC3290 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.

CBC3291 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.

CBC3292 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.

CBC3293 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.

CBC3294 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.

CBC3295 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.

CBC3296 #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.

CBC3297 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.

CBC3298 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.

CBC3299 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.

CBC3300 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.

CBC3301 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.

CBC3302 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.

CBC3303 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.

CBC3304 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.

CBC3306 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.

CBC3307 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.

CBC3308 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.

CBC3309 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.

CBC3310 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.

CBC3311 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.

CBC3312 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.

CBC3313 Compiler internal name &1 has been undefined as a macro.

Explanation: Do not redefine internal compiler names.

User Response: Remove the macro undefinition.

CBC3314 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.

CBC3320 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.

CBC3321 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.

CBC3322 Illegal multibyte character &1.

Explanation: The multibyte character specified is not valid.

User Response: Correct the multibyte character.

CBC3323 ″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.

CBC3324 ″&1″ cannot be converted to ″&2″.

Explanation: The cast between the two types is not allowed.

User Response: Remove the cast.

CBC3327 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.

CBC3328 ″″&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.

CBC3329 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.

CBC3332 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.

CBC3334 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.

CBC3335 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.

CBC3339 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.

CBC3341 #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.

CBC3342 ″″/*″ 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.

CBC3343 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.

CBC3344 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.

CBC3345 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.

CBC3346 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.

CBC3347 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.

CBC3348 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.

CBC3349 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.

CBC3350 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.

CBC3351 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.

CBC3352 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.

CBC3356 Compilation unit is empty.

Explanation: There is no code in the compilation unit.

User Response: Ensure the correct source file is specified. Recompile.

CBC3357 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.

CBC3358 ″&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.

CBC3359 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.

CBC3360 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.

CBC3361 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.

CBC3362 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.

CBC3363 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.

CBC3364 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.

CBC3366 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.

CBC3367 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.

CBC3369 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.

CBC3374 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.

CBC3376 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.

CBC3377 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.

CBC3378 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 ″...″.

CBC3379 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.

CBC3380 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.

CBC3381 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.

CBC3382 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.

CBC3383 Expecting ″&1″ to be an external identifier.

Explanation: The identifier must have external linkage.

User Response: Change the storage class to extern.

CBC3384 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.

CBC3387 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().

CBC3388 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.

CBC3389 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.

CBC3390 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.

CBC3393 &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.

CBC3394 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.

CBC3395 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″.

CBC3396 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.

CBC3397 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.

CBC3398 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.

CBC3399 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.

CBC3400 #pragma &1 must appear only once in any C file.

User Response: Remove all but one of the specified #pragma directives.

CBC3401 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.

CBC3402 &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.

CBC3408 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.

CBC3409 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.

CBC3410 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.

CBC3411 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.

CBC3412 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.

CBC3413 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.

CBC3414 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.

CBC3415 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.

CBC3416 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.

CBC3417 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.

CBC3418 '&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.

CBC3419 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.

CBC3420 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.

CBC3421 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.

CBC3422 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.

CBC3423 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.

CBC3424 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.

CBC3425 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.

CBC3426 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.

CBC3427 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.

CBC3428 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.

CBC3429 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.

CBC3430 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.

CBC3431 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.

CBC3432 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.

CBC3433 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.

CBC3434 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.

CBC3435 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.

CBC3436 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.

CBC3437 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.

CBC3438 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.

CBC3439 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.

CBC3440 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.

CBC3441 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.

CBC3442 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.

CBC3443 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.

CBC3444 The opening brace is redundant.

Explanation: The initialization expression contains extra, possibly unnecessary, braces.

User Response: Remove the extra braces.

CBC3445 The closing brace is redundant.

Explanation: The initialization expression contains extra, possibly unnecessary, braces.

User Response: Remove the extra braces.

CBC3446 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.

CBC3447 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.

CBC3448 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.

CBC3449 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.

CBC3450 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.

CBC3451 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.

CBC3452 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.

CBC3453 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.

CBC3454 &1 condition evaluates to &2.

Explanation: This message traces preprocessor expression evaluation.

User Response: No response required.

CBC3455 defined(&1) evaluates to &2.

Explanation: This message traces preprocessor #ifdef and #ifndef evaluation.

User Response: No response required.

CBC3456 Stop skipping tokens.

Explanation: This messages traces conditional compilation activity.

User Response: No response required.

CBC3457 File &1 has already been included.

Explanation: This #include directive is redundant.

User Response: Remove the #include directive.

CBC3458 #line directive changing line to &1 and file to &2.

Explanation: This message traces #line directive evaluation.

User Response: No response required.

CBC3459 #line directive changing line to &1.

Explanation: This message traces #line directive evaluation.

User Response: No response required.

CBC3460 &1 nesting level is &2.

Explanation: This message traces conditional compilation activity.

User Response: No response required.

CBC3461 Generating precompiled header file &1.

Explanation: This message traces precompiled header generation activity.

User Response: No response required.

CBC3462 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.

CBC3463 Using precompiled header file &1.

Explanation: This message traces precompiled header file generation activity.

User Response: No response required.

CBC3464 Begin skipping tokens.

Explanation: This messages traces conditional compilation activity.

User Response: No response required.

CBC3465 #undef undefining macro name &1.

Explanation: This message traces #undef preprocessor directive evaluation.

User Response: No response required.

CBC3466 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.

CBC3467 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.

CBC3468 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.

CBC3469 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.

CBC3470 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.

CBC3471 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.

CBC3472 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.

CBC3473 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.

CBC3474 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.

CBC3475 The real constant arithmetic expression folds to positive infinity.

Explanation: Constant folding results in an overflow.

User Response: Change the expression.

CBC3476 The real constant arithmetic expression folds to negative infinity.

Explanation: Constant folding results in an overflow.

User Response: Change the expression.

CBC3478 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.

CBC3479 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.

CBC3480 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.

CBC3481 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.

CBC3482 The opening brace before this point is redundant.

Explanation: The initialization expression contains extra, possibly unnecessary, braces.

User Response: Remove the extra braces.

CBC3483 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.

CBC3484 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.

CBC3485 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.

CBC3486 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.

CBC3487 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.

CBC3489 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.

CBC3490 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.

CBC3491 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.

CBC3492 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.

CBC3493 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.

CBC3494 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.

CBC3495 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.

CBC3496 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 Language Reference.

CBC3497 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.

CBC3498 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.

CBC3499 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.

CBC3503 Option ″&1″ is not supported for &2.

Explanation: The option specified is not supported on this operating system.

User Response: Remove the option.

CBC3505 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.

CBC3508 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.

CBC3509 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.

CBC3512 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.

CBC3513 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.

CBC3514 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.

CBC3515 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.

CBC3517 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.

CBC3518 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.

CBC3519 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.

CBC3520 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.

CBC3521 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.

CBC3522 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.

CBC3524 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.

CBC3531 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.

CBC3532 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.

CBC3533 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.

CBC3534 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.

CBC3535 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.

CBC3536 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.

CBC3537 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.

CBC3538 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.

CBC3539 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.

CBC3545 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.

CBC3546 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.

CBC3547 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.

CBC3548 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.

CBC3549 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.

CBC3550 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.

CBC3551 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.

CBC3552 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.

CBC3553 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.

CBC3554 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.

CBC3555 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.

CBC3556 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.

CBC3557 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.

CBC3558 #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.

CBC3559 #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.

CBC3560 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.

CBC3561 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.

CBC3562 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.

CBC3563 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.

CBC3564 Invalid conversion descriptor was encountered during the conversion of the source line.

Explanation: No conversion was performed because conversion descriptor is not valid.

CBC3565 #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.

CBC3566 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.

CBC3567 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.

CBC3568 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.

CBC3569 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.

CBC3570 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.

CBC3571 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.

CBC3572 Expected text &1 was not encountered on option &2.

User Response: Use the correct syntax for specifying the option

CBC3573 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.

CBC3574 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.

CBC3575 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.

CBC3576 Csect name &1 has been truncated to &2.

Explanation: The static, data and test csect names have been truncated to 8 characters.

CBC3577 Obsolete option OPTIMIZE(2) defaults to OPTIMIZE(1).

Explanation: Optimize(2) is no longer supported and has been defaulted to 1.

CBC3578 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.

CBC3585 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.

CBC3586 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.

CBC3600 3600 - 3631 are LE messages.

Explanation: Refer to the LE manuals for further information about these messages

CBC3671 The header file name in the #include directive cannot be empty.

User Response: Specify a non-empty header file name in the #include directive.

CBC3675 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.

CBC3676 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.

CBC3677 Option LONGNAME is turned on because option DLL is specified.

Explanation: Option LONGNAME is turned on by the compiler because DLL option is specified.

CBC3678 Option RENT is turned on because option DLL is specified.

Explanation: Option RENT is turned on by the compiler because DLL option is specified.

CBC3679 Option LONGNAME is turned on because option EXPORTALL is specified.

Explanation: Option LONGNAME is turned on by the compiler because EXPORTALL option is specified.

CBC3680 Option RENT is turned on because option EXPORTALL is specified.

Explanation: Option RENT is turned on by the compiler because EXPORTALL option is specified.

CBC3681 #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.

CBC3682 ″&1″ will not be exported because #pragma variable(&2,NORENT) is specified.

Explanation: Variables with NORENT option cannot be exported.

CBC3683 ″&1″ will not be exported because it does not have external storage class.

Explanation: Only objects with external storage class can be exported.

CBC3684 Exporting function main is not allowed.

Explanation: Main cannot be exported.

User Response: Remove the pragma export for main.

CBC3685 ″&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.

CBC3686 Unexpected keyword(s). One or more keywords were found in an invalid location.

Explanation: One or more keywords were found in an invalied location.

User Response: Remove the keyword(s) or place them immediately to the left of the identifier to which they apply.

CBC3687 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.

CBC3688 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.

CBC3689 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.

CBC3690 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.

CBC3691 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.

CBC3692 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″.

CBC3693 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.

CBC3694 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.

CBC3695 #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.

CBC3708 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.

CBC3709 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.

CBC3710 Option &1 ignored because option &2 specified.

CBC3711 Option &1 ignored.

CBC3712 Duplicate function specifier ″&1″ ignored.

CBC3713 Keyword ″&1″ is not allowed.

CBC3714 #include searching for file &1.

CBC3715 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.

CBC3717 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.

CBC3721 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.

CBC3722 #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.

CBC3723 #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.

CBC3724 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.

CBC3728 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.

CBC3729 ″&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″.

CBC3730 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.

CBC3731 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.

CBC3732 #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.

CBC3733 Processing #include file &1.

Explanation: This message traces #include file processing.

User Response: No response required.

CBC3735 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.

CBC3736 &1 conflicts with previous &2 declaration.

Explanation: The compiler cannot resolve the conflicting declarations.

User Response: Remove one of the declarations.

CBC3737 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.

CBC3739 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.

CBC3740 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.

CBC3741 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.

CBC3750 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.

CBC3751 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.

CBC3752 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.

CBC3754 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.

CBC3755 The &1 option is not supported in this release.

Explanation: The specified option is not supported in this release.

User Response: Remove the option.

CBC3763 Option &1 ignored because #pragma &2 is specified.

CBC3764 Option &1 ignored for variable &2 because #pragma &3 is specified.

CBC3765 &1 digits are required for the universal-character-name ″&2″.

CBC3766 The universal-character-name ″&1″ is not in the allowable range for an identifier.

CBC3767 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.

CBC3790 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.

CBC3791 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.

CBC3792 #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.

CBC3793 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.

CBC3794 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.

CBC3795 Private variable '&1' appears in the &2 clause.

Explanation: Private variable cannot appear in that clause.

User Response: Remove variable from the clause.

CBC3796 &1 construct cannot be nested within &2 construct.

Explanation: OpenMP constructs are incorrectly nested.

User Response: Correct the constructs.

CBC3797 &1 directive cannot appear within &2 construct.

Explanation: Directive is incorrectly nested.

User Response: Correct the directive.

CBC3798 Threadprivate variable '&1' appears in the &2 clause.

Explanation: Threadprivate variable cannot appear in that clause.

User Response: Remove variable from the clause.

CBC3799 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.

CBC3805 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.

CBC3810 #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.

CBC3811 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.

CBC3812 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.

CBC3813 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.

CBC3815 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.

CBC3862 Unable to read &1.

Where: &1 file *CHAR 100

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

CBC3863 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.

CBC3870 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 after truncation.

User Response: Reduce the length of the Program name. Or use #pragma map to map the Program name to a shorter name.

CBC3871 #pragma &1 ignored for function or typedef &2.

User Response: Check what has been specified before and remove the conflicts.

CBC5001 INTERNAL COMPILER ERROR: &1.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your Service Representative.

CBC5002 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.

CBC5003 &1.

Where: &1 is the detailed message text.

Explanation: General error message.

CBC5031 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.

CBC5032 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.

CBC5033 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.

CBC5034 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.

CBC5051 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.

CBC5052 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.

CBC5053 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.

CBC5054 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.

CBC5055 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.

CBC5057 &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.

CBC5101 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.

CBC5102 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.

CBC5103 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.

CBC5104 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.

CBC5105 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.

CBC5106 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.

CBC5107 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.

CBC5108 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.

CBC5109 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.

CBC5110 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.

CBC5111 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.

CBC5112 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.

CBC5113 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.

CBC5114 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.

CBC5115 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.

CBC5116 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.

CBC5117 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.

CBC5118 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.

CBC5119 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.

CBC5120 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.

CBC5121 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.

CBC5122 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.

CBC5123 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.

CBC5130 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.

CBC5131 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.

CBC5132 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.

CBC5141 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.

CBC6000 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.

CBC6001 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.

CBC6002 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.

CBC6003 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.

CBC6004 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.

CBC6005 ″&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.

CBC6010 ″&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.

CBC6011 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.

CBC6012 ″&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.

CBC6013 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.

CBC6020 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.

CBC6021 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.

CBC6022 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.

CBC6023 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

CBC6030 &1

Where: &1 is the detailed message text.

Explanation: General informational message.

CBC6031 &1

Where: &1 is the detailed message text.

Explanation: General warning message.

CBC6032 &1

Where: &1 is the detailed message text.

Explanation: General error message.

CBC6033 &1

Where: &1 is the detailed message text.

Explanation: General severe error message.

CBC6050 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.

CBC6051 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.

CBC6052 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.

CBC6053 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.

CBC6059 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.

CBC6060 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.

CBC6061 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.

CBC6062 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.

CBC6100 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.

CBC6101 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.

CBC6102 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.

CBC6103 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.

CBC6104 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.

CBC6105 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.

CBC6106 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.

CBC6107 ″&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.

CBC6108 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.

CBC6109 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.

CBC6110 ″&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.

CBC6111 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.

CBC6112 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.

CBC6113 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.

CBC6114 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.

CBC6115 ″&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.

CBC6116 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.

CBC6117 ″&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.

CBC6119 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.

CBC6120 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.

CBC6121 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.

CBC6122 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.

CBC6125 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.

CBC6126 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.

CBC6127 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.

CBC6128 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.

CBC6130 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.

CBC6131 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.

CBC6132 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.

CBC6133 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.

CBC6134 ″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.

CBC6140 ″&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&colon.

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

CBC6141 ″&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.

CBC6142 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.

CBC6143 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.

CBC6150 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.

CBC6151 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.

CBC6152 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.

CBC6153 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.

CBC6155 &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.

CBC6160 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.

CBC6161 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.

CBC6170 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.

CBC6171 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.

CBC6172 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.

CBC6173 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.

CBC6174 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.

CBC6175 Duplicate ″IMPORT″ definitions are detected:

Explanation: The listed objects were defined multiple times.

User Response: Define these objects once.

CBC6177 ″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.

CBC6178 ″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.

CBC6180 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.

CBC6181 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.

CBC6182 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.

CBC6183 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.

CBC6184 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.

CBC6185 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.

CBC6200 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.

CBC6201 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.

CBC6202 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.

CBC6203 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.

CBC6204 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.

CBC6205 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.

CBC6206 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.

CBC6210 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.

CBC6211 Duplicate writable static objects are detected:

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

User Response: Define these objects once.

CBC6212 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.

CBC6213 Duplicate objects are detected:

Explanation: The listed objects were defined multiple times.

User Response: Define the objects once.

CBC6220 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.

CBC6221 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.

CBC6229 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.

CBC6230 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.

CBC6231 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.

CBC6232 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.

CBC6233 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.

CBC6240 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.

CBC6241 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.

CBC6242 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.

CBC6243 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.

CBC6244 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.

CBC6245 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.

CBC6246 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.

CBC6247 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.

CBC6248 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.

CBC6250 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.

CBC6260 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.

CBC6261 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.

CBC6262 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.

CBC6263 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.

CBC6265 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.

CBC6266 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.

CBC6267 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.

CBC6268 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.

CBC6269 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.

CBC6270 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.

CBC6271 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.

CBC6280 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. .

CBC6281 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.

CBC6282 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.

CBC6299 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.

CBC6300 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.

CBC6301 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.

CBC6302 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.

CBC6303 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.

CBC6304 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.

CBC6305 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.

CBC6306 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.

CBC6307 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.

CBC6308 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″.

CBC6310 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.

CBC6311 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.

CBC6312 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.

CBC6313 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.

CBC6314 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.

CBC6315 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.

CBC6316 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.

CBC6317 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.

CBC6320 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.

CBC6340 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.

CBC6341 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.

CBC6342 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.

CBC6345 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.

CBC6400 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.

CBC6401 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.

CBC6402 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.

CBC6403 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.

CBC6404 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.

CBC6406 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.

CBC6407 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.

CBC6408 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.

CBC6420 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.

CBC6421 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.

CBC6422 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.

CBC6423 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.

CBC6425 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.

CBC6426 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.

CBC6430 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.

CBC6431 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.

CBC6440 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.

CBC6441 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.

CBC6442 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.

CBC6443 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.

CBC6444 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.

CBC6445 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.

CBC6446 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.

CBC6447 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.

CBC6450 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.

CBC6451 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.

CBC6460 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.

CBC6461 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.

CBC6490 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.

CBC6491 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.

CBC6492 INTERNAL COMPILER ERROR: Error &1 in Procedure &2.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your Service Representative.

CBC6493 INTERNAL COMPILER ERROR: &1.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your Service Representative.

CBC6494 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.

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.

Appendix H. Utility Messages

This appendix 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
- Runtime 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 56. 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 57. 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 58. 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:

CBC9000 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.

CBC9001 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.

CBC9002 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.

CBC9003 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.

CBC9004 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.

CBC9005 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.

CBC9006 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.

CBC9007 Internal Error. Contact your IBM representative.

User Response: Please report this problem.

CBC9008 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.

CBC9009 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.

CBC9020 Licensed Materials - Property of IBM 5647-A01 (C) Copyright IBM Corp. 1994, 1998. All Rights Reserved. US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Explanation: Copyright statement for the message module (do not translate)

User Response: No action required (do not translate)

Appendix I. Layout of the Events File

This appendix specifies the layout of the SYSEVENT file. SYSEVENT is an events file that contains error information and source file statistics. The SYSEVENT file is not the same as the binder's Input Event Log. Use the `EVENTS` compiler option to produce the SYSEVENT file. For more information on the `EVENTS` compiler option, see "EVENTS | NOEVENTS" on page 101.

In the following example, the source file `simple.c` is compiled with the `EVENTS(USERID.LIST(EGEVENT))` compiler option. The file `err.h` is a header file that is included in `simple.c`. Figure 79 is the event file that is generated when `simple.c` is compiled.

```
 1   #include "./err.h"
 2   main() {
 3      add some error messages;
 4      return(0);
 5      here and there;
 6   }
```

Figure 77. simple.c

```
1   add some;
2   errors in the header file;
```

Figure 78. Err.h

```
-------  start simple.events  ------
  FILEID 0 1 0 10 ./simple.c
  FILEID 0 2 1 9 ././err.h
  ERROR 0 2 0 0 1 1 0 0 CBC1AAA E 12 48 Definition of function add require
  FILEEND 0 2 2
  ERROR 0 2 0 0 1 5 0 0 CBC1BBB E 12 35 Syntax error: possible missing '{'
  ERROR 0 1 0 0 3 3 0 0 CBC1CCC E 12 26 Undeclared identifier add.
  ERROR 0 1 0 0 5 8 0 0 CBC1DDD E 12 42 Syntax error: possible missing ';'
  ERROR 0 1 0 0 5 3 0 0 CBC1EEE E 12 27 Undeclared identifier here.
  FILEEND 0 1 6
-------  end   simple.events  ------
```

Figure 79. Sample SYSEVENT file

There are three different record types generated in the event file:

- FILEID
- FILEEND
- ERROR

Description of the Fileid Field

The following is an example of the FILEID field from the sample SYSEVENT file that is shown in Figure 79. Table 59 on page 720 describes the FILEID identifiers.

```
FILEID 0 1 0 10 ./simple.c
       A B C D  E
```

Table 59. Explanation of the FILEID Field Layout

Column	Identifier	Description
A	Revision	Revision number of the event record.
B	File number	Increments starting with 1 for the primary file.
C	Line number	The line number of the # include directive. For the primary source file, this value is 0.
D	File name length	Length of file or dataset.
E	File name	String containing file/dataset name.

Description of the Fileend Field

The following is an example of the FILEEND field from the sample SYSEVENT file that is shown in Figure 79 on page 719. Table 60 describes the FILEEND identifiers.

```
FILEEND 0 1 6
        A B C
```

Table 60. Explanation of the FILEEND Field Layout

Column	Identifier	Description
A	Revision	Revision number of the event record
B	File number	File number that has been processed to end of file
C	Expansion	Total number of lines in the file

Description of the Error Field

The following is an example of the ERROR field from the sample SYSEVENT file that is shown in Figure 79 on page 719. Table 61 describes the ERROR identifiers.

```
ERROR 0 1 0 0 3 3 0 0 CBCMMMM E 12 26 Undeclared identifier add.
      A B C D E F G H I       J K  L  M
```

Table 61. Explanation of the ERROR Field Layout

Column	Identifier	Description
A	Revision	Revision number of the event record.
B	File number	Increments starting with 1 for the primary file.
C	Reserved	Do not build a dependency on this identifier. It is reserved for future use.
D	Reserved	Do not build a dependency on this identifier. It is reserved for future use.

Table 61. Explanation of the ERROR Field Layout (continued)

Column	Identifier	Description
E	Starting line number	The source line number for which the message was issued. A value of 0 indicates the message was not associated with a line number.
F	Starting column number	The column number or position within the source line for which the message was issued. A value of 0 indicates the message is not associated with a line number.
G	Reserved	Do not build a dependency on this identifier. It is reserved for future use.
H	Reserved	Do not build a dependency on this identifier. It is reserved for future use.
I	Message identifier	String Containing the message identifier.

J	Message severity character	`I=Informational`
		`W=Warning`
		`E=Error`
		`S=Severe`
		`U=Unrecoverable`

K	Message severity number	Return code associated with the message.
L	Message length	Length of message text.
M	Message text	String containing message text.

Notices

Programming Interface Information

This publication documents *intended* Programming Interfaces that allow the customer to write z/OS C/C++ programs.

Trademarks

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/`.

Extracts from X/Open Specification, Programming Languages, Issue 4 Release 2, copyright 1988, 1989, February 1992, by the X/Open Company Limited, have been

Glossary

This glossary defines technical terms and abbreviations that are used in z/OS C/C++ documentation. If you do not find the term you are looking for, refer to the index of the appropriate z/OS C/C++ manual or view *IBM Glossary of Computing Terms*, located at:

`http://www.ibm.com/networking/nsg/nsgmain.htm`

This glossary includes terms and definitions from:

- *American National Standard Dictionary for Information Systems*, ANSI/ISO X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI/ISO). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are indicated by the symbol *ANSI/ISO* after the definition.
- *IBM Dictionary of Computing*, SC20-1699. These definitions are indicated by the registered trademark *IBM* after the definition.
- *X/Open CAE Specification, Commands and Utilities, Issue 4. July, 1992*. These definitions are indicated by the symbol *X/Open* after the definition.
- *ISO/IEC 9945-1:1990/IEEE POSIX 1003.1-1990*. These definitions are indicated by the symbol *ISO.1* after the definition.
- *The Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol *ISO-JTC1* after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol *ISO Draft* after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

A

abstract class. (1) A class with at least one pure virtual function that is used as a base class for other classes. The abstract class represents a concept; classes derived from it represent implementations of the concept. You cannot create a direct object of an abstract class, but you can create references and pointers to an abstract class and set them to refer to objects of classes derived from the abstract class. See also *base class*. (2) A class that allows polymorphism. There can be no objects of an abstract class; they are only used to derive new classes.

abstract code unit. See *ACU*.

abstract data type. A mathematical model that includes a structure for storing data and operations that can be performed on that data. Common abstract data types include sets, trees, and heaps.

abstraction (data). A data type with a private representation and a public set of operations (functions or operators) which restrict access to that data type to that set of operations. The C++ language uses the concept of classes to implement data abstraction.

access. An attribute that determines whether or not a class member is accessible in an expression or declaration.

access declaration. A declaration used to restore access to members of a base class.

access mode. (1) A technique that is used to obtain a particular logical record from, or to place a particular logical record into, a file assigned to a mass storage device. *ANSI/ISO*. (2) The manner in which files are referred to by a computer. Access can be sequential (records are referred to one after another in the order in which they appear on the file), access can be random (the individual records can be referred to in a nonsequential manner), or access can be dynamic (records can be accessed sequentially or randomly, depending on the form of the input/output request). *IBM*. (3) A particular form of access permitted to a file. *X/Open*.

access resolution. The process by which the accessibility of a particular class member is determined.

access specifier. One of the C++ keywords: public, private, and protected, used to define the access to a member.

ACU (abstract code unit). A measurement used by the z/OS C/C++ compiler for judging the size of a function. The number of ACUs that comprise a function is proportional to its size and complexity.

addressing mode. See *AMODE*.

address space. (1) The range of addresses available to a computer program. *ANSI/ISO*. (2) The complete range of addresses that are available to a programmer. See also *virtual address space*. (3) The area of virtual storage available for a particular job. (4) The memory locations that can be referenced by a process. *X/Open*. *ISO.1*.

aggregate. (1) An array or a structure. (2) A compile-time option to show the layout of a structure or union in the listing. (3) In programming languages, a structured collection of data items that form a data type. *ISO-JTC1*. (4) In C++, an array or a class with no user-declared constructors, no private or protected non-static data members, no base classes, and no virtual functions.

alert. (1) A message sent to a management services focal point in a network to identify a problem or an impending problem. *IBM*. (2) To cause the user's terminal to give some audible or visual indication that an error or some other event has occurred. When the standard output is directed to a terminal device, the method for alerting the terminal user is unspecified. When the standard output is not directed to a terminal device, the alert is accomplished by writing the alert character to standard output (unless the utility description indicates that the use of standard output produces undefined results in this case). *X/Open*.

alert character. A character that in the output stream should cause a terminal to alert its user via a visual or audible notification. The alert character is the character designated by a '\a' in the C and C++ languages. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the alert function. *X/Open*.

This character is named <alert> in the portable character set.

alias. (1) An alternate label; for example, a label and one or more aliases may be used to refer to the same data element or point in a computer program. *ANSI/ISO*. (2) An alternate name for a member of a partitioned data set. *IBM*. (3) An alternate name used for a network. Synonymous with nickname. *IBM*.

alias name. (1) A word consisting solely of underscores, digits, and alphabetics from the portable file name character set, and any of the following characters: `! % , @`. Implementations may allow other characters within alias names as an extension. *X/Open*. (2) An alternate name. *IBM*. (3) A name that is defined in one network to represent a logical unit name in another interconnected network. The alias name does not have to be the same as the real name; if these names are not the same; translation is required. *IBM*.

alignment. The storing of data in relation to certain machine-dependent boundaries. *IBM*.

alternate code point. A syntactic code point that permits a substitute code point to be used. For example, the left brace ({) can be represented by X'B0' and also by X'C0'.

American National Standard Code for Information Interchange (ASCII). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. *IBM*.

Note: IBM has defined an extension to ASCII code (characters 128–255).

American National Standards Institute (ANSI/ISO). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. *ANSI/ISO*.

AMODE (addressing mode). In z/OS, a program attribute that refers to the address length that a program is prepared to handle upon entry. In z/OS, addresses may be 24 or 31 bits in length. *IBM*.

angle brackets. The characters < (left angle bracket) and > (right angle bracket). When used in the phrase “enclosed in angle brackets”, the symbol < immediately precedes the object to be enclosed, and > immediately follows it. When describing these characters in the portable character set, the names <less-than-sign> and <greater-than-sign> are used. *X/Open*.

anonymous union. A union that is declared within a structure or class and does not have a name. It must not be followed by a declarator.

ANSI/ISO. See *American National Standards Institute*.

API (application program interface). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program. *IBM*.

application. (1) The use to which an information processing system is put; for example, a payroll application, an airline reservation application, a network application. *IBM*. (2) A collection of software components used to perform specific types of user-oriented work on a computer. *IBM*.

application generator. An application development tool that creates applications, application components (panels, data, databases, logic, interfaces to system services), or complete application systems from design specifications.

application program. A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll. *IBM*.

archive libraries. The archive library file, when created for application program object files, has a special symbol table for members that are object files.

argument. (1) A parameter passed between a calling program and a called program. *IBM*. (2) In a function call, an expression that represents a value that the calling function passes to the function specified in the call. (3) In the shell, a parameter passed to a utility as the equivalent of a single string in the *argv* array created by one of the *exec* functions. An argument is one of the options, option-arguments, or operands following the command name. *X/Open*.

argument declaration. See *parameter declaration*.

arithmetic object. (1) A bit field, or an integral, floating-point, or packed decimal (IBM extension) object. (2) A real object or objects having the type float, double, or long double.

array. In programming languages, an aggregate that consists of data objects with identical attributes, each of which may be uniquely referenced by subscripting. *ISO-JTC1*.

array element. A data item in an array. *IBM*.

ASCII. See *American National Standard Code for Information Interchange.*

Assembler H. An IBM licensed program. Translates symbolic assembler language into binary machine language.

assembler language. A source language that includes symbolic language statements in which there is a one-to-one correspondence with the instruction formats and data formats of the computer. *IBM*.

assembler user exit. In the z/OS Language Environment a routine to tailor the characteristics of an enclave prior to its establishment.

assignment expression. An expression that assigns the value of the right operand expression to the left operand variable and has as its value the value of the right operand. *IBM*.

atexit list. A list of actions specified in the z/OS C/C++ `atexit()` function that occur at normal program termination.

auto storage class specifier. A specifier that enables the programmer to define a variable with automatic storage; its scope restricted to the current block.

automatic call library. Contains modules that are used as secondary input to the binder to resolve external symbols left undefined after all the primary input has been processed.

The automatic call library can contain:

- Object modules, with or without binder control statements
- Load modules
- z/OS C/C++ run-time routines (SCEELKED)

automatic library call. The process in which control sections are processed by the binder or loader to resolve references to members of partitioned data sets. *IBM*.

automatic storage. Storage that is allocated on entry to a routine or block and is freed on the subsequent return. Sometimes referred to as *stack storage* or *dynamic storage*.

B

background process. (1) A process that does not require operator intervention but can be run by the computer while the workstation is used to do other work. *IBM*. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command. *IBM*. (3) A process that is a member of a background process group. *X/Open*. *ISO.1*.

background process group. Any process group, other than a foreground process group, that is a member of a session that has established a connection with a controlling terminal. *X/Open*. *ISO.1*.

backslash. The character \. This character is named <backslash> in the portable character set.

base class. A class from which other classes are derived. A base class may itself be derived from another base class. See also *abstract class*.

based on. The use of existing classes for implementing new classes.

binary expression. An expression containing two operands and one operator.

binary stream. (1) An ordered sequence of untranslated characters. (2) A sequence of characters that corresponds on a one-to-one basis with the characters in the file. No character translation is performed on binary streams. *IBM*.

bind. (1) To combine one or more control sections or program modules into a single program module, resolving references between them. (2) To assign virtual storage addresses to external symbols.

binder. The DFSMS/MVS program that processes the output of language translators and compilers into an executable program (load module or program object). It replaces the linkage editor and batch loader in the MVS/ESA, OS/390, or z/OS operating system.

bit field. A member of a structure or union that contains a specified number of bits. *IBM*.

bitwise operator. An operator that manipulates the value of an object at the bit level.

blank character. (1) A graphic representation of the space character. *ANSI/ISO*. (2) A character that represents an empty position in a graphic character string. *ISO Draft*. (3) One of the characters that belong to the *blank* character class as defined via the LC_CTYPE category in the current locale. In the POSIX locale, a blank character is either a tab or a space character. *X/Open*.

block. (1) In programming languages, a compound statement that coincides with the scope of at least one of the declarations contained within it. A block may also specify storage allocation or segment programs for other purposes. *ISO-JTC1*. (2) A string of data elements recorded or transmitted as a unit. The elements may be characters, words or physical records. *ISO Draft*. (3) The unit of data transmitted to and from a device. Each block contains one record, part of a record, or several records.

block statement. In the C or C++ languages, a group of data definitions, declarations, and statements appearing between a left brace and a right brace that are processed as a unit. The block statement is considered to be a single C or C++ statement. *IBM*.

boundary alignment. The position in main storage of a fixed-length field, such as a halfword or doubleword, on a byte-level boundary for that unit of information. *IBM*.

braces. The characters { (left brace) and } (right brace), also known as *curly braces*. When used in the phrase “enclosed in (curly) braces” the symbol { immediately precedes the object to be enclosed, and } immediately follows it. When describing these characters in the portable character set, the names <left-brace> and <right-brace> are used. *X/Open*.

brackets. The characters [(left bracket) and] (right bracket), also known as *square brackets*. When used in the phrase *enclosed in (square) brackets* the symbol [immediately precedes the object to be enclosed, and] immediately follows it. When describing these characters in the portable character set, the names <left-bracket> and <right-bracket> are used. *X/Open*.

break statement. A C or C++ control statement that contains the keyword “`break`” and a semicolon. *IBM*. It is used to end an iterative or a switch statement by exiting from it at any point other than the logical end. Control is passed to the first statement after the iteration or switch statement.

built-in. (1) A function that the compiler will automatically inline instead of making the function call, unless the programmer specifies not to inline. (2) In programming languages, pertaining to a language object that is declared by the definition of the programming language; for example, the built-in function SIN in PL/I, the predefined data type INTEGER in FORTRAN. *ISO-JTC1*. Synonymous with *predefined*. *IBM*.

byte-oriented stream. See *orientation of a stream*.

C

C library. A system library that contains common C language subroutines for file access, string operators, character operations, memory allocation, and other functions. *IBM*.

C or C++ language statement. A C or C++ language statement contains zero or more expressions. A block statement begins with a { (left brace) symbol, ends with a } (right brace) symbol, and contains any number of statements.

All C or C++ language statements, except block statements, end with a ; (semicolon) symbol.

c89 utility. A utility used to compile and bind an application program from the z/OS shell.

C++ class library. A collection of C++ classes.

C++ library. A system library that contains common C++ language subroutines for file access, memory allocation, and other functions.

callable services. A set of services that can be invoked by z/OS Language Environment-conforming high level languages using the conventional z/OS Language Environment-defined call interface, and usable by all programs sharing the z/OS Language Environment conventions.

Use of these services helps to decrease an application's dependence on the specific form and content of the services delivered by any single operating system.

call chain. A trace of all active functions.

caller. A function that calls another function.

cancelability point. A specific point within the current thread that is enabled to solicit cancel requests. This is accomplished using the `pthread_testintr()` function.

carriage-return character. A character that in the output stream indicates that printing should start at the beginning of the same physical line in which the carriage-return character occurred. The carriage-return is the character designated by '\r' in the C and C++ languages. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the beginning of the line. *X/Open*.

case clause. In a C or C++ switch statement, a CASE label followed by any number of statements.

case label. The word `case` followed by a constant integral expression and a colon. When the selector evaluates the value of the constant expression, the statements following the case label are processed.

cast expression. An expression that converts or reinterprets its operand.

cast operator. The cast operator is used for explicit type conversions.

cataloged procedures. A set of control statements placed in a library and retrievable by name. *IBM*.

catch block. A block associated with a try block that receives control when an exception matching its argument is thrown.

char specifier. A char is a built-in data type. In the C++ language, char, signed char, and unsigned char are all distinct data types.

character. (1) A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes. *ANSI/ISO*. (2) A sequence of one or more bytes representing a single graphic symbol or control code. This term corresponds to the ISO C standard term *multibyte character* (multibyte character), where a single-byte character is a special case of the multibyte character. Unlike the usage in the ISO C standard, *character* here has no necessary relationship with storage space, and *byte* is used when storage space is discussed. *X/Open*. *ISO.1*.

character array. An array of type char. *X/Open*.

character class. A named set of characters sharing an attribute associated with the name of the class. The classes and the characters that they contain are dependent on the value of the LC_CTYPE category in the current locale. *X/Open*.

character constant. A string of any of the characters that can be represented, usually enclosed in quotes.

character set. (1) A finite set of different characters that is complete for a given purpose; for example, the character set in ISO Standard 646, 7-bit Coded Character Set for Information Processing Interchange. *ISO Draft*. (2) All the valid characters for a programming language or for a computer system. *IBM*. (3) A group of characters used for a specific reason; for example, the set of characters a printer can print. *IBM*. (4) See also *portable character set*.

character special file. (1) A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. *IBM*. (2) A file that refers to a device. One specific type of character special file is a terminal device file. *X/Open*. *ISO.1*.

character string. A contiguous sequence of characters terminated by and including the first null byte. *X/Open*.

child. A node that is subordinate to another node in a tree structure. Only the root node is not a child.

child enclave. The *nested enclave* created as a result of certain commands being issued from a *parent enclave*.

CICS (Customer Information Control System). Pertaining to an IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases. *IBM*.

CICS destination control table. See *DCT*.

CICS translator. A routine that accepts as input an application containing EXEC CICS commands and produces as output an equivalent application in which each CICS command has been translated into the language of the source.

class. (1) A C++ aggregate that may contain functions, types, and user-defined operators in addition to data. A class may be derived from another class, inheriting the properties of its parent class. A class may restrict access to its members. (2) A user-defined data type. A class data type can contain both data representations (data members) and functions (member functions).

class key. One of the C++ keywords: class, struct and union.

class library. A collection of classes.

class member operator. An operator used to access class members through class objects or pointers to class objects. The class member operators are:

```
. -> .* ->*
```

class name. A unique identifier that names a class type.

class scope. An indication that a name of a class can be used only in a member function of that class.

class tag. Synonym for *class name*.

class template. A blueprint describing how a set of related classes can be constructed.

client program. A program that uses a class. The program is said to be a *client* of the class.

CLIST. A programming language that typically executes a list of TSO commands.

CLLE (COBOL Load List Entry). Entry in the load list containing the name of the program and the load address.

COBCOM. Control block containing information about a COBOL partition.

COBOL (common business-oriented language). A high-level language, based on English, that is primarily used for business applications.

COBOL Load List Entry. See *CLLE*.

COBVEC. COBOL vector table containing the address of the library routines.

coded character set. (1) A set of graphic characters and their code point assignments. The set may contain fewer characters than the total number of possible characters: some code points may be unassigned. *IBM*. (2) A coded set whose elements are single characters; for example, all characters of an alphabet. *ISO Draft*. (3) Loosely, a code. *ANSI/ISO*.

code element set. (1) The result of applying a code to all elements of a coded set, for example, all the three-letter international representations of airport names. *ISO Draft*. (2) The result of applying rules that map a numeric code value to each element of a character set. An element of a character set may be related to more than one numeric code value but the reverse is not true. However, for state-dependent encodings the relationship between numeric code values to elements of a character set may be further controlled by state information. The character set may contain fewer elements than the total number of possible numeric code values; that is, some code values may be unassigned. *X/Open*. (3) Synonym for codeset.

code page. (1) An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, assignment of characters and meanings to 128 code points for a 7-bit code. (2) A particular assignment of hexadecimal identifiers to graphic characters.

code point. (1) A representation of a unique character. For example, in a single-byte character set each of 256 possible characters is represented by a one-byte code point. (2) An identifier in an alert description that represents a short unit of text. The code point is replaced with the text by an alert display program.

codeset. Synonym for code element set. *IBM*.

collating element. The smallest entity used to determine the logical ordering of character or wide-character strings. A collating element consists of either a single character, or two or more characters collating as a single entity. The value of the LC_COLLATE category in the current locale determines the current set of collating elements. *X/Open*.

collating sequence. (1) A specified arrangement used in sequencing. *ISO-JTC1*. *ANSI/ISO*. (2) An ordering assigned to a set of items, such that any two sets in that assigned order can be collated. *ANSI/ISO*. (3) The relative ordering of collating elements as determined by the setting of the LC_COLLATE category in the current locale. The character order, as defined for the LC_COLLATE category in the current locale, defines the relative order of all collating elements, such that each element occupies a unique position in the order. This is the order used in ranges of characters and collating elements in regular expressions and pattern matching. In addition, the definition of the collating weights of characters and collating elements uses collating elements to represent their respective positions within the collation sequence.

collation. The logical ordering of character or wide-character strings according to defined precedence rules. These rules identify a collation sequence between the collating elements, and such additional rules that can be used to order strings consisting or multiple collating elements. *X/Open*.

collection. (1) An abstract class without any ordering, element properties, or key properties. (2) In a general sense, an implementation of an abstract data type for storing elements.

Collection Class Library. A set of classes that provide basic functions for collections, and can be used as base classes.

column position. A unit of horizontal measure related to characters in a line.

It is assumed that each character in a character set has an intrinsic column width independent of any output device. Each printable character in the portable character set has a column width of one. The standard utilities, when used as described in this document set, assume that all characters have integral column widths. The column width of a character is not necessarily related to the internal representation of the character (numbers of bits or bytes).

The column position of a character in a line is defined as one plus the sum of the column widths of the preceding characters in the line. Column positions are numbered starting from 1. *X/Open*.

comma expression. An expression (not a function argument list) that contains two or more operands separated by commas. The compiler evaluates all operands in the order specified, discarding all but the last (rightmost). The value of the expression is the value of the rightmost operand. Typically this is done to produce side effects.

command. A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command processor parameter list (CPPL). The format of a TSO parameter list. When a TSO terminal monitor application attaches a command processor, register 1 contains a pointer to the CPPL, containing addresses required by the command processor.

COMMAREA. A communication area made available to applications running under CICS.

Common Business-Oriented Language. See *COBOL*.

common expression elimination. Duplicated expressions are eliminated by using the result of the previous expression. This includes intermediate expressions within expressions.

compilation unit. (1) A portion of a computer program sufficiently complete to be compiled correctly. *IBM*. (2) A single compiled file and all its associated include files. (3) An independently compilable sequence of high-level language statements. Each high-level language product has different rules for what makes up a compilation unit.

complete class name. The complete qualification of a nested class name including all enclosing class names.

Complex Mathematics library. A C++ class library that provides the facilities to manipulate complex numbers and perform standard mathematical operations on them.

computational independence. No data modified by either a main task program or a parallel function is examined or modified by a parallel function that might be running simultaneously.

concrete class. (1) A class that is not abstract. (2) A class defining objects that can be created.

condition. (1) A relational expression that can be evaluated to a value of either true or false. *IBM*. (2) An exception that has been enabled, or recognized, by the z/OS Language Environment and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware/operating system and result in an interrupt. They can also be detected by language-specific generated code or language library code.

conditional expression. A compound expression that contains a condition (the first expression), an expression to be evaluated if the condition has a nonzero value (the second expression), and an expression to be evaluated if the condition has the value zero (the third expression).

condition handler. A user-written condition handler or language-specific condition handler (such as a PL/I ON-unit or z/OS C/C++ `signal()` function call) invoked by the z/OS C/C++ *condition manager* to respond to conditions.

condition manager. Manages conditions in the common execution environment by invoking various user-written and language-specific *condition handlers*.

condition token. In the z/OS Language Environment, a data type consisting of 12 bytes (96 bits). The condition token contains structured fields that indicate various aspects of a condition including the severity, the associated message number, and information that is specific to a given instance of the condition.

const. (1) An attribute of a data object that declares the object cannot be changed. (2) A keyword that allows you to define a variable whose value does not change. (3) A keyword that allows you to define a parameter that is not changed by the function. (4) A keyword that allows you to define a member function that does not modify the state of the class for which it is defined.

constant. (1) In programming languages, a language object that takes only one specific value. *ISO-JTC1*. (2) A data item with a value that does not change. *IBM*.

constant expression. An expression having a value that can be determined during compilation and that does not change during the running of the program. *IBM*.

constant propagation. An optimization technique where constants used in an expression are combined and new ones are generated. Mode conversions are done to allow some intrinsic functions to be evaluated at compile time.

constructed reentrancy. The attribute of applications that contain external data and require additional processing to make them reentrant. Contrast with *natural reentrancy*.

constructor. A special C++ class member function that has the same name as the class and is used to create an object of that class.

control character. (1) A character whose occurrence in a particular context specifies a control function. *ISO Draft*. (2) Synonymous with nonprinting character. *IBM*. (3) A character, other than a graphic character, that affects the recording, processing, transmission, or interpretation of text. *X/Open*.

control statement. (1) A statement that is used to alter the continuous sequential execution of statements; a control statement may be a conditional statement, such as `if`, or an imperative statement, such as `return`. (2) A statement that changes the path of execution.

controlling process. The session leader that establishes the connection to the controlling terminal. If the terminal ceases to be a controlling terminal for this session, the session leader ceases to be the controlling process. *X/Open*. *ISO.1*.

controlling terminal. A terminal that is associated with a session. Each session may have at most one controlling terminal associated with it, and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal cause signals to be sent to all processes in the process group associated with the controlling terminal. *X/Open*. *ISO.1*.

conversion. (1) In programming languages, the transformation between values that represent the same data item but belong to different data types. Information may be lost because of conversion since accuracy of data representation varies among different data types. *ISO-JTC1*. (2) The process of changing from one method of data processing to another or from one data processing system to another. *IBM*. (3) The process of changing from one form of representation to another; for example to change from decimal representation to binary representation. *IBM*. (4) A change in the type of a value. For example, when you add values having different data types, the compiler converts both values to a common form before adding the values.

conversion descriptor. A per-process unique value used to identify an open codeset conversion. *X/Open*.

conversion function. A member function that specifies a conversion from its class type to another type.

coordinated universal time (UTC). Synonym for Greenwich Mean Time (GMT). See *GMT*.

copy constructor. A constructor that copies a class object of the same class type.

CSECT (control section). The part of a program specified by the programmer to be a relocatable unit, all elements of which are to be loaded into adjoining main storage locations.

Cross System Product. See *CSP*.

CSP (Cross System Product). A set of licensed programs designed to permit the user to develop and run applications using independently defined maps (display and printer formats), data items (records, working storage, files, and single items), and processes (logic). The Cross System Product set consists of two parts: Cross System Product/Application Development (CSP/AD) and Cross System Product/Application Execution (CSP/AE). *IBM*.

current working directory. (1) A directory, associated with a process, that is used in path-name resolution for path names that do not begin with a slash. *X/Open*. *ISO.1*. (2) In the OS/2 operating system, the first directory in which the operating system looks for programs and files and stores temporary files and output. *IBM*. (3) In the z/OS UNIX environment, a directory that is active and that can be displayed. Relative path name resolution begins in the current directory. *IBM*.

cursor. A reference to an element at a specific position in a data structure.

Customer Information Control System. See *CICS*.

D

data abstraction. A data type with a private representation and a public set of operations (functions or operators) which restrict access to that data type to that set of operations. The C++ language uses the concept of classes to implement data abstraction.

data definition (DD). (1) In the C and C++ languages, a definition that describes a data object, reserves storage for a data object, and can provide an initial value for a data object. A data definition appears outside a function or at the beginning of a block statement. *IBM*. (2) A program statement that describes the features of, specifies relationships of, or establishes context of, data. *ANSI/ISO*. (3) A statement that is stored in the environment and that externally identifies a file and the attributes with which it should be opened.

data definition name. See *ddname*.

data definition statement. See *DD statement*.

data member. The smallest possible piece of complete data. Elements are composed of data members.

data object. (1) A storage area used to hold a value. (2) Anything that exists in storage and on which operations can be performed, such as files, programs, classes, or arrays. (3) In a program, an element of data structure, such as a file, array, or operand, that is needed for the execution of a program and that is named or otherwise specified by the allowable character set of the language in which a program is coded. *IBM*.

data set. Under z/OS, a named collection of related data records that is stored and retrieved by an assigned name.

data stream. A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format. *IBM*.

data structure. The internal data representation of an implementation.

data type. The properties and internal representation that characterize data.

Data Window Services (DWS). Services provided as part of the Callable Services Library that allow manipulation of data objects such as VSAM linear data sets and temporary data objects known as *TEMPSPACE*.

DBCS (double-byte character set). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets.

Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. *IBM*.

DCT (destination control table). A table that contains an entry for each extrapartition, intrapartition, and indirect destination. Extrapartition entries address data sets external to the CICS region. Intrapartition destination entries contain the information required to locate the queue in the intrapartition data set. Indirect destination entries contain the information required to locate the queue in the intrapartition data set.

ddname (data definition name). (1) The logical name of a file within an application. The ddname provides the means for the logical file to be connected to the physical file. (2) The part of the data definition before the equal sign. It is the name used in a call to `fopen` or `freopen` to refer to the data definition stored in the environment.

DD statement (data definition statement). (1) In z/OS, serves as the connection between the logical name of a file and the physical name of the file. (2) A job control statement that defines a file to the operating system, and is a request to the operating system for the allocation of input/output resources.

dead code elimination. A process that eliminates code that exists for calculations that are not necessary. Code may be designated as dead by other optimization techniques.

dead store elimination. A process that eliminates unnecessary storage use in code. A store is deemed unnecessary if the value stored is never referenced again in the code.

decimal constant. (1) A numerical data type used in standard arithmetic operations. (2) A number containing any of the digits 0 through 9. *IBM*.

decimal overflow. A condition that occurs when one or more nonzero digits are lost because the destination field in a decimal operation is too short to contain the results.

declaration. (1) In the C and C++ languages, a description that makes an external object or function available to a function or a block statement. *IBM*. (2) Establishes the names and characteristics of data objects and functions used in a program.

declarator. Designates a data object or function declared. Initializations can be performed in a declarator.

default argument. An argument that is declared with a default value in a function prototype or declaration. If a call to the function omits this argument, the default value is used. Arguments with default values must be the trailing arguments in a function prototype argument list.

default clause. In the C or C++ languages, within a switch statement, the keyword default followed by a colon, and one or more statements. When the conditions of the specified case labels in the switch statement do not hold, the default clause is chosen. *IBM*.

default constructor. A constructor that takes no arguments, or, if it takes arguments, all its arguments have default values.

default initialization. The initial value assigned to a data object by the compiler if no initial value is specified by the programmer.

default locale. (1) The C locale, which is always used when no selection of locale is performed. (2) A system default locale, named by locale-related environmental variables.

define directive. A preprocessor directive that directs the preprocessor to replace an identifier or macro invocation with special code.

definition. (1) A data description that reserves storage and may provide an initial value. (2) A declaration that allocates storage, and may initialize a data object or specify the body of a function.

degree. The number of children of a node.

delete. (1) A C++ keyword that identifies a free storage deallocation operator. (2) A C++ operator used to destroy objects created by `new`.

demangling. The conversion of mangled names back to their original source code names. During C++ compilation, identifiers such as function and static class member names are mangled (encoded) with type and scoping information to ensure type-safe linkage. These mangled names appear in the object file and the final executable file. Demangling (decoding) converts these names back to their original names to make program debugging easier. See also *mangling*.

deque. A queue that can have elements added and removed at both ends. A double-ended queue.

dequeue. An operation that removes the first element of a queue.

dereference. In the C and C++ languages, the application of the unary operator * to a pointer to access the object the pointer points to. Also known as *indirection*.

derivation. In the C++ language, to derive a class, called a derived class, from an existing class, called a base class.

derived class. A class that inherits from a base class. All members of the base class become members of the derived class. You can add additional data members and member functions to the derived class. A derived class object can be manipulated as if it is a base class object. The derived class can override virtual functions of the base class.

descriptor. PL/I control block that holds information such as string lengths, array subscript bounds, and area sizes, and is passed from one PL/I routine to another during run time.

destination control table. See *DCT*.

destructor. A special member function that has the same name as its class, preceded by a tilde (˜), and that "cleans up" after an object of that class, for example, freeing storage that was allocated when the object was created. A destructor has no arguments and no return type.

detach state attribute. An attribute associated with a thread attribute object. This attribute has two possible values:

0	Undetached. An undetached thread keeps its resources after termination of the thread.
1	Detached. A detached thread has its resources freed by the system after termination.

device. A computer peripheral or an object that appears to the application as such. *X/Open*. *ISO.1*.

difference. For two sets A and B, the difference (A-B) is the set of all elements in A but not in B. For bags, there is an additional rule for duplicates: If bag P contains an element *m* times and bag Q contains the same element *n* times, then, if *m*>*n*, the difference contains that element *m*-*n* times. If *m*≤*n*, the difference contains that element zero times.

digraph. A combination of two keystrokes used to represent unavailable characters in a C or C++ source program. Digraphs are read as tokens during the preprocessor phase.

directory. (1) In a hierarchical file system, a container for files or other directories. *IBM*. (2) The part of a partitioned data set that describes the members in the data set.

disabled signal. Synonym for *enabled signal*.

display. To direct the output to the user's terminal. If the output is not directed to the terminal, the results are undefined. *X/Open*.

DLL. See *dynamic link library*.

do statement. In the C and C++ compilers, a looping statement that contains the keyword "*do*", followed by a statement (the action), the keyword "*while*", and an expression in parentheses (the condition). *IBM*.

dot. The file name consisting of a single dot character (.). *X/Open*. *ISO.1*.

double-byte character set. See *DBCS*.

double-precision. Pertaining to the use of two computer words to represent a number in accordance with the required precision. *ISO-JTC1*. *ANSI/ISO*.

double-quote. The character ", also known as *quotation mark*. *X/Open*.

This character is named <quotation-mark> in the portable character set.

doubleword. A contiguous sequence of bytes or characters that comprises two computer words and is capable of being addressed as a unit. *IBM*.

dynamic. Pertaining to an operation that occurs at the time it is needed rather than at a predetermined or fixed time. *IBM*.

dynamic allocation. Assignment of system resources to a program when the program is executed rather than when it is loaded into main storage. *IBM*.

dynamic binding. The act of resolving references to external variables and functions at run time. In C++, dynamic binding is supported by using virtual functions.

dynamic link library (DLL). A file containing executable code and data bound to a program at run time. The code and data in a dynamic link library can be shared by several applications simultaneously. Compiling code with the `DLL` option does not mean that the produced executable will be a DLL. To create a DLL, use `#pragma export` or the `EXPORTALL` compiler option.

DSA (dynamic storage area). An area of storage obtained during the running of an application that consists of a register save area and an area for automatic data, such as program variables. DSAs are generally allocated within Language Environment-managed stack segments. DSAs are added to the stack when a routine is entered and removed upon exit in a last in, first out (LIFO) manner. In Language Environment, a DSA is known as a stack frame.

dynamic storage. Synonym for *automatic storage*.

dynamic storage area . See DSA

E

EBCDIC. See *extended binary-coded decimal interchange code.*

effective group ID. An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject to change during the process lifetime, as described in the *exec* family of functions and `setgid()`. *X/Open*. *ISO.1*.

effective user ID. (1) The user ID associated with the last authenticated user or the last `setuid()` program. It is equal to either the real or the saved user ID. (2) The current user ID, but not necessarily the user's login ID; for example, a user logged in under a login ID may change to another user's ID. The ID to which the user changes becomes the effective user ID until the user switches back to the original login ID. All discretionary access decisions are based on the effective user ID. *IBM*. (3) An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject to change during the process lifetime, as described in *exec* and `setuid()`. *X/Open*. *ISO.1*.

elaborated type specifier. A specifier typically used in an incomplete class declaration to qualify types that are otherwise hidden.

element. The component of an array, subrange, enumeration, or set.

element equality. A relation that determines if two elements are equal.

element occurrence. A single instance of an element in a collection. In a unique collection, element occurrence is synonymous with element value.

element value. All the instances of an element with a particular value in a collection. In a nonunique collection, an element value may have more than one occurrence. In a unique collection, element value is synonymous with element occurrence.

else clause. The part of an if statement that contains the word else, followed by a statement. The else clause provides an action that is started when the if condition evaluates to a value of zero (false). *IBM*.

empty line. A line consisting of only a new-line character. *X/Open*.

empty string. (1) A string whose first byte is a null byte. Synonymous with null string. *X/Open*. (2) A character array whose first element is a null character. *ISO.1*.

enabled signal. The occurrence of an enabled signal results in the default system response or the execution of an established signal handler. If disabled, the occurrence of the signal is ignored.

encapsulation. Hiding the internal representation of data objects and implementation details of functions from the client program. This enables the end user to focus on the use of data objects and functions without having to know about their representation or implementation.

enclave. In z/OS Language Environment, an independent collection of routines, one of which is designated as the main routine. An enclave is roughly analogous to a program or run unit.

enqueue. (1) An operation that adds an element as the last element to a queue. (2) Request control of a serially reusable resource.

entry point. The address or label of the first instruction that is executed when a routine is entered for execution.

enumeration constant. In the C or C++ language, an identifier, with an associated integer value, defined in an enumerator. An enumeration constant may be used anywhere an integer constant is allowed. *IBM*.

enumeration data type. (1) In the Fortran, C, and C++ language, a data type that represents a set of values that a user defines. *IBM*. (2) A type that represents integers and a set of enumeration constants. Each enumeration constant has an associated integer value.

enumeration tag. In the C and C++ language, the identifier that names an enumeration data type. *IBM*.

enumeration type. An enumeration type defines a set of enumeration constants. In the C++ language, an enumeration type is a distinct data type that is not an integral type.

enumerator. In the C and C++ language, an enumeration constant and its associated value. *IBM*.

equivalence class. (1) A grouping of characters that are considered equal for the purpose of collation; for example, many languages place an uppercase character in the same equivalence class as its lowercase form, but some languages distinguish between accented and unaccented character forms for the purpose of collation. *IBM*. (2) A set of collating elements with the same primary collation weight.

Elements in an equivalence class are typically elements that naturally group together, such as all accented letters based on the same base letter.

The collation order of elements within an equivalence class is determined by the weights assigned on any subsequent levels after the primary weight. *X/Open*.

escape sequence. (1) A representation of a character. An escape sequence contains the \ symbol followed by one of the characters: `a`, `b`, `f`, `n`, `r`, `t`, `v`, `'`, `"`, `x`, `\`, or followed by one or more octal or hexadecimal digits. (2)

A sequence of characters that represent, for example, nonprinting characters, or the exact code point value to be used to represent variant and nonvariant characters regardless of code page. (3) In the C and C++ language, an escape character followed by one or more characters. The escape character indicates that a different code, or a different coded character set, is used to interpret the characters that follow. Any member of the character set used at runtime can be represented using an escape sequence. (4) A character that is preceded by a backslash character and is interpreted to have a special meaning to the operating system. (5) A sequence sent to a terminal to perform actions such as moving the cursor, changing from normal to reverse video, and clearing the screen. Synonymous with multibyte control. *IBM*.

exception. (1) Any user, logic, or system error detected by a function that does not itself deal with the error but passes the error on to a handling routine (also called throwing the exception). (2) In programming languages, an abnormal situation that may arise during execution, that may cause a deviation from the normal execution sequence, and for which facilities exist in a programming language to define, raise, recognize, ignore, and handle it; for example, (ON-) condition in PL/I, exception in ADA. *ISO-JTC1*.

executable. A load module or program object which has yet to be loaded into memory for execution.

executable file. A regular file acceptable as a new process image file by the equivalent of the *exec* family of functions, and thus usable as one form of a utility. The standard utilities described as compilers can produce executable files, but other unspecified methods of producing executable files may also be provided. The internal format of an executable file is unspecified, but a conforming application cannot assume an executable file is a text file. *X/Open*.

exception handler. (1) Exception handlers are catch blocks in C++ applications. Catch blocks catch exceptions when they are thrown from a function enclosed in a try block. Try blocks, catch blocks, and throw expressions are the constructs used to implement formal exception handling in C++ applications. (2) A set of routines used to detect deadlock conditions or to process abnormal condition processing. An exception handler allows the normal running of processes to be interrupted and resumed. *IBM*.

executable file. A regular file acceptable as a new process image file by the equivalent of the *exec* family of functions, and thus usable as one form of a utility. The standard utilities described as compilers can produce executable files, but other unspecified methods of producing executable files may also be provided. The internal format of an executable file is unspecified, but a conforming application cannot assume an executable file is a text file. *X/Open*.

executable program. A program that has been link-edited and therefore can be run in a processor. *IBM*.

extended binary-coded data interchange code (EBCDIC). A coded character set of 256 8-bit characters. *IBM*.

extended-precision. Pertaining to the use of more than two computer words to represent a floating point number in accordance with the required precision. In z/OS four computer words are used for an extended-precision number.

extension. (1) An element or function not included in the standard language. (2) File name extension.

external data definition. A description of a variable appearing outside a function. It causes the system to allocate storage for that variable and makes that variable accessible to all functions that follow the definition and are located in the same file as the definition. *IBM*.

extern storage class specifier. A specifier that enables the programmer to declare objects and functions that several source files can use.

F

feature test macro (FTM). A macro (`#define`) used to determine whether a particular set of features will be included from a header. *X/Open*. *ISO.1*.

FIFO special file. A type of file with the property that data written to such a file is read on a first-in-first-out basis. Other characteristics of FIFOs are described in `open()`, `read()`, `write()`, and `lseek()`. *X/Open*. *ISO.1*.

file access permissions. The standard file access control mechanism uses the file permission bits. The bits are set at the time of file creation by functions such as `open()`, `creat()`, `mkdir()`, and `mkfifo()` and can be changed by `chmod()`. The bits are read by `stat()` or `fstat()`. *X/Open*.

file descriptor. (1) A positive integer that the system uses instead of the file name to identify an open file. (2) A per-process unique, non-negative integer used to identify an open file for the purpose of file access. *ISO.1*.

The value of a file descriptor is from zero to {OPEN_MAX}—which is defined in <limits.h>. A process can have no more than {OPEN_MAX} file descriptors open simultaneously. File descriptors may also be used to implement directory streams. *X/Open*.

file mode. An object containing the *file mode bits* and file type of a file, as described in `<sys/stat.h>`. *X/Open*.

file mode bits. A file's file permission bits, set-user-ID-on-execution bit (S_ISUID) and set-group-ID-on-execution bit (S_ISGID). *X/Open*.

file permission bits. Information about a file that is used, along with other information, to determine if a process has read, write, or execute/search permission to a file. The bits are divided into three parts: owner, group, and other. Each part is used with the corresponding file class of process. These bits are contained in the file mode, as described in *<sys/stat.h>.* The detailed usage of the file permission bits is described in *file access permissions*. *X/Open*. *ISO.1*.

file scope. A name declared outside all blocks, classes, and function declarations has file scope and can be used after the point of declaration in a source file.

filter. A command whose operation consists of reading data from standard input or a list of input files and writing data to standard output. Typically, its function is to perform some transformation on the data stream. *X/Open*.

first element. The element visited first in an iteration over a collection. Each collection has its own definition for first element. For example, the first element of a sorted set is the element with the smallest value.

flat collection. A collection that has no hierarchical structure.

float constant. (1) A constant representing a nonintegral number. (2) A number containing a decimal point, an exponent, or both a decimal point and an exponent. The exponent contains an e or E, an optional sign (+ or -), and one or more digits (0 through 9). *IBM*.

for statement. A looping statement that contains the word *for* followed by a for-initializing-statement, an optional condition, a semicolon, and an optional expression, all enclosed in parentheses.

foreground process. (1) A process that must run to completion before another command is issued. The foreground process is in the foreground process group, which is the group that receives the signals generated by a terminal. *IBM*. (2) A process that is a member of a foreground process group. *X/Open*. *ISO.1*.

foreground process group. (1) The group that receives the signals generated by a terminal. *IBM*. (2) A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal. Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that controlling terminal. *X/Open*. *ISO.1*.

foreground process group ID. The process group ID of the foreground process group. *X/Open*. *ISO.1*.

form-feed character. A character in the output stream that indicates that printing should start on the next page of an output device. The formfeed is the character designated by '\f' in the C and C++ language. If the formfeed is not the first character of an output line, the result is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the next page. *X/Open*.

forward declaration. A declaration of a class or function made earlier in a compilation unit, so that the declared class or function can be used before it has been defined.

freestanding application. (1) An application that is created to run without the run-time environment or library with which it was developed. (2) An z/OS C/C++ application that does not use the services of the dynamic z/OS C/C++ run-time library or of the Language Environment. Under z/OS C support, this ability is a feature of the System Programming C support.

free store. Dynamically allocated memory. New and delete are used to allocate and deallocate free store.

friend class. A class in which all the member functions are granted access to the private and protected members of another class. It is named in the declaration of another class and uses the keyword friend as a prefix to the class. For example, the following source code makes all the functions and data in class `you` friends of class `me`:

```
class me {
   friend class you;
     // ...
};
```

friend function. A function that is granted access to the private and protected parts of a class. It is named in the declaration of the other class with the prefix `friend`.

function. A named group of statements that can be called and evaluated and can return a value to the calling statement. *IBM*.

function call. An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by the called function. A function call contains the name of the function to which control moves and a parenthesized list of values. *IBM*.

function declarator. The part of a function definition that names the function, provides additional information about the return value of the function, and lists the function parameters. *IBM*.

function definition. The complete description of a function. A function definition contains a sequence of specifiers (storage class, optional type, inline, virtual, optional friend), a function declarator, optional contructor-initializers, parameter declarations, optional const, and the block statement. Inline, virtual, friend, and const are not available with C.

function prototype. A function declaration that provides type information for each parameter. It is the first line of the function (header) followed by a semicolon **(;)**. The declaration is required by the compiler at the time that the function is declared, so that the compiler can check the type.

function scope. Labels that are declared in a function have function scope and can be used anywhere in that function after their declaration.

function template. Provides a blueprint describing how a set of related individual functions can be constructed.

G

Generalization. Refers to a class, function, or static data member which derives its definition from a template. An instantiation of a template function would be a generalization.

Generalized Object File Format (GOFF). It is the strategic object module format for S/390. It extends the capabilities of object modules to contain more information than current object modules. It removes the limitations of the previous object module format and supports future enhancements. It is required for XPLINK.

generic class. Synonym for *class templates*.

global. Pertaining to information available to more than one program or subroutine. *IBM*.

global scope. Synonym for *file scope*.

global variable. A symbol defined in one program module that is used in other independently compiled program modules.

GMT (Greenwich Mean Time). The solar time at the meridian of Greenwich, formerly used as the prime basis of standard time throughout the world. GMT has been superseded by coordinated universal time (UTC).

graphic character. (1) A visual representation of a character, other than a control character, that is normally produced by writing, printing, or displaying. *ISO Draft*. (2) A character that can be displayed or printed. *IBM*.

Graphical Data Display Manager (GDDM). Pertaining to an IBM licensed program that provides a group of routines that allows pictures to be defined and displayed procedurally through function routines that correspond to graphic primitives. *IBM*.

Greenwich Mean Time. See GMT.

group ID. (1) A non-negative integer that is used to identify a group of system users. Each system user is a member of at least one group. When the identity of a group is associated with a process, a group ID value is referred to as a real group ID, an effective group ID, one of the supplementary group IDs or a saved set-group-ID. *X/Open*. (2) A non-negative integer, which can be contained in an object of type *gid_t*, that is used to identify a group of system users. *ISO.1*.

H

halfword. A contiguous sequence of bytes or characters that constitutes half a computer word and can be addressed as a unit. *IBM*.

hash function. A function that determines which category, or bucket, to put an element in. A hash function is needed when implementing a hash table.

hash table. (1) A data structure that divides all elements into (preferably) equal-sized categories, or buckets, to allow quick access to the elements. The hash function determines which bucket an element belongs in. (2) A table of information that is accessed by way of a shortened search key (that hash value). Using a hash table minimizes average search time.

header file. A text file that contains declarations used by a group of functions, programs, or users.

heap storage. An area of storage used for allocation of storage whose lifetime is not related to the execution of the current routine. The heap consists of the initial heap segment and zero or more increments.

hexadecimal constant. A constant, usually starting with special characters, that contains only hexadecimal digits. Three examples for the hexadecimal constant with value 0 would be '\x00', '0x0', or '0X00'.

High Level Assembler. An IBM licensed program. Translates symbolic assembler language into binary machine language.

hiperspace memory file. An IBM file used under z/OS to deal with memory files as large as 2 gigabytes. *IBM*.

hooks. Instructions inserted into a program by a compiler at compile-time. Using hooks, you can set break-points to instruct the Debug Tool to gain control of the program at selected points during its execution.

hybrid code. Program statements that have not been internationalized with respect to code page, especially where data constants contain variant characters. Such statements can be found in applications written in older implementations of MVS, which required syntax statements to be written using code page IBM-1047 exclusively. Such applications cannot be converted from one code page to another using `iconv()`.

I

I18N. Abbreviation for *internationalization*.

identifier. (1) One or more characters used to identify or name a data element and possibly to indicate certain properties of that data element. *ANSI/ISO*. (2) In programming languages, a token that names a data object such as a variable, an array, a record, a subprogram, or a function. *ANSI/ISO*. (3) A sequence of letters, digits, and underscores used to identify a data object or function. *IBM*.

if statement. A conditional statement that contains the keyword if, followed by an expression in parentheses (the condition), a statement (the action), and an optional else clause (the alternative action). *IBM*.

ILC (interlanguage call). A function call made by one language to a function coded in another language. Interlanguage calls are used to communicate between programs written in different languages.

ILC (interlanguage communication). The ability of routines written in different programming languages to communicate. ILC support enables the application writer to readily build applications from component routines written in a variety of languages.

implementation-defined behavior. Application behavior that is not defined by the standards. The implementing compiler and library defines this behavior when a program contains correct program constructs or uses correct data. Programs that rely on implementation-defined behavior may behave differently on different C or C++ implementations. Refer to the z/OS C/C++ books that are listed in “z/OS C/C++ and Related Publications” on page 4 for information about implementation-defined behavior in the z/OS C/C++ environment. Contrast with *unspecified behavior* and *undefined behavior*.

IMS (Information Management System). Pertaining to an IBM database/data communication (DB/DC) system that can manage complex databases and networks. *IBM*.

include directive. A preprocessor directive that causes the preprocessor to replace the statement with the contents of a specified file.

include file. See *header file*.

incomplete class declaration. A class declaration that does not define any members of a class. Until a class is fully declared, or defined, you can only use the class name where the size of the class is not required. Typically an incomplete class declaration is used as a forward declaration.

incomplete type. A type that has no value or meaning when it is first declared. There are three incomplete types: void, arrays of unknown size and structures and unions of unspecified content. A void type can never be completed. Arrays of unknown size and structures or unions of unspecified content can be completed in further declarations.

indirection. (1) A mechanism for connecting objects by storing, in one object, a reference to another object. (2) In the C and C++ languages, the application of the unary operator * to a pointer to access the object to which the pointer points.

indirection class. Synonym for *reference class*.

induction variable. It is a controlling variable of a loop.

inheritance. A technique that allows the use of an existing class as the base for creating other classes.

initial heap. The z/OS C/C++ heap controlled by the HEAP runtime option and designated by a `heap_id` of 0. The initial heap contains dynamically allocated user data.

initializer. An expression used to initialize data objects. The C++ language, supports the following types of initializers:

- An expression followed by an assignment operator that is used to initialize fundamental data type objects or class objects that contain copy constructors.
- A parenthesized expression list that is used to initialize base classes and members that use constructors.

Both the C and C++ languages support an expression enclosed in braces (`{ }`), that used to initialize aggregates.

inlined function. A function whose actual code replaces a function call. A function that is both declared and defined in a class definition is an example of an inline function. Another example is one which you explicitly declared inline by using the keyword `inline`. Both member and nonmember functions can be inlined.

input stream. A sequence of control statements and data submitted to a system from an input unit. Synonymous with input job stream, job input stream. *IBM*.

instance. An object-oriented programming term synonymous with object. An instance is a particular instantiation of a data type. It is simply a region of storage that contains a value or group of values. For example, if a class `box` is previously defined, two instances of a class `box` could be instantiated with the declaration: `box box1, box2;`

instantiate. To create or generate a particular instance or object of a data type. For example, an instance `box1` of class `box` could be instantiated with the declaration: `box box1;`

instruction. A program statement that specifies an operation to be performed by the computer, along with the values or locations of operands. This statement represents the programmer's request to the processor to perform a specific operation.

instruction scheduling. An optimization technique that reorders instructions in code to minimize execution time.

integer constant. A decimal, octal, or hexadecimal constant.

integral object. A character object, an object having an enumeration type, an object having variations of the type `int`, or an object that is a bit field.

Interactive System Productivity Facility. See *ISPF*.

interlanguage call. See *ILC (interlanguage call)*.

interlanguage communication. See *ILC (interlanguage communication)*.

internationalization. The capability of a computer program to adapt to the requirements of different native languages, local customs, and coded character sets. *X/Open*.

Synonymous with *I18N*.

interoperability. The capability to communicate, execute programs, or transfer data among various functional units in a way that requires the user to have little or no knowledge of the unique characteristics of those units.

Interprocedural Analysis. See *IPA*.

interprocess communication. (1) The exchange of information between processes or threads through semaphores, queues, and shared memory. (2) The process by which programs communicate data to each other to synchronize their activities. Semaphores, signals, and internal message queues are common methods of inter-process communication.

I/O Stream library. A class library that provides the facilities to deal with many varieties of input and output.

IPA (Interprocedural Analysis). A process for performing optimizations across compilation units.

ISPF (Interactive System Productivity Facility). An IBM licensed program that serves as a full-screen editor and dialogue manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user. (ISPF)

iteration. The process of repeatedly applying a function to a series of elements in a collection until some condition is satisfied.

J

JCL (job control language). A control language used to identify a job to an operating system and to describe the job's requirement. *IBM*.

K

keyword. (1) A predefined word reserved for the C and C++ languages, that may not be used as an identifier. (2) A symbol that identifies a parameter in JCL.

kind attribute. An attribute for a mutex attribute object. This attribute's value determines whether the mutex can be locked once or more than once for a thread and whether state changes to the mutex will be reported to the debug interface.

L

label. An identifier within or attached to a set of data elements. *ISO Draft*.

Language Environment. Abbreviated form of z/OS Language Environment. Pertaining to an IBM software product that provides a common runtime environment and runtime services to applications compiled by Language Environment-conforming compilers.

last element. The element visited last in an iteration over a collection. Each collection has its own definition for last element. For example, the last element of a sorted set is the element with the largest value.

late binding. Allowing the system to determine the specific class of the object and invoke the appropriate function implementations at run time. Late binding or dynamic binding hides the differences between a group of related classes from the application program.

leaves. Nodes without children. Synonymous with terminals.

lexically. Relating to the left-to-right order of units.

library. (1) A collection of functions, calls, subroutines, or other data. *IBM*. (2) A set of object modules that can be specified in a link command.

linkage editor. Synonym for linker. The linkage editor has been replaced by the *binder* for the MVS/ESA, OS/390, or z/OS operating systems. See *binder*.

Linkage. Refers to the binding between a reference and a definition. A function has internal linkage if the function is defined inline as part of the class, is declared with the inline keyword, or is a nonmember function declared with the static keyword. All other functions have external linkage.

linker. A computer program for creating load modules from one or more object modules by resolving cross references among the modules and, if necessary, adjusting addresses. *IBM*.

link pack area (LPA). In z/OS, an area of storage containing re-enterable routines from system libraries. Their presence in main storage saves loading time.

literal. (1) In programming languages, a lexical unit that directly represents a value; for example, 14 represents the integer fourteen, "APRIL" represents the string of characters APRIL, 3.0005E2 represents the number 300.05. *ISO-JTC1*. (2) A symbol or a quantity in a source program that is itself data, rather than a reference to data. *IBM*. (3) A character string whose value is given by the characters themselves; for example, the numeric literal 7 has the value 7, and the character literal CHARACTERS has the value CHARACTERS. *IBM*.

loader. A routine, commonly a computer program, that reads data into main storage. *ANSI/ISO*.

load module. All or part of a computer program in a form suitable for loading into main storage for execution. A load module is usually the output of a linkage editor. *ISO Draft*.

local. (1) In programming languages, pertaining to the relationship between a language object and a block such that the language object has a scope contained in that block. *ISO-JTC1*. (2) Pertaining to that which is defined and used only in one subdivision of a computer program. *ANSI/ISO*.

local customs. The conventions of a geographical area or territory for such things as date, time, and currency formats. *X/Open*.

locale. The definition of the subset of a user's environment that depends on language and cultural conventions. *X/Open*.

localization. The process of establishing information within a computer system specific to the operation of particular native languages, local customs, and coded character sets. *X/Open*.

local scope. A name declared in a block has scope within the block, and can therefore only be used in that block.

Long name. An external name C++ name in an object module, or and external name in an object module created by the C compiler when the `LONGNAME` option is used. Long names are up to 1024 characters long and may contain both upper-case and lower-case characters.

lvalue. An expression that represents a data object that can be both examined and altered.

M

macro. An identifier followed by arguments (may be a parenthesized list of arguments) that the preprocessor replaces with the replacement code located in a preprocessor `#define` directive.

macro call. Synonym for *macro*.

macro instruction. Synonym for *macro*.

main function. An external function with the identifier `main` that is the first user function—aside from exit routines and C++ static object constructors—to get control when program execution begins. Each C and C++ program must have exactly one function named `main`.

makefile. A text file containing a list of your application's parts. The make utility uses makefiles to maintain application parts and dependencies.

make utility. Maintains all of the parts and dependencies for your application. The make utility uses a makefile to keep the parts of your program synchronized. If one part of your application changes, the make utility updates all other files that depend on the changed part. This utility is available under the z/OS shell and by default, uses the c89 utility to recompile and bind your application.

mangling. The encoding during compilation of identifiers such as function and variable names to include type and scope information. These mangled names ensure type-safe linkage. See also *demangling*.

manipulator. A value that can be inserted into streams or extracted from streams to affect or query the behavior of the stream.

member. A data object or function in a structure, union, or class. Members can also be classes, enumerations, bit fields, and type names.

member function. (1) An operator or function that is declared as a member of a class. A member function has access to the private and protected data members and member functions of objects of its class. Member functions are also called methods. (2) A function that performs operations on a class.

method. In the C++ language, a synonym for *member function*.

migrate. To move to a changed operating environment, usually to a new release or version of a system. *IBM*.

module. A program unit that usually performs a particular function or related functions, and that is distinct and identifiable with respect to compiling, combining with other units, and loading.

multibyte character. A mixture of single-byte characters from a single-byte character set and double-byte characters from a double-byte character set.

multicharacter collating element. A sequence of two or more characters that collate as an entity. For example, in some coded character sets, an accented character is represented by a non-spacing accent, followed by the letter. Other examples are the Spanish elements *ch* and *ll*. *X/Open*.

multiple inheritance. An object-oriented programming technique implemented in the C++ language through derivation, in which the derived class inherits members from more than one base class.

multitasking. A mode of operation that allows concurrent performance, or interleaved execution of two or more tasks. *ISO-JTC1*. *ANSI/ISO*.

mutex. A flag used by a semaphore to protect shared resources. The mutex is locked and unlocked by threads in a program. A mutex can only be locked by one thread at a time and can only be unlocked by the same thread that locked it. The current owner of a mutex is the thread that it is currently locked by. An unlocked mutex has no current owner.

mutex attribute object. Allows the user to manage the characteristics of mutexes in their application by defining a set of values to be used for the mutex during its creation. A mutex attribute object allows the user to create many mutexes with the same set of characteristics without redefining the same set of characteristics for each mutex created.

mutex object. Used to identify a mutex.

N

name space. A category used to group similar types of identifiers.

named pipe. A FIFO file. Named pipes allow transfer of data between processes in a FIFO manner and synchronization of process execution. Allows processes to communicate even though they do not know what processes are on the other end of the pipe.

natural reentrancy. A program that contains no writable static and requires no additional processing to make it reentrant is considered naturally reentrant.

nested class. A class defined within the scope of another class.

nested enclave. A new enclave created by an existing enclave. The nested enclave that is created must be a new main routine within the process. See also *child enclave* and *parent enclave*.

newline character. A character that in the output stream indicates that printing should start at the beginning of the next line. The newline character is designated by '\n' in the C and C++ language. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the next line. *X/Open*.

nickname. Synonym for alias.

nonprinting character. See *control character*.

null character (NUL). The ASCII or EBCDIC character '\0' with the hex value `00`, all bits turned off. It is used to represent the absence of a printed or displayed character. This character is named <NUL> in the portable character set.

null pointer. The value that is obtained by converting the number 0 into a pointer; for example, `(void *) 0`. The C and C++ languages guarantee that this value will not match that of any legitimate pointer, so it is used by many functions that return pointers to indicate an error. *X/Open*.

null statement. A C or C++ statement that consists solely of a semicolon.

null string. (1) A string whose first byte is a null byte. Synonymous with *empty string*. *X/Open*. (2) A character array whose first element is a null character. *ISO.1*.

null value. A parameter position for which no value is specified. *IBM*.

null wide-character code. A wide-character code with all bits set to zero. *X/Open*.

number sign. The character #, also known as *pound sign* and *hash sign*. This character is named <number-sign> in the portable character set.

O

object. (1) A region of storage. An object is created when a variable is defined. An object is destroyed when it goes out of scope. (See also *instance*.) (2) In object-oriented design or programming, an abstraction consisting of data and the operations associated with that data. See also *class*. *IBM*. (3) An instance of a class.

object code. Machine-executable instructions, usually generated by a compiler from source code written in a higher level language (such as the C++ language). For programs that must be linked, object code consists of relocatable machine code.

object module. (1) All or part of an object program sufficiently complete for linking. Assemblers and compilers usually produce object modules. *ISO Draft*. (2) A set of instructions in machine language produced by a compiler from a source program. *IBM*.

object-oriented programming. A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates not on how something is accomplished, but on what data objects comprise the problem and how they are manipulated.

octal constant. The digit 0 (zero) followed by any digits 0 through 7.

open file. A file that is currently associated with a file descriptor. *X/Open*. *ISO.1*.

operand. An entity on which an operation is performed. *ISO-JTC1*. *ANSI/ISO*.

operating system (OS). Software that controls functions such as resource allocation, scheduling, input/output control, and data management.

operator function. An overloaded operator that is either a member of a class or that takes at least one argument that is a class type or a reference to a class type.

operator precedence. In programming languages, an order relation defining the sequence of the application of operators within an expression. *ISO-JTC1*.

orientation of a stream. After application of an input or output function to a stream, it becomes either byte-oriented or wide-oriented. A byte-oriented stream is a stream that had a byte input or output function applied to it when it had no orientation. A wide-oriented stream is a stream that had a wide character input or output function applied to it when it had no orientation. A stream has no orientation when it has been associated with an external file but has not had any operations performed on it.

overflow. (1) A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage. (2) That portion of an operation that exceeds the capacity of the intended unit of storage. *IBM*.

overlay. The technique of repeatedly using the same areas of internal storage during different stages of a program. *ANSI/ISO*. Unions are used to accomplish this in C and C++.

overloading. An object-oriented programming technique that allows you to redefine functions and most standard C++ operators when the functions and operators are used with class types.

P

parameter. (1) In the C and C++ languages, an object declared as part of a function declaration or definition that acquires a value on entry to the function, or an identifier following the macro name in a function-like macro definition. *X/Open*. (2) Data passed between programs or procedures. *IBM*.

parameter declaration. A description of a value that a function receives. A parameter declaration determines the storage class and the data type of the value.

parent enclave. The enclave that issues a call to system services or language constructs to create a nested or child enclave. See also *child enclave* and *nested enclave*.

parent process. (1) The program that originates the creation of other processes by means of `spawn` or `exec` function calls. See also *child process*. (2) A process that creates other processes.

parent process ID. (1) An attribute of a new process identifying the parent of the process. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an implementation-dependent system process. *X/Open*. (2) An attribute of a new process after it is created by a currently active process. *ISO.1*.

partitioned concatenation. Specifying multiple PDSs or PDSEs under one ddname. The concatenated data sets act as one big PDS or PDSE and access can be made to any member with a unique name. An attempted access to a member whose name occurs more than once in the concatenated data sets, returns the first member with that name found in the entire concatenation.

partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. *IBM*.

partitioned data set extended (PDSE). Similar to *partitioned data set*, but with extended capabilities.

path name. (1) A string that is used to identify a file. A path name consists of, at most, {PATH_MAX} bytes, including the terminating null character. It has an optional beginning slash, followed by zero or more file names separated by slashes. If the path name refers to a directory, it may also have one or more trailing slashes. Multiple successive slashes are treated as one slash. A path name that begins with two successive slashes may be interpreted in an implementation-dependent manner, although more than two leading slashes are treated as a single slash. The interpretation of the path name is described in *path name resolution*. *ISO.1*. (2) A file name specifying all directories leading to the file.

path name resolution. Path name resolution is performed for a process to resolve a path name to a particular file in a file hierarchy. There may be multiple path names that resolve to the same file. *X/Open*.

pattern. A sequence of characters used either with regular expression notation or for path name expansion, as a means of selecting various characters strings or path names, respectively. The syntaxes of the two patterns are similar, but not identical. *X/Open*.

period. The character (**.**). The term *period* is contrasted against *dot*, which is used to describe a specific directory entry. This character is named <period> in the portable character set.

permissions. Codes that determine how a file can be used by any users who work on the system. See also *file access permissions*. *IBM*.

persistent environment. A program can explicitly establish a persistent environment, direct functions to it, and explicitly terminate it.

pointer. In the C and C++ languages, a variable that holds the address of a data object or a function. *IBM*.

pointer class. A class that implements pointers.

pointer to member. An operator used to access the address of non-static members of a class.

polymorphism. The technique of taking an abstract view of an object or function and using any concrete objects or arguments that are derived from this abstract view.

portable character set. The set of characters specified in POSIX 1003.2, section 2.4:

<NUL>	
<alert>	
<backspace>	
<tab>	
<newline>	
<vertical-tab>	
<form-feed>	
<carriage-return>	
<space>	
<exclamation-mark>	!
<quotation-mark>	"
<number-sign>	#
<dollar-sign>	$
<percent-sign>	%
<ampersand>	&
<apostrophe>	'
<left-parenthesis>	(
<right-parenthesis>	)
<asterisk>	*
<plus-sign>	+
<comma>	,
<hyphen>	–
<hyphen-minus>	–
<period>	.
<slash>	/
<zero>	0
<one>	1
<two>	2
<three>	3
<four>	4
<five>	5
<six>	6
<seven>	7
<eight>	8
<nine>	9
<colon>	:
<semicolon>	;
<less-than-sign>	<
<equals-sign>	=
<greater-than-sign>	>
<question-mark>	?
<commercial-at>	@
<A>	A
<B>	B
<C>	C
<D>	D
<E>	E
<F>	F
<G>	G
<H>	H
<I>	I
<J>	J
<K>	K
<L>	L
<M>	M
<N>	N
<O>	O
<P>	P
<Q>	Q
<R>	R
<S>	S
<T>	T
<U>	U
<V>	V
<W>	W
<X>	X
<Y>	Y
<Z>	Z
<left-square-bracket>	[
<backslash>	\
<reverse-solidus>	\
<right-square-bracket>	]
<circumflex>	ˆ
<circumflex-accent>	ˆ
<underscore>	_
<low-line>	_
<grave-accent>	`
<a>	a
<b>	b
<c>	c
<d>	d
<e>	e
<f>	f
<g>	g
<h>	h

`<i>`	`i`
`<j>`	`j`
`<k>`	`k`
`<l>`	`l`
`<m>`	`m`
`<n>`	`n`
`<o>`	`o`
`<p>`	`p`
`<q>`	`q`
`<r>`	`r`
`<s>`	`s`
`<t>`	`t`
`<u>`	`u`
`<v>`	`v`
`<w>`	`w`
`<x>`	`x`
`<y>`	`y`
`<z>`	`z`
`<left-brace>`	`{`
`<left-curly-bracket>`	`{`
`<vertical-line>`	`|`
`<right-brace>`	`}`
`<right-curly-bracket>`	`}`
`<tilde>`	`~`

portable file name character set. The set of characters from which portable file names are constructed. For a file name to be portable across implementations conforming to the ISO POSIX-1 standard and to ISO/IEC 9945, it must consists only of the following characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -
```

The last three characters are the period, underscore, and hyphen characters, respectively. The hyphen must not be used as the first character of a portable file name. Upper- and lower-case letters retain their unique identities between conforming implementations. In the case of a portable path name, the slash character may also be used. *X/Open*. *ISO.1*.

portability. The ability of a programming language to compile successfully on different operating systems without requiring changes to the source code.

positional parameter. A parameter that must appear in a specified location relative to other positional parameters. *IBM*.

precedence. The priority system for grouping different types of operators with their operands.

predefined macros. Frequently used routines provided by an application or language for the programmer.

preinitialization. A process by which an environment or library is initialized once and can then be used repeatedly to avoid the inefficiency of initializing the environment or library each time it is needed.

prelinker. A utility provided with z/OS Language Environment that you can use to process application programs that require DLL support, or contain either constructed reentrancy or external symbol names that are longer than 8 characters. You require the prelinker, or its equivalent function which is provided by the binder, to process all C++ applications, or C applications that are compiled with the RENT, DLL, LONGNAME or IPA options. As of Version 2 Release 4, the prelinker was superseded by the binder. See also *binder*.

preprocessor. A phase of the compiler that examines the source program for preprocessor statements that are then executed, resulting in the alteration of the source program.

preprocessor statement. In the C and C++ languages, a statement that begins with the symbol `#` and is interpreted by the preprocessor during compilation. *IBM*.

primary expression. (1) An identifier, parenthesized expression, function call, array element specification, structure member specification, or union member specification. *IBM*. (2) Literals, names, and names qualified by the :: (scope resolution) operator.

printable character. One of the characters included in the print character classification of the LC_CTYPE category in the current locale. *X/Open*.

private. Pertaining to a class member that is only accessible to member functions and friends of that class.

process. (1) An instance of an executing application and the resources it uses. (2) An address space and single thread of control that executes within that address space, and its required system resources. A process is created by another process issuing the `fork()` function. The process that issues the `fork()` function is known as the parent process, and the new process created by the `fork()` function is known as the child process. *X/Open*. *ISO.1*.

process group. A collection of processes that permits the signaling of related processes. Each process in the system is a member of a process group that is identified by the process group ID. A newly created process joins the process group of its creator. *IBM*. *X/Open*. *ISO.1*.

process group ID. The unique identifier representing a process group during its lifetime. A process group ID is a positive integer. (Under ISO only, it is a positive integer *that can be contained in a pid_t*.) A process group ID will not be reused by the system until the process group lifetime ends. *X/Open*. *ISO.1*.

process group lifetime. A period of time that begins when a process group is created and ends when the last remaining process in the group leaves the group, because either it is the end of the last process' lifetime or the last remaining process is calling the `setsid()` or `setpgid()` functions. *X/Open*. *ISO.1*.

process ID. The unique identifier representing a process. A process ID is a positive integer. (Under ISO only, it is a positive integer *that can be contained in a pid_t*.) A process ID will not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID will not be reused by the system until the process group lifetime ends. A process that is not a system process will not have a process ID of 1. *X/Open*. *ISO.1*.

process lifetime. The period of time that begins when a process is created and ends when the process ID is returned to the system. After a process is created with a `fork()` function, it is considered active. Its thread of control and address space exist until it terminates. It then enters an inactive state where certain resources may be returned to the system, although some resources, such as the process ID, are still in use. When another process executes a `wait()` or `waitpid()` function for an inactive process, the remaining resources are returned to the system. The last resource to be returned to the system is the process ID. At this time, the lifetime of the process ends. *X/Open*. *ISO.1*.

program object. All or part of a computer program in a from suitable for loading into main storage for execution. A program object is the output of the z/OS Binder and is a newer more flexible format (e.g. longer external names) than a load module.

protected. Pertaining to a class member that is only accessible to member functions and friends of that class, or to member functions and friends of classes derived from that class.

prototype. A function declaration or definition that includes both the return type of the function and the types of its parameters. See *function prototype*.

public. Pertaining to a class member that is accessible to all functions.

pure virtual function. A virtual function that has a function definition of `= 0;`. See also *abstract classes*.

Q

qualified class name. Any class name or class name qualified with one or more :: (scope resolution) operators.

qualified name. Used to qualify a nonclass type name such as a member by its class name.

qualified type name. Used to reduce complex class name syntax by using typedefs to represent qualified class names.

Query Management Facility (QMF). Pertaining to an IBM query and report writing facility that enables a variety of tasks such as data entry, query building, administration, and report analysis. *IBM*.

queue. A sequence with restricted access in which elements can only be added at the back end (or bottom) and removed from the front end (or top). A queue is characterized by first-in, first-out behavior and chronological order.

quotation marks. The characters " and ‘, also known as *double-quote* and *single-quote* respectively. *X/Open*.

R

radix character. The character that separates the integer part of a number from the fractional part. *X/Open*.

real group ID. The attribute of a process that, at the time of process creating, identifies the group of the user who created the process. This value is subject to change during the process lifetime, as describe in `setgid()`. *X/Open*. *ISO.1*.

real user ID. The attribute of a process that, at the time of process creation, identifies the user who created the process. This value is subject to change during the process lifetime, as described in `setuid()`. *X/Open*. *ISO.1*.

reason code. A code that identifies the reason for a detected error. *IBM*.

reassociation. An optimization technique that rearranges the sequence of calculations in a subscript expression producing more candidates for common expression elimination.

redirection. In the shell, a method of associating files with the input or output of commands. *X/Open*.

reentrant. The attribute of a program or routine that allows the same copy of a program or routine to be used concurrently by two or more tasks.

reference class. A class that links a concrete class to an abstract class. Reference classes make polymorphism possible with the Collection Classes. Synonymous with *indirection class*.

refresh. To ensure that the information on the user's terminal screen is up-to-date. *X/Open*.

register storage class specifier. A specifier that indicates to the compiler within a block scope data definition, or a parameter declaration, that the object being described will be heavily used.

register variable. A variable defined with the register storage class specifier. Register variables have automatic storage.

regular expression. (1) A mechanism to select specific strings from a set of character strings. (2) A set of characters, meta-characters, and operators that define a string or group of strings in a search pattern. (3) A string containing wildcard characters and operations that define a set of one or more possible strings.

regular file. A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system. *X/Open*. *ISO.1*.

relation. An unordered flat collection class that uses keys, allows for duplicate elements, and has element equality.

relative path name. The name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory. See *path name resolution*. *IBM*.

reserved word. (1) In programming languages, a keyword that may not be used as an identifier. *ISO-JTC1*. (2) A word used in a source program to describe an action to be taken by the program or compiler. It must not appear in the program as a user-defined name or a system name. *IBM*.

RMODE (residency mode). In z/OS, a program attribute that refers to where a module is prepared to run. RMODE can be 24 or ANY. ANY refers to the fact that the module can be loaded either above or below the 16M line. RMODE 24 means the module expects to be loaded below the 16M line.

runtime library. A compiled collection of functions whose members can be referred to by an application program during runtime execution. Typically used to refer to a dynamic library that is provided in object code, such that references to the library are resolved during the linking step. The runtime library itself is not statically bound into the application modules.

S

saved set-group-ID. An attribute of a process that allows some flexibility in the assignment of the effective group ID attribute, as described in the `exec()` family of functions and `setgid()`. *X/Open*. *ISO.1*.

saved set-user-ID. An attribute of a process that allows some flexibility in the assignment of the effective user ID attribute, as described in `exec()` and `setuid()`. *X/Open*. *ISO.1*.

scalar. An arithmetic object, or a pointer to an object of any type.

scope. (1) That part of a source program in which a variable is visible. (2) That part of a source program in which an object is defined and recognized.

scope operator (::). An operator that defines the scope for the argument on the right. If the left argument is blank, the scope is global; if the left argument is a class name, the scope is within that class. Synonymous with *scope resolution operator*.

scope resolution operator (::). Synonym for *scope operator*.

semaphore. An object used by multi-threaded applications for signalling purposes and for controlling access to serially reusable resources. Processes can be locked to a resource with semaphores if the processes follow certain programming conventions.

sequence. A sequentially ordered flat collection.

sequential concatenation. Multiple sequential data sets or partitioned data-set members are treated as one long sequential data set. In the case of sequential data sets, you can access or update the data sets in order. In the case of partitioned data-set members, you can access or update the members in order. Repositioning is possible if all of the data sets in the concatenation support repositioning.

sequential data set. A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. *IBM*.

session. A collection of process groups established for job control purposes. Each process group is a member of a session. A process is a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership; see `setsid()`. There can be multiple process groups in the same session. *X/Open*. *ISO.1*.

shell. A program that interprets sequences of text input as commands. It may operate on an input stream or it may interactively prompt and read commands from a terminal. *X/Open*.

This feature is provided as part of the z/OS Shell and Utilities feature licensed program.

Short name. An external non-C++ name in an object module produced by compiling with the NOLONGNAME option. Such a name is up to 8 characters long and single case.

signal. (1) A condition that may or may not be reported during program execution. For example, `SIGFPE` is the signal used to represent erroneous arithmetic operations such as a division by zero. (2) A mechanism by which a process may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term *signal* is also used to refer to the event itself. *X/Open*. *ISO.1*. (3) A method of interprocess communication that simulates software interrupts. *IBM*.

signal handler. A function to be called when the signal is reported.

single-byte character set (SBCS). A set of characters in which each character is represented by a one-byte code. *IBM*.

single-precision. Pertaining to the use of one computer word to represent a number in accordance with the required precision. *ISO-JTC1*. *ANSI/ISO*.

single-quote. The character ‘, also known as *apostrophe*. This character is named <quotation-mark> in the portable character set.

slash. The character /, also known as *solidus*. This character is named <slash> in the portable character set.

socket. (1) A unique host identifier created by the concatenation of a port identifier with a transmission control protocol/Internet protocol (TCP/IP) address. (2) A port identifier. (3) A 16-bit port-identifier. (4) A port on a specific host; a communications end point that is accessible though a protocol family's addressing mechanism. A socket is identified by a socket address. *IBM*.

sorted map. A sorted flat collection with key and element equality.

sorted relation. A sorted flat collection that uses keys, has element equality, and allows duplicate elements.

sorted set. A sorted flat collection with element equality.

source module. A file that contains source statements for such items as high-level language programs and data description specifications. *IBM*.

source program. A set of instructions written in a programming language that must be translated to machine language before the program can be run. *IBM*.

space character. The character defined in the portable character set as <space>. The space character is a member of the space character class of the current locale, but represents the single character, and not all of the possible members of the class. *X/Open*.

spanned record. A logical record contained in more than one block. *IBM*.

specialization. A user-supplied definition which replaces a corresponding template instantiation.

specifiers. Used in declarations to indicate storage class, fundamental data type and other properties of the object or function being declared.

spill area. A storage area used to save the contents of registers. *IBM*.

SQL (Structured Query Language). A language designed to create, access, update and free data tables.

square brackets. The characters [(left bracket) and] (right bracket). Also see *brackets*.

stack frame. The physical representation of the activation of a routine. The stack frame is allocated and freed on a LIFO (last in, first out) basis. A stack is a collection of one or more stack segments consisting of an initial stack segment and zero or more increments.

stack storage. Synonym for *automatic storage*.

standard error. An output stream usually intended to be used for diagnostic messages. *X/Open*.

standard input. (1) An input stream usually intended to be used for primary data input. *X/Open*. (2) The primary source of data entered into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command. *IBM*.

standard output. (1) An output stream usually intended to be used for primary data output. *X/Open*. (2) The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can go to a file or to another command. *IBM*.

statement. An instruction that ends with the character **;** (semicolon) or several instructions that are surrounded by the characters { and }.

static. A keyword used for defining the scope and linkage of variables and functions. For internal variables, the variable has block scope and retains its value between function calls. For external values, the variable has file scope and retains its value within the source file. For class variables, the variable is shared by all objects of the class and retains its value within the entire program.

static binding. The act of resolving references to external variables and functions before run time.

storage class specifier. One of the terms used to specify a storage class, such as auto, register, static, or extern.

stream. (1) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format. (2) A file access object that allows access to an ordered sequence of characters, as described by the ISO C standard. Such objects can be created by the `fdopen()` or `fopen()` functions, and are associated with a file descriptor. A stream provides the additional services of user-selectable buffering and formatted input and output. *X/Open*.

string. A contiguous sequence of bytes terminated by and including the first null byte. *X/Open*.

string constant. Zero or more characters enclosed in double quotation marks.

string literal. Zero or more characters enclosed in double quotation marks.

striped data set. A special data set organization that spreads a data set over a specified number of volumes so that I/O parallelism can be exploited. Record *n* in a striped data set is found on a volume separate from the volume containing record *n - p*, where *n > p*.

struct. An aggregate of elements having arbitrary types.

structure. A construct (a class data type) that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types. A structure can be used in all places a class is used. The initial projection is public.

structure tag. The identifier that names a structure data type.

Structured Query Language. See *SQL*.

stub routine. A routine, within a runtime library, that contains the minimum lines of code required to locate a given routine at run time.

subprogram. In the IPA Link version of the Inline Report listing section, an equivalent term for 'function'.

subscript. One or more expressions, each enclosed in brackets, that follow an array name. A subscript refers to an element in an array.

subsystem. A secondary or subordinate system, usually capable of operating independently of or asynchronously with, a controlling system. *ISO Draft*.

subtree. A tree structure created by arbitrarily denoting a node to be the root node in a tree. A subtree is always part of a whole tree.

superset. Given two sets A and B, A is a superset of B if and only if all elements of B are also elements of A. That is, A is a superset of B if B is a subset of A.

support. In system development, to provide the necessary resources for the correct operation of a functional unit. *IBM*.

switch expression. The controlling expression of a switch statement.

switch statement. A C or C++ language statement that causes control to be transferred to one of several statements depending on the value of an expression.

system default. A default value defined in the system profile. *IBM*.

system process. (1) An implementation-dependent object, other than a process executing an application, that has a process ID. *X/Open*. (2) An object, other than a process executing an application, that is defined by the system, and has a process ID. *ISO.1*.

T

tab character. A character that in the output stream indicates that printing or displaying should start at the next horizontal tabulation position on the current line. The tab is the character designated by '\t' in the C language. If the current position is at or past the last defined horizontal tabulation position, the behavior is unspecified. It is unspecified whether the character is the exact sequence transmitted to an output device by the system to accomplish the tabulation. *X/Open*.

This character is named <tab> in the portable character set.

task library. A class library that provides the facilities to write programs that are made up of tasks.

template. A family of classes or functions with variable types.

template class. A class instance generated by a class template.

Template Declaration. A prototype of a template which can optionally include a template definition.

Template Definition. A blueprint the compiler uses to generate a template instantiation.

template function. A function generated by a function template.

Template Instantiation. Compiler generated code for a class or function using the referenced types and the corresponding class or function template definition.

terminals. Synonym for *leaves*.

text file. A file that contains characters organized into one or more lines. The lines must not contain NUL characters and none can exceed {LINE_MAX}—which is defined in limits.h—bytes in length, including the new-line character. The term *text file* does not prevent the inclusion of control or other unprintable characters (other than NUL). *X/Open*.

thread. The smallest unit of operation to be performed within a process. *IBM*.

throw expression. An argument to the C++ exception being thrown.

tilde. The character ˜. This character is named <tilde> in the portable character set.

token. The smallest independent unit of meaning of a program as defined either by a parser or a lexical analyzer. A token can contain data, a language keyword, an identifier, or other parts of language syntax. *IBM*.

traceback. A section of a dump that provides information about the stack frame, the program unit address, the entry point of the routine, the statement number, and the status of the routines on the call-chain at the time the traceback was produced.

trigraph sequence. An alternative spelling of some characters to allow the implementation of C in character sets that do not provide a sufficient number of non-alphabetic graphics. *ANSI/ISO*.

Before preprocessing, each trigraph sequence in a string or literal is replaced by the single character that it represents.

truncate. To shorten a value to a specified length.

try block. A block in which a known C++ exception is passed to a handler.

type definition. A definition of a name for a data type. *IBM*.

type specifier. Used to indicate the data type of an object or function being declared.

U

ultimate consumer. The target of data in an I/O operation. An ultimate consumer can be a file, a device, or an array of bytes in memory.

ultimate producer. The source of data in an I/O operation. An ultimate producer can be a file, a device, or an array of byes in memory.

unary expression. An expression that contains one operand. *IBM*.

undefined behavior. Action by the compiler and library when the program uses erroneous constructs or contains erroneous data. Permissible undefined behavior includes ignoring the situation completely with unpredictable results. It also includes behaving in a documented manner that is characteristic of the environment, during translation or program execution, with or without issuing a diagnostic message. It can also include terminating a translation or execution, while issuing a diagnostic message. Contrast with *unspecified behavior* and *implementation-defined behavior*.

underflow. (1) A condition that occurs when the result of an operation is less than the smallest possible nonzero number. (2) Synonym for arithmetic underflow, monadic operation. *IBM.*

union. (1) In the C or C++ language, a variable that can hold any one of several data types, but only one data type at a time. *IBM*. (2) For bags, there is an additional rule for duplicates: If bag P contains an element *m* times and bag Q contains the same element *n* times, then the union of P and Q contains that element *m+n* times.

union tag. The identifier that names a union data type.

unnamed pipe. A pipe that is accessible only by the process that created the pipe and its child processes. An unnamed pipe does not have to be opened before it can be used. It is a temporary file that lasts only until the last file descriptor that uses it is closed.

unique collection. A collection in which the value of an element only occurs once; that is, there are no duplicate elements.

unrecoverable error. An error for which recovery is impossible without use of recovery techniques external to the computer program or run.

unspecified behavior. Action by the compiler and library when the program uses correct constructs or data, for which the standards impose no specific requirements. Such action should not cause compiler or application failure. You should not, however, write any programs to rely on such behavior as they may not be portable to other systems. Contrast with *implementation-defined behavior* and *undefined behavior*.

user-defined data type. (1) A mathematical model that includes a structure for storing data and operations that can be performed on that data. Common abstract data types include sets, trees, and heaps. (2) See also *abstract data type*.

user ID. A nonnegative integer that is used to identify a system user. (Under ISO only, a nonnegative integer, which can be contained in an object of type *uid_t*.) When the identity of a user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or (under ISO only, and there optionally) a saved set-user ID. *X/Open*. *ISO.1*.

user name. A string that is used to identify a user. *ISO.1*.

user prefix. In the z/OS environment, the user prefix is typically the user's logon user identification.

V

value numbering. An optimization technique that involves local constant propagation, local expression elimination, and folding several instructions into a single instruction.

variable. In programming languages, a language object that may take different values, one at a time. The values of a variable are usually restricted to a certain data type. *ISO-JTC1*.

variant character. A character whose hexadecimal value differs between different character sets. On EBCDIC systems, such as S/390, these 13 characters are an exception to the portability of the portable character set.

```
<left-square-bracket>  [
<right-square-bracket> ]
<left-brace>           {
<right-brace>          }
<backslash>            \
<circumflex>           ^
<tilde>                ~
<exclamation-mark>     !
<number-sign>          #
<vertical-line>        |
<grave-accent>         `
<dollar-sign>          $
<commercial-at>        @
```

vertical-tab character. A character that in the output stream indicates that printing should start at the next vertical tabulation position. The vertical-tab is the character designated by '\v' in the C or C++ languages. If the current position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the tabulation. *X/Open*. This character is named <vertical-tab> in the portable character set.

virtual address space. In virtual storage systems, the virtual storage assigned to a batched or terminal job, a system task, or a task initiated by a command.

virtual function. A function of a class that is declared with the keyword `virtual`. The implementation that is executed when you make a call to a virtual function depends on the type of the object for which it is called, which is determined at run time.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed and variable length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

visible. Visibility of identifiers is based on scoping rules and is independent of *access.*

volatile attribute. (1) In the C or C++ language, the keyword *volatile*, used in a definition, declaration, or cast. It causes the compiler to place the value of the data object in storage and to reload this value at each reference to the data object. *IBM*. (2) An attribute of a data object that indicates the object is changeable. Any expression referring to a volatile object is evaluated immediately (for example, assignments).

W

while statement. A looping statement that contains the keyword *while* followed by an expression in parentheses (the condition) and a statement (the action). *IBM*.

white space. (1) Space characters, tab characters, form-feed characters, and new-line characters. (2) A sequence of one or more characters that belong to the space character class as defined via the LC_CTYPE category in the current locale. In the POSIX locale, white space consists of one or more blank characters (space and tab characters), new-line characters, carriage-return characters, form-feed characters, and vertical-tab characters. *X/Open*.

wide-character. A character whose range of values can represent distinct codes for all members of the largest extended character set specified among the supporting locales.

wide-character code. An integral value corresponding to a single graphic symbol or control code. *X/Open*.

wide-character string. A contiguous sequence of wide-character codes terminated by and including the first null wide-character code. *X/Open*.

wide-oriented stream. See *orientation of a stream*.

word. A character string considered as a unit for a given purpose. In S/390, a word is 32 bits or 4 bytes.

working directory. Synonym for *current working directory*.

writable static area. See WSA.

write. (1) To output characters to a file, such as standard output or standard error. Unless otherwise stated, standard output is the default output destination for all uses of the term *write*. *X/Open*. (2) To make a permanent or transient recording of data in a storage device or on a data medium. *ISO-JTC1*. *ANSI/ISO*.

WSA (writable static area). An area of memory in the program that is modifyable during program execution. Typically, this area contains global variables and function and variable descriptors for DLLs.

X

XPLINK (Extra Performance Linkage). A new call linkage between functions that has the potential for a significant performance increase when used in an environment of frequent calls between small functions. XPLINK makes subroutine calls more efficient by removing nonessential instructions from the main path.

When all functions are compiled with the XPLINK option, pointers can be used without restriction, which makes it easier to port new applications to S/390.

Z

z/OS UNIX System Services (z/OS UNIX). An element of the z/OS operating system, (formerly known as OpenEdition). z/OS UNIX includes a POSIX system Application Programming Interface for the C language, a shell and utilities component, and a dbx debugger. All the components conform to IEEE POSIX standards (ISO 9945-1: 1990/IEEE POSIX 1003.1-1990, IEEE POSIX 1003.1a, IEEE POSIX 1003.2, and IEEE POSIX 1003.4a).

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 *IBM Online Library Omnibus Edition z/OS Collection*, SK2T-6700, 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
- *z/OS C/C++ Language Reference*, SC09-4764
- *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*, SC09-4763
- *z/OS C/C++ Reference Summary*, SX09-1319
- *OS/390 C/C++ IBM Open Class Library User's Guide*, SC09-2363
- *OS/390 C/C++ IBM Open Class Library Reference*, SC09-2364
- *Debug Tool User's Guide and Reference*, SC09-2137

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

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
- *Programming Guide*, SC26-9049
- *Language Reference*, SC26-9046
- *Diagnosis Guide*, GC26-9047
- *Licensed Program Specifications*, GC26-9044
- *Installation and Customization under z/OS*, GC26-9045
- *Millenium Language Extensions*, GC26-9266

PL/I

- *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*, SC26-8957
- *DB2 Application Programming and SQL Guide*, SC26-8958
- *DB2 Call Level Interface Guide and Reference*, SC26-8959
- *DB2 Command Reference*, SC26-8960
- *DB2 Data Sharing: Planning and Administration*, SC26-8961
- *DB2 Installation Guide*, GC26-8970
- *DB2 Messages and Codes*, GC26-8979
- *DB2 Reference for Remote DRDA Requesters and Servers*, SC26-8964
- *DB2 SQL Reference*, SC26-8966
- *DB2 Utility Guide and Reference*, SC26-8967

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

Printed in the United States of America

SC09-4767-00