

OS/390



# C/C++ User's Guide



OS/390



# C/C++ User's Guide

**Note!**

Before using this information and the product it supports be sure to read the general information under "Notices" on page xv.

**Fourth Edition, September 1998**

This edition applies to Version 2 Release 6 of OS/390 C/C++ (5647-A01) and to all subsequent releases and modifications until otherwise indicated in new editions or other updated documentation. 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. Note that the OS/390 C/C++ publications are available through the OS/390 Library page at: <http://www.s390.ibm.com/os390/bkserv>.

IBM welcomes your comments. You can send your comments electronically to the network ID listed below. Be sure to include your entire network address if you wish a reply.

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

IBMLink: [toribm\(torrcf\)](mailto:toribm(torrcf)@ca.ibm.com)

IBM/PROFS: [torolab4\(torrcf\)](mailto:torolab4(torrcf)@ca.ibm.com)

IBMMAIL: [ibmmail\(caibmwt9\)](mailto:ibmmail(caibmwt9)@ca.ibm.com)

To send your comments by facsimile (attention: RCF coordinator) use the following FAX numbers:

United States and Canada: 416-448-6161

Other Countries: (+1)-416-448-6161

Alternatively, you can use the Reader's Comment Form that is provided at the back of this publication, or mail your comments directly to:

IBM Canada Ltd. Laboratory

Information Development

2G/345/1150/TOR

1150 Eglinton Avenue East

North York, 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.

© Copyright International Business Machines Corporation 1996, 1999. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

Notices . . . . .	xv
Standards . . . . .	xv
Trademarks . . . . .	xvi

---

## Part 1. Introduction . . . . . 1

<b>Chapter 1. About This Book . . . . .</b>	<b>3</b>
IBM OS/390 C/C++ and Related Publications . . . . .	4
Hardcopy Books . . . . .	9
Softcopy Books . . . . .	9
Softcopy Examples . . . . .	9
OS/390 C/C++ on the World Wide Web . . . . .	10
C/C++ News... . . . .	10
How to Read the Syntax Diagrams . . . . .	11
 <b>Chapter 2. About IBM OS/390 C/C++ . . . . .</b>	 <b>15</b>
Changes for Version 2 Release 6. . . . .	15
The C/C++ Compilers . . . . .	16
The C Language . . . . .	16
The C++ Language . . . . .	16
Common Features of the OS/390 C and C++ Compilers . . . . .	17
OS/390 C Compiler Specific Features . . . . .	18
Features That Are Specific to the OS/390 C++ Compiler . . . . .	18
Utilities . . . . .	19
Class Libraries . . . . .	19
Class Library Source . . . . .	20
The Debug Tool . . . . .	20
OS/390 Language Environment . . . . .	21
The Program Management Binder . . . . .	21
OS/390 UNIX System Services (OS/390 UNIX) . . . . .	22
OS/390 C/C++ Applications with OS/390 UNIX C/C++ Functions . . . . .	23
Input and Output . . . . .	24
I/O Interfaces . . . . .	24
File Types . . . . .	25
Additional I/O Features . . . . .	25
The System Programming C Facility. . . . .	26
Interaction with Other IBM Products . . . . .	26
Additional Features of OS/390 C/C++ . . . . .	27
 <b>Chapter 3. Important Changes to the Prelinker Documentation . . . . .</b>	 <b>31</b>

---

## Part 2. User's Reference . . . . . 33

<b>Chapter 4. OS/390 C Example . . . . .</b>	<b>35</b>
Example of an OS/390 C Program . . . . .	35
CBC3UAAM . . . . .	35
CBC3UAAN . . . . .	36
Compiling, Binding, and Running the OS/390 C Example . . . . .	37
Under OS/390 Batch . . . . .	37
Under TSO . . . . .	37
Under the OS/390 Shell . . . . .	38

<b>Chapter 5. OS/390 C++ Examples</b>	39
Example of an OS/390 C++ Program	39
CBC3UBRH	40
CBC3UBRC	41
Compiling, Binding, and Running the OS/390 C++ Example	43
Under OS/390 Batch	43
Under TSO	43
Under the OS/390 Shell	45
Example of an OS/390 C++ Template Program.	45
CLB3ALST.C	46
CLB3ALST.H	46
CLB3AITR.C	47
CLB3AITR.H	47
CLB3AMAX.H	47
CLB3AMAX.C	48
CLB3AMIN.H	48
CLB3AMIN.C	48
CLB3ASTR.H	49
CLB3ATMP.CXX	50
Compiling, Binding, and Running the C++ Template Example	51
Under OS/390 Batch	51
Under TSO	53
Under the OS/390 Shell	53
 <b>Chapter 6. Compiler Options</b>	 55
Specifying Compiler Options	55
IPA Considerations	56
Using Special Characters.	57
Specifying OS/390 C Compiler Options Using #pragma Options	58
Specifying Compiler Options under OS/390 UNIX.	59
Compiler Option Defaults.	59
Summary of Compiler Options	59
Compatibility Options	62
Compiler Options for File Management.	63
Options That Control the Compiler Listing.	64
Options for Debugging and Diagnosing Errors	65
Options That Control the Source Code.	66
Options That Control the Object Code	66
Options That Control the Preprocessor.	68
Options That Control Program Execution	68
Options That Control the IPA Object.	68
Options That Control the IPA Link Step.	69
Direct-to-SOM Options.	70
Portability Options	70
Description of Compiler Options	70
AGGREGATE   NOAGGREGATE.	70
ALIAS   NOALIAS	71
ANSIALIAS   NOANSIALIAS	72
ARCHITECTURE	73
ARGPARSE   NOARGPARSE	74
ATTRIBUTE   NOATTRIBUTE	75
CHECKOUT   NOCHECKOUT	76
CONVLIT   NOCONVLIT	78
CSECT   NOCSECT	79
DEFINE	82
DIGRAPH   NODIGRAPH	82

	DLL   NODLL . . . . .	84
	EVENTS   NOEVENTS . . . . .	85
	EXECOPS   NOEXECOPS . . . . .	86
	EXH   NOEXH. . . . .	87
	EXPMAC   NOEXPMAC . . . . .	88
	EXPORTALL   NOEXPORTALL . . . . .	88
	FASTTEMPINC   NOFASTTEMPINC . . . . .	89
	FLAG   NOFLAG. . . . .	90
	FLOAT . . . . .	91
	GENPCH   NOGENPCH . . . . .	95
	GONUMBER   NOGONUMBER . . . . .	96
	HALT(num) . . . . .	97
	INFO   NOINFO . . . . .	98
	INLINE   NOINLINE. . . . .	99
	INLRPT   NOINLRPT . . . . .	102
	IPA   NOIPA . . . . .	103
	LANGLVL . . . . .	107
	LIBANSI   NOLIBANSI. . . . .	110
	LIST   NOLIST . . . . .	110
	LOCALE   NOLOCALE . . . . .	112
	LONGNAME   NOLONGNAME . . . . .	114
	LSEARCH   NOLSEARCH . . . . .	115
	MARGINS   NOMARGINS . . . . .	121
	MAXMEM   NOMAXMEM . . . . .	123
	MEMORY   NOMEMORY. . . . .	124
	NESTINC   NONESTINC. . . . .	125
	OBJECT   NOOBJECT . . . . .	125
	OE   NOOE. . . . .	127
	OFFSET   NOOFFSET . . . . .	128
	OMVS   NOOMVS . . . . .	129
	OPTFILE   NOOPTFILE . . . . .	129
	OPTIMIZE   NOOPTIMIZE . . . . .	131
	PHASEID . . . . .	133
	PLIST. . . . .	134
	PORT   NOPORT . . . . .	134
	PPONLY   NOPPONLY . . . . .	136
	REDIR   NOREDIR . . . . .	138
	RENT   NORENT . . . . .	139
	ROUND . . . . .	140
	SEARCH   NOSEARCH . . . . .	140
	SERVICE   NOSERVICE. . . . .	142
	SEQUENCE   NOSEQUENCE. . . . .	143
	SHOWINC   NOSHOWINC . . . . .	145
	SOM   NOSOM . . . . .	145
	SOMEINIT   NOSOMEINIT . . . . .	146
	SOMGS   NOSOMGS . . . . .	146
	SOMRO   NOSOMRO. . . . .	147
	SOMVOLATTR   NOSOMVOLATTR. . . . .	148
	SOURCE   NOSOURCE . . . . .	148
	SPILL   NOSPILL . . . . .	150
	SRCMSG   NOSRCMSG. . . . .	151
	SSCOMM   NOSSCOMM . . . . .	151
	START   NOSTART. . . . .	152
	STRICT   NOSTRICT . . . . .	153
	TARGET. . . . .	153
	TEMPINC   NOTEMPINC . . . . .	156

	157
	158
	161
	163
	163
	164
	165
	166
	167
Description of Compatible Compiler Options . . . . .	168
	169
	170
	170
	171
	172
	173
Using the OS/390 C Compiler Listing . . . . .	174
	174
	175
	180
Using the OS/390 C++ Compiler Listing . . . . .	183
	183
	184
	190
Using the IPA Link Step Listing . . . . .	193
	193
	200
 <b>Chapter 7. Binder Options and Control Statements . . . . .</b>	 207
Binder Options . . . . .	207
	207
	207
	207
	208
	208
	208
	209
	209
	209
	209
	210
	210
Binder Control Statements . . . . .	210
	211
	211
	212
	212
	213
	214
	214
 <b>Chapter 8. Runtime Options . . . . .</b>	 217
Specifying Runtime Options . . . . .	217
Using the #pragma runopts Preprocessor Directive . . . . .	217



<b>Chapter 9. Compiling</b>	221
Compiling with IPA	221
The IPA Compile Step	221
The IPA Link Step	222
Input to the OS/390 C/C++ Compiler	223
Primary Input	224
Secondary Input	224
Output from the Compiler	224
Specifying Output Files	225
Valid Input/Output File Types	227
Compiling Under OS/390 Batch	228
Using Cataloged Procedures for OS/390 C	229
Using Cataloged Procedures for OS/390 C++	229
Using Special Characters	230
Using Your Own JCL	230
Specifying Source Files	231
Specifying Include Files	232
Specifying Output Files	232
Compiling Under TSO	233
Using the CC and CXX REXX EXECs	233
Specifying Sequential and Partitioned Data Sets	234
Specifying HFS Files or Directories	235
Using ISPF to Invoke the Compiler	236
Compiling and Binding under the OS/390 Shell	239
Compiling and Binding in One Step with c89 and c++ (or cxx)	242
Using the make Utility	243
Using Feature Test Macros	244
Using Include Files	246
Specifying Include File Names	247
Forming File Names	247
Forming Data Set Names with LSEARCH   SEARCH Options	248
Search Sequence	250
Determining whether the File Name is in Absolute Form	251
Using SEARCH and LSEARCH	253
Search Sequences for Include Files	254
With the NOOE option	255
With the OE option	255
Compiling OS/390 C Source Code Using the SEARCH option	257
Compiling OS/390 C++ Source Code Using the SEARCH option	257
<b>Chapter 10. Using Precompiled Headers</b>	259
Determining the Initial Sequence	259
Matching the Initial Sequence	262
Example - Reusing Sequences	263
Using the GENP and USEP Compiler Options	263
Using an Alternative Initial Sequence	264
Restrictions	264
Organizing Your Source Files	265
Common Header File	266
Global PCH File for the Entire Directory	266
One PCH file for Each Member of the Directory	266
<b>Chapter 11. Using the IPA Link Step with OS/390 C/C++ Programs</b>	267
IPA Linking Your Program	267
Using DD Statements for the Standard Data Sets	268
Primary Input (SYSIN)	269

Location of Compiler and OS/390 Language Environment Library (STEPLIB)	269
Secondary Input (SYSLIB)	269
Output (SYSLIN or SYSPUNCH)	270
Destination of Errors Generated by the IPA Link Step (SYSOUT)	270
Listing (SYSCPRT)	270
Temporary Workspaces for the IPA Link Step (SYSUTx)	271
IPA Link Step Input	271
Primary Input	271
Secondary Input	272
Object File Formats	274
Object Record Formats	275
The IPA Link Step Control File	277
Output from the IPA Link Step	281
Specifying Output Files	281
Mapping Static Symbol Names	283
Running the IPA Link Step Under OS/390 Batch	283
Using the EDCI and CBCI Cataloged Procedures	284
Using Your Own JCL	286
Running the IPA Link Step in OS/390 UNIX	286
Using JCL	286
Invoking IPA from the c89 Utility	287
<b>Chapter 12. Binding OS/390 C/C++ Programs</b>	289
When You Can Use the Binder	289
When You Cannot Use the Binder	289
Your Output is a PDS, not a PDSE	289
CICS	289
MTF	289
IPA	289
Using Different Methods to Bind	290
Single Final Bind	290
Bind Each Compile Unit	291
Build and Use a DLL	292
Rebind a Changed Compile Unit	294
Binding Under OS/390 UNIX	294
OS/390 UNIX Example	295
Single Final Bind Using c89	295
Bind Each Compile Unit Using c89	296
Build and Use a DLL Using c89	297
Rebind a Changed Compile Unit Using c89	297
Binding under OS/390 Batch	299
OS/390 Batch Example	299
Single Final Bind under OS/390 Batch	299
Bind Each Compile Unit under OS/390 Batch	300
Build and Use a DLL under OS/390 Batch	301
Rebind a Changed Compile Unit under OS/390 Batch	303
Writing JCL for the binder	304
Binding Under TSO Using CXXBIND	305
TSO Example	306
Single Final Bind Under TSO	307
Bind Each Compile Unit Under TSO	307
Build and Use a DLL under TSO	308
Rebind a Changed Compile Unit Under TSO	308
<b>Chapter 13. Binder Processing</b>	311
Primary Input Processing	312

C or C++ Object Module as Input . . . . .	312
Secondary Input Processing. . . . .	312
Load Module as Input . . . . .	313
Program Object as input . . . . .	313
Autocall Input Processing (Library Search) . . . . .	313
Incremental Autocall Processing (AUTOCALL Control Statement) . . . . .	313
Final Autocall Processing (SYSLIB) . . . . .	314
Rename Processing . . . . .	314
Generating Aliases for Automatic Library Call (Library Search) . . . . .	315
Dynamic Link Library (DLL) Processing . . . . .	315
Statically bound functions . . . . .	316
Imported Variables . . . . .	316
Imported Functions . . . . .	316
Output Program Object . . . . .	316
Output IMPORT Statements. . . . .	317
Output Listing . . . . .	317
Header . . . . .	318
Input Event Log . . . . .	319
Module Map . . . . .	319
Cross Reference Table . . . . .	321
Imported and Exported Symbols Listing . . . . .	321
Mangled to Demangled Symbol Cross Reference . . . . .	322
Processing Options . . . . .	323
Save Operation Summary . . . . .	323
Save Module Attributes . . . . .	323
Entry Point and Alias Summary . . . . .	324
Long Symbol Abbreviation Table . . . . .	324
DDname vs Pathname Cross Reference Table . . . . .	325
Message Summary Report . . . . .	325
Binder Processing of C/C++ Object to Program Object . . . . .	326
Rebindability . . . . .	327
Error recovery . . . . .	329
Unresolved Symbols . . . . .	329
Significance of Library Search Order . . . . .	330
Duplicates . . . . .	331
Duplicate functions from autocall . . . . .	333
Hunting down references to unresolved symbols . . . . .	333
Non-reentrant DLL Problems . . . . .	333
Code That Has Been Prelinked . . . . .	334
<b>Chapter 14. Running an OS/390 C/C++ Application . . . . .</b>	<b>335</b>
Running an Application Under OS/390 Batch . . . . .	335
Specifying Runtime Options under OS/390 Batch . . . . .	335
Specifying Runtime Options in the EXEC Statement . . . . .	336
Using Cataloged Procedures . . . . .	336
Running an Application under TSO . . . . .	337
Specifying Runtime Options under TSO . . . . .	338
Passing Arguments to the OS/390 C/C++ Application . . . . .	338
Running an Application under OS/390 UNIX. . . . .	339
OS/390 UNIX Application Environments . . . . .	339
Specifying Runtime Options under OS/390 UNIX . . . . .	339
Restriction on Using 24-bit AMODE Programs . . . . .	340
Copying Applications between a PDS and HFS . . . . .	340
Running a Data Set Member from the OS/390 Shell . . . . .	340
Running an OS/390 UNIX Application under OS/390 Batch . . . . .	340

<b>Part 4. Utilities and Tools</b>	<b>343</b>
<b>Chapter 15. Model Tool</b>	<b>345</b>
About the OS/390 C/C++ Model Tool	345
Accessing Library Functions.	345
Method 1.	346
Method 2.	346
Method 3.	348
Method 4.	348
Accessing Pragma Directives	348
Method 1.	348
Method 2.	349
Method 3.	349
<b>Chapter 16. Object Library Utility</b>	<b>351</b>
Creating an Object Library Under OS/390 Batch	351
Creating and Object Library Under TSO	352
Object Library Utility Map.	353
<b>Chapter 17. DLL Rename Utility</b>	<b>357</b>
DLL Redistribution Scenario.	357
Inputs and Outputs	358
Restriction	359
Using the DLL Rename Utility under OS/390 Batch	360
Example of Renaming a DLL under OS/390 Batch	361
Using the DLL Rename Utility under TSO.	361
Specifying DLLRNAME Parameters Directly	361
Specifying DLLRNAME Parameters Using an Input File	362
Example of Renaming a DLL under TSO	363
<b>Chapter 18. Filter Utility.</b>	<b>365</b>
CXXFILT Options.	366
SYMMAP   NOSYMMAP	366
SIDEBYSIDE   NOSIDEBYSIDE	366
WIDTH(width)   NOWIDTH	366
REGULARNAME   NOREGULARNAME	366
CLASSNAME   NOCLASSNAME	367
SPECIALNAME   NOSPECIALNAME	367
Unknown Type of Name	367
Under OS/390 Batch	367
Under TSO	368
<b>Chapter 19. DSECT Conversion Utility</b>	<b>371</b>
DSECT Utility Options	371
SECT	371
BITF0XL   NOBITF0XL	372
COMMENT   NOCOMMENT	373
DEFSUB   NODEFSUB	373
EQUATE   NOEQUATE	373
HDRSKIP   NOHDRSKIP	375
INDENT   NOINDENT	376
LOCALE   NOLOCALE	376
LOWERCASE   NOLOWERCASE	376
OPTFILE   NOOPTFILE	377
PPCOND   NOPPCOND	377
SEQUENCE   NOSEQUENCE	377

UNNAMED   NOUNNAMED . . . . .	378
OUTPUT . . . . .	378
RECFM . . . . .	378
LRECL . . . . .	378
BLKSIZE . . . . .	378
Generation of Structures . . . . .	378
Under OS/390 Batch . . . . .	381
Under TSO . . . . .	382
 <b>Chapter 20. Coded Character Set and Locale Utilities . . . . .</b>	<b>385</b>
Coded Character Set Conversion Utilities . . . . .	385
iconv Utility . . . . .	385
genxlt Utility . . . . .	387
localedef Utility . . . . .	388
 <b>Part 5. OS/390 UNIX Utilities . . . . .</b>	<b>393</b>
 <b>Chapter 21. Archive and Make Utilities . . . . .</b>	<b>395</b>
Archive Libraries . . . . .	395
Creating Archive Libraries . . . . .	395
Creating Makefiles . . . . .	396
 <b>Chapter 22. BPXBATCH Utility . . . . .</b>	<b>397</b>
BPXBATCH Usage . . . . .	397
Parameter . . . . .	398
Usage Notes . . . . .	398
Files . . . . .	399
 <b>Part 6. Appendixes . . . . .</b>	<b>401</b>
 <b>Appendix A. Prelinking and Linking OS/390 C/C++ Programs . . . . .</b>	<b>403</b>
Prelinking an Application . . . . .	403
Using DD Statements for the Standard Data Sets - Prelinker . . . . .	404
Input to the Prelinker . . . . .	406
Prelinker Output . . . . .	406
Mapping long names to S-Names . . . . .	407
Linking an Application . . . . .	408
Using DD Statements for Standard Data Sets—Linkage Editor . . . . .	408
Input to the Linkage Editor . . . . .	409
Output from the Linkage Editor . . . . .	410
Link-Editing Multiple Object Modules . . . . .	412
Building DLLs . . . . .	412
Linking Your Code . . . . .	413
Using DLLs . . . . .	413
Prelinking and Linking an Application Under OS/390 Batch and TSO . . . . .	417
OS/390 Language Environment Prelinker Map . . . . .	418
Processing the Prelinker Automatic Library Call . . . . .	423
References to Currently Undefined Symbols (External References) . . . . .	423
Prelinking and Linking Under OS/390 Batch . . . . .	423
Writing JCL for the Prelinker and Linkage Editor . . . . .	425
Secondary Input to the Linker . . . . .	426
Using Additional Input Object Modules under OS/390 Batch . . . . .	427
Under TSO . . . . .	428
Using CPLINK . . . . .	431
Using LINK . . . . .	433

Prelinking and Link-Editing under the OS/390 Shell . . . . .	434
Using your JCL . . . . .	435
Setting c89 to Invoke the Prelinker . . . . .	437
Using the c89 Utility. . . . .	437
Prelinker Control Statement Processing . . . . .	437
IMPORT Control Statement . . . . .	438
INCLUDE Control Statement . . . . .	438
LIBRARY Control Statement . . . . .	439
RENAME Control Statement . . . . .	440
Reentrancy . . . . .	441
Natural or Constructed Reentrancy . . . . .	441
Using the Prelinker to Make Your Program Reentrant . . . . .	442
Generating a Reentrant Load Module in C . . . . .	442
Generating a Reentrant Load Module in C++ . . . . .	443
Resolving Multiple Definitions of the Same Template Function . . . . .	443
External Variables . . . . .	444
<b>Appendix B. Prelinker and Linkage Editor Options . . . . .</b>	<b>445</b>
Prelinker Options. . . . .	445
DLLNAME(dll-name) . . . . .	445
DUP   NODUP . . . . .	445
ER   NOER . . . . .	445
MAP   NOMAP . . . . .	445
MEMORY   NOMEMORY. . . . .	446
NCAL   NONCAL. . . . .	446
OMVS   NOOMVS . . . . .	446
UPCASE   NOUPCASE . . . . .	447
Linkage Editor Options. . . . .	447
<b>Appendix C. Diagnosing Problems . . . . .</b>	<b>449</b>
Problem Checklist . . . . .	449
When Does the Error Occur? . . . . .	450
The Error Occurs at Compile Time . . . . .	450
The Error Occurs at IPA Link Time . . . . .	451
The Error Occurs at Bind Time. . . . .	452
The Error Occurs at Prelink Time. . . . .	452
The Error Occurs at Link Time. . . . .	453
The Error Occurs at Run Time. . . . .	453
Installation Problems . . . . .	455
<b>Appendix D. IBM Supplied Cataloged Procedures and REXX EXECs . . . . .</b>	<b>457</b>
Tailoring PROCs, REXX EXECs, and EXECs . . . . .	458
Data Sets Used . . . . .	460
Description of Data Sets Used. . . . .	460
Examples Using Cataloged Procedures . . . . .	466
<b>Appendix E. Using Assembler Macros . . . . .</b>	<b>467</b>
CBC3UAAP. . . . .	469
CBC3UAAQ . . . . .	470
CBC3UAAR . . . . .	471
CBC3UAAS. . . . .	472
CBC3UAAT. . . . .	473
CBC3UAAU . . . . .	474
<b>Appendix F. OS/390 C/C++ Compiler Return Codes and Messages . . . . .</b>	<b>475</b>
Return Codes . . . . .	475

Compiler Messages . . . . .	475
<b>Appendix G. Other Return Codes and Messages . . . . .</b>	<b>595</b>
<b>Appendix H. Utility Messages . . . . .</b>	<b>597</b>
DSECT Utility Messages . . . . .	597
Return Codes . . . . .	597
Messages . . . . .	597
DLLRNAME Utility Messages . . . . .	599
Return Codes . . . . .	599
Messages . . . . .	599
CXXFILT Utility Messages . . . . .	600
Return Codes . . . . .	600
Messages . . . . .	600
<b>Appendix I. Other OS/390 C Utilities . . . . .</b>	<b>603</b>
Using the Old Syntax for CC . . . . .	603
Using CMOD . . . . .	604
<b>Appendix J. Layout of the Events File . . . . .</b>	<b>607</b>
Description of the Fileid Field . . . . .	607
Description of the Filend Field . . . . .	608
Description of the Error Field . . . . .	608
<b>Glossary . . . . .</b>	<b>611</b>
<b>Bibliography . . . . .</b>	<b>639</b>
OS/390 . . . . .	639
VS COBOL II Release 4 . . . . .	639
COBOL FOR MVS & VM Release 2. . . . .	639
COBOL for OS/390 & VM Version 2 Release 1. . . . .	639
PL/I for MVS & VM Release 1 Modification 1 . . . . .	640
OS PL/I Version 2 Release 3 . . . . .	640
VS FORTRAN Version 2 Release 6 . . . . .	640
CICS/ESA Version 4 Release 1 . . . . .	640
CICS Transaction Server for OS/390 Release 2 . . . . .	640
DB2 Version 3 Release 1. . . . .	640
DB2 Version 4 Release 1. . . . .	641
DB2 Version 5 Release 1. . . . .	641
IMS/ESA Version 4 Release 1 . . . . .	641
IMS/ESA Version 5 Release 1 . . . . .	641
IMS/ESA Version 6 Release 1 . . . . .	641
QMF Version 3 Release 2 . . . . .	641
VSAM. . . . .	642
<b>INDEX . . . . .</b>	<b>643</b>
<b>Readers' Comments — We'd Like to Hear from You. . . . .</b>	<b>657</b>





---

## Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY, 10594, USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This publication documents *intended* Programming Interfaces that allow the customer to write OS/390 C/C++ programs.

Any interfaces, including service component interfaces, that are not documented in the OS/390 C/C++ publications are not formal interfaces. You should not build any dependencies on these interfaces, as IBM can change or remove interfaces at any time, without notice.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

---

## 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.

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.

Extracts from X/Open Specification, Programming Languages, Issue 4 Release 2, copyright 1988, 1989, February 1992, by the X/Open Company Limited, have been reproduced with the permission of X/Open Company Limited. No further reproduction of this material is permitted without the written notice from the X/Open Company Ltd, UK.

---

## Trademarks

The following terms, which may be denoted by a single asterisk (\*), are trademarks of International Business Machines Corporation in the United States or other countries or both:

AD/Cycle	AFP	AIX
AIX/6000	AT	AS/400
BookManager	C Set ++	C/370
C/MVS	C++/MVS	Common User Access
CICS	CICS/ESA	CICSplex
COBOL/370	CUA	CT
DATABASE 2	DB2	DFSMS
DFSMS/MVS	DFSMSdfp	DRDA
ESCON	GDDM	Hiperspace
IBM	IBMLink	IMS
IMS/ESA	MVS/DFP	MVS/ESA
MVS/SP	MVS/XA	Open Class
OpenEdition	Operating System/2	Operating System/400
OS OPEN	OS/2	OS/390
OS/400	PROFS	PS/2
QMF	RACF	RETAIN
S/370	S/390	SAA
SOM	SOMobjects	SP
SQL/DS	System/370	System/390
System Object Model	Systems Application Architecture	VisualAge
VM/ESA	VSE/ESA	VTAM
3090	3890	400

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation.

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

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



---

## Part 1. Introduction

This part presents introductory concepts on the OS/390 C/C++ product, and discusses the OS/390 C/C++ library. Specifically, it discusses the following:

- “Chapter 1. About This Book” on page 3
- “Chapter 2. About IBM OS/390 C/C++” on page 15
- “Chapter 3. Important Changes to the Prelinker Documentation” on page 31



---

## Chapter 1. About This Book

This edition of the *OS/390 C/C++ User's Guide* is intended for users of the IBM OS/390 C/C++ compiler with the OS/390 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 under the OS/390 operating system.

To use this, or any other OS/390 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 OS/390 Language Environment product and OS/390 UNIX<sup>®</sup> System Services (OS/390 UNIX).

---

## IBM OS/390 C/C++ and Related Publications

This section summarizes the content of the IBM OS/390 C/C++ publications and shows where to find related information in other publications.

Table 1. OS/390 C/C++ Publications

Book Title and Number	Key Sections/Chapters in the Book
<i>OS/390 C/C++ Programming Guide</i> , SC09-2362	<p>Guidance information for:</p> <ul style="list-style-type: none"><li>• C/C++ input and output</li><li>• Debugging OS/390 C programs that use input/output</li><li>• Using linkage specifications in C++</li><li>• Combining C and assembler</li><li>• Creating and using DLLs</li><li>• Using threads in an OS/390 UNIX application</li><li>• Reentrancy</li><li>• Using the decimal data type in C and C++</li><li>• Handling exceptions, error conditions, and signals</li><li>• Optimizing code</li><li>• Optimizing your C/C++ code with Interprocedural Analysis</li><li>• Network communications under OS/390 UNIX</li><li>• Interprocess communications using OS/390 UNIX</li><li>• Structuring a program that uses C++ templates</li><li>• Using environment variables</li><li>• Using System Programming C facilities</li><li>• Library functions for the System Programming C facilities</li><li>• Using runtime user exits</li><li>• Using the OS/390 C multitasking facility</li><li>• Using other IBM products with OS/390 C/C++ (CICS*, CSP, DWS, DB2*, GDDM*, IMS*, ISPF, QMF*)</li><li>• Direct-to-SOM support under OS/390 C/C++</li><li>• Internationalization: locales and character sets, code set conversion utilities, mapping variant characters</li><li>• POSIX character set</li><li>• Code point mappings</li><li>• Locales supplied with OS/390 C/C++</li><li>• Charmap files supplied with OS/390 C/C++</li><li>• Examples of charmap and locale definition source files</li><li>• Converting code from code character set IBM-1047</li><li>• Using built-in functions</li><li>• Programming considerations for OS/390 UNIX C/C++</li></ul>
<i>OS/390 C/C++ User's Guide</i> , SC09-2361	<p>Guidance information for:</p> <ul style="list-style-type: none"><li>• OS/390 C/C++ examples</li><li>• Compiler options</li><li>• Binder options and control statements</li><li>• Specifying OS/390 Language Environment runtime options</li><li>• Compiling, IPA Linking, binding, and running OS/390 C/C++ programs</li><li>• Using precompiled headers</li><li>• Utilities (Object Library, DLL Rename, CXXFILT, DSECT Conversion, Code Set and Locale, ar and make, BPXBATCH)</li><li>• Diagnosing problems</li><li>• Cataloged procedures and REXX EXECs supplied by IBM</li><li>• Error messages and return codes</li></ul>



Table 1. OS/390 C/C++ Publications (continued)

Book Title and Number	Key Sections/Chapters in the Book
<i>OS/390 C/C++ Language Reference</i> , SC09-2360	Reference information for: <ul style="list-style-type: none"> <li>• The C and C++ Languages</li> <li>• Lexical elements of OS/390 C and OS/390 C++</li> <li>• Declarations, expressions and operators</li> <li>• Implicit type conversions</li> <li>• Functions and statements</li> <li>• Preprocessor directives</li> <li>• C++ classes, class members, and friends</li> <li>• C++ overloading, special member functions, and inheritance</li> <li>• C++ templates and exception handling</li> <li>• OS/390 C and OS/390 C++ compatibility</li> </ul>
<i>OS/390 C/C++ Run-Time Library Reference</i> , SC28-1663	Reference information for: <ul style="list-style-type: none"> <li>• C header files</li> <li>• C Library functions</li> </ul>
<i>OS/390 C Curses</i> , SC28-1907	Reference information for: <ul style="list-style-type: none"> <li>• Curses concepts</li> <li>• Key data types</li> <li>• General rules for characters, renditions, and window properties</li> <li>• General rules of operations and operating modes</li> <li>• Use of macros</li> <li>• Restrictions on block-mode terminals</li> <li>• Curses functional interface</li> <li>• Contents of headers</li> <li>• The terminfo database</li> </ul>
<i>OS/390 C/C++ Compiler and Run-Time Migration Guide</i> , SC09-2359	Guidance and reference information for: <ul style="list-style-type: none"> <li>• Common migration questions</li> <li>• Application executable program compatibility</li> <li>• Source program compatibility</li> <li>• Input and output operations compatibility</li> <li>• Class library migration considerations</li> <li>• Changes between releases of OS/390</li> <li>• C/370* V1 to V2 compiler changes</li> <li>• Other migration considerations</li> </ul>
<i>OS/390 C/C++ Reference Summary</i> , SX09-1313	Summary tables for: <ul style="list-style-type: none"> <li>• Character set, trigraphs, digraphs, and keywords</li> <li>• Escape sequences, storage classes</li> <li>• Predefined and derived types, type qualifiers</li> <li>• Operator precedence, redirection symbols</li> <li>• fprintf() format, type characters, and flag characters</li> <li>• fscanf() format and type characters</li> <li>• __amrc structure</li> <li>• Hardware exceptions and signals</li> <li>• Compiler return codes</li> <li>• Compiler options</li> <li>• #pragma directives</li> <li>• Library functions</li> <li>• Utilities</li> </ul>

Table 1. OS/390 C/C++ Publications (continued)

Book Title and Number	Key Sections/Chapters in the Book
<i>OS/390 C/C++ IBM Open Class Library User's Guide</i> , SC09-2363	Guidance information for: <ul style="list-style-type: none"> <li>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</li> <li>Using the I/O Stream Class Library: Introduction, getting started, advanced topics, and manipulators</li> <li>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</li> <li>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 Decimal classes</li> </ul>
<i>OS/390 C/C++ IBM Open Class Library Reference</i> , SC09-2364	Reference information for: <ul style="list-style-type: none"> <li>Complex Mathematics Class Library</li> <li>I/O Stream Class Library</li> <li>Collection Class Library</li> <li>Application Support Class Library</li> </ul>
<i>OS/390 C/C++ SOM-Enabled Class Library User's Guide and Reference</i> , SC09-2366	Guidance and reference information for: <ul style="list-style-type: none"> <li>C++ SOM (RRBC-enabled) versions of Collection and Application Support Class Libraries</li> <li>Cross-language SOM version of the Collection Class Library</li> </ul>
<i>Debug Tool User's Guide and Reference</i> , SC09-2137	Guidance and reference information for: <ul style="list-style-type: none"> <li>Preparing to debug programs</li> <li>Debugging programs</li> <li>Using Debug Tool in different environments</li> <li>Language-specific information</li> <li>Debug Tool reference</li> </ul>
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 IBM OS/390 C/C++ licensed program, including: <ul style="list-style-type: none"> <li>Isolating reportable problems</li> <li>Keywords</li> <li>Preparing an Authorized Program Analysis Report (APAR)</li> <li>Problem identification worksheet</li> <li>Maintenance on OS/390</li> <li>Late changes to OS/390 C/C++ publications</li> </ul>

**Note:** For complete and detailed information on linking and running with OS/390 Language Environment and using the OS/390 Language Environment runtime options, refer to the *OS/390 Language Environment Programming Guide*, SC28-1939. For complete and detailed information on using interlanguage calls, refer to *OS/390 Language Environment Writing Interlanguage Applications*, SC28-1943.

The following table lists the OS/390 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 OS/390 C/C++	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ Compiler and Run-Time Migration Guide</i>, SC09-2359</li> <li>• <i>OS/390 Language Environment Customization</i>, SC28-1941</li> <li>• <i>OS/390 UNIX System Services Planning</i>, SC28-1890</li> <li>• <i>OS/390 Planning for Installation</i>, GC28-1726</li> <li>• OS/390 Task Atlas, available on the OS/390 Library page on the World Wide Web (<a href="http://www.s390.ibm.com/os390/bkserv">http://www.s390.ibm.com/os390/bkserv</a>)</li> </ul>
Installing	<ul style="list-style-type: none"> <li>• OS/390 Program Directory</li> <li>• <i>OS/390 Planning for Installation</i>, GC28-1726</li> <li>• <i>OS/390 Language Environment Customization</i>, SC28-1941</li> </ul>
Coding programs	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ Run-Time Library Reference</i>, SC28-1663</li> <li>• <i>OS/390 C/C++ Language Reference</i>, SC09-2360</li> <li>• <i>OS/390 C/C++ Reference Summary</i>, SX09-1313</li> <li>• <i>OS/390 C/C++ Programming Guide</i>, SC09-2362</li> <li>• <i>OS/390 Language Environment Concepts Guide</i>, GC28-1945</li> <li>• <i>OS/390 Language Environment Programming Guide</i>, SC28-1939</li> <li>• <i>OS/390 Language Environment Programming Reference</i>, SC28-1940</li> <li>• <i>OS/390 C/C++ IBM Open Class Library User's Guide</i>, SC09-2363</li> <li>• <i>OS/390 C/C++ IBM Open Class Library Reference</i>, SC09-2364</li> <li>• <i>OS/390 C/C++ SOM-Enabled Class Library User's Guide and Reference</i>, SC09-2366</li> </ul>
Coding and binding programs with interlanguage calls	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ Programming Guide</i>, SC09-2362</li> <li>• <i>OS/390 C/C++ Language Reference</i>, SC09-2360</li> <li>• <i>OS/390 Language Environment Programming Guide</i>, SC28-1939</li> <li>• <i>OS/390 Language Environment Writing Interlanguage Applications</i>, SC28-1943</li> <li>• <i>DFSMS/MVS Program Management</i>, SC26-4916</li> </ul>
Compiling, binding, and running programs	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ User's Guide</i>, SC09-2361</li> <li>• <i>OS/390 Language Environment Programming Guide</i>, SC28-1939</li> <li>• <i>OS/390 Language Environment Debugging Guide and Run-Time Messages</i>, SC28-1942</li> <li>• <i>DFSMS/MVS Program Management</i>, SC26-4916</li> <li>• OS/390 Messages Database, available on the OS/390 Library page in the World Wide Web (<a href="http://www.s390.ibm.com/os390/bkserv">http://www.s390.ibm.com/os390/bkserv</a>)</li> </ul>
Compiling and binding applications in the OS/390 UNIX environment	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ User's Guide</i>, SC09-2361</li> <li>• <i>OS/390 UNIX System Services User's Guide</i>, SC28-1891</li> <li>• <i>OS/390 UNIX System Services Command Reference</i>, SC28-1892</li> <li>• <i>DFSMS/MVS Program Management</i>, SC26-4916</li> </ul>

Table 2. Publications by Task (continued)

Tasks	Books
Compiling and binding SOM applications with OS/390 SOMobjects*	<ul style="list-style-type: none"> <li>• <i>OS/390 SOMobjects Programmer's Guide</i>, GC28-1859</li> <li>• <i>OS/390 C/C++ Programming Guide</i>, SC09-2362</li> <li>• <i>OS/390 C/C++ User's Guide</i>, SC09-2361</li> </ul>
Debugging programs	<ul style="list-style-type: none"> <li>• README file</li> <li>• <i>Debug Tool User's Guide and Reference</i>, SC09-2137</li> <li>• <i>OS/390 C/C++ User's Guide</i>, SC09-2361</li> <li>• <i>OS/390 C/C++ Programming Guide</i>, SC09-2362</li> <li>• <i>OS/390 Language Environment Programming Guide</i>, SC28-1939</li> <li>• <i>OS/390 Language Environment Debugging Guide and Run-Time Messages</i>, SC28-1942</li> <li>• <i>OS/390 UNIX System Services Messages and Codes</i>, SC28-1908</li> <li>• <i>OS/390 UNIX System Services User's Guide</i>, SC28-1891</li> <li>• <i>OS/390 UNIX System Services Command Reference</i>, SC28-1892</li> <li>• <i>OS/390 UNIX System Services Programming Tools</i>, SC28-1904</li> </ul>
Using shells and utilities in the OS/390 UNIX environment	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ User's Guide</i>, SC09-2361</li> <li>• <i>OS/390 UNIX System Services Command Reference</i>, SC28-1892</li> <li>• <i>OS/390 UNIX System Services Messages and Codes</i>, SC28-1908</li> </ul>
Using sockets library functions in the OS/390 UNIX environment	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ Run-Time Library Reference</i>, SC28-1663</li> </ul>
Porting a UNIX Application to OS/390	<ul style="list-style-type: none"> <li>• <i>OS/390 UNIX System Services Porting Guide</i> This guide contains useful information about supported header files and C functions, sockets in an OS/390 UNIX environment, process management, compiler optimization tips, and suggestions for improving the application's performance after it has been ported. The <i>Porting Guide</i> is available as a PDF file which you can download, or as web pages which you can browse, at the following URL: <a href="http://www.s390.ibm.com/unix/bpxa1por.html">http://www.s390.ibm.com/unix/bpxa1por.html</a></li> </ul>
Performing diagnosis and submitting an Authorized Program Analysis Report (APAR)	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ User's Guide</i>, SC09-2361</li> <li>• CBC.SCBCDOC(APAR) on OS/390 C/C++ product tape</li> </ul>
Quick reference	<ul style="list-style-type: none"> <li>• <i>OS/390 C/C++ Reference Summary</i>, SX09-1313</li> </ul>
Multimedia Tutorial	<ul style="list-style-type: none"> <li>• For a new way of learning C++ programming, you can order the CD-ROM <i>Experience C++: A Multimedia Tutorial</i>, SK2T-1158. This tutorial runs in DOS.</li> </ul>

**Note:** For information on using the prelinker, see "Appendix A. Prelinking and Linking OS/390 C/C++ Programs" on page 403. As of Release 4, this appendix contains information that was previously in the chapter on prelinking and linking OS/390 C/C++ programs in *OS/390 C/C++ User's Guide*. It also contains prelinker information that was previously in the *OS/390 C/C++ Programming Guide*.

---

## Hardcopy Books

The following OS/390 C/C++ books are available in hardcopy:

- *OS/390 C/C++ Run-Time Library Reference*, SC28-1663
- *OS/390 C/C++ User's Guide*, SC09-2361
- *OS/390 C/C++ Programming Guide*, SC09-2362
- *OS/390 C/C++ Reference Summary*, SX09-1313
- *OS/390 C/C++ IBM Open Class Library User's Guide*, SC09-2363
- *OS/390 C Curses*, SC28-1907
- *OS/390 C/C++ Compiler and Run-Time Migration Guide*, SC09-2359
- *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 the *OS/390 C/C++ Compiler and Run-Time Migration Guide*, SC09-2359 at no charge. Feature code 8009 includes the remaining books.

---

## Softcopy Books

All of the OS/390 C/C++ publications (except for the *OS/390 C/C++ Reference Summary*) are available in softcopy book format. The books are available on the tape that accompanies the OS/390 product, and on a CD-ROM called the *IBM Online Library Omnibus Edition: OS/390 Collection*, SK2T-6700.

To read the softcopy books, the BookManager\* Read (Program 5684-062, 5695-046) licensed program must be available on your operating system. BookManager Read provides access to online information as an alternative to hard copy documents. You can read, search, make notes, and select sections of text to print.

Also available are BookManager Read/DOS (Program 73F6-022) for the DOS operating system, and BookManager Read/2 (Program 73F6-023) for the OS/2 operating system. With these products, you can download online books to your workstation and read them.

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 OS/390 home page. The URL for this page is:

<http://www.s390.ibm.com/os390/index.html>

---

## Softcopy Examples

Most of the larger examples in the following books are available in machine-readable form:

- *OS/390 C/C++ Language Reference*, SC09-2360
- *OS/390 C/C++ User's Guide*, SC09-2361
- *OS/390 C/C++ Programming Guide*, SC09-2362
- *OS/390 C/C++ IBM Open Class Library User's Guide*, SC09-2363

- *OS/390 C/C++ IBM Open Class Library Reference*, SC09-2364
- *OS/390 C/C++ SOM-Enabled Class Library User's Guide and Reference*, SC09-2366

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 CBCxxyy or CLBxxyy, where x refers to a publication:

- R and X refer to the *OS/390 C/C++ Language Reference*, SC09-2360
- G refers to the *OS/390 C/C++ Programming Guide*, SC09-2362
- U refers to the *OS/390 C/C++ User's Guide*, SC09-2361
- A refers to the *OS/390 C/C++ IBM Open Class Library User's Guide*, SC09-2363

Examples labelled as CBCxxyy appear in the *OS/390 C/C++ Language Reference*, the *OS/390 C/C++ Programming Guide*, and the *OS/390 C/C++ User's Guide*. Examples labelled as CLBxxyy appear in the *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 the *OS/390 C/C++ IBM Open Class Library Reference*, SC09-2364 and in the *OS/390 C/C++ SOM-Enabled Class Library User's Guide and Reference*, SC09-2366. For the *OS/390 C/C++ SOM-Enabled Class Library User's Guide and Reference*, SC09-2366, the label refers to a member name in the data set *CBC.SCLBXS*.

---

## OS/390 C/C++ on the World Wide Web

Additional information on OS/390 C/C++ is available on the World Wide Web. The URL for the OS/390 C/C++ home page is:

<http://www.software.ibm.com/ad/c390/index.html>

This page contains late-breaking information about the OS/390 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 OS/390 C/C++ information library and the libraries of other OS/390 elements that are available on the Web. The OS/390 C/C++ home page also contains information on active Beta programs, samples that you can download, C/370 product newsletters, and links to other related Web sites.

---

## C/C++ News...

IBM also publishes the *C/370 Compiler Newsletter*. This free newsletter keeps subscribers up to date on the latest product releases. It also provides coding hints and tips, questions and answers, and news about C/370 products and IBM OS/390 C/C++.

To take advantage of this free publication, send your name, full mailing address, and phone number, as follows:

- Send a message electronically to the following network ID :
  - Internet: [inetc370@ca.ibm.com](mailto:inetc370@ca.ibm.com)
  - IBMMAIL: [@caibmrzx](mailto:ibmmail(caibmrzx))
- Mail your request to:

---

## 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—▶▶

The following indicates a continuation; the opposing arrowheads indicate the end of a command, directive, or statement.

▶▶—statement—▶▶

Diagrams of syntactical units other than complete commands, directives, or statements look like this:

▶▶—statement—▶▶

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

▶▶—statement—*required\_item*—▶▶

- Optional items are below the main path.

▶▶—statement—  
          └─*optional\_item*—┘

- 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.

▶▶—statement—  
          └─*required\_choice1*—  
          └─*required\_choice2*—

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.

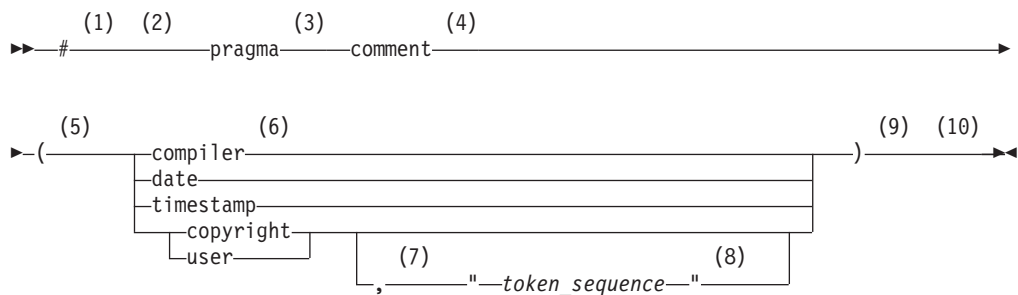


A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

- 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.



#### 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 OS/390 C/C++

The C/C++ feature of the IBM OS/390 licensed program provides support for C and C++ application development on the OS/390 platform. The C/C++ feature is based on the C/C++ for MVS/ESA\* product.

IBM OS/390 C/C++ includes:

- A C compiler (referred to as the OS/390 C compiler)
- A C++ compiler (referred to as the OS/390 C++ compiler)
- A set of C++ class libraries
- Application Support Class and Collection Class Library source
- A mainframe interactive Debug Tool (optional)
- A set of utilities for C/C++ application development

IBM offers the C language on other platforms, such as the AIX\*, IBM Operating System/2\* (OS/2\*), IBM Operating System/400\* Version 3 (OS/400\*), Sun Solaris, VM/ESA\*, VSE/ESA\*, and Windows® operating systems. The AIX, OS/2, OS/400, Sun Solaris, and Windows operating systems also offer the C++ language.

---

### Changes for Version 2 Release 6

OS/390 C/C++ has made the following changes for this release:

- Added support for the Institute of Electrical and Electronics Engineers (IEEE) binary floating-point data type, in conformance with the IEEE 754 standard, as applicable to the S/390\* environment. For details on the OS/390 C/C++ support, see "FLOAT" on page 91. In addition, two related sub-options have been introduced, ARCH(3) and TUNE(3). The two sub-options support the new G5 processor architecture, and IEEE binary floating-point data. Refer to "ARCHITECTURE" on page 73 and "TUNE" on page 161 for details.

Complete IEEE binary floating-point support for OS/390 and its elements requires that you apply small programming enhancements (SPEs) to OS/390 V2R6.0, and to specific releases of some software. These SPEs are delivered as program temporary fixes (PTFs). Consult your System Programmer to ensure that the SPE PTFs you require for IEEE binary floating-point support, as documented in the *OS/390 Planning for Installation* publication, are applied to your system. The *OS/390 Planning for Installation* publication documents the complete software requirements for IEEE binary floating-point support on OS/390.

- Improved the performance of the Binary Coded Decimal (BCD) class library, and its compatibility with the decimal data type in C, and other S/390 languages. For details, see *Using the C++ Decimal Data Type* in the *OS/390 C/C++ Programming Guide*.
- Added support for the long long integer data type. For more details, see the sections on integer declarations in the *OS/390 C/C++ Language Reference*. The run-time library, including functions such as `printf()` and `scanf()`, does not support the long long data type at this time.
- Added a new compiler option, `PORT`, that enables you to increase the syntax checking for the `#pragma pack` directive in your code. This option is helpful when porting code that contains `#pragma pack` directives or packed data from other platforms. For more information on the `PORT` option, see "PORT | NOPORT" on page 134.
- Added a new compiler option, `FASTTEMPINC`, that enables you to improve your compilation time for C++ class templates if you use a large number of recursive

templates in an application. For more information on the FASTEMPINC option, see “FASTTEMPINC | NOFASTTEMPINC” on page 89.

- 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 OS/390 C/C++ compiler feature(s) or to use the DLL Rename Utility.
- The level of optimization you get when you specify the OPT(1), or OPT, compiler option is the same as when you specify OPT(2). For more information on the OPTIMIZATION option, see the “OPTIMIZE | NOOPTIMIZE” on page 131.
- The OS/390 C++ class library header files are now distributed in the hierarchical file system (HFS) in directory /usr/lpp/ioclib/include.
- As part of the name change of *OpenEdition*\* to *OS/390 UNIX System Services*, occurrences of *OpenEdition* have been changed to *OS/390 UNIX System Services* or its abbreviated name, *OS/390 UNIX*, throughout the OS/390 C/C++ information library. *OpenEdition* may continue to appear in messages, panel text, and other code locations.

---

## The C/C++ Compilers

The following sections describe the C and C++ languages and the OS/390 C/C++ compilers.

### The C Language

The C language is a general purpose, versatile, and functional programming language, which 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 OS/390 C++ and OS/390 C, refer to the *OS/390 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 OS/390 C and C++ Compilers

The C or C++ compilers offer many features to help your work:

- Optimization support.
  - Algorithms to take advantage of S/390 architecture to get better optimization for speed and use of computer resources through the OPTIMIZE and IPA compile-time options.
  - The OPTIMIZE compile-time option to instruct the compiler to optimize the machine instructions it generates, to produce faster-running object code, thereby optimizing application performance at run time.
  - Interprocedural Analysis (IPA), to perform optimizations across compilation units, thereby optimizing application performance at run time.
  - The precompiled header facility, to save information from one compilation unit for use in another or to reuse information when re-compiling the source compilation unit, thereby improving performance at compile time.

- DLLs (dynamic link libraries) to reduce application size, and dynamically link to exported variables and functions at run time.

IBM OS/390 C/C++ provides support for generating DLLs in a way similar to the way OS/2 generates DLLs. 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 calls a DLL function, or references a DLL, IBM OS/390 C/C++ provides a load-on-reference DLL. 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. OS/390 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 OS/390, or the OS/390 Language Environment Prelinker (prelinker) and program management binder. The OS/390 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, and Fortran, to enable programmers to integrate OS/390 C/C++ code with existing applications.
- Exploitation of OS/390 and OS/390 UNIX technology.

OS/390 UNIX is an IBM implementation of the open operating system environment, as defined in the XPG4 and POSIX standards.

- When used with OS/390 UNIX and OS/390 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.

## OS/390 C Compiler Specific Features

In addition to the features common to OS/390 C/C++, the OS/390 C compiler provides you with the following capabilities:

- The ability to write portable code that conforms to 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 OS/390 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 OS/390 environment, such as fixed-point (packed) decimal data support

## Features That Are Specific to the OS/390 C++ Compiler

In addition to the features common to OS/390 C/C++, the OS/390 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 OS/390 C++ compiler also conforms to a subset of the C++ ANSI/ISO (Draft) Standard (X3J16/93-0062).
- System Object Model (SOM) support, through the SOM Interface Definition Language (IDL) compiler available with OS/390 SOMobjects. You can use the IDL compiler and associated emitters to create language-specific bindings that define the interface to a SOM object. This enables OS/390 C++ programs to share SOM objects with other languages. In addition, SOM enables release-to-release binary compatibility.

With Direct-to-SOM (DTS) support in the OS/390 C++ compiler, you can generate SOM objects directly from C++ code. You do not need to create and process the IDL first. You can write virtually the same code you do when creating C++ objects.

**Note:** The OS/390 C++ compiler no longer supports IDL generation through the IDL compile-time option. This option instructed the compiler to generate

IDL. Mixed-language or distributed object applications used IDL. If you need IDL for your applications, you should now code it yourself instead of generating it through the IDL compile option.

- C++ template support and exception handling consistent with VisualAge\* C++ product implementations.

---

## Utilities

The OS/390 C/C++ compilers provide the following utilities:

- The Object Library Utility to update partitioned data set (PDS) 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 CXXFILT Utility to map OS/390 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 OS/390 C/C++ data structures
- The C/C++ Model Tool to provide online help for C/C++ #pragma directives and runtime library functions. These functions are other than the C Curses functions, and are at the level that is supplied in OS/390 Release 2

---

## Class Libraries

IBM OS/390 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++ 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, and Time.

The Application Support Class library is available in a C++ SOM version as well as the regular C++ native version.

- 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.

The Collection Class Library is available in a C++ SOM and a cross-language SOM version, as well as the regular C++ native version.

All of the libraries that are described above are thread-safe, except the cross-language SOM version of the Collection Class Library.

All of the libraries that are described above are available in both static and DLL formats. OS/390 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 OS/390 C/C++ compiler feature(s) or to use the DLL Rename Utility.

## Class Library Source

The Class Library Source consists of the following:

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

---

## The Debug Tool

IBM OS/390 C/C++ supports program development by using a mainframe interactive Debug Tool. This optionally available tool allows you to debug applications in their native host environment, such as CICS/ESA, IMS/ESA\*, DB2, and so on. The Debug Tool provides the following support and function:

- Step mode
- Breakpoints
- Monitor
- Frequency analysis
- Dynamic patching

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

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

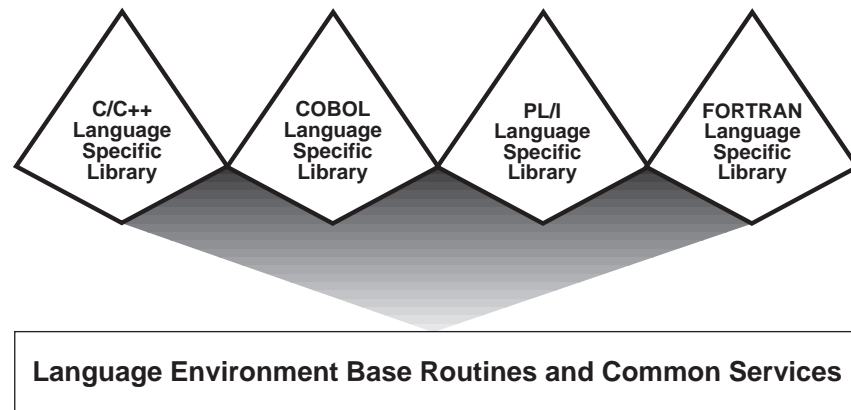


---

## OS/390 Language Environment

IBM OS/390 C/C++ exploits the C/C++ runtime environment and library of runtime services available with OS/390 Language Environment (formerly Language Environment for MVS & VM, Language Environment/370 and LE/370).

OS/390 Language Environment consists of four language-specific runtime libraries, and Base Routines and Common Services; see Figure 1. OS/390 Language Environment establishes a common runtime environment and common runtime services for language products, user programs, and other products.



*Figure 1. Libraries in OS/390 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 OS/390 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. OS/390 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; OS/390 UNIX services are available to an application programmer or program through the OS/390 C/C++ language bindings.
- Access to language-specific library routines, such as the OS/390 C/C++ library functions.

---

## The Program Management Binder

The binder provided with OS/390 combines the object modules, load modules, and program objects comprising an OS/390 application. It produces a single 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 OS/390 Language Environment combines the object modules comprising an OS/390 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, or a PDSE or HFS file.

---

## OS/390 UNIX System Services (OS/390 UNIX)

OS/390 UNIX provides capabilities under OS/390 to make it easier to implement or port applications in an open, distributed environment. OS/390 UNIX Services are available to OS/390 C/C++ application programs through the C/C++ language bindings available with OS/390 Language Environment.

Together, the OS/390 UNIX Services, OS/390 Language Environment, and OS/390 C/C++ compilers provide an application programming interface that supports industry standards.

OS/390 UNIX provides support for both existing OS/390 applications and new OS/390 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)
- OS/390 UNIX Extensions that provide OS/390-specific support beyond the defined standards
- The OS/390 UNIX Shell and Utilities feature, which provides:
  - A shell, based on the Korn Shell and compatible with the Bourne Shell
  - Tools and utilities that conform to the *X/Open Single UNIX Specification*, also known as *X/Open Portability Guide (XPG) Version 4, Issue 2*, and provide OS/390 support. The following utilities are included:

<b>ar</b>	Creates and maintains library archives
-----------	--

- |                 |  |
|-----------------|--|
| <b>BPXBATCH</b> | Allows you to submit batch jobs that run shell commands, scripts, or OS/390 C/C++ executable files in HFS files from a shell session   |
| <b>c89</b>      | Compiles, assembles, and binds OS/390 UNIX C applications  |
| <b>gencat</b>   | Merges the message text source files Messagefile (usually *.msg) into a formatted message Catalogfile (usually *.cat)  |
| <b>lex</b>      | Automatically writes large parts of a lexical analyzer based on a description that is supplied by the programmer   |
| <b>make</b>     | Helps you manage projects containing a set of interdependent files, such as a program with many OS/390 C/C++ source and object files, keeping all such files up to date with one another |
| <b>yacc</b>     | Allows you to write compilers and other programs that parse input according to strict grammar rules  |
- Support for other utilities such as:
- |                  |   |
|------------------|---|
| <b>c++</b>       | Compiles, assembles, and binds OS/390 UNIX C++ applications                                       |
| <b>mkcatdefs</b> | Preprocesses a message source file for input to the gencat utility                                |
| <b>runcat</b>    | Invokes mkcatdefs and pipes the message catalog source data (the output from mkcatdefs) to gencat |
| <b>dspcat</b>    | Displays all or part of a message catalog   |
| <b>dspmsg</b>    | Displays a selected message from a message catalog  |
- The OS/390 UNIX Debugger feature, which provides the dbx interactive symbolic debugger for OS/390 UNIX applications
  - OS/390 UNIX, which provides access to a hierarchical file system (HFS), with support for the POSIX.1 and XPG4 standards
  - OS/390 C/C++ I/O routines, which support using HFS files, standard OS/390 data sets, or a mixture of both
  - Application threads (with support for a subset of POSIX.4a)
  - Support for OS/390 C/C++ DLLs

OS/390 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 OS/390 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 OS/390 UNIX environment can enhance your productivity. Refer to the *OS/390 UNIX System Services User's Guide* for more information on the Shell and Utilities.

---

## OS/390 C/C++ Applications with OS/390 UNIX C/C++ Functions

Most OS/390 UNIX C functions are available at all times. However, to use some OS/390 UNIX C functions, you must run an OS/390 C/C++ program on a system where the OS/390 UNIX kernel is available and active. In some situations, you must also 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 the *OS/390 C/C++ Run-Time Library Reference* for more information about requirements for each function.

You can invoke an OS/390 C/C++ program that uses OS/390 UNIX C functions using the following methods:

- Directly from the OS/390 UNIX Shell.
- From another program, or from the OS/390 UNIX 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(0N)` runtime option.

---

## Input and Output

The C/C++ runtime library that supports the OS/390 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 OS/390 C/C++ extension to the ANSI standard.

#### **TCP/IP Sockets I/O**

OS/390 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 OS/390 UNIX sockets. OS/390 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 OS/390 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 OS/390 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

OS/390 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 the *OS/390 C/C++ Programming Guide*.

### Hierarchical File System Files

When you are running under MVS, TSO (batch and interactive), or IMS environments, OS/390 C/C++ recognizes a Hierarchical File System (HFS) file. The name specified on the `fopen()` or `freopen()` call has to conform to certain rules (described in the *OS/390 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

IBM OS/390 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 OS/390 — including support for multiple members opened for write
- Overlapped I/O support under OS/390 (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 (SP C) facility allows you to build applications that require no dynamic loading of OS/390 Language Environment libraries. It also allows you to tailor your application to better utilize the low-level services available on your operating system. SP C offers a number of advantages:

- You can develop applications that you can execute in a customized environment rather than with OS/390 Language Environment services. Note that if you do not use OS/390 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 OS/390 C language in place of assembler language when writing system exit routines, by using the interfaces that are provided by SP C.
- SP C lets you develop applications featuring a user-controlled environment, in which an OS/390 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 OS/390 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 OS/390 C++ compiler. However, your OS/390 C++ program can use interlanguage calls (ILC) to call OS/390 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 OS/390 C++ applications. OS/390 C++ code preprocessed on CICS/ESA V4 R1 cannot run under CICS/ESA V3 R3.
- DATABASE 2 (DB2)  
DB2 programs manage data that is stored in relational data bases. The IBM DATABASE 2 licensed program runs on OS/390.



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 OS/390 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 OS/390 C or OS/390 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)

OS/390 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 graphics pictures
- Support for many types of display terminals, printers, and plotters

- Query Management Facility (QMF)

OS/390 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.

---

## Additional Features of OS/390 C/C++

Feature	Description
Multibyte Character Support	OS/390 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 OS/390 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 <code>wcstombs()</code> , <code>mbstowcs()</code> , <code>wcsrtombs()</code> , and <code>mbsrtowcs()</code> , as well as the family of wide-character I/O functions. Wide-character data can be represented by the <code>wchar_t</code> data type.

Feature	Description
Extended Precision Floating-Point Numbers	<p>OS/390 C/C++ provides three S/370 floating-point number data types: single precision (32 bits), declared as <code>float</code>; double precision (64 bits), declared as <code>double</code>; and extended precision (128 bits), declared as <code>long double</code>.</p> <p>Extended precision floating-point numbers give greater accuracy to mathematical calculations.</p> <p>As of Release 6, OS/390 C/C++ also supports IEEE 754 floating-point representation. By default, <code>float</code>, <code>double</code>, and <code>long double</code> values are represented in IBM S/390 floating point format. However, the IEEE 754 floating-point representation is used if you specify the <code>FL0AT(IEEE754)</code> compile option. For details on this support, see “FLOAT” on page 91.</p>
Command Line Redirection	You can redirect the standard streams <code>stdin</code> , <code>stderr</code> , and <code>stdout</code> from the command line or when calling programs using the <code>system()</code> function.
National Language Support	OS/390 C/C++ provides message text in either American English or Japanese. You can dynamically switch between the two languages.
Locale Definition Support	OS/390 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 <code>isdigit()</code> , use this information.
Coded Character Set (Code page) Support	The OS/390 C/C++ compiler can compile C/C++ source written in different EBCDIC code pages. In addition, the <code>iconv</code> 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.
Multitasking Facility (MTF)	Multitasking is a mode of operation where your program performs two or more tasks at the same time. OS/390 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 OS/390 to allow a single OS/390 C application program to use more than one processor of a multiprocessing system simultaneously.
Packed Structures and Unions	OS/390 C provides support for packed structures and unions. Structures and unions may be packed to reduce the storage requirements of a OS/390 C program.
Fixed-point (Packed) Decimal Data	<p>OS/390 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 <code>COMP-3</code> or the PL/I data type <code>FIXED DEC</code>, with up to 31 digits of precision.</p> <p>The Application Support Class Library provides the Binary Coded Decimal Class for C++ programs.</p>
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 <code>system()</code> function under OS/390, OS/390 UNIX, and TSO. You can also use the <code>system()</code> function to call EXECs on OS/390 and TSO, or Shell scripts using OS/390 UNIX.
Exploitation of ESA	Support for OS/390, IMS/ESA, Hiperspace expanded storage, and CICS/ESA allows you to exploit the features of the ESA.



Feature	Description
Exploitation of hardware	<p>Use the ARCHITECTURE compiler option to select the minimum level of machine architecture on which your program will run. ARCH(3) enables support for IEEE 754 Binary Floating-Point instructions. ARCH(2) instructs the compiler to generate faster instruction sequences available only on newer machines.</p> <p>Use the TUNE compiler option to optimize your application for a selected machine architecture. Tune(3) optimizes your application for the new G5 processor. TUNE(2) optimizes your application for other architectures. For information on which machines and architectures support the above options, refer to "ARCHITECTURE" on page 73 and "TUNE" on page 161.</p>



---

## Chapter 3. Important Changes to the Prelinker Documentation

Prior to OS/390 C/C++ Version 2 Release 4, examples in this book showed how to use the prelinker and linkage-editor, and sections throughout the book discussed concepts of prelinking and linking. 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 OS/390 C/C++ Programs” on page 403.

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

- **Your output is a PDS, not a PDSE**

If you are using OS/390 batch or TSO, and your output must target a PDS instead of a PDSE, you cannot use the binder.

- **CICS**

CICS does not support PDSEs. If your program targets CICS, you cannot use the binder.

- **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.



---

## Part 2. User's Reference

This part reviews the basic steps for compiling, binding, and running OS/390 C/C++ programs under the OS/390 operating system. It also describes the options available to you at compile, IPA link, bind, and run time.

- "Chapter 4. OS/390 C Example" on page 35
- "Chapter 5. OS/390 C++ Examples" on page 39
- "Chapter 6. Compiler Options" on page 55
- "Chapter 7. Binder Options and Control Statements" on page 207
- "Chapter 8. Runtime Options" on page 217



---

## Chapter 4. OS/390 C Example

This chapter outlines the basic steps for compiling, binding, and running an OS/390 C example program under OS/390 batch, TSO, or the OS/390 shell.

If you have not yet compiled an OS/390 C program, some concepts in this chapter may be unfamiliar. Refer to “Chapter 9. Compiling” on page 221, “Chapter 12. Binding OS/390 C/C++ Programs” on page 289, and “Chapter 14. Running an OS/390 C/C++ Application” on page 335 for a detailed description on compiling, binding, and running an OS/390 C program.

This chapter describes steps to bind an OS/390 C example program. It does not describe the prelink and link steps. If you are using the prelinker, see “Appendix A. Prelinking and Linking OS/390 C/C++ Programs” on page 403.

The example program that this chapter describes is shipped with the OS/390 C compiler in the data set CBC.SCBCSAM.

---

### Example of an OS/390 C Program

The following example shows a simple OS/390 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 254.
- 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 OS/390 C language, see the *OS/390 C/C++ Language Reference* and the *OS/390 C/C++ Run-Time Library Reference*.

---

## Compiling, Binding, and Running the OS/390 C Example

In general, you can compile, bind, and run OS/390 C programs under OS/390 batch, TSO, or the OS/390 shell. You cannot run the IPA Link step under TSO, or under OS/390 batch by using the ISPF interface. For more information, see “Chapter 9. Compiling” on page 221, “Chapter 12. Binding OS/390 C/C++ Programs” on page 289, and “Chapter 14. Running an OS/390 C/C++ Application” on page 335.

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 OS/390 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.

### 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 OS/390 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 OS/390 Language Environment runtime library and the OS/390 C compiler are in the `STEPLIB`, `LPALST`, or `LNKLST` concatenation.
2. Compile the OS/390 C source. You can use the `REXX EXEC CC` to invoke the OS/390 C compiler under TSO as follows:

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

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 233.

3. Perform a bind:

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

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 OS/390 C/C++ Programs” on page 289 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 an OS/390 C/C++ Application” on page 335 for more information.

## Under the OS/390 Shell

1. Ensure that the OS/390 Language Environment runtime library and the OS/390 C compiler are in the STEPLIB, LPALST, or LNKLIST concatenation.
2. Put the source in the HFS. From the OS/390 shell type:

```
tso oput "'cbc.scbsam(cbc3uaam)' '$PWD/cbc3uaam.c'"
tso oput "'cbc.scbsam(cbc3uaan)' '$PWD/cbc3uaan.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 directory, the mkdir command to create a new directory, and the cd command to change the directory.

3. Compile and bind:

```
c89 -o ctof cbc3uaam.c
```

**Note:** You can use c89 to compile source that is stored in a data set.

4. Run the program:

```
./ctof
```

---

## Chapter 5. OS/390 C++ Examples

This chapter outlines the basic steps for compiling, binding, and running OS/390 C++ example programs under OS/390 batch, TSO, or the OS/390 shell.

If you have not yet compiled an OS/390 C++ program, some concepts in this chapter may be unfamiliar. Refer to “Chapter 9. Compiling” on page 221, “Chapter 12. Binding OS/390 C/C++ Programs” on page 289, and “Chapter 14. Running an OS/390 C/C++ Application” on page 335 for a detailed description on compiling, binding, and running an OS/390 C++ program.

The example programs that this chapter describes are shipped with the OS/390 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 an OS/390 C++ Program

The following example shows a simple OS/390 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 OS/390 C++ compiler in data sets `CBC.SCBCSAM(CBC3UBRC)` and `CBC.SCBCSAM(CBC3UBRH)`.

If you need more details on the constructs of the OS/390 C++ language, see the *OS/390 C/C++ Language Reference* or the *OS/390 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 5. 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 5. 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()) {

```

```

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);
}

```

Figure 6. OS/390 C++ Biorhythm Example Program (Part 2 of 2)

---

## Compiling, Binding, and Running the OS/390 C++ Example

In general, you can compile, bind, and run OS/390 C++ programs under OS/390 batch, TSO, or the OS/390 shell. You cannot run the IPA Link step under TSO, or under OS/390 batch by using the ISPF interface. For more information, see “Chapter 9. Compiling” on page 221, “Chapter 12. Binding OS/390 C/C++ Programs” on page 289, and “Chapter 14. Running an OS/390 C/C++ Application” on page 335.

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

### Under OS/390 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 7. JCL to Compile, Bind, and Run the Example Program Using the CBCCBG Procedure*

In Figure 7, 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.

For more information on compiling, binding, and running, see “Chapter 9. Compiling” on page 221, “Chapter 12. Binding OS/390 C/C++ Programs” on page 289, and “Chapter 14. Running an OS/390 C/C++ Application” on page 335.

### 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 OS/390 Language Environment runtime library, the OS/390 class library DLLs, and the OS/390 C++ compiler are in the STEPLIB, dynamic LPA, or Link List concatenation.
2. Compile the OS/390 C++ source. You can use the REXX EXEC CXX to invoke the OS/390 C++ compiler under TSO as follows:

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

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 233.

3. Bind:

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

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 OS/390 C/C++ Programs” on page 289.



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 337.

## Under the OS/390 Shell

1. Ensure that the OS/390 Language Environment runtime library and the OS/390 C++ compiler library are in the STEPLIB, LPALST, or LNKLST concatenation.
2. Put the source in the HFS. From the OS/390 shell type:

```
tso oput "'cbc.scbsam(cbc3ubrc)' '$PWD/cbc3ubrc.C'"
tso oput "'cbc.scbsam(cbc3ubrh)' '$PWD/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 directory.

3. Compile and bind:

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

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

4. Run the program:

```
./bio
```

---

## Example of an OS/390 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 OS/390 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 50) 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 8. Template of a Linked List

## CLB3ALST.H

```
??=ifndef _CBCLIST_H_
??=ifndef __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 9. 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 10. Template of an Iterator

## CLB3Aitr.H

```
??=ifndef _CBCITER_H
??=ifndef __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 11. 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 12. 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 13. OS/390 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 13. OS/390 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 OS/390 batch, TSO, and the OS/390 shell.

### Under OS/390 Batch

To compile, bind, and run the template example program under OS/390 batch, follow these steps:

1. Ensure that OS/390 Language Environment runtime library and the OS/390 C++ compiler are in STEPLIB, LPAIST, or the LNKLIB 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 14. 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 OS/390 Language Environment runtime library, the OS/390 Class Library DLLs, and the OS/390 C++ compiler are in STEPLIB, LPALST, or the LNKST 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 233 for more information.

3. Create a library from the PDS *userhlq*.TEMPINC.OBJ:

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

For more information see “Creating and Object Library Under TSO” on page 352.

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 305.

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 337.

## Under the OS/390 Shell

To compile, bind, and run the template example program under the OS/390 shell, follow these steps:

1. Ensure that OS/390 Language Environment runtime library and the OS/390 C++ compiler are in STEPLIB, LPALST, or the LNKST concatenation.
2. Perform a series of oput commands for all files that are used, as follows:

```
tso oput "'cbc.sclbsam(clb3atmp)'" '$PWD/clb3atmp.C'"  
tso oput "'cbc.sclbsam.h(clb3astr)'" '$PWD/clb3astr.h'"  
tso oput "'cbc.sclbsam.h(clb3aitr)'" '$PWD/clb3aitr.h'"  
tso oput "'cbc.sclbsam.h(clb3amin)'" '$PWD/clb3amin.h'"  
tso oput "'cbc.sclbsam.h(clb3amax)'" '$PWD/clb3amax.h'"  
tso oput "'cbc.sclbsam.h(clb3alst)'" '$PWD/clb3alst.h'"
```

```
tso oput "'cbc.sclbsam.c(clb3aitr)' '$PWD/clb3aitr.c'"
tso oput "'cbc.sclbsam.c(clb3alst)' '$PWD/clb3alst.c'"
tso oput "'cbc.sclbsam.c(clb3amax)' '$PWD/clb3amax.c'"
tso oput "'cbc.sclbsam.c(clb3amin)' '$PWD/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 OS/390 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 OS/390 batch.

See “Chapter 9. Compiling” on page 221, and “Appendix D. IBM Supplied Cataloged Procedures and REXX EXECs” on page 457 for details.

- In your own JCL procedure, by passing a parameter string to the compiler.
- In an options file. See “OPTFILE | NOOPTFILE” on page 129 for details.
- For OS/390 C, in a `#pragma options` preprocessor directive within your source file. See “Specifying OS/390 C Compiler Options Using `#pragma Options`” on page 58 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 `-W` option to pass options to the compiler.
- In the ISPF panels that are used to invoke the OS/390 C/C++ compiler in foreground and background.

The following compiler options are inserted at the bottom of your object module to indicate their status:

ALIAS	(C compile and IPA Link step only)
ANSIALIAS	(C compile and C++ compile only)
ARCHITECTURE	
ARGPARSE	
CONVLIT	
DLL	
EVENTS	
EXH	(C++ compile only)
EXECOPS	
EXPORTALL	(C compile and C++ compile only)
FLOAT	
GENPCH	
GONUMBER	
IPA	
INLINE	(C compile and IPA Link step only)
LANGLVL	
LIBANSI	
LOCALE	
LONGNAME	
OPTIMIZE	

PLIST	
REDIR	(C compile and IPA Link step only)
RENT	(C compile and IPA Link step only)
ROUND	
START	
STRICT	
SOM	(C++ compile only)
SOMEINIT	(C++ compile only)
SOMGS	(C++ compile only)
TARGET	
TEST	
TUNE	
UPCONV	(C compile only)
USEPCH	(C compile and C++ compile only)

## 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 *OS/390 C/C++ Programming Guide* for an overview of IPA before you use the IPA compiler option.

### 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).
- 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.

- You must specify either the LONGNAME compiler option or the #pragma longname preprocessor directive on the IPA Compile step (unless you are using the c89 utility). Otherwise, the compiler generates an unrecoverable error.
- 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 <sup>1</sup>
xxxx	xxxx	xxxx

**Note:** <sup>1</sup>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 OS/390 Shell

The OS/390 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 *OS/390 UNIX System Services Command Reference* for more information.

## Under OS/390 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 OS/390 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 *OS/390 C/C++ Language Reference*.

The #pragma options preprocessor directive must appear before the first OS/390 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	OBJECT
ALIAS	OPTIMIZE
ANSIALIAS	RENT
ARCHITECTURE	SERVICE
CHECKOUT	SPILL
DECK	START
GONUMBER	TEST
INLINE	TUNE
LIBANSI	UPCONV
MAXMEM	XREF

### 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 OS/390 UNIX

The c89 and c++ utilities specify most compiler options when they call the OS/390 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 LANTLRVL(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 -W. For example, specify -Wc,expo to export all functions and variables.

For a complete description of c89, c++ and related utilities, refer to the *OS/390 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 an OS/390 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 OS/390 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.

---

## 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 62 lists options that are compatible 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.

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

Compiler Option (Abbreviated Names are underlined)	IBM Supplied Default	C	C++	Accepted by IPA Link	More Information
<u>AGGREGATE</u>   <u>NOAGGREGATE</u>	NOAGG	✓		✓	See 70
<u>ALIAS</u> [(name)]   <u>NOALIAS</u>	NOALI	✓			See 71

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

Compiler Option (Abbreviated Names are underlined)	IBM Supplied Default	C	C++	Accepted by IPA Link	More Information
<u>ANSIALIAS</u>   <u>NOANSIALIAS</u>	ANS	✓	✓	✓	See 72
<u>ARCHITECTURE</u> ( <i>n</i> )	ARCH(0)	✓	✓	✓	See 73
<u>ARGPARSE</u>   <u>NOARGPARSE</u>	ARG	✓	✓	✓	See 74
<u>ATTRIBUTE</u> [(FULL)]   <u>NOATTRIBUTE</u>	ATT		✓	✓	See 75
<u>CHECKOUT</u> (subopts)   <u>NOCHECKOUT</u>	NOCHE	✓		✓	See 76
<u>CONVLIT</u>   <u>NOCONVLIT</u>	NOCONV	✓	✓	✓	See 78
<u>CSECT</u>   <u>NOCSECT</u>	NOCSE	✓	✓	✓	See 79
<u>DEFINE</u> (name1[=   =def1], name2[=   =def2],...)	no default user definitions	✓	✓	✓	See 82
<u>DIGRAPH</u>   <u>NODIGRAPH</u>	NODIGR		✓		See 82
<u>DLL</u> ( <u>CBA</u>   <u>NOCBA</u> )   <u>NODLL</u> ( <u>CBA</u>   <u>NCBA</u> )	NODLL(NOCBA)	✓		✓	See 84
<u>DLL</u> ( <u>CBA</u>   <u>NOCBA</u> )	DLL(NOCBA)		✓	✓	See 84
<u>EVENTS</u> [(filename)]   <u>NOEVENTS</u>	NOEVENT	✓	✓	✓	See 85
<u>EXECOPS</u>   <u>NOEXECOPS</u>	EXEC	✓	✓	✓	See 86
<u>EXH</u>   <u>NOEXH</u>	EXH		✓		See 87
<u>EXPMAC</u>   <u>NOEXPMAC</u>	NOEXP	✓	✓	✓	See 88
<u>EXPORTALL</u>   <u>NOEXPORTALL</u>	NOEXPO	✓	✓	✓	See 88
<u>FASTTEMPINC</u>   <u>NOFASTTEMPINC</u>	NOFASTT		✓		See 89
<u>FLAG</u> (severity)   <u>NOFLAG</u>	FL(I)	✓	✓	✓	See 90
<u>FLOAT</u> (subopts)	FLOAT[(HEX, FOLD, NOMAF, NORRM, NOAFP)]	✓	✓	✓	See 91
<u>GENPCH</u> [(filename)]   <u>NOGENPCH</u> (filename)	NOGENP	✓	✓	✓	See 95
<u>GONUMBER</u>   <u>NOGONUMBER</u>	NOGONUM	✓	✓	✓	See 96
<u>HALT</u> (num)	HALT(16)	✓	✓	✓	See 97
<u>INFO</u> (subopts)   <u>NOINFO</u>	NOINFO		✓		See 98
<u>INLINE</u> (subopts)   <u>NOINLINE</u> [(subopts)]	NOINL(AUTO, NOREPORT, 100, 1000)	✓		✓	See 99
<u>INLRPT</u> [(filename)]   <u>NOINLRPT</u> [(filename)]	NOINLR	✓	✓	✓	See 102
<u>IPA</u> [(subopts)]   <u>NOIPA</u> [(subopts)]	NOIPA(NOLINK, OBJECT, NOOPT, NOLIST, NOGONUMBER, NOATTRIBUTE, NOXREF, LEVEL(1),NOMAP, DUP, ER, NONCAL, NOUPCASE, NOCONTROL	✓	✓	✓	See 103
<u>LANGLVL</u> (ANSI SAA  SAAL2 COMPAT EXTENDED COMMONC)	LANG(EXTENDED)	✓	✓		See 107
<u>LIBANSI</u>   <u>NOLIBANSI</u>	NOLIB	✓		✓	See 110
<u>LIST</u> [(filename)]   <u>NOLIST</u> [(filename)]	NOLIS	✓	✓	✓	See 110



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

Compiler Option (Abbreviated Names are underlined)	IBM Supplied Default	C	C++	Accepted by IPA Link	More Information
<u>LOCALE</u> [(name)]   <u>NOLOCALE</u>	NOLOC	✓	✓	✓	See 112
<u>LONGNAME</u>   <u>NOLONGNAME</u>	NOLO (C only), LO (C++ only)	✓	✓	✓	See 114
<u>LSEARCH</u> (subopts)   <u>NOLSEARCH</u>	NOLSE	✓	✓	✓	See 115
<u>MARGINS</u>   <u>NOMARGINS</u>	NOMAR	✓	✓	✓	See 121
<u>MARGINS</u> (m,n)   <u>NOMARGINS</u> (C compile and IPA Link step )	V-format: NOMAR F-format: MAR(1,72)	✓		✓	See 121
<u>MAXMEM</u>   <u>NOMAXMEM</u>	MAXMEM(2097152)	✓	✓	✓	See 123
<u>MEMORY</u>   <u>NOMEMORY</u>	MEM	✓	✓	✓	See 124
<u>NESTINC</u> (num)   <u>NONESTINC</u>	NONEST	✓	✓	✓	See 125
<u>OBJECT</u> [(filename)]   <u>NOOBJECT</u> [(filename)]	OBJ	✓	✓	✓	See 125
<u>OE</u> [(filename)]   <u>NOOE</u> [(filename)]	NOOE	✓	✓	✓	See 127
<u>OFFSET</u>   <u>NOOFFSET</u>	NOOF	✓	✓	✓	See 128
<u>OPTFILE</u> [(filename)]   <u>NOOPTFILE</u> [(filename)]	NOOPTF	✓	✓	✓	See 129
<u>OPTIMIZE</u> [(level)]   <u>NOOPTIMIZE</u>	NOOPT	✓	✓	✓	See 131
<u>PHASEID</u>   <u>NOPHASEID</u>	NOPHASEID	✓	✓	✓	See 133
<u>PLIST</u> (HOST   OS)	PLIST(HOST)	✓	✓	✓	See 134
<u>PORT</u> (PPS   NOPPS)   <u>NOPORT</u> (PPS   NOPPS)	NOPORT(NOPPS)		✓	✓	See 134
<u>PPONLY</u> [(subopts)]   <u>NOPPONLY</u> [(subopts)]	NOPP	✓	✓	✓	See 136
<u>REDIR</u>   <u>NOREDIR</u>	RED	✓	✓	✓	See 138
<u>RENT</u>   <u>NORENT</u>	NORENT	✓		✓	See 139
<u>ROUND</u> (opt)	ROUND(N)	✓	✓	✓	See 140
<u>SEARCH</u> (opt1,opt2,...)   <u>NOSEARCH</u>	NOSE	✓	✓	✓	See 140
<u>SERVICE</u> (string)   <u>NOSERVICE</u>	NOSERV	✓	✓	✓	See 142
<u>SEQUENCE</u>   <u>NOSEQUENCE</u>	NOSEQ	✓	✓	✓	See 143
<u>SEQUENCE</u> (m,n)   <u>NOSEQUENCE</u>	V-format: NOSEQ F-format: SEQ(73,80)	✓		✓	See 143
<u>SHOWINC</u>   <u>NOSHOWINC</u>	NOSHOW	✓	✓	✓	See 145
<u>SOM</u>   <u>NOSOM</u>	NOSOM		✓		See 145
<u>SOMEINIT</u>   <u>NOSOMEINIT</u>	SOMEI		✓		See 146
<u>SOMGS</u>   <u>NOSOMGS</u>	NOSOMG		✓		See 146
<u>SOMRO</u> (classname)   <u>NOSOMRO</u>	NOSOMR		✓		See 147
<u>SOMVOLATTR</u>   <u>NOSOMVOLATTR</u>	NOSOMV		✓		See 148
<u>SOURCE</u> [(filename)]   <u>NOSOURCE</u> [(filename)]	NOSO	✓	✓	✓	See 148
<u>SPILL</u>   <u>NOSPILL</u>	SPILL(128)	✓	✓	✓	See 150
<u>SRCMSG</u>   <u>NOSRCMSG</u>	NOSRCM		✓		See 151
<u>SSCOMM</u>   <u>NOSSCOMM</u>	NOSS	✓		✓	See 151
<u>START</u>   <u>NOSTART</u>	STA	✓	✓	✓	See 152

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

Compiler Option (Abbreviated Names are underlined)	IBM Supplied Default	C	C++	Accepted by IPA Link	More Information
<u>STRICT</u>   <u>NOSTRICT</u>	STRICT	✓	✓	✓	See 153
<u>TARGET</u> (suboption)	TARG(LE)	✓	✓	✓	See 153
<u>TEMPINC</u> [(filename)]   <u>NOTEMPINC</u> [(filename)]	TEMPINC(TEMPINC) for PDS TEMPINC(./tempinc) for HFS		✓		See 156
<u>TERMINAL</u>   <u>NOTERMINAL</u>	TERM	✓	✓	✓	See 157
<u>TEST</u> [(subopts)]   <u>NOTEST</u> [(subopts)]	<ul style="list-style-type: none"> <li>C default: NOTEST (HOOK, SYM, BLOCK, LINE, PATH)</li> <li>C++ default: NOTEST(HOOK)</li> </ul>	✓	✓	✓	See 158
<u>TUNE</u> (n)	TUN(3)	✓	✓	✓	See 161
<u>UNDEFINE</u> (name)	no action	✓	✓	✓	See 163
<u>UPCONV</u>   <u>NOUPCONV</u>	NOUPC	✓		✓	See 163
<u>USEPCH</u> [(filename)]   <u>NOUSEPCH</u> [(filename)]	NOUSEP	✓	✓	✓	See 164
<u>WSIZEOF</u>   <u>NOWSIZEOF</u>	NOWSIZEOF	✓	✓	✓	See 165
<u>XREF</u>   <u>NOXREF</u>	NOXR	✓	✓	✓	See 166
<u>XSOMINC</u> [(subopts)]   <u>NOXSOMINC</u>	NOXS		✓		See 167

## 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++	Accepted by IPA Link	Replacement Option	More Information
<u>DECK</u>   <u>NODECK</u>	NODECK	✓		✓	OBJECT	See 169 and 125
<u>HWOPTS</u> (STRING   NOSTRING)   <u>NOHWOPTS</u>	NOHWO	✓		✓	ARCH	See 170 and 73
<u>OMVS</u> [(filename)]   <u>NOOMVS</u>	NOOMVS	✓		✓	OE	See 129 and 127
<u>SYSLIB</u> (pdsname-list)	no action	✓	✓	✓	SEARCH	See 140 and 170
<u>SYSPATH</u> (path1,path2,...)   <u>NOSYSPATH</u>	NOSYS		✓		SEARCH	See 140 and 171
<u>USERLIB</u> (pdsname-list)	no action	✓	✓	✓	LSEARCH	See 115 and 172
<u>USERPATH</u> (path1,path2,...)   <u>NOUSERPATH</u>	NOUSER		✓		LSEARCH	See 115 and 173

## 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	C++	Accepted by IPA Link	More Information
DECK	Produces an object module, and stores it in the data set associated with SYSPUNCH. <b>Use OBJECT instead of DECK.</b>	✓		✓	See 169 and 125
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 89
GENPCH	Generates precompiled header files.	✓	✓	✓	See 95
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 106
LSEARCH	Specifies the libraries or disks to be scanned for user include files.	✓	✓	✓	See 115
MEMORY	Improves compile-time performance by using a MEMORY file in place of a work file, if possible.	✓	✓	✓	See 124
OBJECT	Produces an object module, and stores it in the file that you specify, or in the data set associated with SYSLIN.	✓	✓	✓	See 125
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 127
OMVS	<b>The options OMVS and OE perform the same function. Use the OE option instead of the OMVS option.</b> 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 129 and 127
OPTFILE	Directs the compiler to look for compiler options in the file specified.	✓	✓	✓	See 129
SEARCH	Specifies the libraries or disks to be scanned for system include files.	✓	✓	✓	See 140
SYSLIB	Specifies the PDSs to be scanned for system include files. <b>Use SEARCH instead of SYSLIB.</b>	✓	✓	✓	See 140 and 170
SYSPATH	Specifies search paths to be scanned for system include files. <b>Use SEARCH instead of SYSPATH.</b>		✓		See 140 and 171

Table 6. Compiler Options for File Management (continued)

Option	Description	C	C++	Accepted by IPA Link	More Information
TEMPINC	Places template instantiation files in the PDS or HFS directory specified.		✓		See 156
USEPCH	Instructs the compiler to use precompiled header files.	✓	✓	✓	See 164
USERLIB	Specifies the PDSs to be scanned for your own include files. <b>Use LSEARCH instead of USERLIB.</b>	✓	✓	✓	See 115 and 172
USERPATH	Specifies search paths to be scanned for your own include files. <b>Use LEARCH instead of USERPATH.</b>		✓		See 115 and 173

## 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 7. Compiler Options That Control Listings

Option	Description	C	C++	Accepted by IPA Link	More Information
AGGREGATE	Lists structures and unions, and their sizes. The IPA Link step accepts but ignores this option.	✓		✓	See 70
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 75
EXPMAC	Lists all expanded macros. You must use the SOURCE option with EXPMAC.	✓	✓	✓	See 88
INLINE(,REPORT,,)	Generates a report on the status of inlined functions.	✓		✓	See 99
INLRPT	Generates a report on the status of inlined functions.	✓	✓	✓	See 102
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 103
LIST	Includes the object module in the compiler listing, in assembler-like code.	✓	✓	✓	See 110
OFFSET	Lists offset addresses relative to entry points of functions. The LIST option must be used with OFFSET.	✓	✓	✓	See 128
SHOWINC	Lists include files if SOURCE option specified.	✓	✓	✓	See 145

Table 7. Compiler Options That Control Listings (continued)

Option	Description	C	C++	Accepted by IPA Link	More Information
SOURCE	Lists source file.	✓	✓	✓	See 148
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. For IPA Link, generates the External Symbol Cross Reference, and 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 166

## Options for Debugging and Diagnosing Errors

These options help you to detect and correct errors in your OS/390 C/C++ program.

Table 8. Compiler Options for Debugging and Diagnostics

Option	Description	C	C++	Accepted by IPA Link	More Information
CHECKOUT	Gives informational messages for possible programming errors. The IPA Link step accepts but ignores this option.	✓		✓	See 76
EVENTS	Produces an events file that contains error information and source file statistics. The IPA Link step accepts but ignores this option.	✓	✓	✓	See 85
FLAG	Specifies the lowest severity level to be listed.	✓	✓	✓	See 90
GONUMBER	Generates line number tables for Debug Tool and error trace backs. The TEST option turns on GONUMBER.	✓	✓	✓	See 96
INFO	Generates informational messages.		✓		See 98
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 103
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 103
PHASEID	Causes each compiler module (phase) to issue an informational message which identifies the compiler phase module name, product id, and build level.	✓	✓	✓	See 133
SERVICE	Places a string in the object module, which is displayed in the traceback if the application fails abnormally.	✓	✓	✓	See 142
SRCMSG	Adds source code lines to diagnostic messages.		✓		See 151

Table 8. Compiler Options for Debugging and Diagnostics (continued)

Option	Description	C	C++	Accepted by IPA Link	More Information
TERMINAL	Directs diagnostic messages to be displayed on the terminal.	✓	✓	✓	See 157
TEST	Generates information that the Debug Tool needs to debug your program.	✓	✓	✓	See 158

## Options That Control the Source Code

These options allow you to control your OS/390 C/C++ source code.

Table 9. Summary of Compiler Options Used for Source Code Control

Option	Description	C	C++	Accepted by IPA Link	More Information
HALT	Specifies that the compiler stop processing files when it returns an error severity level of <i>n</i> or above.	✓	✓	✓	See 97
LANGLVL	Specifies the language standard to be used.	✓	✓		See 107
MARGINS	Identifies position of source to be scanned by the compiler.	✓	✓	✓	See 121
NESTINC	Specifies the number of nested include files to be allowed.	✓	✓	✓	See 125
SEQUENCE	Specifies the columns used for sequence numbers.	✓	✓	✓	See 143
SSCOMM	Allows comments to be specified by two slashes (//). The IPA Link step accepts but ignores this option.	✓		✓	See 151
UPCONV	Preserves <i>unsignedness</i> during OS/390 C/C++ type conversions. The IPA Link step accepts but ignores this option.	✓		✓	See 163
WSIZEOF	Causes the sizeof operator to return the widened size for function return types	✓	✓	✓	See 165

## Options That Control the Object Code

These options are used to control how your OS/390 C/C++ object code is produced.

Table 10. Summary of Compiler Options Used for Object Code Control

Option	Description	C	C++	Accepted by IPA	More Information
ALIAS	Generates ALIAS binder control statements for each required entry point.	✓			See 71
ANSIALIAS	Specifies whether type-based aliasing is to be used during optimization.	✓	✓	✓	See 72

Table 10. Summary of Compiler Options Used for Object Code Control (continued)

Option	Description	C	C++	Accepted by IPA	More Information
CSECT	Instructs the compiler to generate csect names in the output object module.	✓	✓	✓	See 79
DLL	Generates object code for DLLs or DLL applications.	✓		✓	See 84
EXECOPS	Allows runtime options to be passed to your program.	✓	✓	✓	See 86
EXH	Controls the generation of C++ exception handling code.		✓		See 87
EXPORTALL	Exports all externally defined functions and variables.	✓	✓	✓	See 88
FLOAT	Switches floating-point representation between IEEE and hexadecimal.	✓	✓	✓	See 91
HWOPTS	Generates code for different hardware features. Use ARCH instead of this option.	✓		✓	See 170
INLINE	Inlines user functions into source and helps maximize optimization.	✓		✓	See 99
NOINLINE	Disables inlining of user functions into source.	✓	✓	✓	See 99
IPA	Instructs the compiler to perform Interprocedural Analysis (IPA) processing.	✓	✓	✓	See 103
IPA(LEVEL)	Indicates the level of IPA optimization that the IPA Link step should perform.	✓	✓	✓	See 103
LIBANSI	Indicates whether functions with the name of an ANSI C library function are in fact ANSI C library functions.	✓	✓	✓	See 110
LONGNAME	Provides support for external names of mixed case and up to 1024 characters long.	✓	✓	✓	See 114
MAXMEM	Limits the amount of memory used for local tables of specific, memory intensive optimization.	✓	✓	✓	See 123
OBJECT	Produces an object module, and stores it in the file that you specify, or in the data set associated with SYSLIN.	✓	✓		125
OPTIMIZE	Improves runtime performance by introducing optimizations during code generation.	✓	✓	✓	See 131
RENT	Generates reentrant code. The IPA Link step accepts but ignores this option.	✓		✓	See 139
ROUND	Sets the rounding mode for binary floating point numbers.	✓	✓	✓	See 140
SPILL	Specifies the size of the spill area to be used for compilation.	✓	✓	✓	See 150
START	Generates a CEESTART whenever necessary.	✓	✓	✓	See 152
STRICT	Affects the precision of floating point calculations	✓	✓	✓	See 153
TARGET	Generates an object module for the targeted operating system or runtime library.	✓	✓	✓	See 153
TUNE	Specifies the architecture for which the executable program will be optimized.	✓	✓	✓	See 161

---

## Options That Control the Preprocessor

These options specify how the preprocessor runs.

Table 11. Summary of Compiler Options for Preprocessor

Option	Description	C	C++	Accepted by IPA Link	More Information
ARCHITECTURE	Specifies the architecture for which the executable program instructions are to be generated.	✓	✓	✓	See 73
CONVLIT	Turns on string literal codepage conversion.	✓	✓	✓	See 78
DEFINE	Defines preprocessor macro names.	✓	✓	✓	See 82
LOCALE	Specifies the locale to be used at compile time.	✓	✓	✓	See 112
PPONLY	Specifies that only the preprocessor is to be run and not the compiler.	✓	✓	✓	See 136
UNDEFINE	Removes any value its argument may have.	✓	✓	✓	See 163

---

## Options That Control Program Execution

These options control the execution of your program

Table 12. Summary of Compiler Options for Program Execution

Option	Description	C	C++	Accepted by IPA Link	More Information
ARGPARSE	Parses arguments provided on the invocation line.	✓	✓	✓	See 74
EXECOPS	Allows you to specify runtime options on the invocation line.	✓	✓	✓	See 86
PLIST	Specifies that the original operating system parameter list should be available.	✓	✓	✓	See 134
REDIR	Allows redirection of stderr, stdin, and stdout from the invocation line.	✓	✓	✓	See 138
TARGET	Generates an object module for the specified runtime environment.	✓	✓	✓	See 153

---

## Options That Control the IPA Object

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

Table 13. Compiler Options for IPA Object Control

Option	Description	C	C++	IPA	More Information
IPA(ATTRIBUTE)	Saves information about symbol storage offsets in the IPA object file.	✓	✓	✓	See 104



Table 13. Compiler Options for IPA Object Control (continued)

Option	Description	C	C++	IPA	More Information
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 104
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 104
IPA(OBJECT)	Indicates whether a conventional (non-IPA)/IPA object is to be produced during the IPA Compile step.	✓	✓	✓	See 104
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 104
IPA(XREF)	Saves information about symbol storage offsets in the IPA object file.	✓	✓	✓	See 104

## Options That Control the IPA Link Step

These options control the IPA Link step.

Table 14. Compiler Options for IPA Link Control

Option	Description	C	C++	IPA	More Information
IPA(LINK)	Instructs the compiler to perform IPA Link processing.	✓	✓	✓	See 106
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 106
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 106

---

## Direct-to-SOM Options

These options control the generation of SOM objects from your OS/390 C++ code.

Table 15. Summary of Compiler Options for SOM

Option	Description	C	C++	Accepted by IPA Link	More Information
SOM	Turns on implicit SOM mode, and causes som.hh to be included in a program.		✓		See 145
SOMGS	Instructs the compiler to disable direct access to attributes.		✓		See 146
SOMRO	Causes the release order to the specified classes to be written to standard output.		✓		See 147
SOMEINIT	Instructs the compiler to initialize SOM classes “early”, before the main function.		✓		See 146
SOMVOLATTR	Instructs the compiler to generate volatile _get and _set methods.		✓		See 148
XSOMINC	Instructs the compiler to exclude header files when implicit SOM mode is turned on.		✓		See 167

---

## Portability Options

These options allow you to port your C++ code to the OS/390 C++ compiler.

Table 16. Summary of Compiler Options for Portability

Option	Description	C	C++	Accepted by IPA Link	More Information
PORT	Adjusts the error recovery action that the compiler takes when it encounters an ill-formed #pragma pack directive.		✓		See 134

---

## Description of Compiler Options

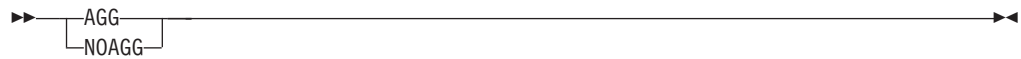
The following sections describe the compiler options and their usage. Compiler options are listed alphabetically. Syntax diagrams show the abbreviated forms of the compiler options.

### AGGREGATE | NOAGGREGATE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

DEFAULT: NOAGGREGATE

CATEGORY: Listing



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.

### 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓			✓	✓

DEFAULT: NOALIAS

CATEGORY: Object Code Control



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 to the NAME card.
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 *DFSMS/MVS Program Management*.

### Effect on IPA Compile Step

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

### Effect on IPA Link Step

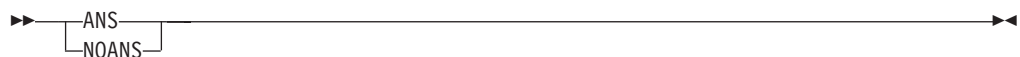
If you specify the ALIAS option on the IPA Link step, the IPA Link step generates an unrecoverable error.

## ANSIALIAS | NOANSIALIAS

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

DEFAULT: ANSIALIAS

CATEGORY: Source Code Analysis



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

The following are not subject to type-based aliasing:

- Signed and unsigned types. For example, a pointer to a *signed int* can point to an *unsigned int*.
- Character pointer types can point to any type.
- Types qualified as *volatile* or *const*. 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.

**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 107 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.

**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**

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: ARCHITECTURE(0)

CATEGORY: Object Code Control

►►—ARCH—(—n—)—————►►

The ARCHITECTURE option specifies the architecture for which the executable program's *instructions* are to be generated. It allows the optimizer to take advantage of specific hardware instruction sets. A subparameter specifies the group to which a model number belongs.

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 is optimized for 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-Exx, and 9672-Pxx
- 2 Produces code that is optimized for the following and follow on models:
  - 9672-Rx2, 9672-Rx3, 9672-Rx4, and 2003
- 3 Produces code that is optimized for the 9672 G5 and follow on models

**Note:** Code that is compiled at ARCH(1) runs on machines in the arch(1) group and later machines, including those in the arch(2) group. It may not run on earlier machines. Code that is compiled at ARCH(2) may not run on arch(1) or earlier machines.

## 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 161.

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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: ARGPARSE

CATEGORY: Program Execution



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**

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓	✓		✓

DEFAULT: NOATTRIBUTE

CATEGORY: Listing



The ATTRIBUTE option produces a Cross Reference listing that shows the attributes for each symbol, and an External Symbol Cross Reference 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 does not produce an attribute listing.

## Effect on IPA Compile Step

The `ATTRIBUTE` 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 `ATTRIBUTE` option for the IPA Link step, the IPA Link step generates an External Symbol Cross Reference listing section. It also generates a Storage Offset Listing if you created the IPA objects with the C compiler and specified the `XREF`, `IPA(ATTR)`, or `IPA(XREF)` option, and the IPA Link step did not coalesce the symbols for the current partition.

## CHECKOUT | NOCHECKOUT

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

DEFAULT: `NOCHECKOUT`

CATEGORY: `Debug/Diagnostic`



where:

`subopts` is one of the suboptions that are shown in Table 17.

The `CHECKOUT` option specifies that the compiler is to produce informational messages that indicate possible programming errors. The messages can help OS/390 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 17. *CHECKOUT* Suboptions, Abbreviations, and Descriptions

CHECKOUT Suboption	Abbreviated Name	Description
ACCURACY   NOACCURACY	AC   NOAC	Assignments of long values to variables that are not long
ENUM   <u>NOENUM</u>	EN   NOEN	Usage of enumerations
EXTERN   <u>NOEXTERN</u>	EX   NOEX	Unused variables that have external declarations



Table 17. CHECKOUT Suboptions, Abbreviations, and Descriptions (continued)

CHECKOUT Suboption	Abbreviated Name	Description
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
PORT   NOPORT	POR   NOPOR	Nonportable usage of the OS/390 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 *OS/390 C/C++ Language Reference* for more information regarding this pragma directive.

## 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.

## CONVLIT | NOCONVLIT

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOCONV

CATEGORY: Preprocessor



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 *OS/390 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"`

If you specify either SOM or PPOONLY 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**

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

DEFAULT: NOCSECT

CATEGORY: Object Code Control



The CSECT option ensures that the object module contains named CSECTs. 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 *OS/390 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. If the LONGNAME compiler option was not in effect when you specified `CSECT(qualifier)`, the compiler turns it on, and issues a warning message.

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. If the file name is more than 8 characters in length, *csectname* consists of the first 8 file name characters starting from the left.

**code CSECT** is named with the source file name in uppercase.

**data CSECT** is named with the source file name 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. The test CSECT name is always truncated to eight characters.

For example, if you compile `/u/cricket/project/mem1.ext.c` with the option CSECT, the test CSECT will have the name `$mem1.ex`

## 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 CSECT(qualifier), the compiler turns it on, and issues a warning 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*.

**Notes:**

1. The qualifier suboption takes advantage of the binder's capabilities, and may not generate names acceptable to the OS/390 Language Environment Prelinker.
2. The `#` that is appended as part of the `#C`, `#S`, or `#T` suffix is not locale-sensitive.
3. 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.
4. If the source file is either sequential or instream in your JCL, you must do one of the following to name your CSECT:
  - Specify a non-null suboption for the CSECT compiler option
  - Use the `#pragma csect` directive

Otherwise, you will receive an error message.

## 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:

- If you specify the CSECT option without a qualifier, 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 a warning each time.
- If you specify `CSECT(qualifier)`, the form of the CSECT name that IPA Link generates is altered. See "The IPA Link Step Control File" on page 277 for details.
- 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 message each time.
- 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.

## DEFINE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: no default user definitions

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 57 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))
```

### 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

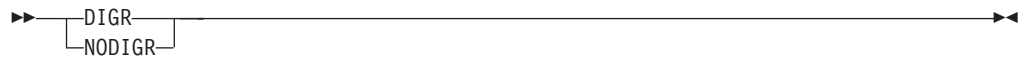
The IPA Link step accepts but ignores the `DEFINE` option.

## DIGRAPH | NODIGRAPH

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			✓

DEFAULT: NODIGRAPH

CATEGORY: Source Code Analysis



The DIGRAPH option allows you to use additional digraphs and keywords. A digraph is a combination of keys that produces a character not available on some keyboards. Table 18 shows the digraphs that OS/390 C++ supports:

Table 18. Digraphs

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

Table 19 shows additional keywords that OS/390 C++ supports:

Table 19. Additional Keywords

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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: NODLL (NOCBA) for C compile and IPA Link step

DLL (NOCBA) for C++ Compile

CATEGORY: Object Code Control



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 older applications compiled without the DLL option. This accommodation accounts for the incompatibility of function pointers created with and without the DLL compiler option. 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 *OS/390 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 *OS/390 C/C++ Programming Guide*.

### Notes:

1. You must use the LONGNAME and RENT options with the DLL option. If you use the DLL option without RENT and LONGNAME, the OS/390 C compiler automatically turns them on.



2. OS/390 C++ code is always DLL code. You cannot specify NODLL for OS/390 C++ code.

## 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)`.

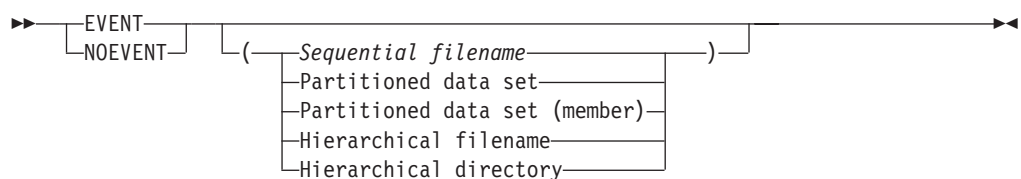
You should only export subprograms (functions and C++ methods) or variables that you need for the interface to the final DLL. If you export subprograms or variables unnecessarily (for example, by using the `EXPORTALL` option), you severely limit IPA optimization. Global variables are not coalesced, and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

## EVENTS | NOEVENTS

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOEVENTS

CATEGORY: Debug/Diagnostic



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 J. Layout of the Events File" on page 607.

### 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

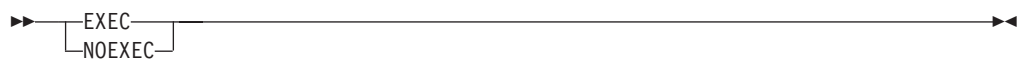
The IPA Link step accepts the EVENT option, but ignores it.

## EXECOPS | NOEXECOPS

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: EXECOPS

CATEGORY: Object Code Control and Program Execution



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.

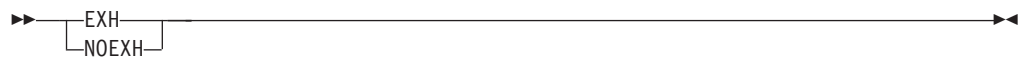
If you specify this option on both the IPA Compile and the IPA Link steps, the setting on the IPA Link step overrides the setting on the IPA Compile step. This situation occurs whether you use EXECOPS and NOEXECOPS as compiler options, or specify them by using the `#pragma runopts` directive on the IPA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

## EXH | NOEXH

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: EXH

CATEGORY: Object Code Control



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 is not ANSI conformant.

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 an LE 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

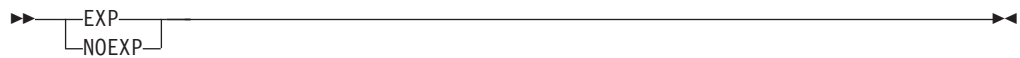
The IPA Link step issues a diagnostic message if you specify the EXH option for that step.

## EXPMAC | NOEXPMAC

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOEXPMAC

CATEGORY: Listing



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.

### 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOEXPORTALL

CATEGORY: Object Code Control



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 *OS/390 C/C++ Language Reference*.

- 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 OS/390 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.

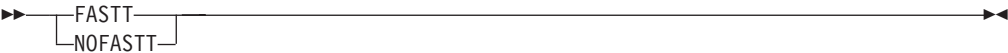
If you use the `EXPORTALL` option during the IPA Compile step, you severely limit IPA optimization. Refer to “DLL | NODLL” on page 84 for more information about the effects of this option on IPA processing.

## FASTTEMPINC | NOFASTTEMPINC

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOFASTT

CATEGORY: File Management



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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: FLAG (I)

CATEGORY: Debug/Diagnostic



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 F. OS/390 C/C++ Compiler Return Codes and Messages” on page 475 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT:

FLOAT(HEX, FOLD, NOMAF, NORRM, NOAFP\*)

\*dependent on ARCH() level

CATEGORY: Object Code Control



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 HEXADECEIMAL. In previous releases of OS/390 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. This makes floating-point calculations faster and more accurate, but the results may not be exactly equivalent to those produced by the two discrete instructions. This option may affect the precision of floating-point intermediate results.

**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

**Note:** To enable the AFP option, you must apply small programming enhancements (SPEs) to OS/390 V2R6.0, and to specific releases of some software. These SPEs are delivered as program temporary fixes (PTFs). Consult your System Programmer to ensure that the SPE PTFs that you require for IEEE binary floating-point support as documented in the *Planning for Installation* publication are applied to your system. The *Planning for Installation* publication documents the complete software requirements for IEEE binary floating-point support on OS/390.

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 *OS/390 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.

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. Subprograms that have the same RRM value are then placed in that partition.

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.

You can override the setting of `STRICT` 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.

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

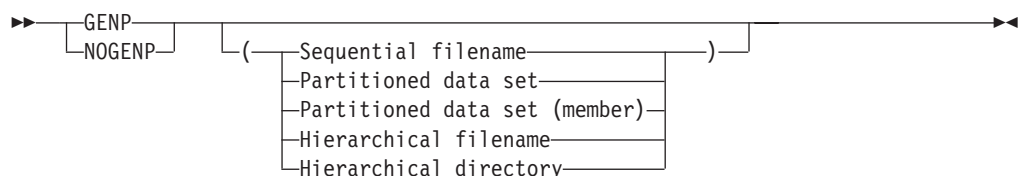
**Note:** The inlining of subprograms (C functions, C++ functions and methods) is inhibited if the `FLOAT` suboptions (including the floating-point mode), and the `ROUND` and `STRICT` options are not all compatible between compilation units. Calls between incompatible compilation units result in reduced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

## GENPCH | NOGENPCH

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: `NOGENPCH`

CATEGORY: File Management



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 SYSPCH ddname if you allocated one. If you did not allocate SYSPCH, 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 263.

#### 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 136, “SHOWINC | NOSHOWINC” on page 145, and “EXPMAC | NOEXPMAC” on page 88.
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 164.

### 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

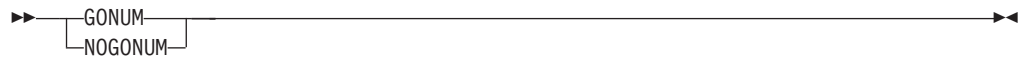
The IPA Link Step accepts the GENP option, but ignores it.

## GONUMBER | NOGONUMBER

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: NOGONUMBER

CATEGORY: Debug/Diagnostic



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.

### 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

### 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 57 and “LIST | NOLIST” on page 110 for more information.

## HALT(num)

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

DEFAULT: 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 F. OS/390 C/C++ Compiler Return Codes and Messages” on page 475 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**

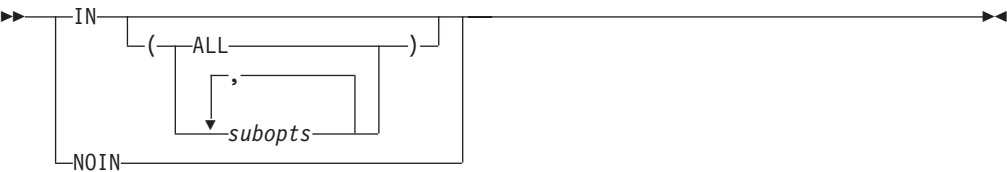
The HALT option affects the IPA Link step in a way similar to the way it affects the IPA Compile step, but the message severity levels may be different. Also, the severity levels for the IPA Link step and a C++ compilation include the “unrecoverable” level.

**INFO | NOINFO**

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	↙			

DEFAULT: 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).

### 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.

## INLINE | NOINLINE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		✓

DEFAULT for C Compile:

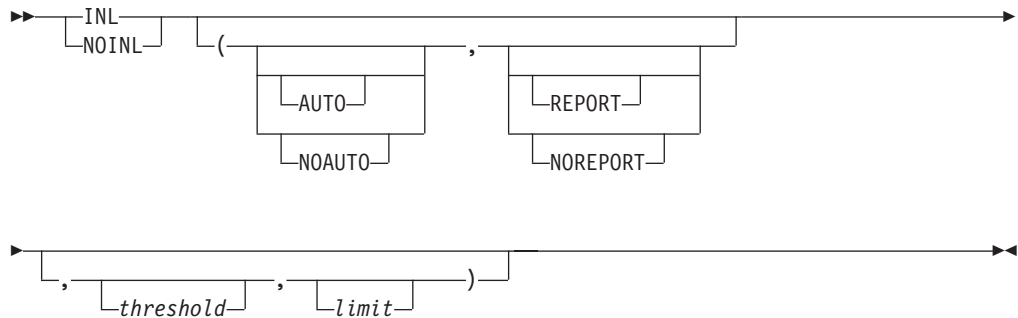
- If NOOPT is in effect: NOINLINE (AUTO,REPORT,100,1000)  
NOOPT is the default for C compile
- If OPT is in effect: INLINE(AUTO,NOREPORT,100,1000)

DEFAULT for 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.

CATEGORY: Object Code Control



The `INLINE` option instructs the compiler to place the code for selected functions at the point of call; this is called *inlining*. It eliminates the linkage overhead and exposes the entire inlined function 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 function, it removes the non-inlined instance of the function.
- If the compiler inlines all invocations of an externally visible function, it does not remove the non-inlined instance of the function. 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 *OS/390 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 functions within the *threshold* and *limit*.

If you specify `NOAUTO`, the inliner only inlines those functions specified with the `#pragma inline` directive. The `#pragma inline` and `#pragma noinline` directives allow you to determine which functions 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 *OS/390 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 192, “Inline Report” on page 182, and “Inline Report for IPA Inliner” on page 202.

The default is NOREPORT

***threshold***

The maximum relative size of a function 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 function; the OS/390 C compiler translates your OS/390 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 function can grow before auto-inlining stops. For C compile, the default for *limit* is 1000 ACUs for a function. For the IPA Link step, the default for *limit* is 8000 ACUs for that function. 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 functions when you are developing a program, you can specify the NOINLINE option on a #pragma options preprocessor directive. When you want to inline your functions, 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 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 functions will be inlined even if you have #pragma inline directives in your code.

## 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` function 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` function specifiers.

- If the IPA Link step inlines all invocations of a function, it removes the non-inlined instance of the function, unless the function 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 functions and externally visible functions in the same manner.

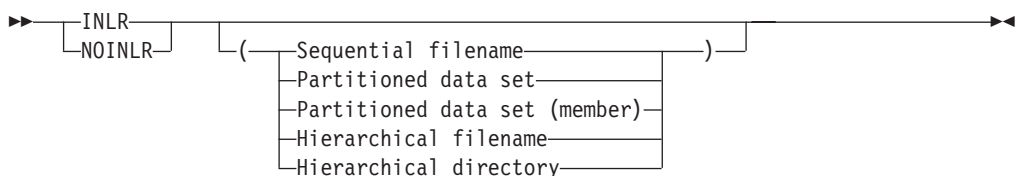
The IPA inliner has the inlining capabilities of the compilation unit inliner. In addition, the IPA inliner detects complex recursion, and may inline it. If you specify the `INLRPT` option, the IPA Link listing contains the IPA Inline Report section. This section is similar to the report that the compilation 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

DEFAULT: `NOINLRPT`

CATEGORY: Listing



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 functions 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)

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

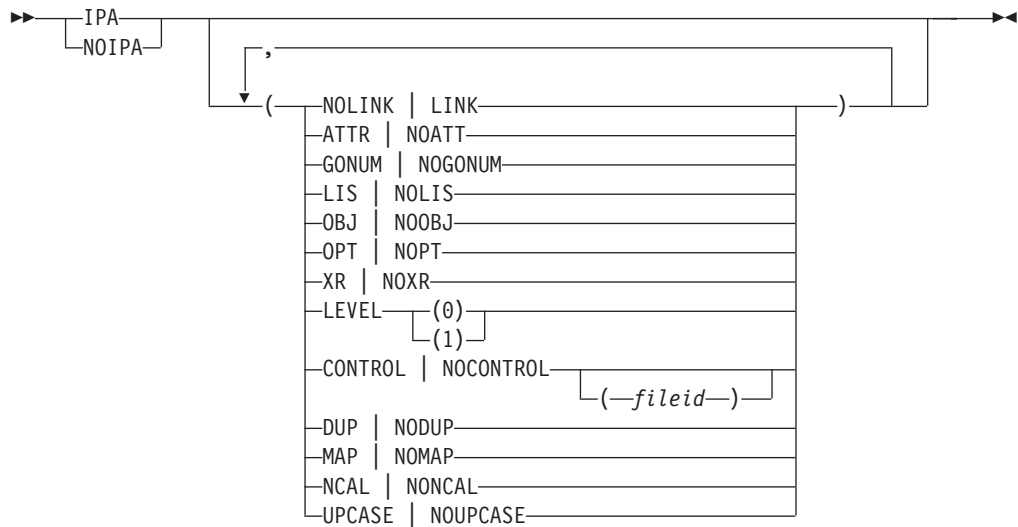
If you specify the INLRPT option on the IPA Link step, the IPA Link step listing contains an IPA Inline Report section. Refer to “INLINE | NOINLINE” on page 99 for more information about generating an IPA Inline Report section.

IPA | NOIPA

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: NOIPA

CATEGORY: Object Code Control/IPA Link Control



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 listing section after IPA Compile step source analysis is complete. It also does not generate a Storage Offset 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.

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 96 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 57.

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 110 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 57.

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

Indicates whether the IPA Compile step produces a non-IPA object in addition to the IPA object as part of the object file.

The default is IPA(OBJECT). The abbreviations are IPA(OBJ|NOOBJ).

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 listing section after IPA Compile step source

analysis is complete. It also does not cause the compiler to generate a Storage Offset or External Symbol Cross Reference listing section during IPA Compile step code generation.

Refer to “XREF | NOXREF” on page 166 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).

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)

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 function pruning and program partitioning only. IPA performs alias analysis quickly, with some loss of precision.

If you specify LEVEL(1), IPA performs all of the optimizations that it does at LEVEL(0), as well as function inlining and global variable coalescing. IPA performs more precise alias analysis for pointer dereferences and function calls.

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 193 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(UPCASE). The abbreviations are IPA(UPC|NOUPC).

Refer to the Interprocedural Analysis chapter in the *OS/390 C/C++ Programming Guide* for an overview and more details about Interprocedural Analysis.

## LANGLVL

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT:   LANGLVL(EXTENDED)

CATEGORY:   Source Code Control



The `LANGVL` 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 `LANGVL` suboptions are:

#### `LANGVL(ANSI)`

Indicates language constructs that are defined by ANSI. Some non-ANSI stub routines will exist even if you specify `LANGVL(ANSI)`, for compatibility with previous releases. The macro `__ANSI__` is defined as 1.

#### Notes:

1. You cannot use the compiler options `LANGVL(ANSI)` and `NOEXH` together, because `NOEXH` breaks ANSI conformance. If you specify either of the following, the compiler issues a warning message to indicate that it ignores `NOEXH`:
  - `NOEXH LANGVL(ANSI)`
  - `LANGVL(ANSI) NOEXH`
2. When you specify `LANGVL(ANSI)`, the compiler can still read and analyze the `_Packed` keyword in OS/390 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 `LANGVL(ANSI)` is specified at compile time, and the language level of the code will be ANSI.

#### `LANGVL(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 OS/390 C++. The macro `__COMPAT__` is defined as 1.

#### `LANGVL(COMMONC)`

Indicates language constructs that are defined by XPG, many of which `LANGVL(EXTENDED)` already supports. `LANGVL(ANSI)` and `LANGVL(EXTENDED)` do not support the following, but `LANGVL(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 function-like macro is equivalent to the ANSI/ISO token concatenation operator



The COMMONC suboption is available only for OS/390 C. The macro `__COMPAT__` 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 ANSI conformance code. Using `LANGLVL(COMMONC)` and `ANSIALIAS` together may have undesirable effects on your code at a high optimization level. See “`ANSIALIAS` | `NOANSIALIAS`” on page 72 for more information.

#### `LANGLVL(EXTENDED)`

Indicates all language constructs available with OS/390 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 OS/390 C. See the *OS/390 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 OS/390 C. See the *OS/390 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

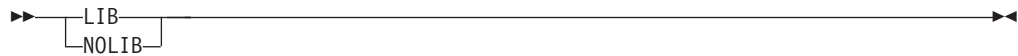
The IPA Link Step accepts but ignores the `LANGLVL` option.

## LIBANSI | NOLIBANSI

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: NOLIBANSI

CATEGORY: Code Optimization



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.

### 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

If you specify the LIBANSI 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()`.

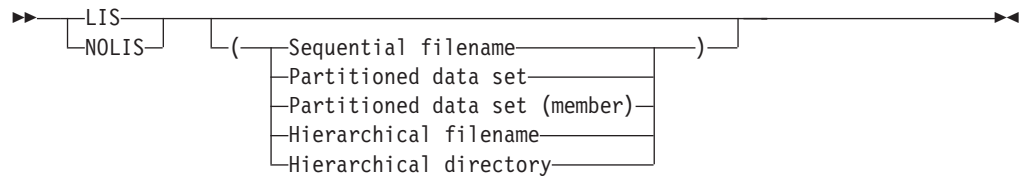
If you specify this option on both the IPA Compile and the IPA Link steps, the setting on the IPA Link step overrides the setting on the IPA Compile step. This applies whether you use LIBANSI and NOLIBANSI as compiler options or specify them using the `#pragma runopts` directive (on the IPA Compile step).

## LIST | NOLIST

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: NOLIST

CATEGORY: Listing



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.lst) LIST
CXX HELLO (LIST(/hello.lst)
```

If you specify data set names in an OS/390 C/C++ program, with the SOURCE, LIST or INLRPT options, all the listing sections are combined into the last data set name specified.

#### 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 57 and “GONUMBER | NOGONUMBER” on page 96 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 57 and “GONUMBER | NOGONUMBER” on page 96 for more information.

## LOCALE | NOLOCALE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: NOLOCALE

CATEGORY: Preprocessor



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 `OS/390 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 *OS/390 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT for C : NOLONGNAME

DEFAULT for C++ : LONGNAME

CATEGORY: Object Code Control



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 OS/390 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

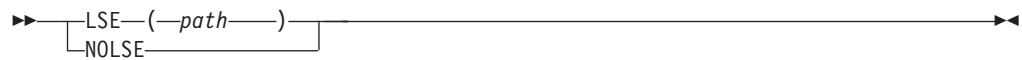
The IPA Link step ignores this option if you specify it, and uses the LONGNAME option for all partitions it generates.

## LSEARCH | NOLSEARCH

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOLSEARCH

CATEGORY: File Management



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 116 for details.

The `#include "filename"` format of the `#include` C/C++ preprocessor directive indicates user include files. See “Using Include Files” on page 246 for a description of the `#include` preprocessor directive.

For further information on library search sequences, see “Search Sequences for Include Files” on page 254.

### 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)
```

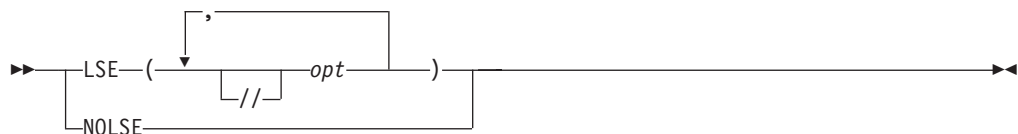
The compiler uses the following search sequence to look for your include files:

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 251 for more details on absolute `#include filename`.

## Additional Syntax

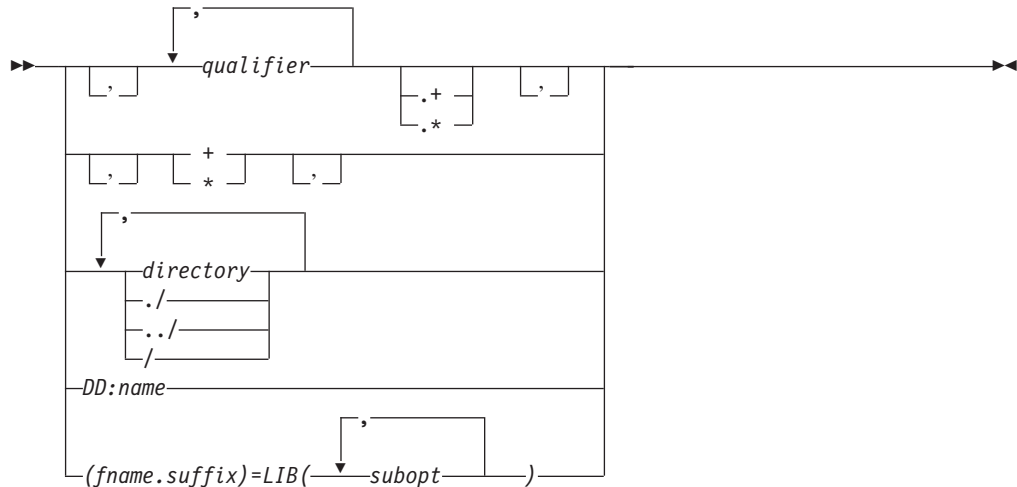


You must use the double slashes (`//`) to specify data set library searches when you specify the `o` compiler option. (You may use them regardless of the `o` option).



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:



*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.

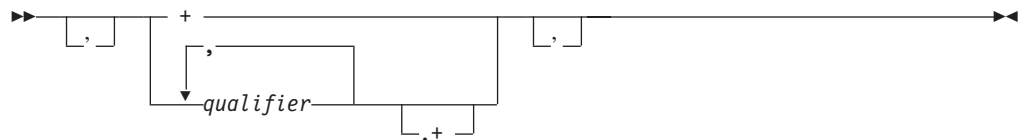
When the `#include filename` matches the pattern of *fname.suffix*, the search continues according to the subopts in the order specified. An asterisk (\*) in *fname* or *suffix* matches anything. If the compiler does not find the file, it attempts other searches according to the remaining options in LSEARCH.

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 253 for additional information on HFS files.

```
LSEARCH(/u/mike/myfiles)
#include "new/headers.h"
```

## Specifying Sequential Data Sets and PDSs

**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:



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.

- For the PDS file name:
  1. All the *paths* and slashes (slashes are replaced by periods)
  2. All the periods and *qualifiers* after the leftmost *qualifier*
- For the PDS member name, the leftmost *qualifier* is used as the member name

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 20 on page 120.

See “Forming Data Set Names with LSEARCH | SEARCH Options” on page 248 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)*

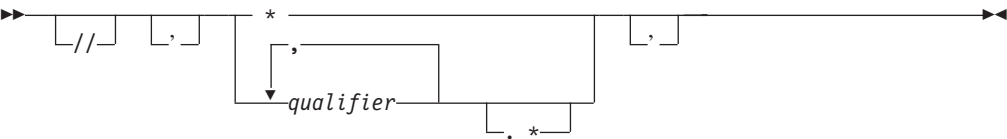
Also see the third example in Table 20.

**Examples:** The following example shows you how to specify a PDS search path:

Table 20. 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)	<i>userid.CC(A)</i>
#include "A.B.D"	LSEARCH(CC.+)	<i>userid.CC.B.D(A)</i>
#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(+)	<i>userid.A.B.H(DD)</i>
#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 21. 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

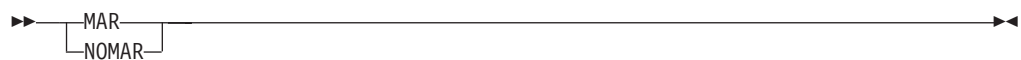
The IPA Link step accepts the LSEARCH option, but ignores it.

## MARGINS | NOMARGINS

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT for C++: NOMARGINS

### OS/390 C++



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.

In an OS/390 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.

If the MARGINS option is specified along with the SOURCE option in an OS/390 C program, only the range specified on the MARGINS option is shown in the compiler source listing.

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.

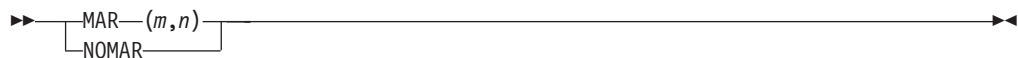
## Options for OS/390 C

DEFAULT for C:

- F-format: MARGINS (1,72)
- V-format: NOMARGINS

CATEGORY: Source Code Control

### OS/390 C



In an OS/390 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 OS/390 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 OS/390 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.

#### Notes:

1. The MARGINS option does not reformat listings.
2. If your program uses the `#include` preprocessor directive to include OS/390 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 OS/390 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: MAXMEM (2097152), or MAXMEM (\*), or MAXMEM (0)

CATEGORY: Object Code Control



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.

## 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: MEMORY

CATEGORY: File Management



The MEMORY option specifies that the compiler is to use a MEMORY file in place of a work-file if possible. See the *OS/390 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.

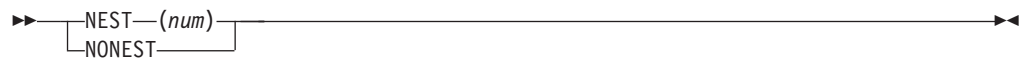


## NESTINC | NONESTINC

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NONESTINC

CATEGORY: Source Code Control



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

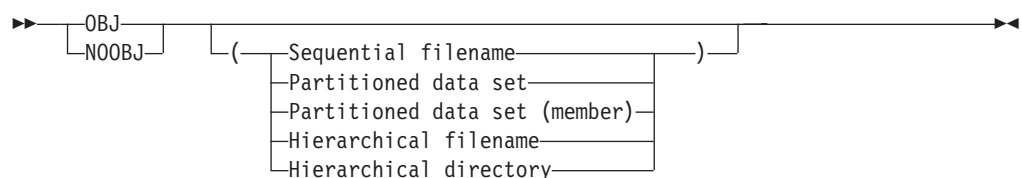
The IPA Link step accepts the NESTINC option, but ignores it.

## OBJECT | NOOBJECT

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: OBJECT

CATEGORY: File Management and Object Code Control



The OBJECT option specifies whether the compiler is to produce an object module.

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 `N0OBJ` 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 `N0OBJ`. For example, the following specifications have the same result:

```
CXX HELLO (N0OBJ(/hello.obj) OBJ
CXX HELLO (OBJ(/hello.obj)
```

If you specify `OBJ` and `N0OBJ` 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 (N0OBJ(/hello.obj) OBJ(/n1.obj) N0OBJ(/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.

**OS/390 C:** For OS/390 C programs, see Table 22 on page 169 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

The `OBJECT` suboption directs the IPA Compile step to generate an IPA or a combined IPA/conventional object module. 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 `N0OBJECT` 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 `OBJECT` suboption of the IPA option. Refer to “IPA | NOIPA” on page 103 for information about the `IPA(OBJECT)` option.

## 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 the *OS/390 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 103 for information about the `IPA(OBJECT)` option.

OE | NOOE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

DEFAULT: NOOE

CATEGORY: Source Code Control



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 OS/390 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

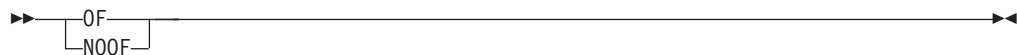
On the IPA Link step, the `OE` option controls the display of file names.

## OFFSET | NOOFFSET

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: `NOOFFSET`

CATEGORY: Listing



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.

### 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.

## OMVS | NOOMVS

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		✓

Default: `NOOMVS`

CATEGORY: Source Code Control



In the OS/390 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 OS/390 C and OS/390 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 127.

### 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.

## OPTFILE | NOOPTFILE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: `NOOPTFILE`

CATEGORY: File Management



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 lf, lv, and lt 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(PROJ10PT) EXPORTALL'
```

If the file PROJ10PT 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(PROJ10PT) LONGNAME OPTFILE(PROJ20PT) LIST'
```

If the file PROJ10PT contains SO LIST and the file PROJ20PT 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.

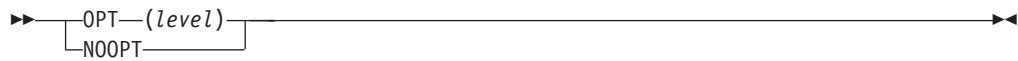
## OPTIMIZE | NOOPTIMIZE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT:

- C and C++ compile: NOOPTIMIZE
- IPA Link: OPTIMIZE

CATEGORY: Object Code Control



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** The optimization done at OPTIMIZE(1) and at OPTIMIZE(2) is identical.
- 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 the conformance to the ANSI standard may affect the global optimization of your program. You should have at least 8MB of memory to compile at this optimization level.

#### **no level**

OPTIMIZE specified with no level defaults to OPTIMIZE(2).

Inlining of functions in conjunction with other optimizations provides optimal run time performance. See “INLINE | NOINLINE” on page 99 for more information about the INLINE option and the *OS/390 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 72.

### **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 for the OPTIMIZE and LEVEL suboptions of the IPA option in “IPA | NOIPA” on page 103 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.

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 103 for information about using the OPTIMIZE option under IPA.

PHASEID

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOPHASEID

CATEGORY: Debug/Diagnostic



If you specify the PHASEID option, it causes each compiler module (phase) to issue an informational message as the phase begins execution. This message identifies compiler phase module name, product id, 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 module(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

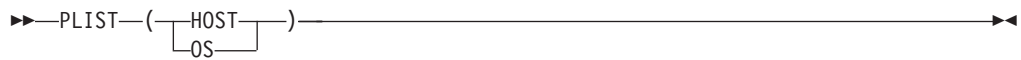
The IPA Link step uses the PHASEID option that you specify for that step.

## PLIST

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: PLIST (HOST)

CATEGORY: Program Execution



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 *OS/390 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓	✓		

DEFAULT: NOPORT (NOPPS)

CATEGORY: Portability



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` 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

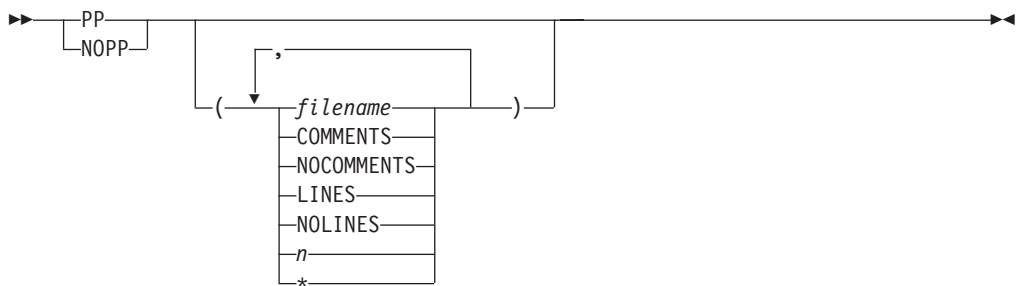
The IPA link step accepts the PORT option but ignores it.

## PPONLY | NOPPONLY

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOPPONLY

CATEGORY: Preprocessor



The PPOPTION 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 PPOPTION 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.

<i>n</i>	If a parameter <i>n</i> , which is an integer between 2 and 32760 inclusive, is specified, all lines are folded at column <i>n</i> .
*	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 PPOONLY suboptions are cumulative. If you specify suboptions in multiple instances of PPOONLY and NOPPOONLY, 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 (NOPPOONLY(/aa.exp) PPOONLY(LINES) PPOONLY(NOLINES)
```

```
CXX HELLO (PPOONLY(/aa.exp,LINES,NOLINES)
```

```
CXX HELLO (PPOONLY(/aa.exp,NOLINES)
```

All #line and #pragma preprocessor directives (except for margins and sequence directives) remain. When you specify PPOONLY(\*), #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 PPOONLY option, the compiler turns on the TERMINAL option. If you specify the SHOWINC, XREF, AGGREGATE, or EXPMAC options with the PPOONLY option, the compiler issues a warning, and ignores the options.

If you specify the PPOONLY 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")
```

where ??= 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 *OS/390 C/C++ Programming Guide*.

The NOPPOONLY option specifies that both the preprocessor and the compiler are to be run against the source file.

If you specify both PPOONLY and NOPPOONLY, the last one that is specified is used.

## Effect on IPA Compile Step

The PPOONLY 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

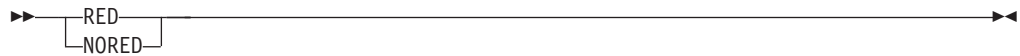
The IPA Link step accepts the PPOONLY option, but ignores it.

## REDIR | NOREDIR

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: REDIR

CATEGORY: Program Execution



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()`.

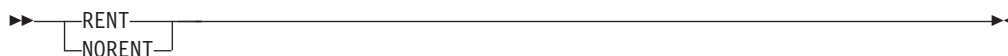
If you specify this option on both the IPA Compile and the IPA Link steps, the setting on the IPA Link step overrides the setting on the IPA Compile step. This situation occurs whether you use REDIR and NOREDIR as compiler options or specify them using the `#pragma runopts` directive (on the IPA Compile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

## RENT | NORENT

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓	✓	✓

DEFAULT: NORENT

CATEGORY: Object Code Control



The RENT option specifies that the compiler is to take code that is not naturally reentrant and make it reentrant. Refer to the *OS/390 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 OS/390 C/C++ Programs” on page 289, or the prelinker, which is described in “Appendix A. Prelinking and Linking OS/390 C/C++ Programs” on page 403.

### 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. OS/390 C++ code always uses constructed reentrancy.
3. RENT variables reside in the modifiable writable static area for both OS/390 C and OS/390 C++ programs.
4. NORENT variables reside in the code area (which may be write protected) for both OS/390 C and OS/390 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.

### 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.

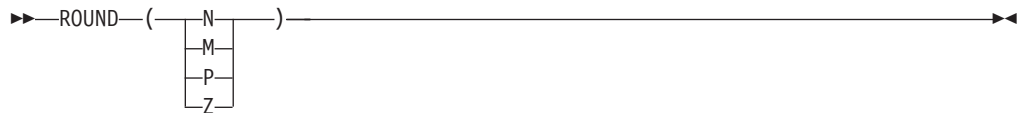
If you generate an IPA Link listing by using the LIST or MAP compiler option, the IPA Link step generates a Partition Map listing section for each partition. If any symbols within a partition are reentrant, the options section of the Partition Map displays the RENT compiler option.

## ROUND

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: ROUND(N)

CATEGORY: Object Code Control



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 91 for further information.

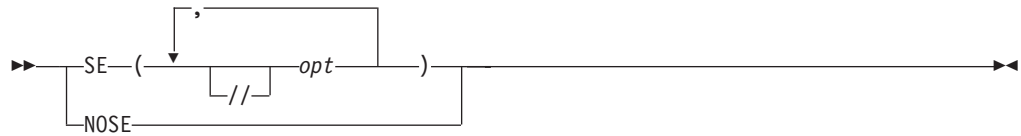
## SEARCH | NOSEARCH

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOSEARCH



## 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 246 for a description of the `#include` preprocessor directive.

For further information on library search sequences, see “Search Sequences for Include Files” on page 254.

The suboptions for the `SEARCH` option are identical to those for the `LSEARCH` option, as described on page “`LSEARCH | NOSEARCH`” on page 115.

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 251 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

DEFAULT: NOSERVICE

CATEGORY: Debug/Diagnostic

►►—SERV—(—string—)—————►►

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 OS/390 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.
- 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.
- 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.

### 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

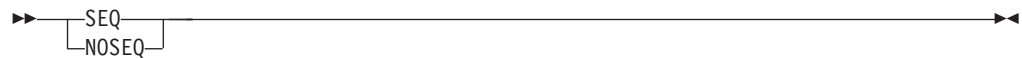
If you specify the **SERVICE** option on the IPA Compile step, or specify `#pragma options(SERVICE)` in your code, it has no effect on the IPA Link step. Only the **SERVICE** option you specify on the IPA Link step affects the generation of the service string for that step.

## SEQUENCE | NOSEQUENCE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

CATEGORY: Source Code Control

DEFAULT for C++ NOSEQUENCE



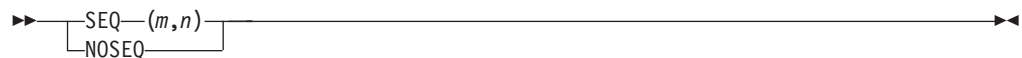
The SEQUENCE option defines the section of the input record that is to contain sequence numbers. No attempt is made to sort the input lines or records into the specified sequence or to report records out of sequence.

For OS/390 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.

### Options for OS/390 C

DEFAULT

- for C(F-Format): SEQUENCE (73,80)
- for C(V-Format): NOSEQUENCE



Under OS/390 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 OS/390 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 OS/390 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**

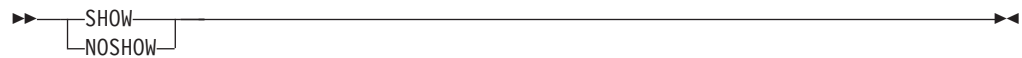
The IPA Link step accepts the SEQUENCE option, but ignores it.

## SHOWINC | NOSHOWINC

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOSHOWINC

CATEGORY: Listing



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.

### 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

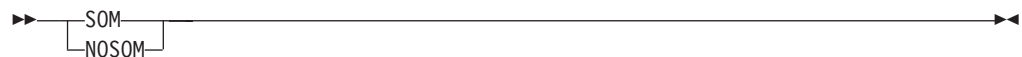
The IPA Link step accepts the SHOWINC option, but ignores it.

## SOM | NOSOM

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOSOM

CATEGORY: Direct-to-SOM



The SOM option turns on implicit SOM mode, and also causes the file `<som.hh>` to be included. It is equivalent to placing `#pragma SOMAsDefault(on)` at the start of the translation unit.

All classes are implicitly derived from `SOMObject` until a `#pragma SOMAsDefault(off)` is encountered.

For further details see the *OS/390 C/C++ Programming Guide*.

## Effect on IPA Compile Step

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

## Effect on IPA Link Step

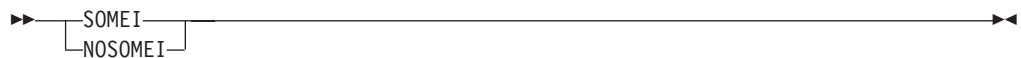
The IPA Link step issues a diagnostic message if you specify the SOM option for that step.

## SOMEINIT | NOSOMEINIT

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: SOMEINIT

CATEGORY: Direct-to-SOM



The SOMEINIT option instructs the compiler to initialize SOM classes *early*, before the main function. This reduces the size of the generated object module, by avoiding unnecessary checks to determine whether or not the SOM class is initialized.

With either the SOMEINIT or NOSOMEINIT option in effect, any reference to a static member of a SOM class initializes the class *early*.

This option has no effect if SOM mode is off.

## Effect on IPA Compile Step

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

## Effect on IPA Link Step

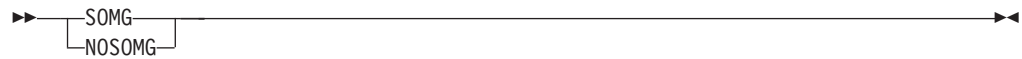
The IPA Link step issues a diagnostic message if you specify the SOMEINIT option for that step.

## SOMGS | NOSOMGS

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOSOMGS

CATEGORY: Direct-to-SOM



The SOMGS option instructs the compiler to disable direct access to attributes. Instead, the get and set methods are used. This is equivalent to specifying `#pragma SOMNoDataDirect(on)` as the first line of the translation unit.

The default NOSOMGS means that the direct data access method will be used. This option has no effect if SOM mode is off.

### Effect on IPA Compile Step

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

### Effect on IPA Link Step

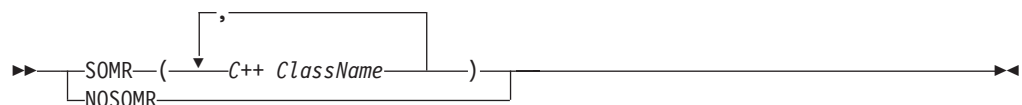
The IPA Link step issues a diagnostic message if you specify the SOMSGS option for that step.

## SOMRO | NOSOMRO

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOSOMRO

CATEGORY: Direct-to-SOM



The SOMRO option instructs the compiler to write the release order of the specified classes to standard output in the form of a `SOMReleaseOrder` pragma. You can capture the output from this option when developing new SOM classes, and include the pragma in the class definition.

The SOMRO option has no effect if SOM mode is off.

For more information, see the *OS/390 C/C++ Programming Guide*.

### Effect on IPA Compile Step

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

### Effect on IPA Link Step

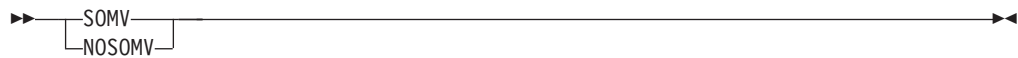
The IPA Link step issues a diagnostic message if you specify the SOMRO option for that step.

## SOMVOLATTR | NOSOMVOLATTR

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOSOMVOLATTR

CATEGORY: Direct-to-SOM



The SOMVOLATTR option tells the compiler to generate volatile versions of the SOM attribute `_get` and `_set` methods.

The following example shows the differences between the `_get` and `_set` methods generated by OS/390 C++ without SOMVOLATTR, and the `_get` and `_set` methods generated by OS/390 C++ with SOMVOLATTR. For example, consider the following class:

```
class Temp : public SOMObject {
public:
    int a;
    #pragma SOMAttribute(a);
};
```

The OS/390 C++ compiler without the SOMVOLATTR option generates the following function signatures for the accessor methods:

```
int Temp::_get_a ( )
void Temp::_set_a ( int )
```

The OS/390 C++ compiler with the SOMVOLATTR option generates the following function signatures for the accessor methods:

```
int Temp::_get_a ( ) const volatile
void Temp::_set_a ( int ) volatile
```

### Effect on IPA Compile Step

The SOMVOLATTR 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 SOMVOLATTR option for that step.

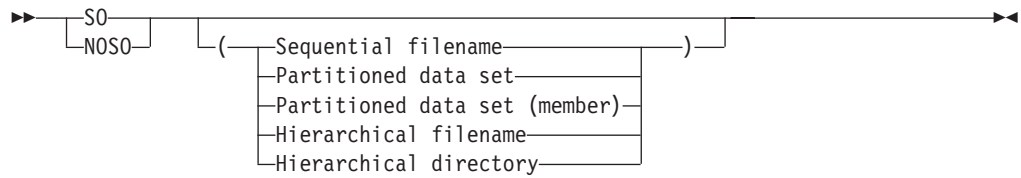
## SOURCE | NOSOURCE

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOSOURCE



## CATEGORY: Listing



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 `SYSPRT` 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.lst) SO
CXX HELLO (SO(/hello.lst)
```

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.lst) SO(/n1.lst) NOSO(/test.lst) SO
CXX HELLO (SO(/test.lst)
```

### 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

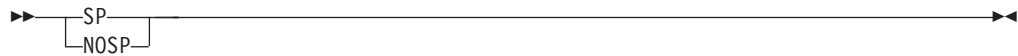
The IPA Link step accepts the `SOURCE` option, but ignores it.

## SPILL | NOSPILL

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		✓

DEFAULT: SPILL(128)

CATEGORY: Object Code Control



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 into temporary storage that is called the spill area.

You may have to expand the spill area; if so, you will receive a compiler message telling you the size to which you should increase the spill area. Once you know the spill area that your source program requires, you can add a `#pragma options(SPILL( size))` directive to your source. The maximum spill area size is 3900. Typically, you will only need to specify this option when compiling very large programs with **OPTIMIZE**.

### Notes:

1. There is an upper limit of 4096 bytes for the combined area for your spill area, local variables and arguments passed to called functions at **OPT**. For best use of your stack, do not pass large arguments, such as structures, by value.
2. If you specify **NOSPILL**, the compiler defaults to **SPILL(128)**.

### 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, it uses the value of the option that you specify. The IPA Link step Prolog and Partition Map listing sections display the value of the option that you specify.

If you do not specify the option for the IPA Link step, the value used for a partition is the maximum **SPILL** option that you specified during the IPA Compile step for any compilation unit that provides code for that partition. The Prolog section of the IPA Link step listing does not display the value of the option, but the Partition Map listing section does.

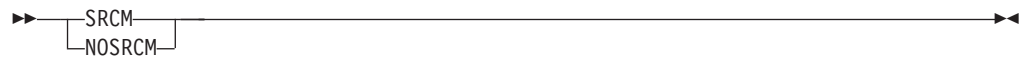
The option value that you specified for each IPA object file on the IPA Compile step appears in the IPA Link step Compiler Options Map listing section.

## SRCMSG | NOSRCMSG

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOSRCMSG

CATEGORY: Debug/Diagnostic



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.

NOSRCMSG indicates that the source lines are not added to the diagnostic messages.

### 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

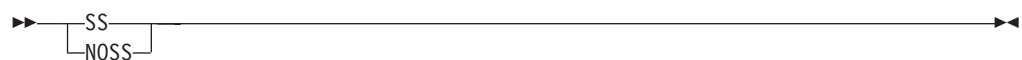
The IPA Link step issues a diagnostic message if you specify the SRCMSG option.

## SSCOMM | NOSSCOMM

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

DEFAULT: NOSSCOMM

CATEGORY: Source Code Control



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 OS/390 C program in your JCL stream DLM, be sure to change the delimiters so that your comments are recognized as OS/390 C comments and not as JCL statements. For example:

```
//COMPILE.SYSIN DD DATA,DLM=@@
#include <stdio.h>
void main(){
```

```
// OS/390 C comment
printf("hello world\n");
// A nested OS/390 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**

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT:   START

CATEGORY:   Object Code Control



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.

**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**

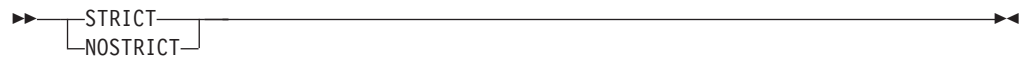
The IPA Link step uses the value of the START option that you specify for that step. It does not use the value that you specify for the IPA Compile step.

## STRICT | NOSTRICT

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: STRICT

CATEGORY: Object Code Control



The **STRICT** option instructs the compiler to perform computational operations in a rigidly defined order so that the results are always determinable and recreatable.

**NOSTRICT** allows the compiler to reorder certain computations for better performance. However, the end result may not be exactly the same as when **STRICT** is specified.

In IEEE floating-point mode, **NOSTRICT** turns on **FLOAT(MAF)** unless you explicitly specify **FLOAT(NOMAF)**.

### 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

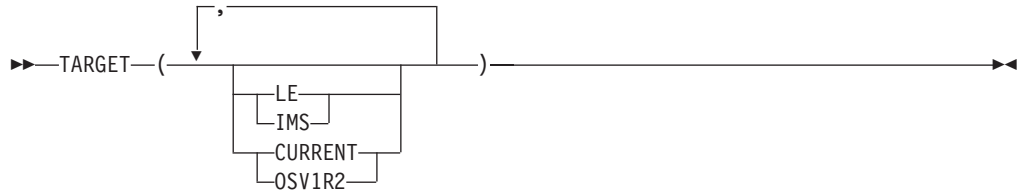
The IPA Link step merges and optimizes the application's code, and then divides it into sections for code generation. Each of these sections is a partition. The IPA Link step uses information from the IPA Compile step to ensure that an object is included in a compatible partition. See "FLOAT" on page 91 for more information on the effect of the **STRICT** option on the IPA Link step.

## TARGET

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: TARGET(LE, CURRENT)

CATEGORY: Program Execution and Object Code Control



The TARGET option specifies the runtime environment and OS/390 release for which the object module is to be generated. To use this option, select a runtime environment of either LE or IMS. Then select the OS/390 Release, CURRENT or OSV1R2. If you do not select a runtime environment or OS/390 Release the default is TARGET (LE,CURRENT) .

TARGET ( )            Generates object code to run under OS/390 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 OS/390 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 the topic TARGET Runtime Environment Suboptions (LE,IMS).

The following suboptions target the OS/390 Release at program runtime:

TARGET (CURRENT)      Generates object code to run under the current version of OS/390. This is the default.

TARGET (OSV1R2)       Generates object code to run under OS/390 Version 1 Release 2 and subsequent releases.

For more information about these suboptions refer to "TARGET OS/390 Release Suboptions (CURRENT, OSV1R2)".

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 the TARGET compile option more than once, the compiler uses the last set of specified suboptions.

## **TARGET OS/390 Release Suboptions (CURRENT, OSV1R2)**

The Target(OSV1R2) compiler option will allow you to generate code that can be executed on a OS/390 Version 1 Release 2 system and subsequent versions. In order to use this compiler option, you must utilize data sets for the Language Environment and Class Libraries from the appropriate level of OS/390.

For example, to generate code to execute on an OS/390 V1R3 system, use V1R3 Language Environment and Class Library datasets during the assembly, compilation and link-edit phases of application development on the OS/390 V2R6 system.

This compiler option will not allow you to exploit new functions provided on the higher level OS/390 system, but rather allow you to build an application on a higher level OS/390 system and execute on a lower level system.

**Restrictions for C/C++:** All input libraries used during the application build process must be the appropriate level for the target OS/390 system.

- These system input libraries include any LE or C++ libraries, such as header, object and Load Module (SCEEKLED) libraries.
- 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 Suboption	Restrictions
CURRENT	None.
OSV1R2	<ul style="list-style-type: none"> <li>• The macro <code>_TARGET_LIB_</code> is set to <code>X'21020000'</code>. The <code>_BFP_</code> (add work macro) is not defined.</li> <li>• The compiler disables the language feature <code>long long</code> and issues error messages if encountered in the source code.</li> <li>• The compiler ignores the following options and issues a warning message: <ul style="list-style-type: none"> <li><code>FLOAT()</code>            Default behaviour is <code>FLOAT(HEX)</code>.</li> <li><code>ROUND()</code>            Default behaviour is <code>ROUND(N)</code>.</li> <li><code>DLL(CBA)</code>            Default behaviour is <code>DLL(NOCBA)</code>.</li> </ul> </li> </ul>

**Restrictions for C:** TARGET(OSV1R2) is not permitted in a `#pragma options` directive.

If you specify TARGET(OSV1R2) on the command line, and one or more of the disallowed options appears in a `#pragma` directive, 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.

The user is responsible for ensuring that all IPA Object files are compiled with 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 CURRENT and OSV1R2 suboptions. 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 LE 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:

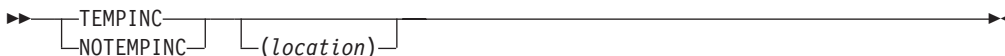
- It overrides the TARGET value that you specified for the IPA Compile step.
- It overrides the value that you specified for #pragma runopts(ENV). If you specify TARGET(LE) or TARGET(), the IPA Link step sets the value of #pragma runopts(ENV) to OS/390. If you specify TARGET(IMS), the IPA Link step sets 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: PDS TEMPINC(TEMPINC) or HFS directory TEMPINC(./tempinc)

CATEGORY: File Management



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 89 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

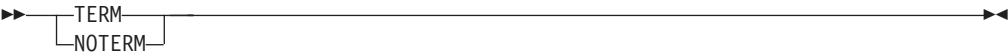
The IPA Link step issues a diagnostic message if you specify the TEMPINC option for that step.

TERMINAL | NOTERMINAL

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT:    TERMINAL

CATEGORY:   Debug/Diagnostic



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 OS/390 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

CATEGORY: Debug/Diagnostic

DEFAULT: NOTEST(HOOK)

### OS/390 C /C++



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	<ul style="list-style-type: none"> <li>For C++ compile, generates all possible hooks</li> <li>For C compile, generates all possible hooks based on current settings of BLOCK, LINE, and PATH suboptions.</li> <li>For IPA Link, generates Function Entry, Function Exit, Function Call, and Function Return hooks</li> <li>For C++ compile, generates symbol information</li> <li>For C compile, generates symbol information unless NOSYM is specified</li> <li>For IPA Link, does not generate symbol information</li> </ul>	<ul style="list-style-type: none"> <li>Generates Function Entry, Function Exit, Function Call and Function Return hooks</li> <li>Does not generate symbol information</li> </ul>
NOHOOK	<ul style="list-style-type: none"> <li>Does not generate any hooks</li> <li>For C++ compile, generates symbol information.</li> <li>For C compile, generates symbol information based on the current settings of SYM and BLOCK</li> <li>For IPA Link, does not generate any symbol information</li> </ul>	<ul style="list-style-type: none"> <li>Does not generate any hooks</li> <li>Does not generate symbol information</li> </ul>

The TEST suboptions generate symbol tables and program hooks that the Debug Tool needs to debug your program. The choices you make when compiling your program affect the amount of Debug Tool function available during your debugging session.

**Trace Program Execution:** To look at the flow of your code to aid in problem determination, use the H00K suboption with OPT in effect. Function entry, function exit, function call, and function return hooks are generated. No symbol information is generated.

When NOOPT is in effect, and you use the H00K suboption, the debugger runs slower, but all the Debug Tool commands such as AT ENTRY \* are available.

**Using TEST / NOTEST:** For OS/390 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.

If you specify the NOTEST option, debugging information is not generated.

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.

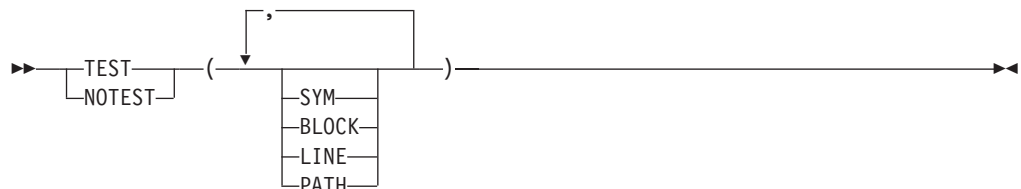
If you specify the TEST option, the compiler turn on GONUMBER.

**Note:** To debug the following types of code, use the latest Remote Debugger available from the OS/390 C/C++ Beta site at <http://www.software.ibm.com/ad/c390/cmvsbeta.htm>:

- IEEE code
- code that uses the long long data type
- code that runs in a POSIX environment

## Additional OS/390 C Compile Syntax

DEFAULT for C compile : NOTEST (H00K,SYM,BLOCK,LINE,PATH)



In addition to H00K and N0H00K, the following suboptions are available for C code. You can use these suboptions along with the H00K suboption for finer control when debugging your program (default suboptions are underlined):

<u>SYM</u>   NOSYM	Generates symbol table information
<u>BLOCK</u>   NOBLOCK	Generates symbol information for nested blocks
<u>LINE</u>   NOLINE	Generates line number hooks
<u>PATH</u>   NOPATH	Generates path breakpoints
ALL	Is equivalent to TEST (HOOK,SYM,BLOCK,LINE,PATH)
NONE	Is equivalent to TEST (HOOK,NOSYM,NOBLOCK,NOLINE,NOPATH)

## Additional OS/390 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).
- 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.
- 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.
- PATH** Generates hooks at all path points.
- 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.

- 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(LINE, BLOCK, 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(NOLINE, NOBLOCK, NOPATH, NOSYM).

For more information on debugging your program, see the *Debug Tool User's Guide and Reference* .

### 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: TUNE(3). If TUNE level is lower than ARCH, TUNE is forced to ARCH.

CATEGORY: Object Code Control

►►—TUN(n)—◄◄

The TUNE option specifies the architecture for which the executable program will be optimized. The option will allow the optimizer to take advantage of architectural differences such as scheduling of instructions.

You 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 and 9021-xx2
- 2** This option generates code that is executable on all models but that is optimized for the following and follow on models:
- 9672-Rx2, 9672-Rx3, 9672-Rx4, and 2003
  - 9672-Rx1, 9672-Exx, and 9672-Pxx
- 3** This option is the default. Produces code that is optimized for the 9672 G5 models

A comment that indicates the level of the TUNE option will be generated in your object module to aid you in diagnosing your program.

### **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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: 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.

### 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

DEFAULT: NOUPCONV

CATEGORY: Source Code Control

```

>> UPC
    NOUPC

```

The UPCONV option causes the OS/390 C compiler to follow unsignedness preserving rules when doing OS/390 C/C++ type conversions; that is, when widening all integral types (char, short, int, long). Use this option when compiling older OS/390 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.

## Effect on IPA Link Step

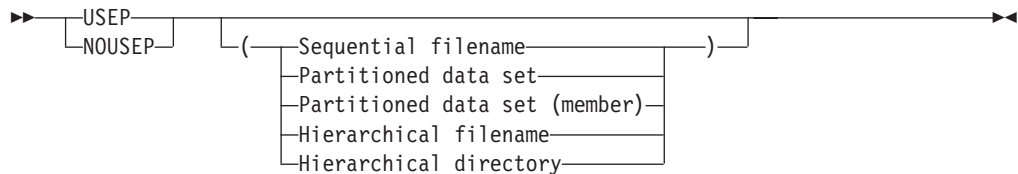
The IPA Link step accepts UPCONV option, but ignores it.

## USEPCH | NOUSEPCH

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOUSEPCH

CATEGORY: File Management



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 263.

### 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

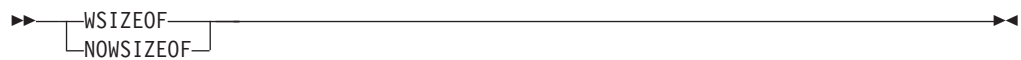
The IPA Link step accepts the USEP option, but ignores it.

## WSIZEOF | NOWSIZEOF

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: NOWSIZEOF

CATEGORY: Source Code Control



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 OS/390 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 *OS/390 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

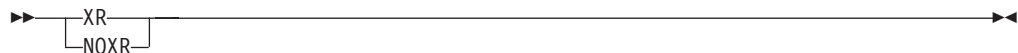
The IPA Link step accepts the WSIZEOF option, but ignores it.

## XREF | NOXREF

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓	✓	✓

DEFAULT: NOXREF

CATEGORY: Listing



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.

For C, a separate offset listing of the variables will appear after the cross reference table.

**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:

XREF(xxx)

## Effect on IPA Compile Step

For C, if you specify the XREF, IPA(ATTRIBUTE), or IPA(XREF) option for the IPA Compile step, the compiler saves symbol storage offset information in the IPA object file. No such information is produced for the regular object module that is produced by using IPA(OBJECT). The XREF|NOXREF compiler option and #pragma options(XREF|NOXREF) have the same effect on IPA.

For C++, this option has the same effect on the IPA Compile step as it does on a regular C++ compilation.

## Effect on IPA Link Step

If you specify the XREF option for the IPA Link step, it generates an External Symbol Cross Reference listing section for each partition.

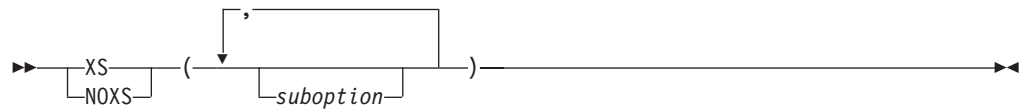
The IPA Link step creates a Storage Offset listing section if you created your IPA objects with the C compiler and the XREF, IPA(ATTR), or IPA(XREF) option, and if IPA did not coalesce the symbols for the current partition.

## XSOMINC | NOXSOMINC

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOXSOMINC

CATEGORY: Direct-to-SOM



The XSOMINC option allows you to exclude header files when implicit SOM mode is turned on. By excluding header files, you prevent the classes in those header files from deriving from the SOMObject class when implicit SOM mode is turned on. You can exclude header files for sets of partitioned data sets (PDSs) or Hierarchical File System (HFS) directories.

*suboption* may be one of the following:

- A set of partitioned data sets that are specified as data set qualifiers followed by a plus sign. If the 0E compiler option is in effect, you must prefix the first data set qualifier with a double slash (/). The double slash is optional if you do not specify the 0E option.
- An HFS directory or directories. The directory name must end with a slash. If you do not specify an ending slash, the compiler appends one.

The following examples illustrate the use of the different suboptions:

### Example 1- set of PDSs, quotes used around qualifiers

If you specify the following, any PDS name that starts with the qualifiers myuserid.TEST is excluded:

```
XSOMINC('myuserid.TEST.+')
```

For example, the compiler excludes the following data sets:

```
myuserid.TEST
myuserid.TEST.CXXHDR
```

### Example 2- set of PDSs, no quotes used around qualifiers

If you do not use single quotes around the data set qualifiers, the compiler prefixes the data set qualifiers with your default high-level qualifier.

For example, if you specify XSOMINC(TEST.+), and your default high-level qualifier is myuserid, the compiler excludes the following data sets:

```
myuserid.TEST
myuserid.TEST.CXXHDR
```

### Example 3-absolute HFS file name

If you specify an absolute HFS file name (one with a directory name) on an `#include` statement, the compiler compares that name against the suboptions specified on the `XSOMINC` option. If the directory names match, the compiler excludes the classes in the specified file.

For example, if you specify `XSOMINC(/dirname/)` and `#include "/dirname/a.h"`, the directory names match, and the compiler excludes `/dirname/a.h`.

### Example 4-relative HFS file name

If you specify a relative HFS file name (one without a directory name) on a `#include` statement, the compiler combines this file name with the suboptions specified for the `SEARCH` or `LSEARCH` options. The compiler compares the resulting file name to what you specified for the `XSOMINC` option, and if the directory names match, excludes the file.

For example, if you specify the following:

```
XSOMINC(/dirname/)
LSEARCH(/dirname/)
#include "new/a.h."
```

The resulting file name is `/dirname/new/a.h`. The directory name matches what you specified on the `XSOMINC` option, and the compiler excludes the file.

If you specify `XSOMINC` more than once, the compiler will use all of the `XSOMINC` suboptions to determine what to exclude. If you specify the `NOXSOMINC` option, the compiler does not use previously specified `XSOMINC` options. However, the compiler will use `XSOMINC` options that you specify after the `NOXSOMINC` option.

Although the default is `NOXSOMINC`, system programmers should set the default to `XSOMINC (/'high-level-qualifier.SCEEH.+ ')`, which instructs the compiler to exclude classes in all standard OS/390 C/C++ header files from inheriting from `SOMobject` when implicit `SOM` mode is on.

This option has no effect if implicit `SOM` mode is off.

### Effect on IPA Compile Step

The `XSOMINC` 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 `XSOMINC` option for that step.

---

## 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 in your code. For new programs, you should use the replacement option that is listed in Table 5 on page 62. 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		

DEFAULT: NODECK

CATEGORY: File Management



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 OS/390 C/C++ programs, use the OBJECT option. Table 22 details the relationship between the DECK and OBJECT options. For a description of OBJECT see “OBJECT | NOOBJECT” on page 125.

Table 22. Relationship between DECK and OBJECT

DECK   NODECK Option	OBJECT   NOBJECT 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
<b>Note:</b> The defaults are OBJECT and NODECK		

### 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓		✓		✓

DEFAULT: NOHWOPTS

CATEGORY: Preprocessor



**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 OS/390 C programs.

See “ARCHITECTURE” on page 73.

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.

### 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 73 for more information.

## SYSLIB

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: no action

CATEGORY: File Management

►►SYSL—(—*pdsnames-list*—)◄◄

**Note:** When compiling new OS/390 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 allocated, 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOSYSPATH

CATEGORY: File Management

►►SYS—(—*pathlist*—)◄◄  
NOSYS

**Note:** When compiling new OS/390 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, OS/390 C++ supports SYSLIB ddnames so that header files can be searched in exactly the same way they are in OS/390 C. The compiler uses SYSLIB to search for include file **after** it performs a search using the OS/390 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 246.

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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
✓	✓	✓		

DEFAULT: no action

CATEGORY: File Management



►►—USERL—(—*pdsnames-list*—)—————►►

**Note:** When compiling new OS/390 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

C	C++	Accepted by IPA Link	Special IPA Processing	
			IPA Compile	IPA Link
	✓			

DEFAULT: NOUSERPATH

CATEGORY: File Management

►►—USER—┐—————►►  
         └—(—*pathlist*—)——┘  
      NOUSER—┘

**Note:** When compiling new OS/390 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 an OS/390 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 246.

### 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 OS/390 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 193 for information about IPA Link step listings.

## Example of an OS/390 C Compiler Listing

Figure 15 shows an example of an OS/390 C compiler listing.

```

15647A01 V2 R6 M00 OS/390 C          'TSCTEST.OSV2R6M0.SCBCSAM(CBC3UAAM)'          05/25/1998 17:14:44 Page 1

          * * * * * P R O L O G * * * * *

Compile Time Library . . . . . : 22060000
Command options:
Program name. . . . . : 'TSCTEST.OSV2R6M0.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
                          : *TUNE(3) *ARCH(0) *SPILL(128) *MAXMEM(2097152)
                          : *TARGET(LE) *FLAG(1) *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)
                          : *FLOAT(HEX,FOLD,NOAFP) *STRICT
                          : *NOCSECT
                          : *NOEVENTS
                          : *OBJECT
                          : *NOGENPCH
                          : *NOUSEPCH
                          : *NOOPTFILE
                          : *NOSERVICE
                          : *NOOE
                          : *NOIPA
                          : *SEARCH(///'CEE.SCEEH.+')
                          : *NOLSEARCH
                          : *NOLOCALE *HALT(16) *PLIST(HOST)
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 * * * * *

15647A01 V2 R6 M00 OS/390 C          'TSCTEST.OSV2R6M0.SCBCSAM(CBC3UAAM)'          05/25/1998 17:14:44 Page 2

          * * * * * S O U R C E * * * * *

LINE  STMT          *...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8...+...9...+...*  SEQNBR INCNO
1      #include <stdio.h> 1
2
3      #include "cbc3uaan.h" 3
4
5      void convert(double); 5
6
7      int main(int argc, char **argv) 7
8      { 8
9          double c_temp; 9
10
11          if (argc == 1) { /* get Celsius value from stdin */ 10
12              int ch; 11
13
14              printf("Enter Celsius temperature: \n"); 12
15
16              if (scanf("%f", &c_temp) != 1) { 13
17                  printf("You must enter a valid temperature\n"); 14
18              } 15
19              else {
20                  convert(c_temp);
21              }
22          }
23          else { /* convert the command-line arguments to Fahrenheit */
24              int i;
25

```

Figure 15. Example of an OS/390 C listing (Part 1 of 6)

```

26      6      for (i = 1; i < argc; ++i) {
27      7          if (sscanf(argv[i], "%f", &c_temp) != 1)
28      8              printf("%s is not a valid temperature\n", argv[i]);
29      9          else
30      10             convert(c_temp);
31      11     }
32     }
33 }
34
35 void convert(double c_temp) {
36     double f_temp = (c_temp * CONV + OFFSET);
37     printf("%5.2f Celsius is %5.2f Fahrenheit\n", c_temp, f_temp);
38 }

```

\*\*\*\*\* E N D O F S O U R C E \* \* \* \* \*

15647A01 V2 R6 M00 OS/390 C 'TSCTEST.OSV2R6M0.SCBCSAM(CBC3UAAM)' 05/25/1998 17:14:44 Page 3

\*\*\*\*\* I N C L U D E S \* \* \* \* \*

INCLUDE FILES --- FILE# NAME

1	TSCTEST.CEE180.SCEEH.H(STDIO)
2	TSCTEST.CEE180.SCEEH.H(FEATURES)
3	TSCTEST.OSV2R6M0.SCBCSAM(CBC3UAAM)

\*\*\*\*\* E N D O F I N C L U D E S \* \* \* \* \*

15647A01 V2 R6 M00 OS/390 C 'TSCTEST.OSV2R6M0.SCBCSAM(CBC3UAAM)' 05/25/1998 17:14:44 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	1-1:116	Class = typedef, Length = 8 Type = array[2] of pointer to unsigned char 1-1:119, 1-1:273, 1-1:274, 1-1:275
__abend	1-1:529	Type = struct with no tag in union at offset 0
__alloc	1-1:539	Type = struct with no tag in union at offset 0
__amrc_ptr	1-1:561	Class = typedef, Length = 4 Type = pointer to struct __amrc_type
__amrc_type	1-1:557	Class = typedef, Length = 220 Type = struct __amrc_type 1-1:561
:		
:		
vprintf		Class = extern Type = function returning int 1-1:274
vsprintf		Class = extern Type = function returning int 1-1:275
FILE	1-1:73	Class = typedef, Length = 4 Type = struct _ffile 1-1:227, 1-1:228, 1-1:229, 1-1:230, 1-1:231, 1-1:232, 1-1:233, 1-1:234, 1-1:235, 1-1:237, 1-1:238, 1-1:239, 1-1:241, 1-1:242, 1-1:243, 1-1:244, 1-1:245, 1-1:246, 1-1:247, 1-1:249, 1-1:250, 1-1:255, 1-1:260, 1-1:262, 1-1:263, 1-1:270, 1-1:272, 1-1:273, 1-1:474, 1-1:475, 1-1:476, 1-1:478, 1-1:568

\*\*\*\*\* 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 15. Example of an OS/390 C listing (Part 2 of 6)

## \*\*\*\*\* STRUCTURE MAPS \*\*\*\*\*

Aggregate map for:		Total size: 8 bytes
.....		
__rrds_key_type		
=====		
Offset Bytes(Bits)	Length Bytes(Bits)	Member Name
=====		
0	4	__fill
4	4	__recnum
=====		

Aggregate map for:		Total size: 4 bytes
.....		
__code		
=====		
Offset Bytes(Bits)	Length Bytes(Bits)	Member Name
=====		
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
=====		

.  
 .  
 .

## \*\*\*\*\* END OF STRUCTURE MAPS \*\*\*\*\*

## \*\*\*\*\* MESSAGE SUMMARY \*\*\*\*\*

Total	Informational(00)	Warning(10)	Error(30)	Severe Error(40)
0	0	0	0	0

\*\*\*\*\* END OF MESSAGE SUMMARY \*\*\*\*\*

Figure 15. Example of an OS/390 C listing (Part 3 of 6)

## Inline Report (Summary)

Reason: P : #pragma noline 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	16	0	2/2	convert
A	T,N	E	114 (74)	2/2	0	main

Mode = AUTO Inlining Threshold = 100 Expansion Limit = 1000

## 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

Figure 15. Example of an OS/390 C listing (Part 4 of 6)

OFFSET	OBJECT	CODE	LINE#	FILE#	PSEUDO	ASSEMBLY	LISTING
Timestamp and Version Information							
000000	F1F9	F9F8				=C'1998'	Compiled Year
000004	F0F5	F2F5				=C'0525'	Compiled Date MMDD
000008	F1F7	F1F4	F4F4			=C'171444'	Compiled Time HHMMSS
00000E	F0F2	F0F6	F0F0			=C'020600'	Compiler Version
Timestamp and Version End							

OFFSET	OBJECT	CODE	LINE#	FILE#	PSEUDO	ASSEMBLY	LISTING
PPA1: Entry Point Constants							
000018	1CCE	A106				=F'483303686'	Flags
00001C	0000	02F0				=A(PPA2-main)	
000020	0000	0000				=F'0'	No PPA3
000024	0000	0000				=F'0'	No EPD
000028	FFFC	0000				=F'-262144'	Register save mask
00002C	0000	0000				=F'0'	Member flags
000030	90					=AL1(144)	Flags
000031	0000	00				=AL3(0)	Callee's DSA use/8
000034	0040					=H'64'	Flags
000036	0012					=H'18'	Offset/2 to CDL
000038	0000	0000				=F'0'	Reserved
00003C	5000	00E9				=F'1342177513'	CDL function length/2
000040	0000	0040				=F'64'	CDL function EP offset
000044	3826	0000				=F'942014464'	CDL prolog
000048	4009	00E0				=F'1074331872'	CDL epilog
00004C	0000	0000				=F'0'	CDL end
000050	0004	****				AL2(4),C'main'	
PPA1 End							
000058			00001		*	#include <stdio.h>	
00005C	01C3	C5C5	00002		*		
000060	0000	00D8	00003		*	#include "cbc3uaan.h"	
000064	FFFF	FFC0	00004		*		
000068	47F0	F001	00005		*	void convert(double);	
00006C	58F0	C31C	00006		*		
000070	184E		00007		*	int main(int argc, char **argv)	
000072	05EF		00007			main DS 0D	
000074	0000	0000	00007			B 34(,r15)	
000078	07F3		00007			CEE eyecatcher	
00007A	90EB	D00C	00007			DSA size	
:						=A(PPA1-main)	
:						B 1(,r15)	
:						L r15,796(,r12)	
:						LR r4,r14	
:						BALR r14,r15	
:						=F'0'	
:						BR r3	
:						STM r14,r11,12(r13)	
000338							
End of Literals							
*** General purpose registers used: 110110000011111							
*** 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							
000338	411CCCC	CCCCCCCC	42200000	00000000		.....	
PPA2: Compile Unit Block							
000348	0300	2202				=F'50340354'	Flags
00034C	FFFF	FCB8				=A(CEESTART-PPA2)	
000350	0000	0000				=F'0'	No PPA4
000354	FFFF	FCB8				=A(TIMESTAMP-PPA2)	
000358	0000	0000				=F'0'	No primary
00035C	0000	0000				=F'0'	Flags
PPA2 End							

Figure 15. Example of an OS/390 C listing (Part 5 of 6)

```

15647A01 V2 R6 M00 OS/390 C          'TSCTEST.OSV2R6M0.SCBCSAM(CBC3UAAM)'          05/25/1998 17:14:44   Page   38

      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
      MAIN          PC    1  000000 000360          CONVERT        PC    2  000000 000090
      LD            LD    0  000058 000001          LD            LD    0  000288 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
15647A01 V2 R6 M00 OS/390 C          'TSCTEST.OSV2R6M0.SCBCSAM(CBC3UAAM)'          05/25/1998 17:14:44   Page   39

      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
15647A01 V2 R6 M00 OS/390 C          'TSCTEST.OSV2R6M0.SCBCSAM(CBC3UAAM)'          05/25/1998 17:14:44   Page   40

      * * * * *   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   * * * * *
      * * * * *   E N D   O F   C O M P I L A T I O N   * * * * *

```

Figure 15. Example of an OS/390 C listing (Part 6 of 6)

## OS/390 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 193 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 *OS/390 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 90 , and “Appendix F. OS/390 C/C++ Compiler Return Codes and Messages” on page 475.

## 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 function.

The summary contains information such as:

- Name of each defined function. Function names appear in alphabetical order.
- Reason for action on a function:
  - A #pragma noline was specified for the function.
  - A #pragma inline was specified for the function.
  - Auto-inlining acted on the function.
  - There was no reason to inline the function.
- Action on a function:
  - Function was inlined at least once.
  - Function was not inlined because of initial size constraints.
  - Function was not inlined because of expansion beyond size constraint.
  - Function was a candidate for inlining, but was not inlined.
  - Function was a candidate for inlining, but was not referenced.
  - The function is directly recursive, or some calls have mismatching parameters.
- Status of original function after inlining:
  - Function is discarded because it is no longer referenced and is defined as static internal.
  - Function was not discarded for various reasons :
    - Function is external. (It can be called from outside the compilation unit.)
    - Some call to this function remains.
    - Function has its address taken.
- Initial relative size of function (in Abstract Code Units (ACU)).
- Final relative size of function (in ACUs) after inlining.
- Number of calls within the function and the number of these calls that were inlined into the function.
- Number of times the function is called by others in the compile unit and the number of times the function 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 function such as:

- Functions that it calls
- Functions that call it
- Functions 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 function with automatic storage increases the automatic storage of the function 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 of identifiers.

---

## **Using the OS/390 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 193 for information about IPA Link step listings.

## Example of an OS/390 C++ Compiler Listing

Figure 16 shows an example of a OS/390 C++ compiler listing. Vertical ellipses indicate sections that have been truncated.

```
5647A01 V2 R6 M00 OS/390 C++          'TS14576.CXX(CBC3UBRC)'          07/16/1998 12:39:09
                                     * * * * * P R O L O G * * * * *
Compiler options. . . . . : ANSIALIAS      ARGPARSE      NODIGRAPH      NOEVENTS      EXECOPS      EXH
                          : NOEXPMAC      NOEXPORTALL    NOFASTTEMPINC  NOGONUMBER     INLRPT      NOLIBANSI
                          : LIST          LONGNAME      NOMARGINS      MEMORY        OBJECT      NOOFFSET
                          : REDIR         NOSEQUENCE    NOSHOWINC      NOSOM          SOMEINIT    NOSOMGS
                          : NOSOMVOLATTR  SOURCE        NOSRCMSG      START         TERMINAL    NOWSIZEOF
                          : XREF          NOATTRIBUTE
                          : ARCH(0)      FLAG(I)       HALT(16)      MAXMEM(2097152) NESTINC(255) OPTIMIZE(1)
                          : PLIST(HOST)  SPILL(128)    TARGET(LE)    TUNE(3)
                          : NOCSECT
                          : DLL(NOCALLBACKANY)
                          : FLOAT(HEX,FOLD,NOAFP) STRICT
                          : NOINFO
                          : NOIPA
                          : NOGENPCH
                          : LONGLVL(EXTENDED) NOTEST(HOOK)
                          : NOLOCALE
                          : NOOE
                          : NOPORT
                          : NOPONLY
                          : NOSERVICE
                          : TEMPINC
                          : OPTFILE(DD:OPTS)
                          : SEARCH(///'CEE.SCEEH.+' ,///'CBC.SCLBH.+')
                          : NOUSEPCH

5647A01 V2 R6 M00 OS/390 C++          'TS14576.CXX(CBC3UBRC)'          07/16/1998 12:39:09
                                     * * * * * 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  |
7  | #include <iostream.h>
8  | #include <iomanip.h>
9  | #include <stdio.h>
10 | #include <math.h>
11 | #include <time.h>
12 |
13 | #include "cbc3ubrh.h"           // Biorhythn Class and Date Class
14 |
15 | int main(void) {
16 |     BioRhythm bio;
17 |     int code;
18 |
19 |     if (!bio.ok()) {
20 |         cerr << "Error in birthdate specification - format is yyyy/mm/dd";
21 |         code = 8;
22 |     }
23 |     else {
24 |         cout << bio; // write out birthdate for bio
25 |         code = 0;
26 |     }
27 |     return(code);
28 | }
29 |
30 | static ostream& operator<<(ostream& os, BioRhythm& bio) {
31 |     os << "Total Days : " << bio.AgeInDays() << "\n";
32 |     os << "Physical : " << bio.Physical() << "\n";
33 |     os << "Emotional : " << bio.Emotional() << "\n";
34 |     os << "Intellectual: " << bio.Intellectual() << "\n";
35 |
36 |     return(os);
37 | }
```

Figure 16. Example of an OS/390 C++ Compiler Listing (Part 1 of 7)

```

38 |
39 | Date::Date() {
40 |     time_t lTime;
41 |     struct tm *newTime;
42 |
43 |     time(&lTime);
44 |     newTime = gmtime(&lTime);
45 |
46 |     curYear = newTime->tm_year + 1900;
47 |     curDay = newTime->tm_yday + 1;
48 | }
49 |
50 | BirthDate::BirthDate(const char *birthText) {
51 |     strcpy(text, birthText);
52 | }
53 |
54 | BirthDate::BirthDate() {
55 |     cout << "Please enter your birthdate in the form yyyy/mm/dd\n";
56 |     cin >> setw(dateLen+1) >> text;
57 | }
58 |
59 | Date::DaysSince(const char *text) {
60 |
61 |     int year, month, day, totDays, delim;
62 |     int daysInYear = 0;
63 |
64 |     int rc = sscanf(text, "%4d%c%2d%c%2d",
65 |                    &year, &delim, &month,
66 |                    &delim, &day);
67 |
68 |     --month;
69 |     if (rc != 5 || year < 0 || year > 9999 ||
70 |         month < 0 || month > 12 ||
71 |         day < 1 || day > 31 ||
72 |         day > numDays[month]) {
73 |         return(-1);
74 |     }
75 |     else {
76 |         for (int i=0; i<month; ++i) {
77 |             daysInYear += numDays[i];
78 |         }
79 |         daysInYear += day;
80 |     }
81 |
82 |     totDays = (curDay - daysInYear) + (curYear - year)*365 - 1;
83 |
84 |     // now, correct for leap year
85 |     if (((year % 4 == 0 && year % 100 != 0) ||
86 |         (year % 400 == 0)) && month <= 2) {
87 |         ++totDays;
88 |     }
89 |
90 |     for (int i=year+1; i < curYear; ++i) {
91 |         if ((i % 4 == 0 && i % 100 != 0) || i % 400 == 0) {
92 |             ++totDays;
93 |         }
94 |     }
95 |     return(totDays);
96 | }

```

\* \* \* \* \* E N D O F S O U R C E \* \* \* \* \*

Figure 16. Example of an OS/390 C++ Compiler Listing (Part 2 of 7)

```

5647A01 V2 R6 M00 OS/390 C++ 'TS14576.CXX(CBC3UBRC)' 07/16/1998 12:39:09
      * * * * * C R O S S   R E F E R E N C E   L I S T I N G   * * * * *
__valist :
   2:116 (D)   2:119 (R)   2:273 (R)   2:274 (R)   2:275 (R)
__amrc_type :
   2:553 (D)   2:557 (R)
__amrc2_type :
   2:566 (D)   2:570 (R)
__device_t :
   2:382 (D)   2:430 (R)
__fabs :
   11:74 (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 R6 M00 OS/390 C++ 'TS14576.CXX(CBC3UBRC)' 07/16/1998 12:39:09
      * * * * * I N C L U D E S   * * * * *
1 = TSCTEST.OSV2R6M0.SCLBH.H(IOSTREAM)
2 = TSCTEST.CEE190.SCEEH.H(STDIO)
3 = TSCTEST.CEE190.SCEEH.H(FEATURES)
4 = TSCTEST.CEE190.SCEEH.H(MEMORY)
5 = TSCTEST.CEE190.SCEEH.H(STRING)
6 = TSCTEST.CEE190.SCEEH.H(WCHAR)
7 = TSCTEST.CEE190.SCEEH.H(TIME)
8 = TSCTEST.OSV2R6M0.SCLBH.H(IOMANIP)
9 = TSCTEST.OSV2R6M0.SCLBH.H(IOSTREAM)
10 = TSCTEST.OSV2R6M0.SCLBH.H(GENERIC)
11 = TSCTEST.CEE190.SCEEH.H(MATH)
12 = TS14576.H(CBC3UBRH)
      * * * * * E N D   O F   I N C L U D E S   * * * * *

5647A01 V2 R6 M00 OS/390 C++ 'TS14576.CXX(CBC3UBRC)' 07/16/1998 12:39:09
      * * * * * 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 16. Example of an OS/390 C++ Compiler Listing (Part 3 of 7)

## Inline Report (Summary)

Reason: P : #pragma noline 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	26	0	2/2	Date::Date()
F	I	D	138 (17)	2/2	1/1	BioRhythm::BioRhythm()
A	I	E	99 (31)	2/2	1/1	BirthDate::BirthDate()
A	N	E	47 (16)	1/1	0	BirthDate::BirthDate(const char*)
F	N	A	37 (9)	1/1	0	BioRhythm::__dfdt()
F	I	D	21	0	2/2	BioRhythm:: BioRhythm()
A	T	-	179 (62)	3/3	1/0	operator<<(ostream&,BioRhythm&)
F	I	E	30	0	1/1	operator>>(istream&,const smanip_int&)
P	-	-	215 (44)	3/2	0	main
F	I	D	17	0	3/3	BioRhythm::Cycle(int)
F	I	D	12	1/0	1/1	BirthDate::DaysOld()
A	L	-	216	0	1/0	Date::DaysSince(const char*)
F	I	D	34 (10)	1/1	1/1	BioRhythm::Emotional()
F	I	D	34 (10)	1/1	1/1	BioRhythm::Intellectual()
F	I	D	34 (10)	1/1	1/1	BioRhythm::Physical()

Mode = AUTO Inlining Threshold = 100 Expansion Limit = 2000

## 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(2,2) : BirthDate::BirthDate()(1,1) 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(1,1) : BioRhythm::BioRhythm()(1,1)

Defined Function : BirthDate::BirthDate(const char\*)  
 Calls To(1,1) : Date::Date()(1,1)  
 Called From : 0  
 .  
 .  
 .

## Inline Report (Additional Information)

INFORMATIONAL CBC5052: Function specified is (or grows) too large to be inlined: operator<<(ostream&,BioRhythm&

INFORMATIONAL CBC5052: Function specified is (or grows) too large to be inlined: Date::DaysSince(const char\*)

Figure 16. Example of an OS/390 C++ Compiler Listing (Part 4 of 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  F1F9  F9F8                      =C'1998'           Compiled Year
000004  F0F7  F1F6                      =C'0716'           Compiled Date MMDD
000008  F1F2  F3F9  F0F9                      =C'123909'         Compiled Time HHMMSS
00000E  F0F2  F0F6  F0F0                      =C'020600'         Compiler Version

Timestamp and Version End

5647A01 V2 R6 M00 OS/390 C++ 'TS14576.CXX(CBC3UBRC)': operator>>(is...) 07/16/1998 12:39:09
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
000018  1CCE  A109                      =F'483303689'      Flags
00001C  0000  0F88                      =A(PPA2-operator>>(istream&,const smanip_int&))
000020  0000  0000                      =F'0'              No PPA3
000024  0000  0000                      =F'0'              No EPD
000028  FF00  0000                      =F'-16777216'      Register save mask
00002C  0000  0001                      =F'1'              Member flags
000030  E0                      =AL1(224)          Flags
000031  0000  01                      =AL3(1)            Callee's DSA use/8
000034  0040                      =H'64'             Flags
000036  0012                      =H'18'             Offset/2 to CDL
000038  0000  0000                      =F'0'              Offset of state variable
00003C  5000  0041                      =F'1342177345'     CDL function length/2
000040  0000  0060                      =F'96'             CDL function EP offset
000044  1826  0000                      =F'405143552'      CDL prolog
000048  200A  0037                      =F'537526327'      CDL epilog
00004C  0000  0000                      =F'0'              CDL end
000050  0026  ****                      AL2(38),C'operator>>(istream&,const smanip_int&)'

PPA1 End

operator>>(istream&,const smanip_int
&)
000078                                00138  |  8      DS      0D
000078  47F0  F001                                00138  |  8      B      1(,r15)
00007C  01C3  C5C5                                CEE eyecatcher
000080  0000  00C8                                DSA size
000084  FFFF  FFA0                      =A(PPA1-operator>>(istream&,const smanip_int&))
000088  90E5  D00C                                00138  |  8      STM    r14,r5,12(r13)
00008C  58E0  D04C                                00138  |  8      L      r14,76(,r13)
000090  4100  E0C8                                00138  |  8      LA      r0,200(,r14)
000094  5500  C314                                00138  |  8      CL      r0,788(,r12)
000098  4140  F04C                                00138  |  8      LA      r4,76(,r15)
00009C  47D0  F03A                                00138  |  8      BNH     58(,r15)
0000A0  58F0  C31C                                00138  |  8      L      r15,796(,r12)
0000A4  184E                                00138  |  8      LR      r4,r14
0000A6  05EF                                00138  |  8      BALR   r14,r15
0000A8  0000  0008                      =F'8'
0000AC  0540                                00138  |  8      BALR   r4,r0
0000AE  4140  4016                                00138  |  8      LA      r4,22(,r4)
0000B2  5000  E04C                                00138  |  8      ST      r0,76(,r14)
0000B6  9210  E000                                00138  |  8      MVI     0(r14),16
0000BA  50D0  E004                                00138  |  8      ST      r13,4(,r14)
0000BE  5800  D014                                00138  |  8      L      r0,20(,r13)
0000C2  18DE                                00138  |  8      LR      r13,r14
0000C4                                End of Prolog

.
.
.

*** General purpose registers used: 1111110000001111
*** Floating point registers used: 1010101000000000
*** Size of register spill area: 128(max) 0(used)
*** Size of dynamic storage: 200
*** Size of executable code: 130

.
.
.

```

Figure 16. Example of an OS/390 C++ Compiler Listing (Part 5 of 7)



## EXTERNAL SYMBOL DICTIONARY

TYPE	ID	ADDR	LENGTH	NAME
SD	1	000000	001018	@STATICP
PR	2	000000	000058	@STATIC
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	@@DLI
LD	0	000078	000001	__rs__FR7istreamRC10smanip_int
LD	0	000448	000001	main
LD	0	000640	000001	__ct__4DateFv
LD	0	000758	000001	DaysSince__4DateFPCc
LD	0	0009A0	000001	__ct__9BirthDateFv
LD	0	000B40	000001	__ct__9BirthDateFPCc
ER	10	000000		CEESG003
ER	11	000000		CBCSG003
ER	12	000000		DaysSince__4DateFPCc
ER	13	000000		@@TRT
UR	14	000648		@STATICP
UR	15	000000		sscanf
UR	16	000000		__ls__7ostreamFd
UR	17	000000		CEETDSIN
UR	18	000000		fmod
UR	19	000000		__ls__7ostreamFi
UR	20	000000		cerr
ER	21	000000		@@TRGLOR
UR	22	000000		__dl__FPv
UR	23	000000		cin
UR	24	000000		cout
UR	25	000000		gmtime
UR	26	000000		__rs__7istreamFPC
UR	27	000000		setw__Fi
UR	28	000000		__ls__7ostreamFPCc
UR	29	000000		time
ER	30	000000		CEESTART
SD	31	000000	000008	@@PPA2
SD	32	000000	00000C	CEEMAIN
ER	33	000000		EDCINPL
ER	34	000000		main

Figure 16. Example of an OS/390 C++ Compiler Listing (Part 6 of 7)

## EXTERNAL SYMBOL CROSS REFERENCE

ORIGINAL NAME	EXTERNAL SYMBOL NAME
@STATICP	@STATICP
@@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
BCSG003	BCSG003
@@TRT	@@TRT
sscanf	sscanf
ostream::operator<<(double)	__ls__7ostreamFd
__sin	CEETDSIN
fmod	fmod
ostream::operator<<(int)	__ls__7ostreamFi
cerr	cerr
@@TRGLOR	@@TRGLOR
operator delete(void*)	__dl__FPv
cin	cin
cout	cout
gmtime	gmtime
istream::operator>>(char*)	__rs__7istreamFPC
setw(int)	setw__Fi
ostream::operator<<(const char*)	__ls__7ostreamFPCc
time	time
CEESTART	CEESTART
@@PPA2	@@PPA2
CEEMAIN	CEEMAIN
EDCINPL	EDCINPL

\*\*\*\*\* END OF COMPILATION \*\*\*\*\*

Figure 16. Example of an OS/390 C++ Compiler Listing (Part 7 of 7)

## OS/390 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 193 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 *OS/390 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 F. OS/390 C/C++ Compiler Return Codes and Messages” on page 475.

## 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 function.

The summary contains information such as:

- Name of each defined function. Function names appear in alphabetical order.
- Reason for action on a function:
  - A `#pragma noline` was specified for that function. The P indicates that inlining could not be performed.
  - A `#pragma inline` was specified for that function. The F indicates that the function was declared inline.
  - Auto-inlining acted on that function.
  - There was no reason to inline the function.
- Action on a function:
  - Function was inlined at least once.
  - Function was not inlined because of initial size constraints.
  - Function was not inlined because of expansion beyond size constraint.
  - Function was a candidate for inlining, but was not inlined.
  - Function was a candidate for inlining, but was not referenced.
  - This function is directly recursive, or some calls have mismatching parameters.
- Status of original function after inlining:
  - Function is discarded because it is no longer referenced and is defined as static internal.
  - Function was not discarded for various reasons :
    - Function is external. (It can be called from outside the compilation unit.)
    - Some call to this function remains.
    - Function has its address taken.
- Initial relative size of function (in Abstract Code Units (ACU)).
- Final relative size of function (in ACUs) after inlining.
- Number of calls within the function and the number of these calls that were inlined into the function.
- Number of times the function is called by others in the compile unit and the number of times this function 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 function such as:

- What functions it calls
- What functions call it
- In which functions 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 function with automatic storage would increase the automatic storage of the function 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 18. Filter Utility” on page 365.

---

## 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 17 on page 194 shows an example of an IPA Link step listing.

```

15647A01 V2 R6 M00 OS/390 C/C++ IPA          'TSIPA.TEST.LINKCNTL(INCLCNTL)'          06/22/1998 15:13:03 Page 1

      * * * * * P R O L O G * * * * *

Compile Time Library . . . . . : 22060000
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
                           : *FLAG(I)          *NOTEST(NOSYM,NOBLOCK,NOLINE,NOPATH,HOO)      *OPTIMIZE(1)
                           : *INLINE(AUTO,REPORT,1000,8000)
                           : *OBJECT          *OPTFILE(DD:OPTION)      *NOSERVICE  *NOOE       *NOLOCALE

*HALT(16)
      : *IPADBG(TRACETPO)

      * * * * * E N D   O F   P R O L O G * * * * *

15647A01 V2 R6 M00 OS/390 C/C++ IPA          'TSIPA.TEST.LINKCNTL(INCLCNTL)'          06/22/1998 15:13:03 Page 2

      * * * * * O B J E C T   F I L E   M A P * * * * *

*ORIGIN  IPA  FILE ID  FILE NAME
P        Y        1  TSIPA.TEST.LINKCNTL(INCLCNTL)
PI       Y        2  TSIPA.TEST.PASS1.OBJ(INCLMAIN)
PI       Y        3  TSIPA.TEST.PASS1.OBJ(INCLRTN1)
x 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 R6 M00 OS/390 C/C++ IPA          'TSIPA.TEST.LINKCNTL(INCLCNTL)'          06/22/1998 15:13:03 Page 3

      * * * * * C O M P I L E R   O P T I O N S   M A P * * * * *

SOURCE FILE ID  COMPILER OPTIONS
1
  *NOALIAS      *ANSIALIAS  *ARCH(0)      *ARGPARSE      *NODLL(NOCALLBACKANY)  *ENV(MVS)  *EXECOPS
  *FLOAT(HEX,FOLD,NOAFP)
  *NOGONUMBER   *IPA(NOLINK,NOOBJECT,NOCOMPRESS)      *NOLOCALE      *LONGNAME      *NOLIBANSI  *NOLIST
  *MAXMEM(2097152)
  *OPTIMIZE(2) *PLIST(HOST) *REDIR          *NORENT        *NOSTART      *SPILL(128) *STRICT      *NOTEST
  *TUNE(3)
  *XREF

2
  *NOALIAS      *ANSIALIAS  *ARCH(0)      *ARGPARSE      *NODLL(NOCALLBACKANY)  *ENV(MVS)  *EXECOPS
  *FLOAT(HEX,FOLD,NOAFP)
  *NOGONUMBER   *IPA(NOLINK,NOOBJECT,NOCOMPRESS)      *NOLOCALE      *LONGNAME      *NOLIBANSI  *NOLIST
  *MAXMEM(2097152)
  *OPTIMIZE(2) *PLIST(HOST) *REDIR          *NORENT        *NOSTART      *SPILL(128) *STRICT      *NOTEST
  *TUNE(3)
  *XREF

3
  *NOALIAS      *ANSIALIAS  *ARCH(0)      *ARGPARSE      *NODLL(NOCALLBACKANY)  *ENV(MVS)  *EXECOPS
  *FLOAT(HEX,FOLD,NOAFP)
  *NOGONUMBER   *IPA(NOLINK,NOOBJECT,NOCOMPRESS)      *NOLOCALE      *LONGNAME      *NOLIBANSI  *NOLIST
  *MAXMEM(2097152)
  *OPTIMIZE(2) *PLIST(HOST) *REDIR          *NORENT        *NOSTART      *SPILL(128) *STRICT      *NOTEST
  *TUNE(3)
  *XREF

      * * * * * E N D   O F   C O M P I L E R   O P T I O N S   M A P * * * * *

```

Figure 17. Example of an IPA Link Step Listing (Part 1 of 7)

## \*\*\*\*\* I N L I N E R E P O R T \*\*\*\*\*

## IPA Inline Report (Summary)

Reason: P : #pragma noline 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	-	76 (44)	2/2	0	main
A	I	D	0 (24)	0	1/1	Incl_Rtn1
A	I	D	0 (8)	0	1/1	Incl_Rtn2

Mode = AUTO      Inlining Threshold = 1000      Expansion Limit = 8000

## IPA Inline Report (Call Structure)

Defined Subprogram : main  
 Calls To(2,2) : Incl\_Rtn2(1,1)  
                   Incl\_Rtn1(1,1)  
 Called From : 0

Defined Subprogram : Incl\_Rtn2  
 Calls To : 0  
 Called From(1,1) : main(1,1)

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 17. Example of an IPA Link Step Listing (Part 2 of 7)

## \*\*\*\*\* PARTITION MAP \*\*\*\*\*

## PARTITION 0

## PARTITION CSECT NAMES:

Code: none  
 Static: none  
 Test: none

## PARTITION DESCRIPTION:

Initialization data partition

## COMPILER OPTIONS FOR PARTITION 0:

*NOALIAS	*ARCH(0)	*ARGPARSE	*NOCSECT	*NODLL	*ENV(MVS)	*EXECOPS	*FLOAT(HEX,FOLD,NOAFP)	*NOGONUMBER
*IPA(LINK)								
*NOLIBANSI	*NOLOCALE	*LONGNAME	*MAXMEM(2097152)		*OPTIMIZE(1)	*PLIST(HOST)	*REDIR	*NORENT
*START								*SPILL(128)
*STRICT	*NOTEST	*TUNE(3)						

## SYMBOLS IN PARTITION 0:

*TYPE	FILE ID	SYMBOL
D	1	gb1

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

\*\*\*\*\* END OF PARTITION MAP \*\*\*\*\*

Figure 17. Example of an IPA Link Step Listing (Part 3 of 7)



```

15647A01 V2 R6 M00 OS/390 C/C++ IPA Partition 0 06/22/1998 15:13:03 Page 7
OFFSET OBJECT CODE LINE# FILE# PSEUDO ASSEMBLY LISTING
15647A01 V2 R6 M00 OS/390 C/C++ IPA Partition 0 06/22/1998 15:13:03 Page 8
EXTERNAL SYMBOL DICTIONARY
TYPE ID ADDR LENGTH NAME
SD 1 000000 000000 @STATICP
SD 2 000000 000004 gbl
15647A01 V2 R6 M00 OS/390 C/C++ IPA Partition 0 06/22/1998 15:13:03 Page 9
EXTERNAL SYMBOL CROSS REFERENCE
ORIGINAL NAME EXTERNAL SYMBOL NAME
@STATICP @STATICP
gbl gbl
15647A01 V2 R6 M00 OS/390 C/C++ IPA Partition 0 06/22/1998 15:13:03 Page 10
***** STORAGE OFFSET LISTING *****
IDENTIFIER DEFINITION ATTRIBUTES
<SEQNBR>--<FILE NO>:<FILE LINE NO>
gbl 5-1:5 Class = external definition, Location = CSECT GBL, Length = 4
***** END OF STORAGE OFFSET LISTING *****
15647A01 V2 R6 M00 OS/390 C/C++ IPA Partition 1 06/22/1998 15:13:03 Page 11
***** PARTITION MAP *****
PARTITION 1 OF 1
PARTITION SIZE:
Actual: 1116
Limit: 102400
PARTITION CSECT NAMES:
Code: none
Static: none
Test: none
PARTITION DESCRIPTION:
Primary partition
COMPILER OPTIONS FOR PARTITION 1:
*NOALIAS *ARCH(0) *ARGPARSE *NOCSECT *NODLL *ENV(MVS) *EXECOPS *FLOAT(HEX,FOLD,NOAFP) *NOGONUMBER
*IPA(LINK) *NOLOCALE *LONGNAME *MAXMEM(2097152) *OPTIMIZE(1) *PLIST(HOST) *REDIR *NORENT *SPILL(128)
*START
*STRICT *NOTEST *TUNE(3)
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
***** END OF PARTITION MAP *****

```

Figure 17. Example of an IPA Link Step Listing (Part 4 of 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	F1F9	F9F8				=C'1998'	Compiled Year
000004	F0F6	F2F2				=C'0622'	Compiled Date MMDD
000008	F1F5	F0F4	F5F4			=C'150454'	Compiled Time HHMMSS
00000E	F0F2	F0F6	F0F0			=C'020600'	Compiler Version
Timestamp and Version End							

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							
000018	1CCE	A106				=F'483303686'	Flags
00001C	0000	00D0				=A(PPA2-main)	
000020	0000	0000				=F'0'	No PPA3
000024	0000	0000				=F'0'	No EPD
000028	FF00	0000				=F'-16777216'	Register save mask
00002C	0000	0000				=F'0'	Member flags
000030	90					=AL1(144)	Flags
000031	0000	00				=AL3(0)	Callee's DSA use/8
000034	02C0					=H'704'	Flags
000036	0012					=H'18'	Offset/2 to CDL
000038	0000	0000				=F'0'	Reserved
00003C	5000	0061				=F'1342177377'	CDL function length/2
000040	0000	0040				=F'64'	CDL function EP offset
000044	3824	0000				=F'941883392'	CDL prolog
000048	4009	0058				=F'1074331736'	CDL epilog
00004C	0000	0000				=F'0'	CDL end
000050	0004	****				AL2(4),C'main'	
PPA1 End							
000058			00010	1	main	DS	0D
000058	47F0	F022	00010	1		B	34(,r15)
00005C	01C3	C5C5					CEE eyecatcher
000060	0000	00C0					DSA size
000064	FFFF	FFC0					=A(PPA1-main)
000068	47F0	F001	00010	1		B	1(,r15)
00006C	58F0	C31C	00010	1		L	r15,796(,r12)
000070	184E		00010	1		LR	r4,r14
000072	05EF		00010	1		BALR	r14,r15
000074	0000	0000					=F'0'
000078	07F3		00010	1		BR	r3
00007A	90E5	D00C	00010	1		STM	r14,r5,12(r13)
00007E	58E0	D04C	00010	1		L	r14,76(,r13)
000082	4100	E0C0	00010	1		LA	r0,192(,r14)
000086	5500	C314	00010	1		CL	r0,788(,r12)
00008A	4130	F03A	00010	1		LA	r3,58(,r15)
00008E	4720	F014	00010	1		BH	20(,r15)
000092	5000	E04C	00010	1		ST	r0,76(,r14)
000096	9210	E000	00010	1		MVI	0(r14),16
00009A	50D0	E004	00010	1		ST	r13,4(,r14)
00009E	18DE		00010	1		LR	r13,r14
0000A0			End of Prolog				
0000A0			00014	1	@1L2	DS	0H
0000A0	4150	0001	00016	1		LA	r5,1
0000A4	5050	D098	00016	1		ST	r5,@PARM.i2(,r13,152)
0000A8	5810	308E	00010	2	+	L	r1,=A(gbl)(,r3,142)
0000AC	5800	1000	00010	2	+	L	r0,gbl(,r1,0)
0000B0	1C40		00010	2	+	MR	r4,r0
0000B2	1805		00010	2	+	LR	r0,r5
0000B4	5000	1000	00010	2	+	ST	r0,gbl(,r1,0)
0000B8	5800	D098	00012	2	+	L	r0,@PARM.i2(,r13,152)
0000BC	8900	0001	00012	2	+	SLL	r0,1
0000C0	5000	D09C	00012	2	+	ST	r0,@IRET1(,r13,156)
0000C4	5000	D0A0	00012	2	+	ST	r0,k(,r13,160)
0000C8	4100	0000	00018	1		LA	r0,0

Figure 17. Example of an IPA Link Step Listing (Part 5 of 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
0000CC	5000	D0A4	00018	1	ST	r0,@CIV0(,r13,164)	
0000D0			00018	1	DS	0H	
0000D0	5800	D0AC	00019	1	L	r0,j(,r13,172)	
0000D4	5000	D0A8	00019	1	ST	r0,@PARM.i0(,r13,168)	
0000D8	5800	D0A0	00019	1	L	r0,k(,r13,160)	
0000DC	5000	D0B0	00019	1	ST	r0,@PARM.j1(,r13,176)	
0000E0	5850	D0A8	00008	3 +	L	r5,@PARM.i0(,r13,168)	
0000E4	1C40		00008	3 +	MR	r4,r0	
0000E6	1805		00008	3 +	LR	r0,r5	
0000E8	5000	D0B4	00008	3 +	ST	r0,@IRET0(,r13,180)	
0000EC	5000	D0A0	00008	3 +	ST	r0,k(,r13,160)	
0000F0	5800	D0A4	00008	3 +	L	r0,@CIV0(,r13,164)	
0000F4	4A00	3088	00008	3 +	AH	r0='H'1'	
0000F8	5000	D0A4	00008	3 +	ST	r0,@CIV0(,r13,164)	
0000FC	5500	308A	00008	3 +	CL	r0='F'10'	
000100	4740	303E	00008	3 +	BL	@1L4	
000104	58F0	D0A0	00022	1	L	r15,k(,r13,160)	
000108			00022	1	DS	0H	
Start of Epilog							
000108	180D		00022	1	LR	r0,r13	
00010A	58D0	D004	00022	1	L	r13,4(,r13)	
00010E	58E0	D00C	00022	1	L	r14,12(,r13)	
000112	9825	D01C	00022	1	LM	r2,r5,28(r13)	
000116	051E		00022	1	BALR	r1,r14	
000118	0707		00022	1	NOPR	7	
Start of Literals							
00011A	0001					=H'1'	
00011C	0000	000A				=F'10'	
000120	0000	0000				=A(gbl)	
000124							
End of Literals							
*** General purpose registers used: 1101110000001111							
*** Floating point registers used: 0000000000000000							
*** Size of register spill area: 128(max) 0(used)							
*** Size of dynamic storage: 192							
*** Size of executable code: 194							
000124	0000	0000					
PPA2: Compile Unit Block							
000128	0300	2202				=F'50340354'	Flags
00012C	FFFF	FED8				=A(CEESTART-PPA2)	
000130	0000	0000				=F'0'	No PPA4
000134	FFFF	FED8				=A(TIMESTMP-PPA2)	
000138	0000	0000				=F'0'	No primary
00013C	0000	0000				=F'0'	Flags
PPA2 End							

Figure 17. Example of an IPA Link Step Listing (Part 6 of 7)

## EXTERNAL SYMBOL DICTIONARY

TYPE	ID	ADDR	LENGTH	NAME
SD	1	000000	000140	@STATICP
LD	0	000058	000001	main
ER	2	000000		CEESG003
ER	3	000000		gb1
ER	4	000000		CEESTART
SD	5	000000	000008	@PPA2
SD	6	000000	00000C	CEEMAIN
ER	7	000000		EDCINPL
ER	8	000000		main

## EXTERNAL SYMBOL CROSS REFERENCE

ORIGINAL NAME	EXTERNAL SYMBOL NAME
@STATICP	@STATICP
main	main
CEESG003	CEESG003
gb1	gb1
CEESTART	CEESTART
@PPA2	@PPA2
CEEMAIN	CEEMAIN
EDCINPL	EDCINPL

## \*\*\*\*\* STORAGE OFFSET LISTING \*\*\*\*\*

IDENTIFIER	DEFINITION	ATTRIBUTES <SEQNBR>-<FILE NO>:<FILE LINE NO>	
gb1	5-1:5	Class = external reference, Location = CSECT GBL,	Length = 4
k	12-1:12	Class = automatic, Location = 160(r13),	Length = 4
j	12-1:12	Class = automatic, Location = 172(r13),	Length = 4

## \*\*\*\*\* END OF STORAGE OFFSET LISTING \*\*\*\*\*

## \*\*\*\*\* SOURCE FILE MAP \*\*\*\*\*

*ORIGIN	OBJECT FILE ID	SOURCE FILE ID	SOURCE FILE NAME
P	2	1	TSIPA.TEST.C(INCLMAIN) - Compiled by 5647A01 V2 R6 M00 OS/390 C on 06/22/1998 15:04:54
P	3	2	TSIPA.TEST.C(INCLRTN1) - Compiled by 5647A01 V2 R6 M00 OS/390 C on 06/22/1998 15:05:05
P	4	3	TSIPA.TEST.C(INCLRTN2) - Compiled by 5647A01 V2 R6 M00 OS/390 C on 06/22/1998 15:05:13

ORIGIN: P=primary input PI=primary INCLUDE

## \*\*\*\*\* END OF SOURCE FILE MAP \*\*\*\*\*

## \*\*\*\*\* MESSAGE SUMMARY \*\*\*\*\*

TOTAL	UNRECOVERABLE (U)	SEVERE (S)	ERROR (E)	WARNING (W)	INFORMATIONAL (I)
0	0	0	0	0	0

## \*\*\*\*\* END OF MESSAGE SUMMARY \*\*\*\*\*

## \*\*\*\*\* END OF COMPILATION \*\*\*\*\*

Figure 17. 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 OS/390 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:
  - You specified `#pragma noline` for the subprogram.
  - You specified `#pragma inline` for the subprogram.
  - 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 96, “IPA | NOIPA” on page 103, and “LIST | NOLIST” on page 110 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. This section only displays variable information from C compilation units.



If you specify the XREF, IPA(XREF), or IPA(ATTRIBUTE) option along with the IPA(OBJECT) option for the IPA Compile step, and the compilation unit includes variables, the IPA Link step may generate a Storage Offset listing.

If you specify the 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#.

## 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 90, and “Appendix F. OS/390 C/C++ Compiler Return Codes and Messages” on page 475.

## 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 *DFSMS/MVS 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 OS/390 C/C++ supplied cataloged procedures, the CXXBIND REXX exec, and the c89, cc, and c++ utilities are indicated only where they differ from the binder default.

#### ALIASES(ALL | NO)

DEFAULT: 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. See "COMPAT(PM1 | PM2 | PM3 | CURRENT | CURR)" on page 208 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 315.

#### CALL(YES | NO)

DEFAULT: CALL(YES)

**Note:** If you use the -r option with c89, cc or c++, the binder uses the CALL(NO) option.

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 314).

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)

Binder DEFAULT: CASE(UPPER)

**Note:** The default that is provided by the OS/390 C/C++ cataloged procedures, CXXBIND, and the c89, cc, and c++ utilities is 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:** OS/390 C++ does not support the CASE(UPPER) option. Use CASE(MIXED) for C++ code.

## COMPAT(PM1 | PM2 | PM3 | CURRENT | CURR)

DEFAULT: COMPAT(CURRENT)

The COMPAT option specifies the compatibility level of the binder. If you do not specify it, the default is COMPAT(CURRENT), which is the current level of the binder. For OS/390 C/C++ code you cannot specify a compatibility level lower than PM3.

## DYNAM(DLL | NO)

Binder DEFAULT: DYNAM(NO)

**Note:** The default that is provided by the OS/390 C/C++ cataloged procedures, CXXBIND, and the c89, cc, and c++ utilities is DYNAM(DLL).

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)

DEFAULT: LET(4)

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)

DEFAULT: 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)

DEFAULT: MAP(NO)

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 than 100 characters.

## REUS(NONE | SERIAL | RENT)

Binder DEFAULT: REUS(NONE)

**Note:** The default that is provided by the OS/390 C/C++ cataloged procedures, CXXBIND, and the c89, cc, and c++ utilities is REUS(RENT).

If you use the -g option with c89, cc, or c++, the binder uses the option REUS(SERIAL).

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 333.

**SERIAL** specifies that a single user can share the code, but it is loaded into a modifiable area of storage. 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).

## UPCASE(YES | NO)

DEFAULT: 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 314 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)

DEFAULT: XREF(NO)

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.
<i>dll-name</i>	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.
<i>identifier</i>	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.

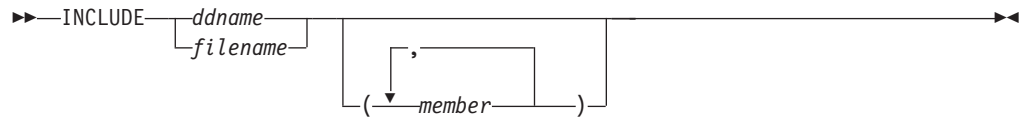
If your program exports symbols, the binder may also generate an output file of corresponding IMPORT control statements. See “Output IMPORT Statements” on page 317.

## 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:





*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:



*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:



*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:



*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 315. 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)

<i>member_name</i>	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 314.

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.

<i>new-name</i>	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 314.



---

## Chapter 8. Runtime Options

This chapter describes how to specify runtime options and `#pragma runopts` preprocessor directives available to you with OS/390 C/C++ and OS/390 Language Environment. For a detailed description of the OS/390 Language Environment runtime options and information about how to apply them in different environments, refer to the *OS/390 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 option of the EXEC PGM=*your-program-name* JCL statement
  - On the exported `_CEE_RUNOPTS` environment variable under the OS/390 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 OS/390 Language Environment, refer to the *OS/390 Language Environment Programming Reference*.

### Using the `#pragma runopts` Preprocessor Directive

You can use the `#pragma runopts` preprocessor directive to specify OS/390 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 OS/390 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 *OS/390 C/C++ Language Reference*.

---

## Part 3. Compiling, Binding, and Running OS/390 C/C++ Programs

This part describes how to compile, bind, and run an OS/390 C/C++ program using OS/390 Language Environment in the following sections:

- “Chapter 9. Compiling” on page 221
- “Chapter 10. Using Precompiled Headers” on page 259
- “Chapter 11. Using the IPA Link Step with OS/390 C/C++ Programs” on page 267
- “Chapter 12. Binding OS/390 C/C++ Programs” on page 289
- “Chapter 13. Binder Processing” on page 311
- “Chapter 14. Running an OS/390 C/C++ Application” on page 335





---

## Chapter 9. Compiling

This chapter describes how to compile your program with the OS/390 C/C++ compiler and OS/390 Language Environment. For specific information about compiler options see "Chapter 6. Compiler Options" on page 55.

The OS/390 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 OS/390 batch, TSO, or the OS/390 shell.

---

### 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 OS/390 C/C++ program across compilation units, and to perform optimizations that are not otherwise available with the OS/390 C/C++ compiler. You should refer to the *OS/390 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 18 shows the flow of processing for a regular compilation:

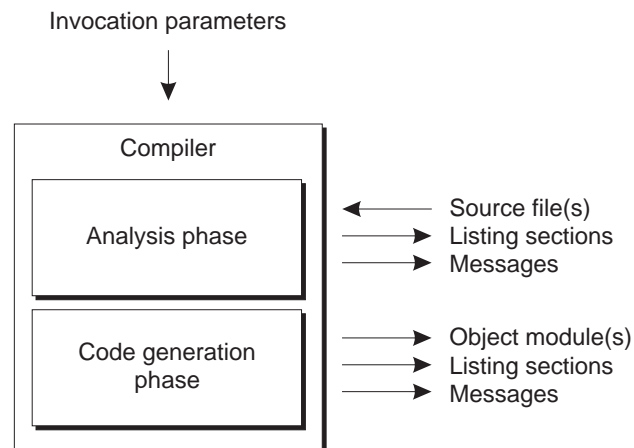


Figure 18. Flow of regular compiler processing

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 IPA-optimized or a combined IPA-optimized and conventional object. Figure 19 shows the flow of IPA Compile step processing:

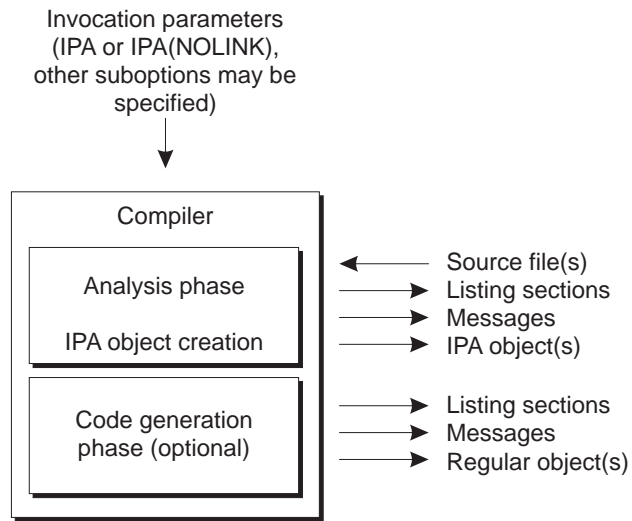


Figure 19. IPA Compile step processing

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.

Specify the IPA(LINK) compiler option to invoke the IPA Link step once for your program as a whole. Figure 20 on page 223 shows the flow of IPA Link step processing:

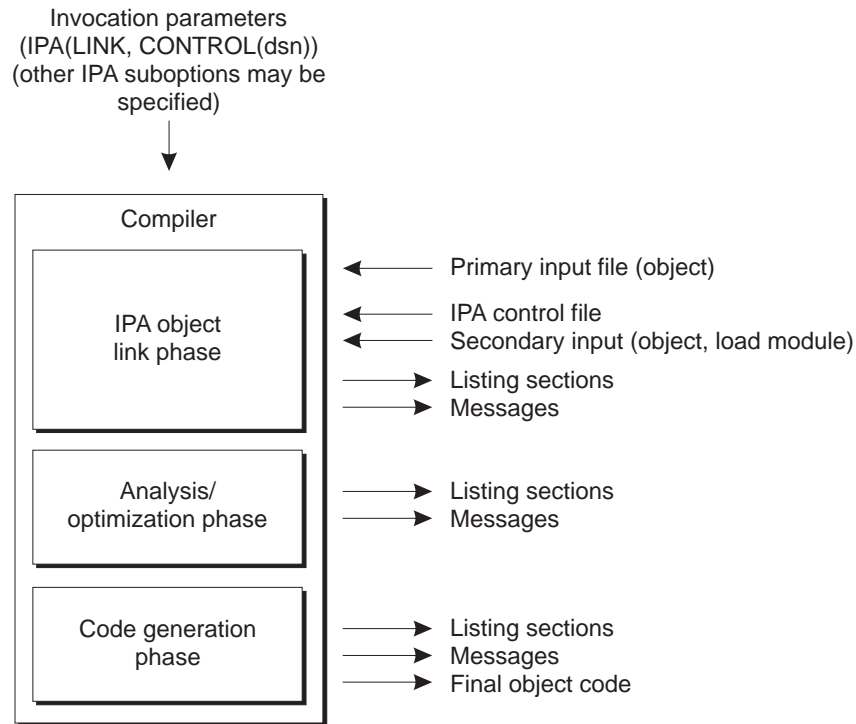


Figure 20. IPA Link step processing

Only c89, c++ and OS/390 batch (without the ISPF interface) support the IPA Link step. Refer to “Chapter 11. Using the IPA Link Step with OS/390 C/C++ Programs” on page 267 for information about the IPA Link step.

## Input to the OS/390 C/C++ Compiler

The following sections describe how to specify input to the OS/390 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 OS/390 C/C++ Programs” on page 267.

If you are compiling an OS/390 C++ program or an OS/390 C program, input for the compiler consists of the following:

- Your OS/390 C/C++ source program
- The OS/390 C/C++ standard header files including IBM-supplied Class Library header files
- Your header files

When you invoke the OS/390 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 LNKLIB concatenations. LPA can be from either

specific modules (IEALPAXx) or a list (LPALSTxx). See the cataloged procedures shipped with the product in “Appendix D. IBM Supplied Cataloged Procedures and REXX EXECs” on page 457.

**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 an OS/390 C++ or OS/390 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 OS/390 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 an absolute directory

## Secondary Input

For an OS/390 C++ or OS/390 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 OS/390 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 115 and “SEARCH | NOSEARCH” on page 140. For more information on naming `#include` files, see “Specifying Include File Names” on page 247. For information on how the compiler searches for `#include` files, see “Search Sequences for Include Files” on page 254. For more information on include files, refer to “Using Include Files” on page 246.

---

## Output from the Compiler

You can specify compiler output files 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

For valid combinations of input file types and output file types, refer to Table 25 on page 227.

## Specifying Output Files

You can use compile options to specify compilation output files as follows:

Table 23. 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 24 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 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 24. Default Suffixes for Output File Types

Output File Type.	OS/390 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 OS/390 data set if the source resides in a data set.
2. If you have specified the 0E option, see “OE | NOOE” on page 127 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

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 `SYSPRT` DD name, if you allocated it. Otherwise, the compiler generates a default file name as described in “`LIST | NOLIST`” on page 110.

## 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 125.

**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 `N0OBJECT` suboption of the IPA compiler option). You can use the object file that the `IPA(NOLINK,N0OBJECT)` compiler option creates as input to the IPA Link step only. It contains an external reference to `@@D0IPA`, 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,N0OBJECT)` option, the binder issues an error message.

Refer to “Valid Input/Output File Types” on page 227 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 `SYST10` DD statement. If you did not allocate `SYST10`, the compiler generates a default file name, as described in “`PPONLY | NOPPONLY`” on page 136.

## 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 156 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 25. 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
<b>Sequential Data Set, for example A.B</b>	<ol style="list-style-type: none"> <li>1. If file exists as a sequential data set, overwrites it</li> <li>2. If file does not exist, creates sequential data set</li> <li>3. Otherwise compilation fails</li> </ol>	<ol style="list-style-type: none"> <li>1. If PDS does not exist, creates the PDS and adds a member into the data set</li> <li>2. If PDS exists and member does not exist, adds member in the PDS</li> <li>3. If PDS and member both exist, then overwrites the member.</li> </ol>	<ol style="list-style-type: none"> <li>1. If the directory does not exist, compilation fails</li> <li>2. If the directory exists but the file does not exist, creates file</li> <li>3. If the file exists, overwrites the file.</li> </ol>	Not supported
<b>A member of a PDS using (member), for example A.B(C)</b>	<ol style="list-style-type: none"> <li>1. If the file exists as a sequential data set, overwrites it</li> <li>2. If the file exists as a PDS, creates or overwrites member</li> <li>3. If file does not exist, creates PDS and member</li> </ol>	<ol style="list-style-type: none"> <li>1. If PDS does not exist, creates PDS and member</li> <li>2. If PDS exists and member does not exist, adds member</li> <li>3. If PDS exists and member also exists, overwrites it</li> </ol>	<ol style="list-style-type: none"> <li>1. If directory does not exist, compilation fails</li> <li>2. If directory exists and the file with the specified filename does not exist, creates file</li> <li>3. If the directory exists and the file exists, overwrites file</li> </ol>	<ol style="list-style-type: none"> <li>1. If directory does not exist, compilation fails</li> <li>2. If directory exists and the file with the filename <i>MEMBER.ext</i> does not exist, creates file</li> <li>3. If directory exists and the file with the filename <i>MEMBER.ext</i> also exists, overwrite file</li> </ol>

Table 25. 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
<b>All members of a PDS, for example A.B</b>	<ol style="list-style-type: none"> <li>1. If file exists as a PDS, creates or overwrites members</li> <li>2. If file does not exist, creates PDS and members</li> <li>3. Otherwise compilation fails</li> </ol>	Not Supported	Not Supported	<ol style="list-style-type: none"> <li>1. If directory does not exist, compilation fails</li> <li>2. If directory exists and the files with the filenames <i>MEMBER.ext</i> do not exist, creates files</li> <li>3. If directory exists and the files with the filenames <i>MEMBER.ext</i> exist, overwrites them</li> </ol>
<b>HFS file, for example /a/b/d.c</b>	<ol style="list-style-type: none"> <li>1. If file exists as a sequential data set, overwrites it</li> <li>2. If file does not exist, creates sequential data set</li> <li>3. Otherwise compilation fails</li> </ol>	<ol style="list-style-type: none"> <li>1. If PDS does not exist, creates the PDS and stores a member into the data set</li> <li>2. If PDS exists and member does not exist, then add the member in the PDS</li> <li>3. If PDS and member both exist, then overwrites the member.</li> </ol>	<ol style="list-style-type: none"> <li>1. If the directory does not exist, compilation fails</li> <li>2. If the directory exists but the file does not exist, creates file</li> <li>3. If the file exists, overwrites the file.</li> </ol>	<ol style="list-style-type: none"> <li>1. If the directory does not exist, compilation fails</li> <li>2. If the directory exists and the file does not exist, creates it</li> <li>3. If the directory exists and the file exists, overwrites it</li> </ol>
<b>HFS Directory, for example a/b/</b>	Not supported	Not supported	Not supported	<ol style="list-style-type: none"> <li>1. If the directory does not exist, compilation fails</li> <li>2. If the directory exists and the files to be written do not exist, creates them</li> <li>3. If the directory exists and the files to be written already exist, overwrites them</li> </ol>

## Compiling Under OS/390 Batch

To compile your OS/390 C/C++ source program under OS/390 batch, you can either use cataloged procedures that IBM supplies, or write your own JCL statements.



## Using Cataloged Procedures for OS/390 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
EDCLIB	Compile and maintain an object library
EDCCL	Compile and link-edit
EDCCPLG	Compile, prelink, link-edit, and run
EDCCLG	Compile, link-edit, and run

### 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 OS/390 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
CBCBG	Bind and run
CBCI	Run the IPA Link step
CBCCCL	Compile, prelink, and link
CBCCCLG	Compile, prelink, link, and run

See “Appendix D. IBM Supplied Cataloged Procedures and REXX EXECs” on page 457 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 an OS/390 C program:

```
//jobname JOB acctno,name...
//COMPILE EXEC PGM=CBCDRVR,
// PARM='/SEARCH(''CEE.SCEEH.'') NOOPT SO OBJ OPTFILE(DD:CPATH)'
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN=CBC.SCBCCMP,DISP=SHR
//SYSLIN DD DSN=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 21. JCL for Compiling an OS/390 C Program (for NOOPT, SOURCE, and OBJ)*

The following example shows sample JCL for compiling an OS/390 C++ program:

```

//jobname JOB acctno,name...
//COMPILE EXEC PGM=CBCCDRVR,
// PARM='/CXX SEARCH(''CEE.SCEEH.''',''CBC.SCLBH.''),NOOPT,S0,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 22. JCL for Compiling an OS/390 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 an OS/390 C/C++ 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 *OS/390 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 an OS/390 C/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 *OS/390 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 *OS/390 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 DSNNAME 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 *OS/390 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 OS/390 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

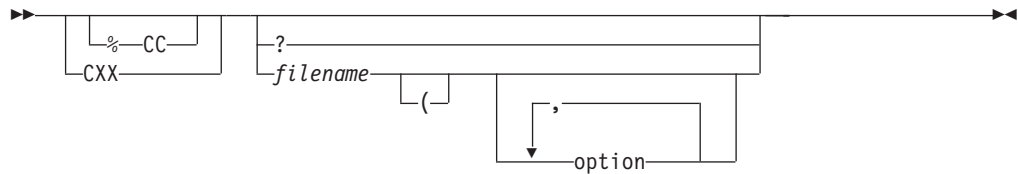
All methods of foreground execution call the CC or CXX REXX EXECs supplied by IBM.

**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 LNKLIST.
- 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 OS/390 C compiler, and the CXX REXX EXEC to invoke the OS/390 C++ compiler. These REXX EXECs share the same syntax:



where

**%** ensures that the REXX EXEC CC is invoked, not the OS/390 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 *OS/390 C/C++ Programming Guide*.

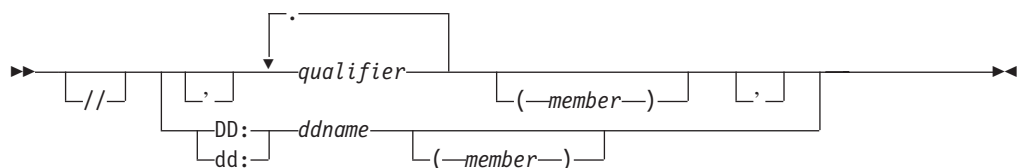
If you invoke either CC or CXX with no arguments or with only a single question mark, the appropriate preceding syntax diagram is displayed.

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

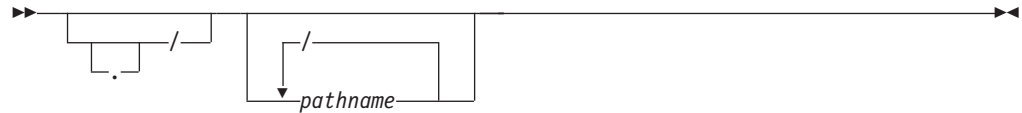
To specify a sequential or partitioned data set for your source file use the following syntax:



**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 59 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 OS/390 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 IBM OS/390 C/C++ option (19). The OS/390 C/C++ FOREGROUND UTILITIES panel is presented, as shown in Figure 23.

```

----- 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 23. Foreground IBM OS/390 C/C++ Utility Panel*

3. Select 1 to get the FOREGROUND OS/390 C COMPILE panel, or select 2 to get the FOREGROUND OS/390 C++ COMPILE panel as shown in Figure 24 on page 237.
4. Enter information such as your source data, password, object data set name, compiler options, and additional input libraries (as necessary).



```

----- FOREGROUND OS/390 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 24. Foreground IBM OS/390 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 59.

```

----- Foreground OS/390 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 OS/390 C++ panel]

```

Figure 25. Foreground IBM OS/390 C++ Compile Panel (2)

**Note for OS/390 C:** The only difference in the appearance of the panels for OS/390 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 225 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 225 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 136 for more information.
- 8** For OS/390 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 OS/390 C/C++ Compiler option to show the BATCH IBM OS/390 C/C++ COMPILE panels. These panels are similar to the FOREGROUND IBM OS/390 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 236 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 under the OS/390 Shell

An OS/390 UNIX C/C++ program 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.

Use the c89 utility to compile and bind a OS/390 UNIX System Services C application program from the OS/390 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 242.

You can also use the `cc exec` to compile a OS/390 UNIX System Services C application program from the OS/390 shell. For more information, see *OS/390 UNIX System Services Command Reference* .

Use the `c++` utility to compile and bind an OS/390 UNIX System Services C++ application program from the OS/390 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 *OS/390 UNIX System Services Command Reference*.

**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 OS/390 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 21. Archive and Make Utilities” on page 395 and *OS/390 UNIX System Services Programming Tools*.

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 OS/390 C/C++ source module is an HFS file, the object file is created in the working directory.
- If the OS/390 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 OS/390 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 OS/390 C/C++ source is located in an OS/390 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 OS/390 C application source to produce only object files. c89 recognizes that a file is an OS/390 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 OS/390 C source to create the default object file usersource.o in your working HFS directory, specify:

```
c89 -c usersource.c
```
  - To compile OS/390 C source to create an object file as a member in the PDS 'KENT.APPROG.OBJ', specify:

```
c89 -c "'/'kent.approg.c(usersrc)'"
```
- Compiling OS/390 C++ application source to produce only object files. c++ recognizes that a file is an OS/390 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 OS/390 C++ source to create the default object file usersource.o in your working HFS directory, specify:

```
c++ -c usersource.C
```
  - To compile OS/390 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 OS/390 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)'"
```

#### OS/390 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 OS/390 UNIX System Services 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 barry1.c > barry1.lst
c++ -V barry1.C > barry1.lst
```

### 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 the *OS/390 UNIX System Services Command Reference* 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 the make Utility

You can use the `make` utility to control the build of your OS/390 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 OS/390 C/C++ source files that are in data sets, you must use the `c89` utility directly.

See the *OS/390 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 *OS/390 UNIX System Services Programming Tools*.

## Using Feature Test Macros

Many of the symbols that are defined in headers are “protected” by a feature test macro. These “protected” symbols are invisible to the application unless the user defines the feature test macro with `#define`, using either of the following methods:

- In the source code before including any header files
- On the compilation command

Note that the `LANGVL` compiler option does not define or undefine these macros.

Table 26 summarizes the relationships between the feature test macros and the standards. ‘Yes’ indicates that a feature test macro makes visible the symbols that are related to a standard.

Table 26. Feature Test Macros and Standards

Feature Test Macro	POSIX .1	POSIX .1a	POSIX .2	POSIX .4a	XPG4 .2	XPG4 .2 Ext
<code>_POSIX_SOURCE</code>	Yes					
<code>_POSIX1_SOURCE 1</code>	Yes					
<code>_POSIX1_SOURCE 2</code>	Yes	Yes				
<code>_POSIX_C_SOURCE 1</code>	Yes					
<code>_POSIX_C_SOURCE 2</code>	Yes		Yes			
<code>_XOPEN_SOURCE</code>	Yes		Yes		Yes	
<code>_XOPEN_SOURCE _EXTENDED 1</code>	Yes		Yes		Yes	Yes
<code>_OPEN_SYS</code>	Yes	Yes	Yes	Yes		
<code>_OPEN_SYS_IPC _EXTENSIONS</code>	Yes		Yes		Yes	
<code>_OPEN_SYS_PTY _EXTENSIONS</code>	Yes	Yes	Yes		Yes	Yes
<code>_OPEN_THREADS</code>	Yes	Yes		Yes		
<code>_OPEN_SOURCE 1</code>	Yes	Yes	Yes	Yes		
<code>_OPEN_SOURCE 2 or _ALL_SOURCE</code>	Yes	Yes	Yes	Yes	Yes	Yes
<code>_OE_SOCKETS</code>	Yes	Yes				
<code>_OPEN_SYS SOCK_EXT</code>	Yes	Yes		Yes	Yes	
<code>_ALL_SOURCE_NO_THREADS</code>	Yes	Yes	Yes		Yes	Yes
<code>_OPEN_SOURCE 3</code>	Yes	Yes	Yes	Yes	Yes	Yes

The OS/390 C/C++ compiler supports the following feature test macros:

- `_POSIX_SOURCE`

When defined to any value with `#define`, it indicates that symbols that are required by POSIX.1 are made visible. Additional symbols can be made visible if POSIX.1 explicitly allows the symbol to appear in the header in question.

- `_POSIX1_SOURCE`

– When defined to 1, it has the same meaning as `_POSIX_SOURCE`.



- When defined to 2, both the POSIX.1a symbols and the POSIX.1 symbols are made visible. Additional symbols can be made visible if POSIX.1a explicitly allows the symbol to appear in the header in question.
- `_POSIX_C_SOURCE`
  - When defined to 1, it indicates that symbols required by POSIX.1 are made visible. Additional symbols can be made visible if POSIX.1 explicitly allows the symbol to appear in the header in question.
  - When defined to 2, both the POSIX.1 and POSIX.2 symbols are made visible. Additional symbols can be made visible if POSIX.2 explicitly allows the symbol to appear in the header in question.
- `_OPEN_SYS`

When defined to 1, this indicates that symbols required by POSIX.1, POSIX.1a, and POSIX.2 are made visible. Any symbols defined by the `_OPEN_THREAD` macro are also made visible. Additional symbols can be made visible if any of these standards explicitly allows the symbol to appear in the header in question or if the symbol is defined to be an extension.
- `_OPEN_THREADS`

When defined to 1, this indicates that symbols required by POSIX.1, POSIX.1a, and POSIX.4a are made visible.
- `_XOPEN_SOURCE`

Defines the functionality defined in the XPG/4 standard dated July 1992.
- `_XOPEN_SOURCE_EXTENDED`

When defined to 1, this defines the functionality defined in the XPG/4 standard plus the set of “Common APIs for UNIX-based Operating Systems”, April, 1994, draft.
- `_OPEN_SYS_IPC_EXTENSIONS`

Defines extensions to the X/Open InterProcess Communications functions. When `_OPEN_SYS_IPC_EXTENSIONS` is defined, the POSIX.1, POSIX.1a, and the XPG4 symbols are visible. This macro should be used in conjunction with `_XOPEN_SOURCE`.
- `_OPEN_SYS_PTY_EXTENSIONS`

Defines extensions to the X/Open Pseudo TTY functions. When `_OPEN_SYS_PTY_EXTENSIONS` is defined, the POSIX.1, POSIX.1a, XPG4, and XPG4.2 symbols are visible. This macro should be used in conjunction with `_XOPEN_SOURCE_EXTENDED` defined to 1.
- `_OPEN_SOURCE`

When defined to 1, this defines all of the functionality that was available on OS/390 OpenEdition in MVS 5.1. This macro is equivalent to specifying `_OPEN_SYS`.

When defined to 2, this defines all of the functionality that is available on OS/390 OpenEdition in MVS 5.2.2, including XPG4, XPG4.2, and all of the extensions.
- `_ALL_SOURCE`

Defines all of the functionality that is available on OS/390 UNIX System Services, including XPG4, XPG4.2, and all of the extensions. In addition, defining `_ALL_SOURCE` makes visible a number of symbols which are not permitted under ANSI, POSIX or XPG4, but which are provided as an aid to porting C-language applications to OS/390 UNIX System Services.
- `_OPEN_DEFAULT`

When defined to 0, and if no other feature test macro is defined, then all symbols will be visible. If in addition to `_OPEN_DEFAULT` only POSIX and/or XPG4 feature

test macros are defined, then only the symbols so requested will be visible. Otherwise, additional symbols (e.g., those visible when the `LANGVLV(EXTENDED)` compiler option is specified), may be exposed.

When defined to 1, this provides the base level of OS/390 UNIX System Services functionality, which includes POSIX.1, POSIX.1a, and POSIX.2.

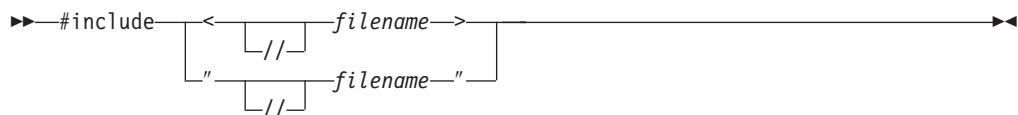
- `_OE_SOCKETS`  
Defines a BSD-like socket interface for the function prototypes and structures involved. This can be used with `_XOPEN_SOURCE_EXTENDED 1` and the XPG4.2 socket interfaces will be replaced with the BSD-like interfaces.
- `OPEN_MSGQ_EXT`  
Defines an interface which enables use of `select()`, `selectex()` and `poll()` to monitor message and file descriptors.
- `_MSE_PROTOS`  
The `_MSE_PROTOS` feature test macro does the following:
  1. Selects behavior for a multibyte extension support (MSE) function declared in `wchar.h` as specified by ISO/IEC 9899:1990/Amendment 1:1994 instead of behavior for the function as defined by CAE Specification, System Interfaces and Headers, Issue 4, July 1992 (XPG4)
  2. Exposes declaration of an MSE function declared in `wchar.h` which is specified by ISO/IEC 9899:1990/Amendment 1:1994 but not by XPG4.
- `_ALL_SOURCE_NO_THREADS`  
Provides the same function as `_ALL_SOURCE`, except it does not expose threading services (`_OPEN_THREADS`).
- `_VARARG_EXT_`  
Allows users of the `va_arg`, `va_end`, and `va_start` macros to define the `va_list` type differently.
- `_OPEN_SYS_SOCK_EXT`  
Defines the interface for function prototypes and structures for the extended sockets and bulk mode support.
- `_SHARE_EXT_VARS`  
Provides access to an application's POSIX and XPG4 external variables from a dynamically loaded module such as a DLL.

---

## 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.

*OS/390 C/C++ Language Reference* describes the `#include` directive. Its syntax is:



The angle brackets specify system include files, and double quotation marks specify user include files.

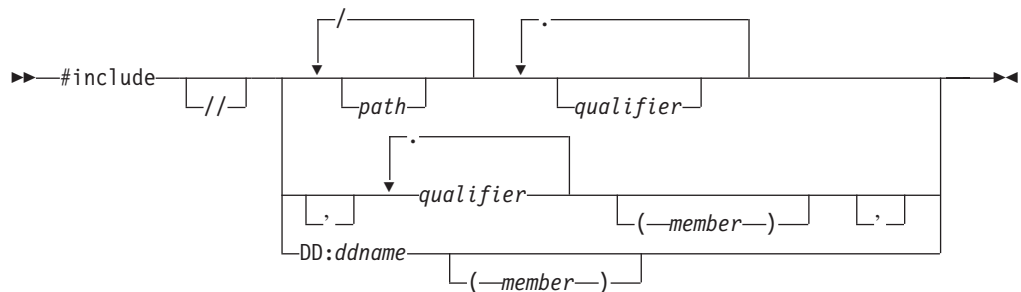
When you use the `#include` directive, you must be aware of the following:

- The *library search sequence*, the search order that C/C++ uses to locate the file. See “Search Sequences for Include Files” on page 254 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 *OS/390 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 115 and “SEARCH | NOSEARCH” on page 140.

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.

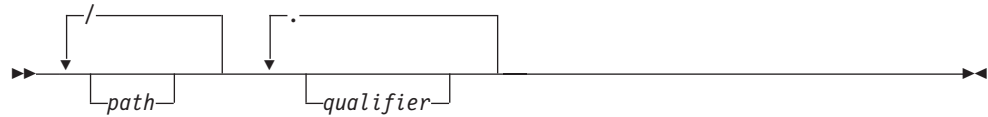
### Note:

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.

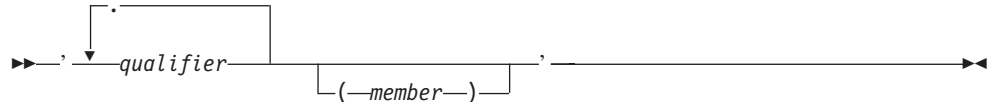
## Forming File Names

Refer to “Determining whether the File Name is in Absolute Form” on page 251 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, it performs the following conversion:

- For the first format:



1. *qualifier* and *path* are uppercased
  2. each *qualifier* and *path* is truncated to 8 characters
  3. invalid characters and characters not valid for a data set name are converted to at signs (@, hex 7c)
- For the second format:



1. *qualifier* and *member* are uppercased
  2. invalid characters are converted to at signs (@, hex 7c)
- For the third format:



1. DD:, *ddname*, and *member* are uppercased
2. invalid characters are converted to at signs (@, hex 7c)

## 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)
```

### 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 119 to understand how to

specify sequential data sets. All *qualifiers* 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.AFIL.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 119 to understand how PDSs are specified. When this is the case then all the *paths*, slashes (replaced by periods), and any *qualifiers* 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(AFIL)
```

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 an OS/390 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(AFIL)
```

## 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 27. Include filename Conversions When NOOE Is Specified

#include Directive	Converted Name
Example 1. This <i>filename</i> 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 <i>filename</i> 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 <i>filename</i> is in absolute form. SEARCH is ignored.	
#include <'COMIC/BOOK.OLDIES.K'>	COMIC/BOOK.OLDIES.K
Example 3.	
SEARCH(LIB1.*,LIB2.+,LIB3) #include "sys/abc/xx"	<ul style="list-style-type: none"> <li>• first <i>opt</i> in SEARCH SEQUENTIAL FILE = <i>userid</i>.LIB1.XX</li> <li>• second <i>opt</i> in SEARCH PDS = <i>userid</i>.LIB2.SYS.ABC(XX)</li> <li>• third <i>opt</i> in SEARCH PDS = <i>userid</i>.LIB3(XX)</li> </ul>
Example 4.	
SEARCH(LIB1.*,LIB2.+,LIB3) #include "Sys/ABC/xx.x"	<ul style="list-style-type: none"> <li>• first <i>opt</i> in SEARCH SEQUENTIAL FILE = <i>userid</i>.LIB1.XX.X</li> <li>• second <i>opt</i> in SEARCH PDS = <i>userid</i>.LIB2.SYS.ABC.X(XX)</li> <li>• third <i>opt</i> in SEARCH PDS = <i>userid</i>.LIB3(XX)</li> </ul>
Example 5.	
SEARCH(LIB1.*,LIB2.+,LIB3) #include <sys/name_1>	<ul style="list-style-type: none"> <li>• first <i>opt</i> in SEARCH SEQUENTIAL FILE = <i>userid</i>.LIB1.NAME@1</li> <li>• second <i>opt</i> in SEARCH PDS = <i>userid</i>.SYS(NAME@1)</li> <li>• third <i>opt</i> in SEARCH PDS = <i>userid</i>.LIB3(NAME@1)</li> </ul>
Example 6.	
SEARCH(LIB1.*,LIB2.+,LIB3) #include <Name2/App1.App2.H>	<ul style="list-style-type: none"> <li>• first <i>opt</i> in SEARCH SEQUENTIAL FILE = <i>userid</i>.LIB1.APP1.APP2.H</li> <li>• second <i>opt</i> in SEARCH PDS = <i>userid</i>.LIB2.NAME2.APP2.H(APP1)</li> <li>• third <i>opt</i> in SEARCH PDS = <i>userid</i>.LIB3(APP1)</li> </ul>
Example 7. The PDS member named YEAREND of the library associated with the ddname PLANLIB is used. A library search is not performed when <i>filename</i> 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:

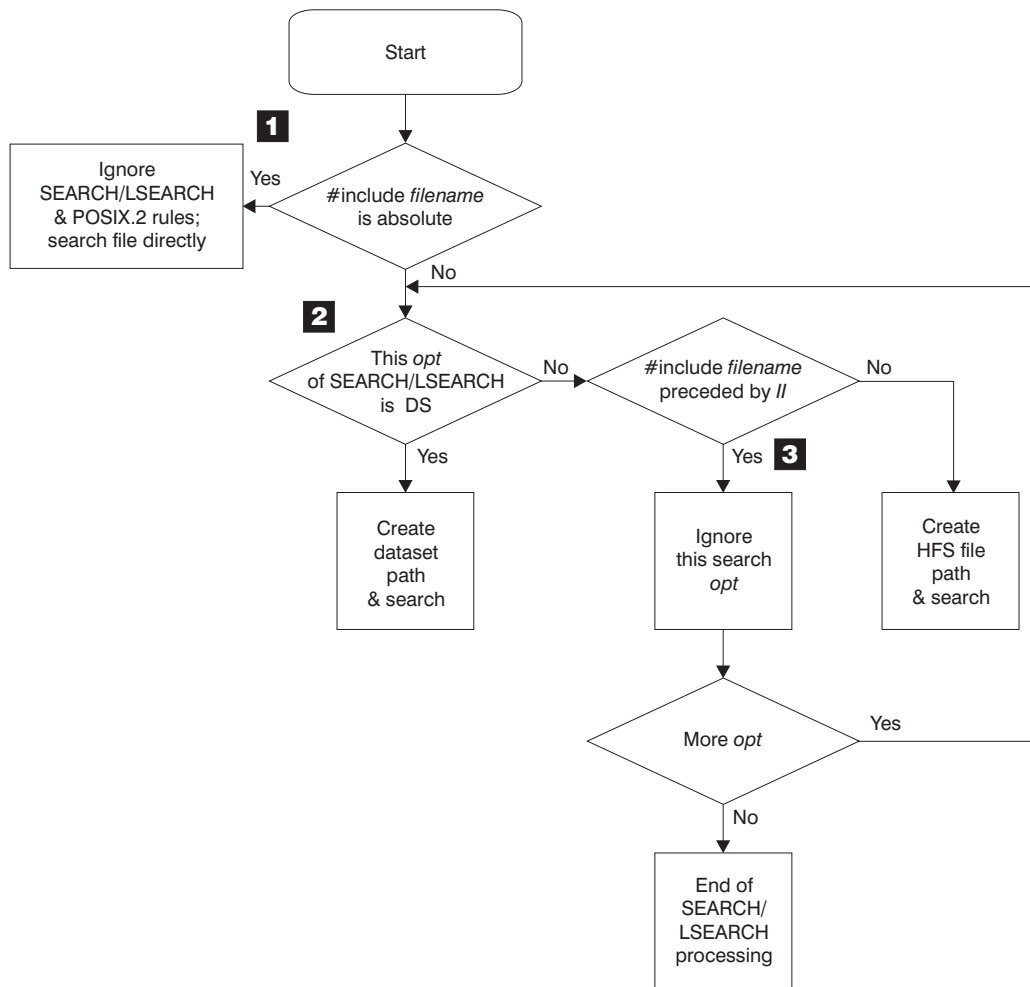


Figure 26. Overview of Include File Searching

- 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 27 on page 252 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 28 on page 253.
- 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:

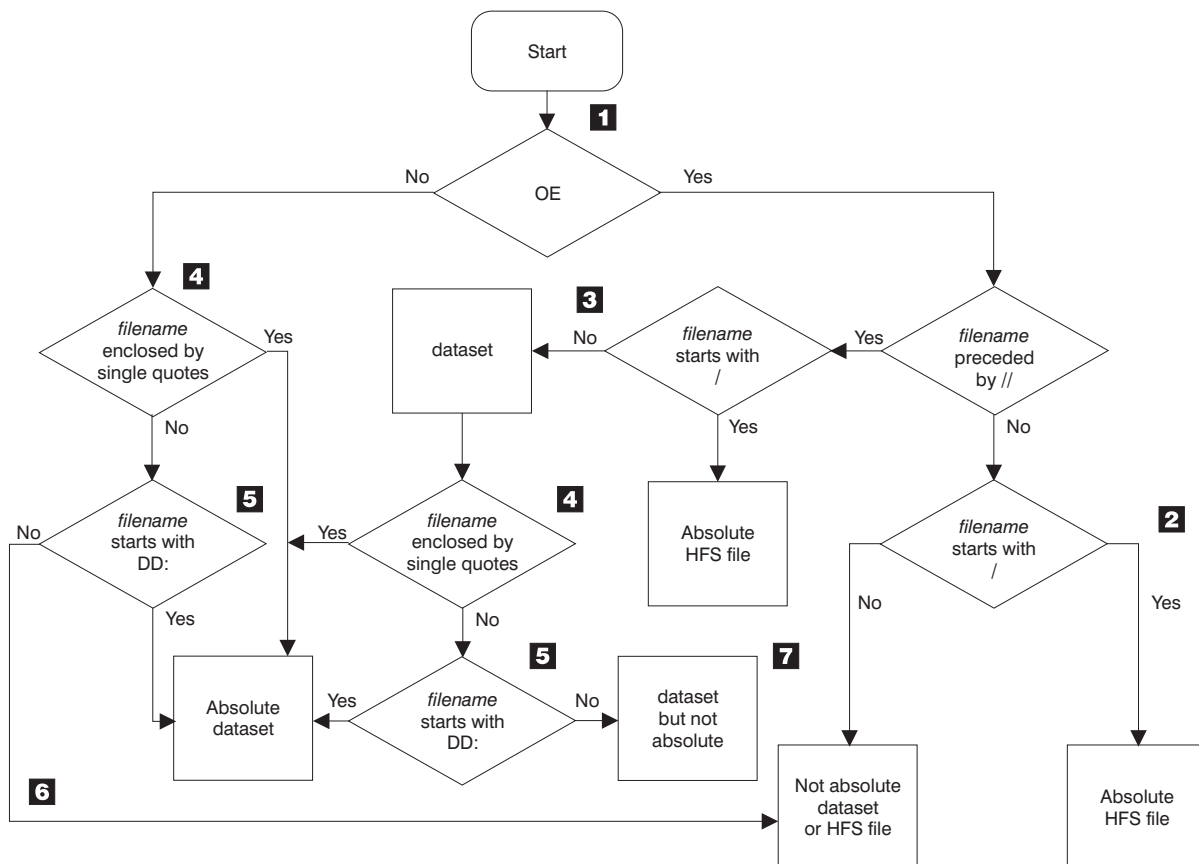


Figure 27. Testing If filename Is In Absolute Form

- 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 *opts* 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 *opts* in SEARCH are used to find system include files and the *opts* 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:

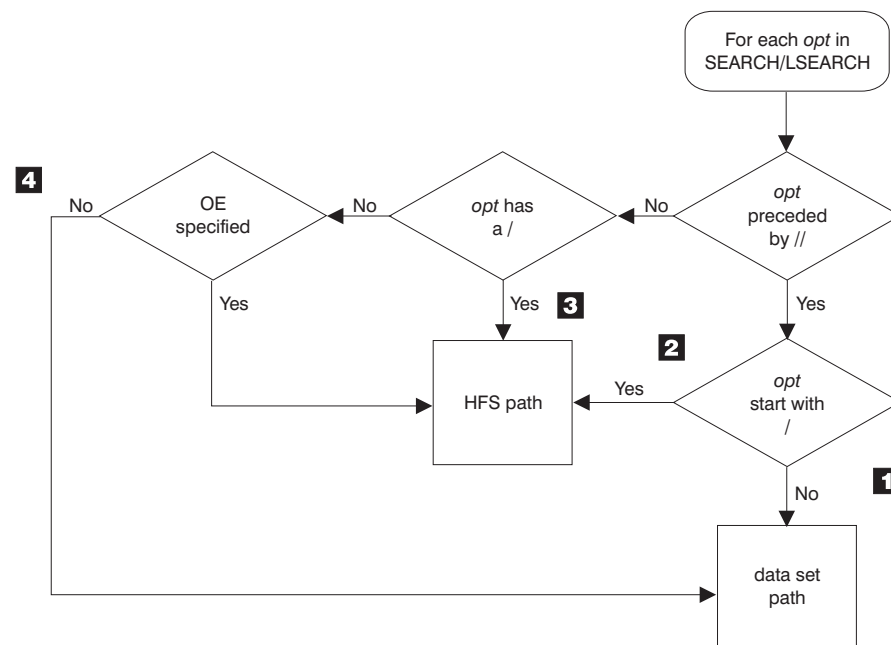


Figure 28. Determining if the SEARCH/LSEARCH opt is an HFS path

### 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 115 and "SEARCH | NOSEARCH" on page 140 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 251 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 28. 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 251 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 29. Examples of Search Order for OS/390 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 OS/390 UNIX System Services 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 OS/390 UNIX System Services 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 OS/390 C Source Code Using the SEARCH option

The following data sets contain the commonly-used system header files for C: <sup>1</sup>

- 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.+')
```

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 OS/390 C++ Source Code Using the SEARCH option

The following data sets contain the commonly-used system header files for OS/390 C++: <sup>1</sup>

- 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.+')
```

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.

---

1. The high-level qualifier may be different at 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 265 for tips on getting the most improvement.

**Note:** A precompiled header may not 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 264. 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 29 on page 261 illustrates how the initial sequence can vary, depending on whether any macros are defined on the command line.



sider 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
#include <stdio.h>
#include "h1.h"

/* Comments are OK */
#include "h2.h"
#include "h3.h"
main() {
.
.
.
}
```

Figure 29. 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 30. 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 <code>int f(int);</code> occurs before any <code>#include</code> 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 OS/390 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 30. 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

Source file type	MVS File	HFS File
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 OS/390 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 or 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, 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 OS/390 C/C++ Programs

This chapter shows how to use the IPA (Interprocedural Analysis) Link step with your OS/390 C/C++ program. Before reading this chapter, refer to the *OS/390 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, OS/390 C/C++ applications contain references to OS/390 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 OS/390 C/C++, refer to “Chapter 16. Object Library Utility” on page 351. For information on binding object modules under OS/390 UNIX System Services, refer to “Chapter 12. Binding OS/390 C/C++ Programs” on page 289.

---

## Using DD Statements for the Standard Data Sets

The IPA Link step uses certain ddnames. Table 31 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 460.

*Table 31. Data Sets Used by the IPA Link Step*

ddname	Type	Function
SYSIN <sup>1</sup>	Input	Primary input
STEPLIB <sup>1, 5</sup>	Utility Library	Location of the OS/390 C/C++ compiler (which provides the IPA Link step) and the OS/390 Language Environment data sets
SYSLIB	Library	Data set for runtime library (SCEELKEX,SCEELKED) <sup>2</sup> Optional data sets for secondary input <sup>4</sup>
SYSLIN <sup>1</sup>	Output	Output data set for the object module, if the OBJECT compiler option is specified
SYSPUNCH <sup>1</sup>	Output	Output data set for the object module, if the DECK and NOOBJECT compiler options are specified
SYSOUT <sup>4</sup>	Output	Destination of diagnostic messages generated by the IPA Link step
SYSCPRT <sup>4</sup>	Output	IPA Link step listing, generated if the IPA(MAP), LIST, or XREF option is specified.
User-specified <sup>3</sup>	Input	Additional object modules and load modules
SYSUT1, SYSUT4-9, SYSUT14 <sup>1</sup>	Output	Work data sets



Table 31. Data Sets Used by the IPA Link Step (continued)

ddname	Type	Function
<b>Notes:</b>		
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 OS/390 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 OS/390 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 OS/390 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 an OS/390 C/C++ Application” on page 335.

## 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 OS/390 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 232.

**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 OS/390 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 193 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
  - OS/390 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.

**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 OS/390 C/C++ IPA Link step. You must recompile your source code with a later release of the OS/390 C/C++ IPA Compile step before attempting to use the current release of the OS/390 C/C++ IPA Link step.

Refer to “Object Record Formats” on page 275 for more information about the different types of object records.

### IPA Linking Multiple Object Modules

OS/390C/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
- OS/390 Language Environment libraries, if any of the OS/390 Language Environment library functions are needed to resolve external references

Refer to "Object Record Formats" on page 275 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 OS/390 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 OS/390 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 OS/390 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 OS/390 C/C++ Object Library Utility, see "Chapter 16. Object Library Utility" on page 351.

## 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

OS/390 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 OS/390 C/C++ programs under OS/390 Language Environment, refer to the *OS/390 Language Environment Programming Guide*.

OS/390 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 *OS/390 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 *OS/390 C/C++ Programming Guide*.

## Object File Formats

The High Level Assembler (HLASM) and other OS/390 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 OS/390 C/C++ compiler LONGNAME option is in effect. IPA Link accepts input in object file format. The OS/390 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 OS/390 Binder. IPA Link does NOT supported this format as input.

Refer to *DFSMS/MVS 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 OS/390 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
- 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 32. IPA Link Control Statements

Start Column	End Column	Field Length	Description
--------------	------------	--------------	-------------



Table 32. IPA Link Control Statements (continued)

1	1	1	Record type indicator <ul style="list-style-type: none"> <li>blank-Control</li> <li>0X02-Binary</li> <li>***-Comment</li> </ul>
2	71	70	Record data
72	72	1	Control statement continuation indicator <ul style="list-style-type: none"> <li>EBCDIC blank character-No continuation (required for last record)</li> <li>non-blank EBCDIC character-Continuation</li> </ul>
73	80	8	Record sequence number (optional field, contents not verified)

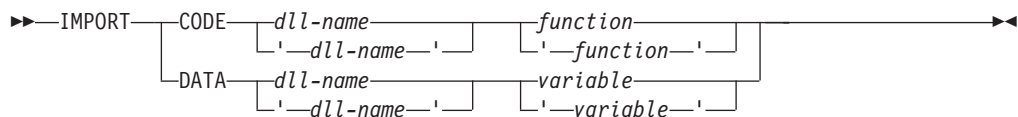
You can delimit character strings with blanks, commas, or parentheses. If character strings contain imbedded blanks, you must enclose the strings in single quotes. If you want to enclose a name in single quotation marks, and it contains a single quotation mark, replace the single quotation mark with 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.

**IMPORT Control Statement:** The IMPORT control statement has the following syntax:

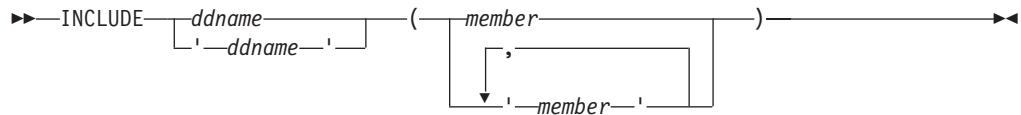


- dll-name* The directory name (primary member or alias) or HFS filename of the load module or program object that contains the imported function or variable. The maximum length of a dll-name is 1024 characters. The maximum length of an HFS filename is 255 bytes.
- variable* An exported variable name. 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:



*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:



*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.

## The 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:

### **csect=csect\_names**

Supplies information that the IPA Link step uses to name the CSECTs for each partition that it creates. The *csect\_names* 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:
 

<b>C</b>	code
<b>S</b>	static data
<b>T</b>	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 *nameprefix* is the qualifier you specified for CSECT and *qualifier* is the name you specified in the IPA Link Step Control File:
 

<b>qualifier#nameprefix#C</b>	code
<b>qualifier#nameprefix#S</b>	static data
<b>qualifier#nameprefix#T</b>	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 qualifier you specified for CSECT:
 

<b>nameprefix#C</b>	code
<b>nameprefix#S</b>	static data
<b>nameprefix#T</b>	test

The IPA Link step issues an error message if you specify the CSECT compiler 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 value 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.

**inline**=*name[,name]*

Specifies a list of functions that are desirable for the compiler to inline. The functions may or may not be inlined.

<b>inline=name[,name] from name[,name]</b>	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.
<b>noinline=name[,name]</b>	Specifies a list of functions that the compiler will not inline.
<b>noinline=name[,name] from name[,name]</b>	Specifies a list of functions that the compiler will not inline, if the functions are called from a particular function or list of functions.
<b>exits=name[,name]</b>	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.
<b>lowfreq=name[,name]</b>	Specifies names of functions that are expected to be called infrequently. These functions are typically error handling or trace functions.
<b>partition=small medium large unsigned-integer</b>	<p>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.</p> <p>For a finer degree of control, you can use an <i>unsigned-integer</i> 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.</p> <p>The size of a CSECT cannot exceed 16 MB.</p> <p>The default for this directive is <i>medium</i>.</p>
<b>safe=name[,name]</b>	Specifies a list of "safe" functions. These are functions that do not indirectly call a visible (not missing) function either through a direct call or a function pointer.
<b>isolated=name[,name]</b>	Specifies a list of "isolated" functions. These are functions that do not directly reference global variables accessible to visible functions. IPA assumes that functions that are bound from shared libraries are isolated.
<b>pure=name[,name]</b>	Specifies a list of "pure" functions. 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. 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. These are functions that are not safe, isolated, or pure.

**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:

<b>safe</b>	Specifies that the missing functions are "safe". See the description for the <code>safe</code> directive, above.
<b>isolated</b>	Specifies that the missing functions are "isolated". See the description for the <code>isolated</code> directive, above.
<b>pure</b>	Specifies that the missing functions are "pure". See the description for the <code>pure</code> directive, above.
<b>unknown</b>	Specifies that the missing functions are "unknown". See the description for the <code>unknown</code> 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.

---

## 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 25 on page 227.

## Specifying Output Files

You can use compiler options to specify the output files for IPA Link, as follows:

*Table 33. 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 34. OS/390 creates HFS files in the current working directory.

The IPA Link step uses the following default suffixes:

*Table 34. 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 *OS/390 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 34. 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 OS/390 Shells and Utilities manual for more information.

**Notes:**

1. If you have specified the 0E option, see “OE | NOOE” on page 127 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 110.

## 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 125.

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 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 193 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 OS/390 Batch

The following diagram shows the basic IPA Link step process for your C/C++ application.

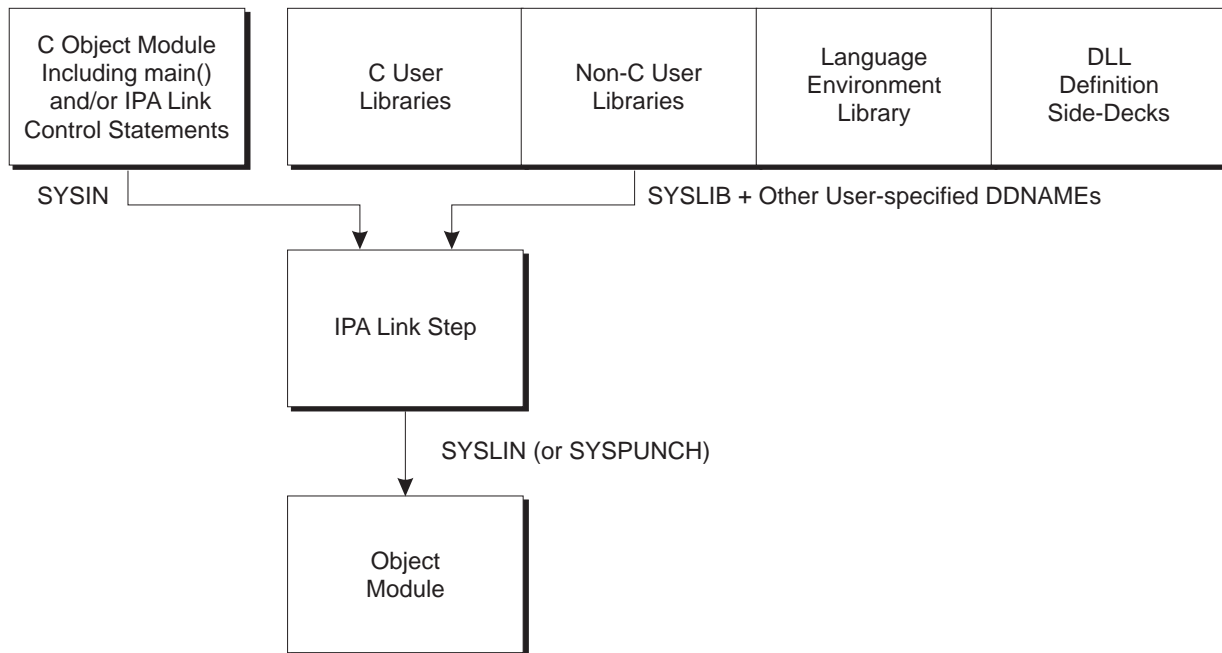


Figure 31. Basic IPA Link Step Processing

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 OS/390 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=*
//SYSPRT DD SYSOUT=*
//SYSIN DD DATA,DLM=@@
  INCLUDE OBJECT(TESTFILE)
  INCLUDE LIBRARY(DECODE)
@@

```

Figure 32. IPA Linking a Program under OS/390 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 OS/390 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.SALES LIB,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 DSN=userid.IPAOBJ,DISP=SHR
//      DD DSN=...
      :

//      DD *
      INCLUDE ddname(member)
      LIBRARY ADDLIB(CPGM10)
/*

```

Figure 33. 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=CBCDRV,PARM='/IPA(MAP,LIST,DUP,ER,NONCAL)'
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
//      DD DSN=CEE.SCBCCMP,DISP=SHR
//SYSLIN DD DSN=userid.MYPROG.OBJ,DISP=SHR
//SYSLIB DD DSN=userid.SECOND.LOAD,DISP=SHR
//      DD DSN=CEE.SCEELKD,DISP=SHR
//OBJECT DD DSN=userid.WORK.OBJECT,DISP=SHR
//LIBRARY DD SYSOUT=userid.LIBRARY.OBJECT,DISP=SHR
//SYSOUT DD SYSOUT=*
//SYSPRT DD SYSOUT=*
//SYSUT1 DD DSN=...
//SYSUT4 DD DSN=...

...
//SYSIN DD *
      INCLUDE OBJECT(TESTFILE)
      INCLUDE LIBRARY(DECODE)
/*

```

Figure 34. JCL for Running the IPA Link Step

---

## Running the IPA Link Step in OS/390 UNIX

Processing your application under OS/390 UNIX System Services is the same as processing it under OS/390 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 DSN=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=*
//SYSPRT DD SYSOUT=*
//SYSIN DD *
//   INCLUDE DDOBJ1
//   INCLUDE DDOBJ2
//   LIBRARY DDLIB3
//   LIBRARY DDLIB4
/*

```

Figure 35. Using JCL for IPA Linking an OS/390 UNIX Application

## 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 the *OS/390 UNIX System Services Command Reference* for more information about the c89 utility.

---

## Chapter 12. Binding OS/390 C/C++ Programs

This chapter describes how to bind your programs using the binder (the DFSMS/MVS program management binder) in the OS/390 batch, OS/390 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 OS/390 batch or TSO:

If you can use PDSEs, you can use the binder. If you want to use PDSs, you must use the prelinker for the following:

- C++ code
- C code compiled with the LONGNAME, RENT, or DLL options

For more information on the prelinker, see “Appendix A. Prelinking and Linking OS/390 C/C++ Programs” on page 403.

---

### When You Cannot Use the Binder

The following are the restrictions to using the binder.

#### Your Output is a PDS, not a PDSE

If you are using OS/390 batch or TSO, and your output must target a PDS instead of a PDSE, you cannot use the binder.

#### CICS

CICS does not support PDSEs. If you have to target CICS, you cannot use the binder.

#### 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,NOBJECT)` 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 36 on page 291 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.

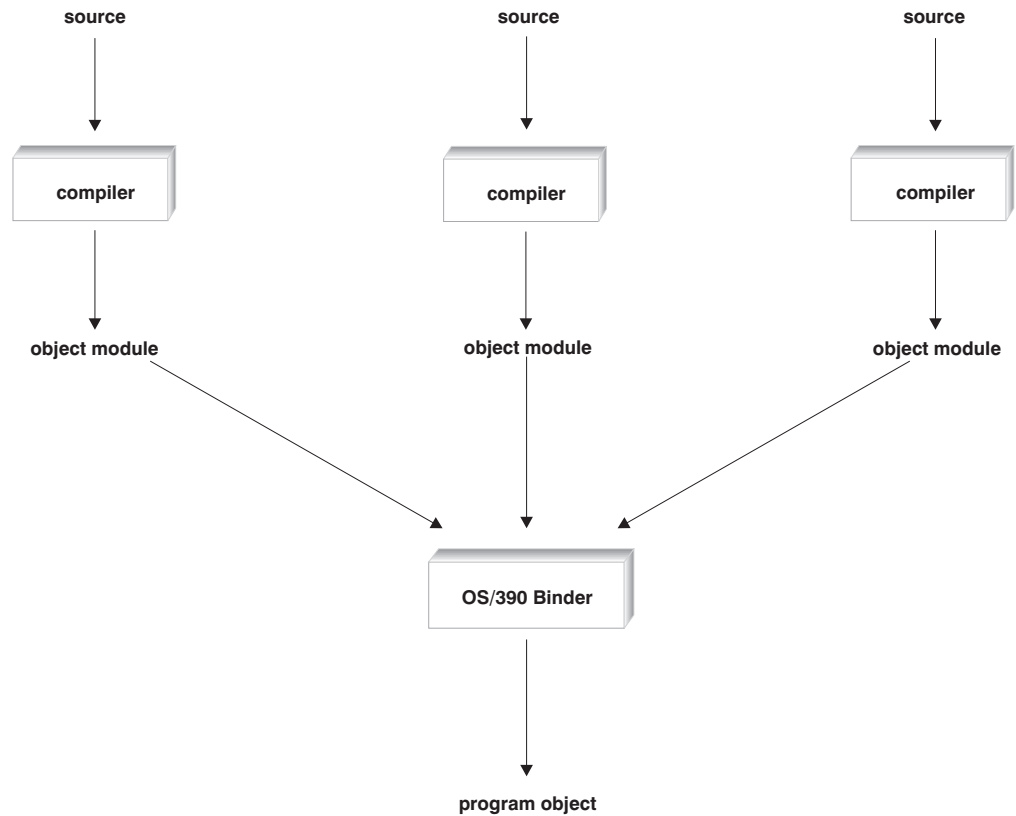


Figure 36. Single Final Bind

## Bind Each Compile Unit

If you have changed the source in a compile unit, you can use the method that is shown in Figure 37 on page 292. 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.

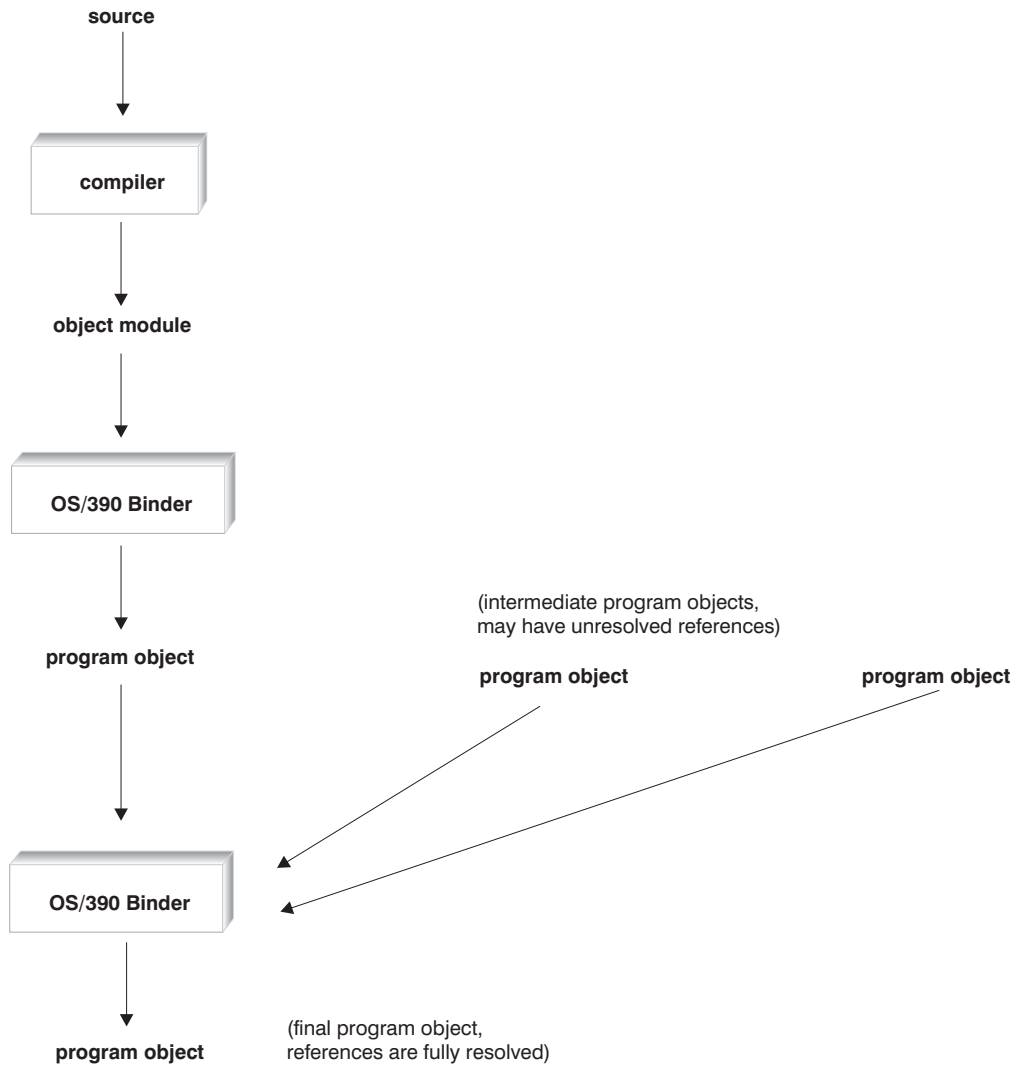


Figure 37. Bind Each Compile Unit

## Build and Use a DLL

You can use the method that is shown in Figure 38 on page 293 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.



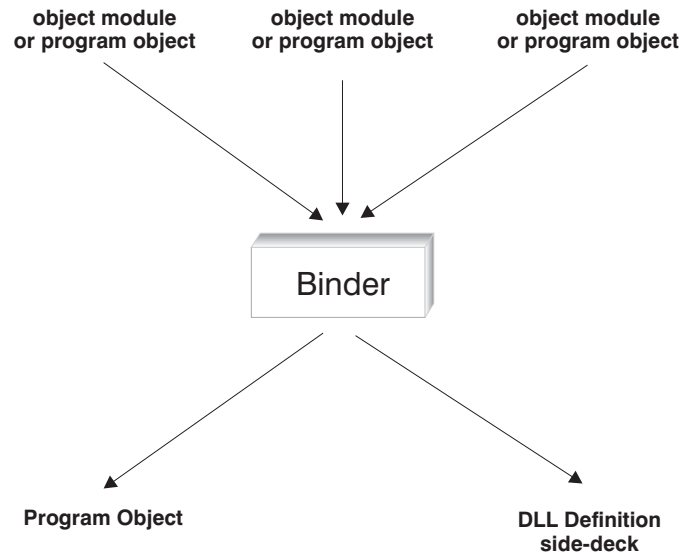


Figure 38. Build a DLL

You can use the method that is shown in Figure 39 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.

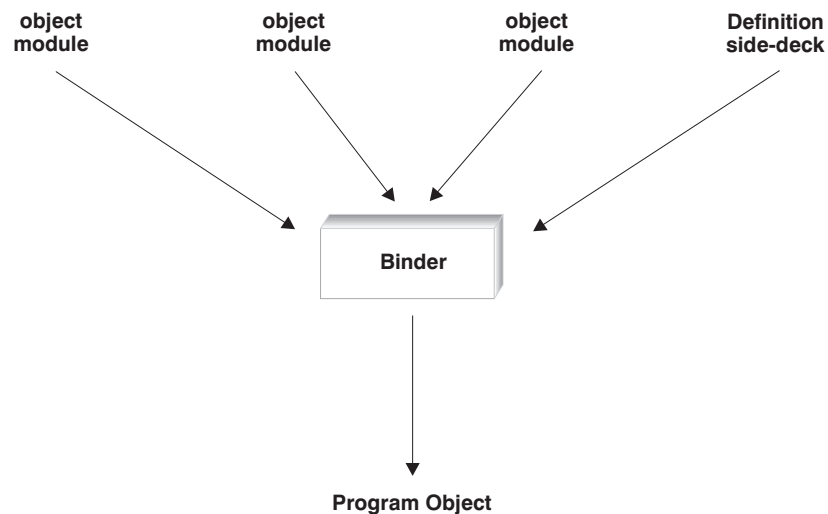


Figure 39. Build an application that uses a DLL

## Rebind a Changed Compile Unit

You can use the method shown in Figure 40 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.

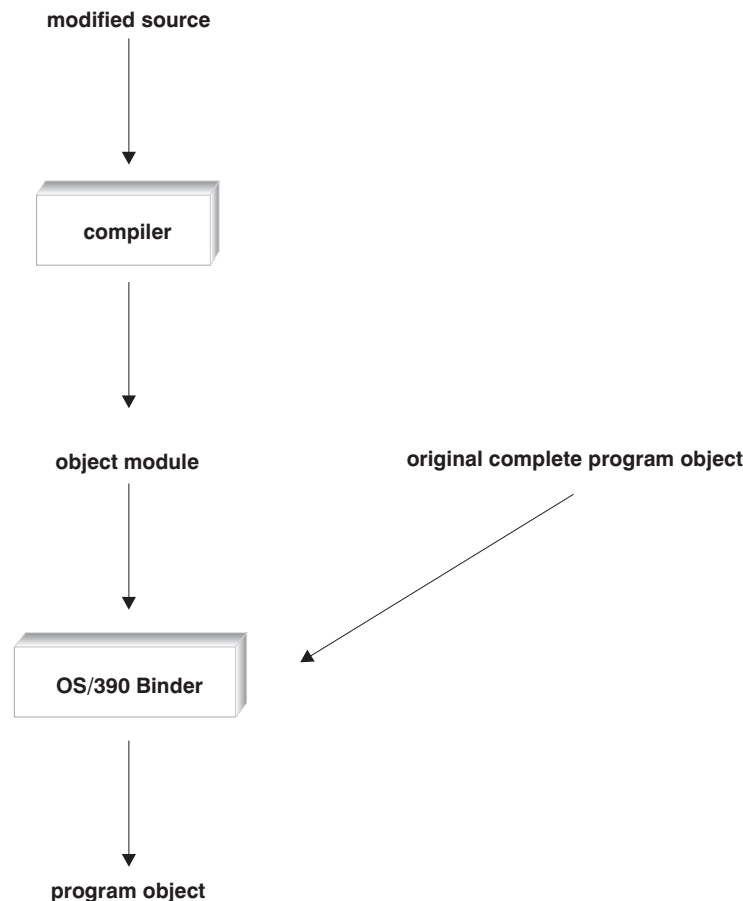


Figure 40. Rebinding a Changed Compile Unit

---

## Binding Under OS/390 UNIX

The `c89` and `c++` utilities are the interface to the compiler and the binder for OS/390 UNIX System Services C/C++ applications. You can use `c89` and `c++`, to compile and bind a OS/390 UNIX System Services C/C++ program in one step, or to bind application object modules after compilation.

Typically, you invoke the `c89` and `c++` utilities from the OS/390 shell. For more information on these utilities, see the *OS/390 UNIX System Services Command Reference*.

## OS/390 UNIX Example

The example source files `unit0.c`, `unit1.c`, and `unit2.c` that are shown in Figure 41, are used to illustrate all of the OS/390 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 41. Example Source Files

## Single Final Bind Using `c89`

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 *OS/390 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 *OS/390 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 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 rebinding. 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:

```
touch myprog.old
cp -m myprog myprog.old
c89 -o myprog unit1.o myprog.old
rm myprog.old
```

The touch command creates myprog if it does not exist, because myprog.old is needed for the rebind step. First myprog is saved under myprog.old; then myprog is overwritten with the result of the bind of unit1.o and the old executable. Like named sections in unit1.o replace those in the old 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
CFLAGS = -W c,"CSECT(myprog)"
.PRECIOUS : myprog 3
myprog : $(SRCS)
    @if [ -e myprog ]; then OLD=myprog; \
    else OLD=; \
    fi; \
    CMD="$(CC) -Wc,csect $(CFLAGS) -o myprog.new $? $$OLD"; \
    echo $$CMD; $$CMD; \ 4
    if [ $$? -eq 0 ]; then \
    mv -f myprog.new myprog; \ 5
    fi;
    -@rm -f $(?:b:+"o") myprog.new 6
```

- 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
- 5** if any files were compiled, rename myprog.new to myprog
- 6** remove .o files and old program

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 and make, see *OS/390 UNIX System Services Command Reference* . For a make tutorial, see *OS/390 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 under OS/390 Batch

You can use the following procedures, which the OS/390 C/C++ compiler supplies, to invoke the binder:

Procedure name	Description
EDCCB	C Compile and Bind steps
EDCCBG	C Compile, Bind, and Go steps
CBCB	C++ Bind step
CBCBG	C++ Bind and Go steps
CBCCB	C++ Compile and Bind steps
CBCCBG	C++ Compile, Bind, and Go steps

If you want to generate DLL code, you must use the binder DYNAM(DLL) option. All the OS/390 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.

## OS/390 Batch Example

Figure 42 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 OS/390 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 42. Example Source Files*

## Single Final Bind under OS/390 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 OS/390 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 OS/390 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 OS/390 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 rebinding. 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
//SYSLIN DD DISP=SHR,DSN=USERID.PLAN9.OBJ(P1)
// DD DISP=SHR,DSN=CBC.SCLBSID(ISTREAM)
//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. The Language Environment runtime libraries SCEELKED and SCEELKEX 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.

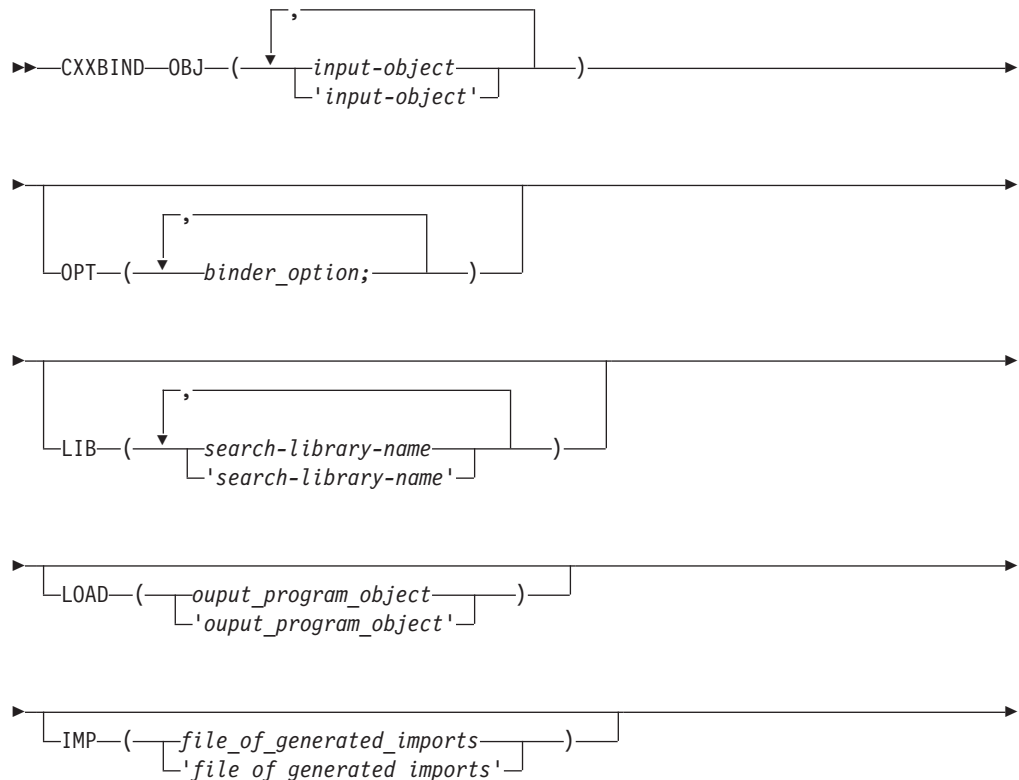
## Binding Under TSO Using CXXBIND

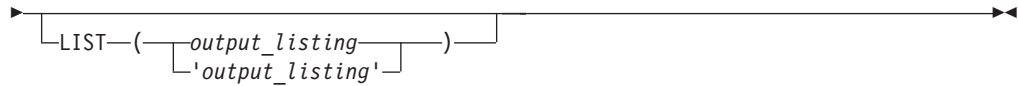
This section describes how to bind your OS/390 C++ or OS/390 C program in TSO by invoking the CXXBIND REXX EXEC. This REXX EXEC invokes the binder and creates an executable program object.

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:





OBJ	<p>You must <b>always</b> specify the input file names by using the OBJ keyword parameter. Each input file must be one of the following:</p> <ul style="list-style-type: none"> <li>• An object module that can be a PDS member, a sequential data set, or an HFS file</li> <li>• A load module that is a PDS member</li> <li>• A program object that can be a PDSE member or an HFS file</li> <li>• A text file that contains binder statements. The file can be a PDS member, a sequential data set, or an HFS file</li> </ul>
OPT	<p>Use the OPT keyword parameter to specify binder options. For example, if you want the binder to use the MAP option, specify the following:</p> <pre>CXXBIND OBJ(PLAN9.OBJ(PROG3)) OPT('MAP')...</pre>
LIB	<p>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.</p> <p>The default libraries that are used are the C run-time libraries CEE.SCEELKED, CEE.SCEELKEX, and the C++ base library, CEE.SCEECPP. The default library names are added to the DD SYSLIB concatenation if library names are specified with the LIB keyword parameter.</p>
LOAD	<p>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.</p>
IMP	<p>Use the IMP keyword parameter to specify where the generated IMPORT control statements should be written.</p>
LIST	<p>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.</p>

## TSO Example

Figure 43 on page 307 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 43. 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(PAN9.OBJ) CSECT(MYPROG)
CXXBIND OBJ(PAN9.OBJ(UNIT2)) OPT('LET,CALL(NO)')
LOAD(PAN9.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(PAN9.LOADE(UNIT0), PAN9.LOADE(UNIT1), PAN9.LOADE(UNIT2))
LOAD(PAN9.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 PAN9.C(UNIT1) and PAN9.C(UNIT2) into DLL PAN9.LOADE(ONETWO) which exports functions f1(), f2(), f3() and f4(). Then build PAN9.C(UNIT0) into a program which dynamically links to functions f1() and f4() defined in the DLL.

1. Compile PAN9.C(UNIT1) and PAN9.C(UNIT2) to generate the object modules PAN9.OBJ(UNIT1) and PAN9.OBJ(UNIT2) which have functions to be exported. Use the CC REXX exec as follows:

```
CC PAN9.C(UNIT1) OBJECT(PAN9.OBJ) EXPORTALL, LONGNAME, DLL, CSECT(MYPROG)
CC PAN9.C(UNIT2) OBJECT(PAN9.OBJ) EXPORTALL, LONGNAME, DLL, CSECT(MYPROG)
```

2. Bind PAN9.OBJ(UNIT1) and PAN9.OBJ(UNIT2) to generate the DLL PAN9.LOADE(ONETWO):

```
CXXBIND OBJ(PAN9.LOADE(UNIT0), PAN9.LOADE(UNIT1)) IMP (PAN9.IMP(ONETWO))
LOAD(PAN9.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 PAN9.C(UNIT0) so that it may import unresolved symbols, and bind with PAN9.IMP(ONETWO), which is the definition side-deck containing IMPORT control statements from the DLL's build:

```
CC PAN9.C(UNIT0) OBJECT(PAN9.OBJ) CSECT(MYPROG), DLL
CXXBIND OBJ(PAN9.LOADE(UNIT0), PAN9.IMP(ONETWO)) LOAD(PAN9.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 OS/390 C/C++ object module or program object. You cannot rebind load modules unless they are naturally reentrant, i.e. C code compiled with the NODLL, NOLONGNAME, and NORENT compiler options.

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.

OS/390 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 rebindingable, provided all the sections are named. You can use the CSECT compiler option or the #pragma csect directive to name a section. See "CSECT | NOCSECT" on page 79.

**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.

---

## 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, 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 rebindingable.

---

## 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 334 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 294 and “Rebindability” on page 327.

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 211. 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 input (DD SYSLIN).

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.

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; For example, '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.

## 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 214.
4. If the name corresponds to a Language Environment function, the binder maps 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 320.

## 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. 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 208 for more information on the DYNAM(DLL) binder option.

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. All C++ code generates 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++ mangled names are demangled when they appear in messages and listings.

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 45 on page 318. Figure 44 on page 318 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 44. 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
//SYSDSFSD 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 45. 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:

```
DFSMS/MVS V1 R4.0 BINDER      11:13:38 TUESDAY JUNE  3, 1997
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 209 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 OFFSET	CLASS OFFSET	NAME	TYPE	LENGTH	DDNAME	SOURCE 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 314. 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.

CROSS - REFERENCE TABLE									
TEXT CLASS = C_CODE									
REFERENCE				TARGET					
CLASS	SECT/PART(ABBREV)	ELEMENT OFFSET	TYPE	SYMBOL(ABBREV)	SECTION (ABBREV)	ELEMENT 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 324.

In the example above, you can see that c2 and c3 are to be dynamically linked

```
*** IMPORTED AND EXPORTED SYMBOLS ***
```

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 to their corresponding mangled names.

```
*** MANGLED TO DEMANGLED SYMBOL CROSS REFERENCE ***
```

```
-----
MANGLED NAME
DE-MANGLED NAME
-----
```

```
b2__Fv
b2()
```

```
c3__Fv
c3()
```

```
*** END OF MANGLED TO DEMANGLED CROSS REFERENCE ***
```

## 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
SIDFILE	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
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 rebindingable. 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 rebinding 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.

## Rebindingability

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 rebind 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 46:

```
/* 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 46. 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.

---

## 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 314

page 314. 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 213.

## 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,"D01")
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 %d\n",i1,i2,i3,i4);
    return 0;
}

```

```

/* file: doit2.c */
int int1 = 11;
#pragma csect(code,"D02")
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 D01.

```

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 D01 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 D01.
IEW2308I 1112 SECTION D02 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 D01 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 D01.
IEW2308I 1112 SECTION D02 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
/*
```

## 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 OS/390 Prelinker and the binder use different methods to manage the Writable Static Area. The OS/390 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 an OS/390 C/C++ Application

This chapter gives an overview of how to run an OS/390 C/C++ program under OS/390 batch, TSO, and the OS/390 Shell.

OS/390 Language Environment provides a common runtime environment for C, C++, COBOL, PL/I, and FORTRAN. For detailed instructions on running existing and new OS/390 C/C++ programs under OS/390 Language Environment, refer to the *OS/390 Language Environment Programming Guide*. The *OS/390 C/C++ Programming Guide* also describes how to run an OS/390 C/C++ program in a CICS environment.

---

### Running an Application Under OS/390 Batch

You must have the Language Environment Library SCEERUN available before you try to run your application under OS/390 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 OS/390 Batch

When you run an OS/390 C/C++ application, you can override the default values for a set of OS/390 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 OS/390 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 336.
- On the GPARM parameter of the cataloged procedures that are supplied by IBM. Refer to "Using Cataloged Procedures" on page 336.
- The `#pragma runopts` statement in your source code.
- The CEEUOPT facility that is provided by OS/390 Language Environment.
- In the assembler user exit. For more information, refer to the *OS/390 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 217.

You should establish the required settings of the options for all OS/390 C/C++ programs that you execute on a production basis. Each time the program is run, the default runtime options that were selected during OS/390 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 OS/390 batch. Partitioned data set member `MEDICAL.ILLNESS.LOAD(SYMPTOMS)` contains your OS/390 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 47. Running your program under OS/390 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:

```
//G01 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 OS/390 C/C++ compiler to run your program. Each procedure listed below includes an execution step:

For OS/390 C programs:

EDCCBG            Compile, bind, and run.

For OS/390 C++ programs:

CBCBG            Bind and run.

CBCCBG           Compile, bind, and run.

CBCG                      Run

For more information on these cataloged procedures, see “Appendix D. IBM Supplied Cataloged Procedures and REXX EXECs” on page 457.

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 OS/390 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 217.

---

## 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 LNKLIST.
- Have your system programmer change the LOGON PROC so the libraries are added to the STEPLIB for the TSO session.
- 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 *OS/390 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 217.

If you want to pass your parameters as mixed case, you must use the ASIS runtime option. See “Passing Arguments to the OS/390 C/C++ Application” for more information on passing mixed case parameters.

## Passing Arguments to the OS/390 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 74 or the *OS/390 C/C++ Language Reference*.

The case of the characters in `argv` depends on you invoked how your OS/390 C/C++ program, as shown in the following table.

Table 35. Case sensitivity of arguments under TSO

How an OS/390 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
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 OS/390 UNIX

This section discusses how to run your OS/390 UNIX System Services C/C++ application.

### OS/390 UNIX Application Environments

You can run your OS/390 UNIX System Services C/C++ application programs from the following environments:

- OS/390 shell
- OS/390 ISPF Shell (ISHELL)
- TSO/E

To call a OS/390 UNIX System Services application program that resides in an HFS file from the TSO/E READY prompt, you must use the BPXBATCH utility.

- OS/390 batch

To run a OS/390 UNIX System Services application program that resides in an HFS file, you must use the BPXBATCH utility with the JCL EXEC statement.

- OS/390 shell through OS/390 batch or TSO

By using the IBM-supplied BPXBATCH program, you can run a OS/390 UNIX System Services 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 OS/390 batch environment or under TSO.

### Specifying Runtime Options under OS/390 UNIX

When invoking a program from the OS/390 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 OS/390 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 OS/390 C/C++ program from within a OS/390 UNIX System Services application, ensure that the executable resides in a PDS or PDSE member.

Any new OS/390 UNIX System Services OS/390 C/C++ applications you develop should be 31-bit AMODE.

## Copying Applications between a PDS and HFS

If you have a OS/390 UNIX System Services C/C++ application as a PDS member and want to place it in the HFS, you can use the OS/390 UNIX System Services TSO/E command `0PUTX` to copy the member into an HFS file.

If you have a OS/390 UNIX System Services C/C++ application as an HFS file and want to place it in a PDS, you can use the OS/390 UNIX System Services TSO/E command `0GETX` 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)"
```

**OS/390 C++ Note:** To use the TSO utility `0GET` 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
```

For a description of these commands, see the *OS/390 UNIX System Services Command Reference*. For examples of using these commands to copy data sets to HFS files, see *OS/390 UNIX System Services User's Guide*.

## Running a Data Set Member from the OS/390 Shell

If your OS/390 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 an OS/390 UNIX Application under OS/390 Batch

### Using the BPXBATCH Utility

Use the IBM-supplied `BPXBATCH` program to run a OS/390 UNIX System Services C/C++ application under OS/390 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 22. BPXBATCH Utility” on page 397.

## 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 48 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 48. 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 OS/390 UNIX Executable to Run under OS/390 Batch**

If your program requires OS/390 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 OS/390 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 OS/390.

- “Chapter 15. Model Tool” on page 345
- “Chapter 16. Object Library Utility” on page 351
- “Chapter 17. DLL Rename Utility” on page 357
- “Chapter 18. Filter Utility” on page 365
- “Chapter 19. DSECT Conversion Utility” on page 371
- “Chapter 20. Coded Character Set and Locale Utilities” on page 385



---

## Chapter 15. Model Tool

The OS/390 C/C++ Model Tool provides online help for developers who use the OS/390 C/C++ compiler and the C/C++ runtime library of OS/390 Language Environment. You can start the Model Tool from an ISPF edit session. You can obtain help for library functions and pragma directives. You can also access the standard menu that provides help for ISPF functions.

---

### About the OS/390 C/C++ Model Tool

- The Model Tool contains library function and #pragma information that is current as of OS/390 Release 2.
- To use the OS/390 C/C++ Model Tool, you or your system programmer must customize your OS/390 system. The ISPLIB and ISPSLIB data sets that are shipped with OS/390 C/C++ must be installed on your system, and must be allocated to your TSO session.
- You should keep in mind that not all functions and pragmas are supported under both C and C++. The versions of the menus that are displayed, and the functions and pragmas that you can select from, depend upon whether you are developing a C or a C++ program. The Model Tool determines the language by checking the low-level qualifier of the data set that you are editing. You can use the MODEL CLASS command to link a low-level qualifier to a language. For more information on this command and the MODEL command in general, refer to the *OS/390 ISPF Edit and Edit Macros*.

Refer to the OS/390 Program Directory for more information about installing and customizing the Model Tool.

- If you use the OS/390 C/C++ Model Tool, you cannot invoke ISPF models directly from the command line. You must select the ISPF menu from the main Model Tool menu.
- If the data set or data set member that you are editing is not empty when you request pragma or library function information, you must indicate where to add the information. To do this, type a or b in the prefix area.
- Functions that are described in the OS/390 C Curses manual are not included in the Model Tool.
- You cannot invoke the `log()` function by typing `log` on the menu of functions beginning with the letter `l`. You must enter the option number instead.
- Functions in the `xti.h` header file are not accessible directly from the command line or from the main Model Tool menu.
- You can use the UP and DOWN commands to scroll through any menu if "More" appears in the upper right corner. You can obtain help for the Model Tool by selecting PF1 from any menu.

---

### Accessing Library Functions

You can use the Model Tool in several ways to obtain help for library functions. The following sections describe these methods. Method 1 is the fastest if you already know the name of the library function that you want.

**Note:** The following menus are for C applications. There are similar menus for C++ applications. Whether C or C++ help is displayed depends upon the low-level qualifier of the data set you are editing. Refer to "About the OS/390 C/C++ Model Tool" for more information.

## Method 1

Enter MODEL funcname (where funcname is a function name, such as fopen), on the command line of your edit session. The Model Tool drops read-only information into your edit session. For example, if you enter MODEL fopen, information is displayed as shown in Figure 49:

```
File Edit Confirm Menu Utilities Compilers Test Help
-----
EDIT      USERID.C(TEST)                      Columns 00001 00072
Command ==> _                               Scroll ==> PAGE
***** ***** Top of Data *****
=NOTE=   #include <stdio.h>
=NOTE=   FILE *fopen(const char *filename, const char *mode);
***** ***** Bottom of Data *****
```

Figure 49. Function information

NOTE in the prefix area marks the information that is read-only. To make it disappear enter RESET on the command line, or end the edit session. If you want to keep any of the read-only information, use the MD command in the corresponding prefix area.

## Method 2

Enter MODEL (without a function name) on the command line of your edit session. The main Model Tool menu is displayed, as shown in Figure 50.

```
OPTION ==> _

          F  LIBRARY FUNCTIONS SUPPORTED BY C
          P  PRAGMA DIRECTIVES SUPPORTED BY C
          I  ISPF FUNCTIONS AVAILABLE UNDER C/C++

Enter END command to cancel MODEL command.
```

Figure 50. Main Model Tool menu

Next, enter F in the option field of the menu. An alphabetical menu is displayed, as shown in Figure 51 on page 347:

```

----- Functions Supported by C-Select a Menu of Functions -----
OPTION ==>

Functions are grouped according to the first character of the function
name. Enter a letter to retrieve the menu of functions whose names
begin with that letter. For instance, for a list of functions whose
names start with the letter a, enter letter a in the option field.
For functions whose names begin with a number sign, enter a # sign.

You may also bypass the function menus and access function information
directly by entering the name of the function in the option field. For
instance, for information on the fopen() function, enter fopen in the
option field.

Enter END command to return to previous menu.

```

Figure 51. Alphabetical menu

From this menu, you can select a menu that contains all of the functions whose names start with a particular letter. For example, selecting option 1 displays the menu containing functions whose names begin with the letter a, as shown in Figure 52:

```

----- function names beginning with a -----
OPTION ==> _
More: +

1 abort-Stop a program
2 abs-Calculate integer absolute value
3 accept-Accept a new connection on a socket
4 access-Determine whether a file can be accessed
5 acos-Calculate arccosine
6 acosh-Calculate hyperbolic arccosine
7 advance-Pattern match given a compiled regular expression
8 alarm-Set an alarm
9 asctime-Convert time to character string
10 asin-Calculate arcsine
11 asinh-Calculate hyperbolic arcsine
12 assert-Verify condition
13 atan-Calculate arctangent
14 atan2-Calculate arctangent
15 atanh-Calculate hyperbolic arctangent
16 atexit-Register program termination function
17 __atof-Perform ISO8859-1 to EBCDIC string conversion
18 __atof_l-Perform ISO8859-1 to EBCDIC conversion

Enter END command to return to alphabetical menu.

```

Figure 52. Menu of functions whose names begin with letter a

You can then select an option, or enter the name of a function that appears on the menu. The Model Tool drops read-only help into your edit session. Proceed as in method 1.

## Method 3

Enter a function name in the option field of the main Model Tool menu. The Model Tool drops read-only help into your edit session. Proceed as in method 1.

## Method 4

Enter a function name in the option field of the alphabetical menu. The Model Tool drops read-only help into your edit session. Proceed as in method 1.

For complete information about library functions, refer to the *OS/390 C/C++ Run-Time Library Reference*.

---

## Accessing Pragma Directives

You can obtain help for #pragma statements in one of several ways. Method 1 is the fastest if you already know the name of the pragma that you want.

**Note:** The following menus are for C applications. There are similar menus for C++ applications. Whether C or C++ help is displayed depends upon the low-level qualifier of the data set you are editing. Refer to “About the OS/390 C/C++ Model Tool” on page 345 for more information.

## Method 1

Enter MODEL pragname (where pragname is a pragma name such as chars), on the command line of your edit session. The Model Tool drops modifiable #pragma statements and read-only help into your edit session, as shown in Figure 53: Modifiable information is highlighted. You can alter or delete it to suit your program.

```
File Edit Confirm Menu Utilities Compilers Test Help
-----
EDIT      USERID.C(TEST)                      Columns 00001 00072
Command ==> _                               Scroll ==> PAGE
***** ***** Top of Data *****
=NOTE=    The #pragma chars directive specifies that the compiler is to treat
=NOTE=    all char objects as signed or unsigned.
=NOTE=
000001    #pragma chars(unsigned)
000002    #pragma chars(signed)
=NOTE=
=NOTE=    This pragma must appear before any statements in a file.
=NOTE=    Once specified, it applies to the rest of the file and cannot be
=NOTE=    turned off.
=NOTE=
=NOTE=    If a source file contains any functions that you want to be
=NOTE=    compiled without #pragma chars, place these functions in a
=NOTE=    different file.
=NOTE=
=NOTE=    The default character type behaves like an unsigned char.
***** ***** Bottom of Data *****
```

Figure 53. #pragma information

You will be given multiple versions of a #pragma statement if different options are available for the pragma.



NOTE in the prefix area marks the information that is read-only. Read-only information disappears when you enter RESET on the command line, or end the edit session, as shown in Figure 54:

If you want to keep any of the read-only information, use the MD command in the

```

File Edit Confirm Menu Utilities Compilers Test Help
-----
EDIT      USERID.C(TEST)                                Columns 00001 00072
Command ==> _                                           Scroll ==> PAGE
***** ***** Top of Data *****
000001    #pragma chars(unsigned)
000002    #pragma chars(signed)
***** ***** Bottom of Data *****

```

Figure 54. Saved #pragma information

corresponding prefix area. The read-only information is not exhaustive; for complete information about a pragma, refer to the *OS/390 C/C++ Language Reference* .

## Method 2

Enter MODEL (without a pragma name) on the command line of your edit session. Enter option P in the option field of the main Model Tool menu that appears. A menu that contains a list of pragmas is displayed, as shown in Figure 55. From this menu, you can select an option, or enter the name of the pragma. The

```

----- Pragmas Supported by C -----
OPTION ==> _

The following #pragma preprocessor directives may be used with the
OS/390 C compiler:

1 chars          15 margins
2 checkout       16 options
3 comment        17 pack
4 csect          18 page
5 environment    19 pagesize
6 export         20 runopts
7 filetag        21 sequence
8 hdrstop        22 skip
9 inline         23 strings
10 langlvl       24 subtitle
11 linkage        25 target
12 longname      26 title
13 map           27 variable
14 margins

Enter END command to cancel MODEL command

```

Figure 55. pragma menu

Model Tool drops modifiable #pragma statements and read-only help into your edit session. Proceed as in method 1.

## Method 3

Enter a pragma name in the option field of the main Model Tool menu. The Model Tool drops modifiable #pragma statements and read-only help into your edit session. Proceed as in method 1.



---

## Chapter 16. Object Library Utility

This chapter describes how to use the Object Library Utility to update libraries of object modules. On OS/390, 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 L-names, S-names, or writable static data. The Object Library Utility creates information, such as the members that contain defined L-names, S-names, or writable static data. This information is stored in a special member of the library that this chapter refers to as the C370LIB 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 OS/390 batch and TSO.

---

### Creating an Object Library Under OS/390 Batch

Under OS/390 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 460.

To compile the OS/390 C program WALTER.SOURCE(SUB1) for L-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 EDCLIB,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 OS/390 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 OS/390 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:

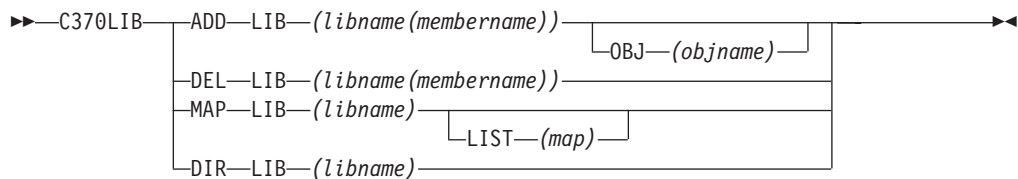
```
//ADDDIR EXEC EDCLIB,
//      LIBRARY='LUCKY13.CXX.OBJMATH',
//      OPARM='ADD MA191',
//      OBJECT='DSNAME=LUCKY13.CXX.OBJ(OBJ191),DISP=SHR'
```

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 and Object Library Under TSO

The Object Library Utility has the following syntax:



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( <i>libname</i> )	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( <i>map</i> )	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 OS/390 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 *OS/390 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 an L-name. An L-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 S-name. A S-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 17. DLL Rename Utility

This chapter describes the DLL Rename utility, which is part of OS/390 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 OS/390 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.

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 OS/390 batch, TSO, and OS/390 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 413, and the *OS/390 C/C++ Programming Guide*.

---

### DLL Redistribution Scenario

Here is an example of a DLL redistribution situation.

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 OS/390 Batch” on page 360 or “Using the DLL Rename Utility under TSO” on page 361.

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 memberse 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 OS/390 C/C++ compiler and bound with the binder.

**Note:** The DLL Rename utility does not support PDSE members built using the OS/390 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 OS/390 batch, input comes from the SYSIN data set.

If you run this utility interactively with TSO, the output goes to your terminal. If you use OS/390 batch or TSO batch, output goes to the first of these data sets for which there is a definition:

- DD:SYSPRINT
- DD:SYSTEM
- 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 56. 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 57. Example of Output from DLLRNAME Utility*

---

## Using the DLL Rename Utility under OS/390 Batch

The following is an example of JCL that you can use to rename a DLL under OS/390 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 OS/390 Language Environment runtime options. If you want to receive messages in Kanji, use the NATLANG option. For detailed information about OS/390 Language Environment runtime options, see the *OS/390 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 <i>newname</i> 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:SYSTEM, 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.

## Example of Renaming a DLL under OS/390 Batch

To rename the DLL described in “DLL Redistribution Scenario” on page 357,

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
/*
```

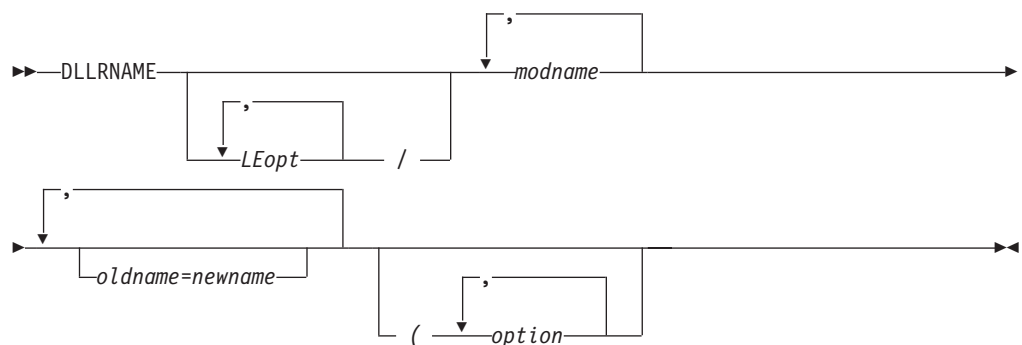
3. Ship PAHPGM1 to your customers with the PAHZIOST DLL.

---

## 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 OS/390 Language Environment runtime options. If you want to receive messages in Kanji, use the NATLANG option. For detailed information about OS/390 Language Environment runtime options, see the *OS/390 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:

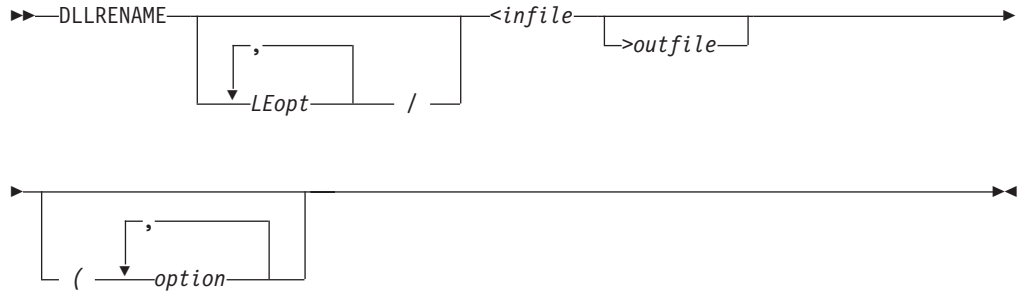
**NOREPORT** Does not generate output, unless you are performing a query.

**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 OS/390 Language Environment runtime options. If you want to receive messages in Kanji, use the NATLANG option. For detailed information about OS/390 Language Environment runtime options, see the *OS/390 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:SYSTEM, 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 <i>newname</i> 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 modnames and lists the DLLs that they load.

The OS/390 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 357, 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 18. Filter Utility

This chapter describes how to use the CXXFILT utility to convert mangled names to demangled names.

When OS/390 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

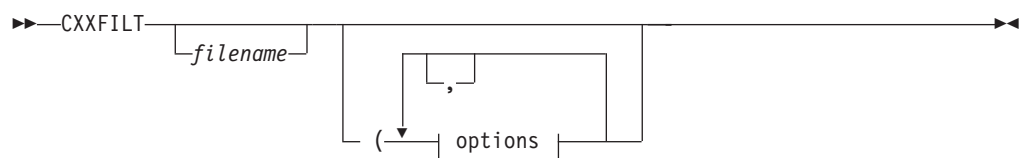
Names that appear within the context of a function name or a member variable. For example, the mangled name `__ls__7ostreamFPCc` is demangled as `ostream::operator<<(const char*)`.

### class names

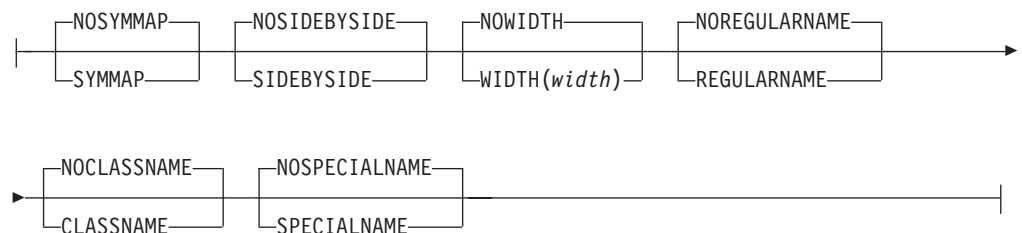
Includes stand-alone class names that do not appear within the context of a function name or a member variable. For example, the stand-alone class name `Q2_1X1Y` is demangled as `X::Y`.

### special names

Special compiler-generated class objects. For example, the compiler-generated symbol name `__vft1X` is demangled as `X::virtual-fn-table-ptr`.



### options:



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__3F00F`, then the SIDEBYSIDE option would produce this result:

```
F00::pr()          pr__3F00Fvxxxx
```

### 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__3F00Fv`.

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 OS/390 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__3F00Fvxxxx  
__1s__7ostreamFPCc  
__vft1X  
/*
```

The output is:

```
F00::pr()      xxxx  
ostream::operator<<(const char*)  
__vft1X
```

C++ Symbol Mapping

demangled	mangled
-----	-----
F00::pr()	pr__3F00Fv
ostream::operator<<(const char*)	__1s__7ostreamFPCs

**Notes:**

1. Because the trailing characters xxxx in the input name pr\_\_3F00Fvxxxx 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__3F00Fv
__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__3F00Fvxxxx
reply< F00::pr()      xxxx
user> __1s__7ostreamFPCc
reply> ostream::operator<<(const char*)
user> __vft1X
reply> __vft1X
user> /*

reply> C++ Symbol Mapping
reply>
reply> demangled                                mangled
reply> -----                                -----
reply> F00::pr()                                pr__3F00Fv
reply> ostream::operator<<(const char*)        __1s__7ostreamFPCs
```

**Notes:**

1. Because the trailing characters xxxx in the input name pr\_\_3F00Fvxxxx 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.
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__3F00Fv
__vft1X
```

For example, enter the following command:

```
cxxfilt FILE1 (SPECIALNAME WIDTH(35) SIDEBYSIDE
```

The above command produces the following output:

```
F00::pr()                pr__3F00Fv
X::virtual-fn-table-ptr   __vft1X
```

CXXFILT terminates when it reads the end-of-file.



## Chapter 19. 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 36. DSECT Utility Options, Abbreviations, and IBM-Supplied Defaults

DSECT Utility Option	Abbreviated Name	IBM Supplied Default
SECT[( <i>name</i> ,...)]	None	SECT(ALL)
BITF0XL   NOBITF0XL	BITF   NOBITF	NOBITF0XL
COMMENT[( <i>delim</i> ,...)]   NOCOMMENT	COM   NOCOM	COMMENT
DEFSUB   NODEFSUB	DEF   NODEF	DEFSUB
EQUATE[( <i>suboptions</i> ,...)]   NOEQUATE	EQU   NOEQU	NOEQUATE
HDRSKIP[( <i>length</i> )]   NOHDRSKIP	HDR( <i>length</i> )   NOHDR	NOHDRSKIP
LOCALE( <i>name</i> )   NOLOCALE	LOC   NOLOC	NOLOCALE
INDENT[( <i>count</i> )]   NOINDENT	IN( <i>count</i> )   NOIN	INDENT(2)
LOWERCASE   NOLOWERCASE	LC   NOLC	LOWERCASE
OPTFILE( <i>filename</i> )   NOOPTFILE	OPTF   NOOPTF	NOOPTFILE
PPCOND[( <i>switch</i> )]   NOPPCOND	PP( <i>switch</i> )   NOPP	NOPPCOND
SEQUENCE   NOSEQUENCE	SEQ   NOSEQ	NOSEQUENCE
UNNAMED   NOUNNAMED	UNN   NOUNN	NOUNNAMED
OUTPUT[( <i>filename</i> )]	OUT[( <i>filename</i> )]	OUTPUT(DD:EDCDSECT)
RECFM[( <i>rcfm</i> )]	None	C/C++ Library defaults
LRECL[( <i>lrec</i> )]	None	C/C++ Library defaults
BLKSIZE[( <i>blksize</i> )]	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.

```
SECTIONNAME 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 unions 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    0F,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 OS/390 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 58. Running the DSECT Utility under OS/390 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.

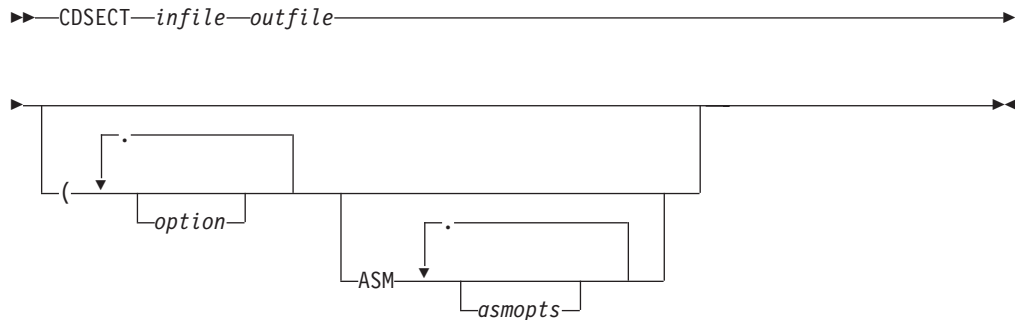
The parameters to the EDCDSECT procedure are:

Table 37. 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:



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 59. 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 20. 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 *OS/390 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 OS/390 C/C++, or to build code set conversions for existing code pages.

The `genxlt` utility runs under OS/390 batch and TSO. The `iconv` utility runs under OS/390 Batch, TSO, and the OS/390 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 OS/390 shell, see *OS/390 UNIX System Services Command Reference* for details on syntax and uses. For more information on the `iconv()` function, refer to the *OS/390 C/C++ Run-Time Library Reference*.

### Under OS/390 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

<b>OUTFILE</b>	The data set name for the output data set
<b>FROMC</b>	The name of the code set in which the input data is encoded
<b>TOC</b>	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 has the record length and record format of the existing data set. If the data set does not exist, and you have not specified the DCB attributes, the record format defaults to variable. If the output data set has a fixed record format, all the records created must have the same length as the output data set. Otherwise the iconv utility will fail. 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.

For more information, refer to the *OS/390 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:

<i>infile</i>	The input data set name.
<i>outfile</i>	The output data set name.
<i>fromcode</i>	The name of the code set in which the input data is encoded.
<i>tocode</i>	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 has the record length and record format of the existing data set. If the data set does not exist, and you have not specified the DCB attributes, the record format defaults to variable. If the output data set has a fixed record format, all the records created must have the same length as the output data set. Otherwise the iconv utility will fail. 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.

For more information, refer to the *OS/390 C/C++ Programming Guide*.

## Under the OS/390 Shell

**iconv** **[--sc]** **--f** *oldset* **--t** *newset* [*file ...*] **iconv** **-l**[**--v**]

**iconv** 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 *OS/390 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 *OS/390 C/C++ Run-Time Library Reference* for more information about the character that is used for converting incorrect characters.

See *OS/390 C/C++ Programming Guide* for a list of code pages that the OS/390 shell supports.

You can use **iconv** to convert singlebyte data or doublebyte data.

### Options:

- |                          |  |
|--------------------------|--|
| <b>-c</b>                | 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. |
| <b>--f</b> <i>oldset</i> | <i>oldset</i> 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.                                     |
| <b>-l</b>                | Lists code sets in the internal table. This option is not supported.   |
| <b>-s</b>                | Suppresses all error messages about faulty encodings.  |
| <b>--t</b> <i>newset</i> | Specifies the destination code set for the output. <i>newset</i> can be either the code set name or a pathname to a file that contains an external code set.                               |
| <b>-v</b>                | Specifies verbose output.  |

## genxlt Utility

Under TSO, you specify the options on the command line. Under OS/390 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 *OS/390 C/C++ Programming Guide*.

## Under OS/390 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:

<code>INFILE</code>	The data set name for the file that contains the character conversion information.
<code>OUTFILE</code>	The data set name for the output file that is to contain the link-edited conversion table.
<code>GOPT</code>	Options for the <code>genxlt</code> 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. :edl

For example:

```
GENXLT GENXLT.SOURCE(EDCUEAEY) GENXLT.LOADLIB(EDCUEAEY) DBCS
```

## localedef Utility

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 `localedef` utility reads the locale definition file and produces a locale object that the locale specific library functions can use. You invoke it by either a JCL procedure or a TSO CLIST, and call the runtime function `setlocale()` to activate it during the application's execution.

The options for the `localedef` utility 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.

Under OS/390 batch, you specify the options on the EXEC PARM and separate them by spaces or commas.

Under TSO, the you sepcify the options on the command line.

<b>CHARMAP</b> ( <i>name</i> )	Specifies the member name of the file that contains the definition of the encoded character set. If you do not specify this option, the <code>localedef</code> utility assumes the encoded character set IBM-1047.  The name that is specified for the CHARMAP is the member name within a partitioned dataset, with the – (dash) sign converted to an @ (at) sign.
<b>FLAG</b> ( <u>W</u>   <u>E</u> )	The FLAG option controls whether <code>localedef</code> issues warning messages. If you specify FLAG(W), <code>localedef</code> issues warning and error messages. If you specify FLAG(E), <code>localedef</code> issues only the error messages.
<b>BLDERR</b>   <u><b>NOBLDERR</b></u>	If you specify the BLDERR option, <code>localedef</code> generates the locale even if it detects errors. If you specify the NOBLDERR option, <code>localedef</code> 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 and code set codes, refer to the *OS/390 C/C++ Programming Guide*. For information on using the `localedef` utility under OS/390 UNIX System Services, refer to the *OS/390 UNIX System Services Command Reference*.

## Under OS/390 Batch

Under OS/390 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 OS/390 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	The data set name for the output partitioned data set and member that is to contain the link-edited locale
LOPT	The options for the <code>localedef</code> utility

For example:

```
//LOCALDEF EXEC PROC=EDCLDEF,
//          INFILE='FRED.LOCALE.SOURCE(EDC$EUEY)',
//          OUTFILE='FRED.LOCALE.LODLIB(EDC$EUEM)',
//          LOPT='CHARMAP(IBM-297)'
```

## 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 OS/390 C/C++ compiler to compile the C source generated
3. Invokes the Linkage Editor to build the locale into a loadable module

The general form of the LOCALDEF CLIST is as follows:

```
▶▶—LOCALDEF—infile—outfile—┐
                               └LOPT(—loptions—)┘▶▶
```

where:

<i>infile</i>	The data set name for the data set that contains the locale definition information
<i>outfile</i>	The data set name for the output partitioned data set and member that is to contain the link-edited locale.
<i>loptions</i>	The options for the localedef utility.

In the following example, the input source is LOCALE.SOURCE(EDC\$EUEY), the output library is LOCALE.LODLIB(EDC\$EUEM) for en\_us.IBM-297, and options are CHARMAP(IBM-297):

```
LOCALEDEF LOCALE.SOURCE(EDC$EUEY) LOCALE.LODLIB(EDC$EUEM) LOPT(CHARMAP(IBM-297))
```

## Under the OS/390 Shell

**localedef** [**-c**] [**-f** *charmap*] [**-i** *sourcefile*] *name*

localedef converts source definitions for locale categories into a format usable by functions and utilities.

OS/390 C/C++ provides localedef and ships it with the compiler. This command requires the installation of OS/390 Language Environment. It also requires the installation of OS/390 UNIX System Services so that you can use the c89 utility. For more information, refer to *OS/390 UNIX System Services Planning*.

The C/C++ compiler also includes the TSO/E command LOCALDEF. The OS/390 shell does not support LOCALDEF. Use >localedef instead.

### Options:

<b>-c</b>	Creates permanent output even if there were warning messages. Normally, localedef does not create permanent output when it has issued warning messages.
-----------	---

- 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. If it contains no slashes, the locale is public, and **localedef** uses the `NLSPATH` environment variable to convert *name* to a full pathname. If *name* contains one or more slashes, `localedef` interprets it as a full pathname of where to store the created definition.



---

## Part 5. OS/390 UNIX Utilities

This part contains information about the OS/390 UNIX System Services utilities.

- “Chapter 21. Archive and Make Utilities” on page 395
- “Chapter 22. BPXBATCH Utility” on page 397



---

## Chapter 21. Archive and Make Utilities

This chapter describes the OS/390 UNIX System Services archive (ar) and make utilities. There are several other useful OS/390 UNIX System Services utilities such as gencat and mkcatdefs. For information on their syntax and use, refer to the *OS/390 UNIX System Services Command Reference*.

The OS/390 Shell and Utilities provide two utilities that you can use to simplify the task of creating and managing OS/390 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 OS/390 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 autocal library mechanism is analogous to the use of Object Libraries for object file for data sets. For more information, see "Chapter 16. Object Library Utility" on page 351.

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 *OS/390 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 OS/390 C/C++ compiler, and the binder, to recompile and rebind an updated application program.

See the *OS/390 UNIX System Services Programming Tools*, and the *OS/390 UNIX System Services Command Reference* for a detailed discussion of the shell `make` utility and how to best take advantage of its function.



---

## Chapter 22. 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 OS/390 C/C++ executable files that reside in hierarchical file system (HFS) files through the OS/390 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.

---

### BPXBATCH Usage

The BPXBATCH program allows you to submit OS/390 batch jobs that run shell commands, scripts, or OS/390 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 OS/390 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 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 OS/390 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 OS/390 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 *OS/390 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 OS/390 C/C++ Programs

Instead of using the prelinker and linkage editor, you can use the binder. See "Chapter 12. Binding OS/390 C/C++ Programs" on page 289 for more information.

This chapter shows how to prelink and link your programs under OS/390 with the OS/390 Language Environment. The OS/390 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 OS/390 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 OS/390 UNIX System Services.

If you do not need to prelink your application, continue to the information in "Linking an Application" on page 408. For information on creating object libraries in OS/390 C++, refer to "Chapter 16. Object Library Utility" on page 351. For information on prelinking and linking object modules under OS/390 UNIX System Services, refer to "Prelinking and Link-Editing under the OS/390 Shell" on page 434.

---

### Prelinking an Application

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

OS/390 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 *OS/390 C/C++ Programming Guide* .

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.

## 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 38. For details on the attributes of specific data sets see “Description of Data Sets Used” on page 460.

Table 38. 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 <sup>2</sup>	Utility Library	Location of prelinker and OS/390 Language Environment runtime data sets
SYSLIB	Library	Secondary input
SYSDEFSD <sup>1</sup>	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 <sup>1</sup>	Input	Obtain additional object modules and load modules
<b>Notes:</b> <sup>1</sup> Required output from the prelinker if you are exporting variables or functions. <sup>2</sup> 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 a OS/390 UNIX System Services environment, 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 438.)



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 (SYSMSG)**

With this DD statement name you provide the prelinker with the information it needs to generate error messages and the prelinker map.

## **Prelinker and OS/390 Language Environment Library (STEPLIB)**

To prelink your program the system must be able to locate the data sets that contain the prelinker and OS/390 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 an OS/390 C/C++ Application” on page 335.

## **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 232.

**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 (SYSRINT)**

If the prelinker encounters problems in its attempt to prelink your program, it generates error messages and places them in the SYSRINT 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 OS/390 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 S-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 OS/390 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 OS/390 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 440.
2. If the long name was found to have a matching short name, the same name is chosen. For example, `D0TOTALS` 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 *OS/390 Language Environment Debugging Guide and Run-Time Messages* 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 39 shows the five data set names and their characteristics.

*Table 39. Data Sets Used for Linking*

ddname	Type	Function
SYSLIN	Input	Primary input data, normally the output of the prelinker
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 <sup>1</sup>	Library	Secondary input
User-specified	Input	Obtain additional object modules and load modules
<b>Notes:</b> <sup>1</sup> 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 introduce it. (See “INCLUDE Control Statement” on page 438.)

## 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 (SYSLOAD)

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 SYSLOAD, 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
- OS/390 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 OS/390 C/C++, refer to “Description of Data Sets Used” on page 460.

## 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 *DFSMS/MVS 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 40.

Table 40. 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 *DFSMS/MVS Program Management*.

The instructions for link-edit processing vary, depending on whether you are running under OS/390 batch or TSO.

**Note:** For information on link-editing modules for interlanguage calls, refer to the *OS/390 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 OS/390 C/C++ programs under OS/390 Language Environment, refer to the *OS/390 Language Environment Programming Guide*.

The Language Environment Library is *dynamic*. This means that many of the functions, such as library functions, available in OS/390 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 OS/390 Language Environment and must be specified as one of the libraries to be searched during autocall.

## Link-Editing Multiple Object Modules

OS/390 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 *OS/390 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 18. Filter Utility” on page 365.

**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 `SYSDDEFSD`. 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 60. 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 60. 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 G0 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 OS/390 Batch and TSO

Figure 61 shows the basic prelinking and linking process for your C or C++ application.

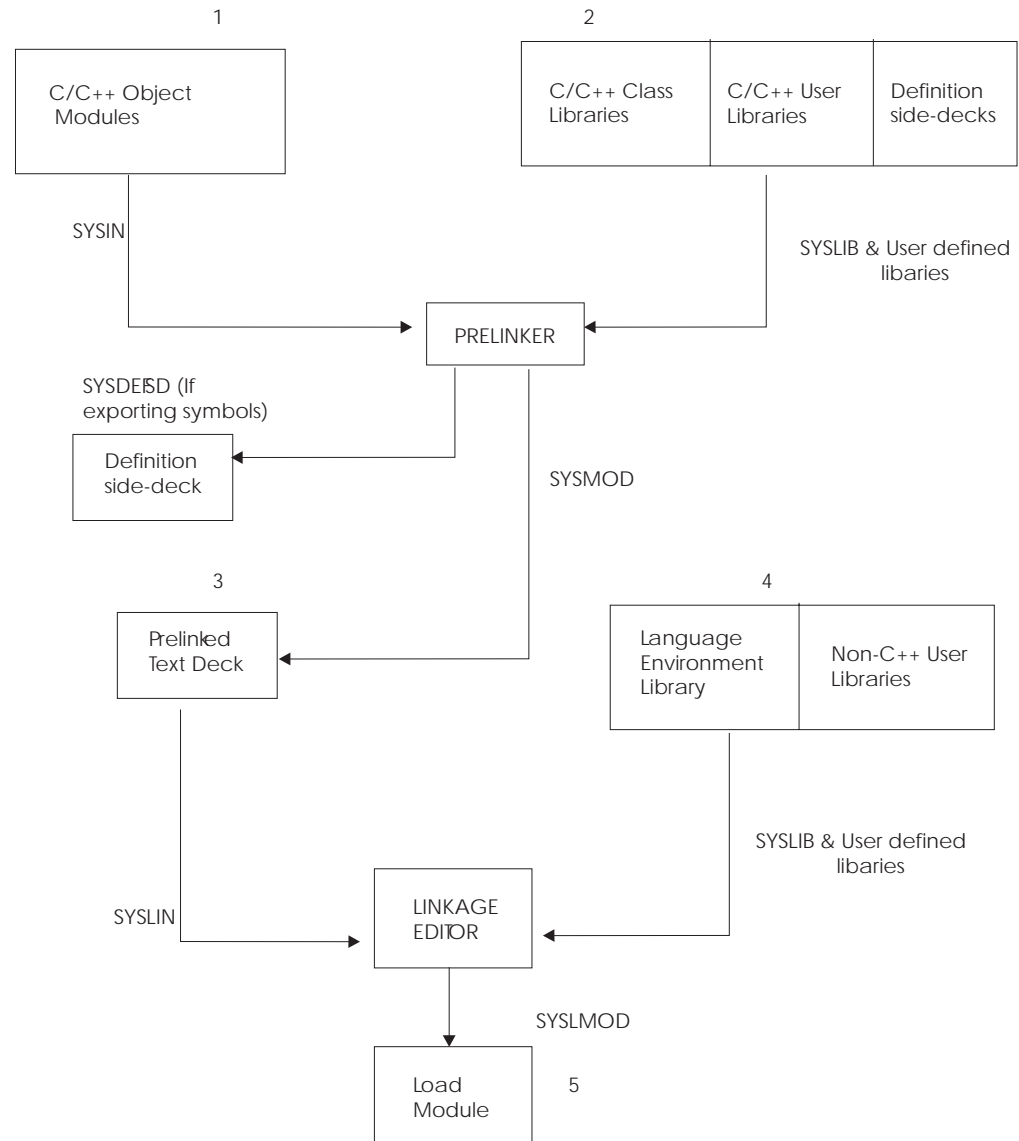


Figure 61. Basic Prelinker and Linkage Editor Processing

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 OS/390 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 retrieve it at any time to run in the job that created it, or in any other job.

---

## OS/390 Language Environment Prelinker Map

When you use the MAP prelinker option, the OS/390 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 63 on page 419, the definition side-deck USERID.DLL.IMPORTS(EXPONLY) contains the record IMPORT DATA 'EXPONLY' year.

The map that is shown in Figure 64 on page 419 was created by compiling the program that is shown in Figure 62. Figure 64 on page 419 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 62. OS/390 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 63. Example of JCL Used to Generate the Example Prelinker Map for an OS/390 C++ program.

```

=====
|                                     |
|                               Prelinker Map 1 |
|                                     |
| CPLINK:5645001 V1 R7 M00 IBM Language Environment 1997/01/20 14:45:28 |
|                                     |
|-----|
| Command Options. . . . . : NONCAL    NOMEMORY ER      DUP      MAP
|                          : NOOMVS    NOUPCASE
|-----|
|                                     |
|                               Object Resolution Warnings 2 |
|                                     |
|-----|

WARNING EDC4015: Unresolved references are detected:
CEESTART @@TRGLOR CEESG003

```

Figure 64. Prelinker Map (Part 1 of 3)

```

=====
|                                     File Map 3                                     |
=====

*ORIGIN  FILE ID  FILE NAME

      P      00001  DD:SYSIN
      A      00002  SHARE.CEE160.SCEECPP(EDC400BA)
      IN      00003  *** DESCRIPTORS ***

*ORIGIN:  P=primary input      PI=primary INCLUDE      SI=secondary I NCLUDE
          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>

=====
|                                     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 64. 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  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()
      D      00001  THIS@INT  this_int_not_in_writable_static
      P              @@TRGLOR  @@TRGLOR
      P              CEESG003  CEESG003
      P      00002  CBCSG003  CBCSG003

*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 64. 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 OS/390 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 OS/390 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 OS/390 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 OS/390 C/C++ Object Library Utility, see “Chapter 16. Object Library Utility” on page 351.

## 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 64 on page 419.

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 440 for a complete description of the RENAME control statement.

Unresolved requests generate error or warning messages to the prelinker map.

## Prelinking and Linking Under OS/390 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 OS/390 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. IBM Supplied Cataloged Procedures and REXX EXECs” on page 457 for more information on REXX EXECs and their uses.

The following example shows the general job control procedure for link-editing a program under OS/390 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
//SYSMOD  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 65. Link-Editing a Program under OS/390 Batch*

You can use one of the following IBM-supplied cataloged procedures that include a prelink and link step to link your C++ program:

CBCCCL   Compile, prelink, and link  
CBCL      Prelink and link  
CBCCCLG   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. IBM Supplied Cataloged Procedures and REXX EXECs” on page 457.

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 445, for linkage editor options see “Linkage Editor Options” on page 447.

### Example of JCL to Prelink and Link

Figure 66 on page 426 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
//SYSMMSG DD DSN=CEE.SCEMSGP(EDCPMSG),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 66. Creating a Load Module under OS/390 Batch

**Note:** For an OS/390 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 OS/390 batch by submitting your own JCL to the operating system or by using the IBM cataloged procedures. See “Appendix D. IBM Supplied Cataloged Procedures and REXX EXECs” on page 457 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 DSN=CEE.SCEELKED,DISP=SHR
// DD DSN=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 67 on page 427.

```

:
//SYSLIN DD      DSNAME=&&GOFIL,DISP=(SHR,DELETE)
//          DD      *
//          INCLUDE ddname(member)
//          LIBRARY ADDLIB(CPGM10)
/*

```

*Figure 67. Linkage Editor Control Statements*

As 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 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 OS/390 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:

```

:
//SYSLIN DD      DSNAME=&&GOFIL,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 OS/390 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 “Appendix I. Other OS/390 C Utilities” on page 603.

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.

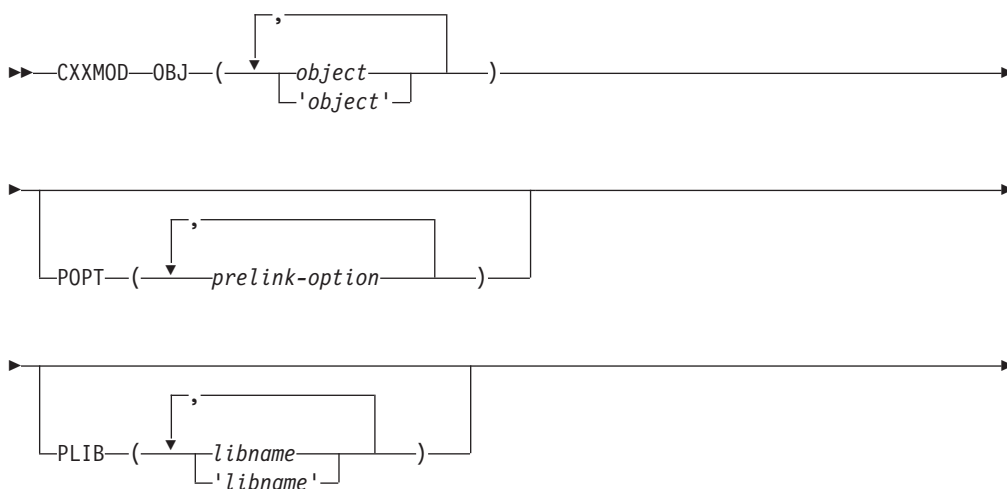
Any linkage editor processing beyond the basic processing described above must be specified by linkage editor control statements in the primary input. The IBM-supplied catalog procedures and REXX EXECs use the DLL versions of the IBM-supplied class libraries by default.

To link-edit your OS/390 C program under TSO, use either the CXXMOD, CMOD, or the LINK command. It is recommended that you use CXXMOD, particularly when linking OS/390 C and OS/390 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 “Appendix I. Other OS/390 C Utilities” on page 603.

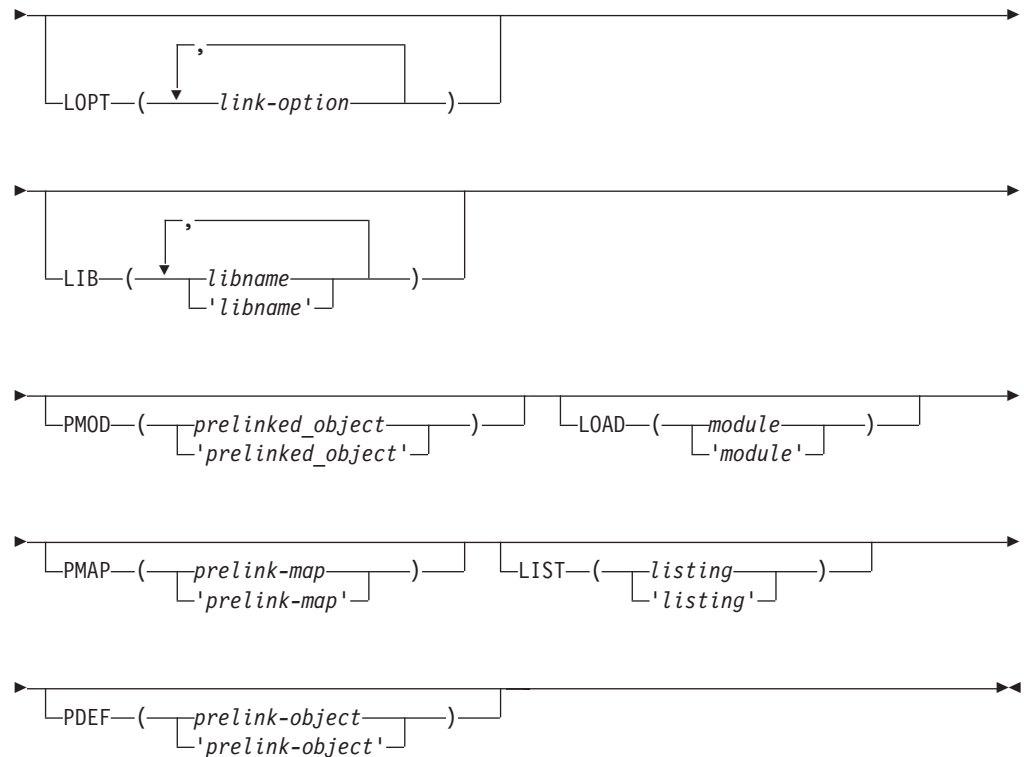
### Prelinking and Linking under TSO

This section describes how to prelink and link your OS/390 C++ or OS/390 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

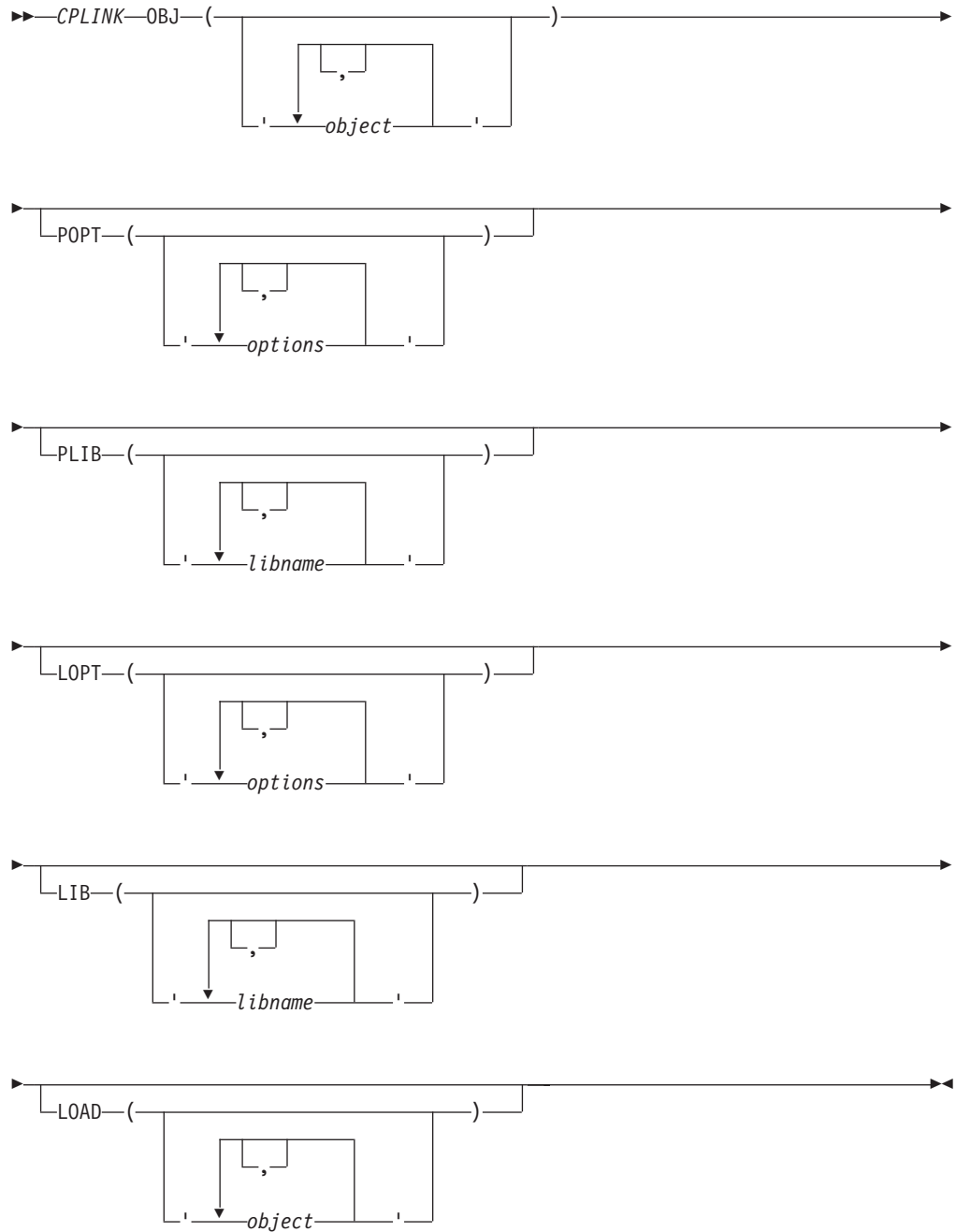
In the following example, the user prefix is RYAN and the input object module members MAIN and FN are in the PDS called 'RYAN.ACCOUNT.OBJ'. A prelink map is to be generated and placed in 'RYAN.ACCOUNT.MAP(SALES)'. The load module will be placed in a PDS member called 'GROUP.ACCOUNT.LOAD(SALES)'. The 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 OS/390 Language Environment stub library and the partitioned data set (library) SALES LIB 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	<p>specifies a string of prelink options.</p> <p>The prelinker options available for CPLINK are the same as for OS/390 batch. For example, if you want the prelinker to use the MAP option, specify the following:</p> <pre>CPLINK file name POPT('MAP')..</pre> <p>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.</p>
PLIB	<p>specifies the library names that the prelinker uses for the automatic library call facility.</p>
LOPT	<p>specifies a string of linkage editor options.</p> <p>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:</p> <pre>CPLINK file name POPT('MAP') LOPT('NOMAP...')</pre>
LIB	<p>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.</p>
LOAD	<p>specifies an output data set name.</p> <p>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 <i>DFSMS/MVS Program Management</i>.</p>

## 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.

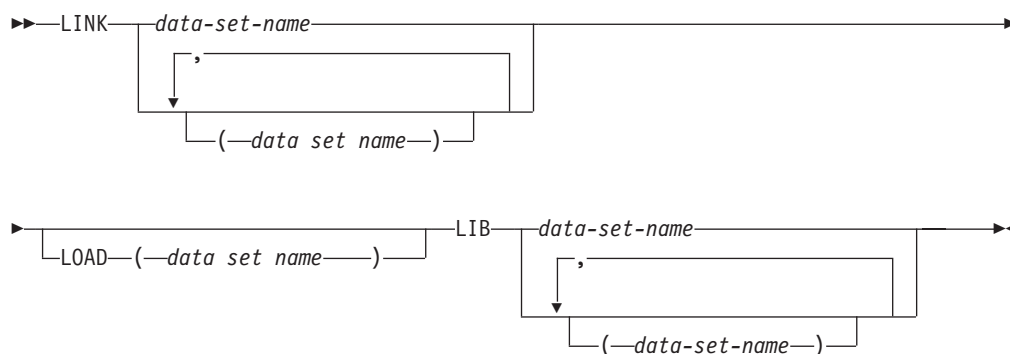
```
CPLINK OBJ('C.OBJ(INCCOMM)')
```

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).

Figure 68. Example of Prelinking under OS/390 Batch

Figure 69. Example of Prelinking under TSO

The general form of the TSO LINK command is:



A request coded this way searches CEE.SCEELKED and SALES LIB.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 410.

For more information about using the TSO command LINK, see *OS/390 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 OS/390 Shell

You can prelink and link your application under the shell by using 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 445.

## Using your JCL

The example JCL in Figure 70 links to an archive library and to OS/390 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
//SYMSGSGS DD DISP=SHR,DSN=CEE.SCEMSGGP(EDCPMSGGE)
//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 70. Using OMVS to prelink and link*

The JCL in Figure 70 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 CEER00TA 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(MEMH00)

*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                 |
=====

      *REASON  FILE ID  OUTPUT
                        ESD NAME  INPUT NAME

      P        00001  CEESTART  CEESTART
      P        00001  CEEMAIN   CEEMAIN
      D        00001  MAIN      main
      D        00003  FOO       foo
      D        00003  GOO       goo
      D        00004  HOO       hoo
      P        00002  CEESG003  CEESG003
      P        00002  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 71. 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++, cc, and the {\_STEPS} environment variable, see *OS/390 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 445.

---

## 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

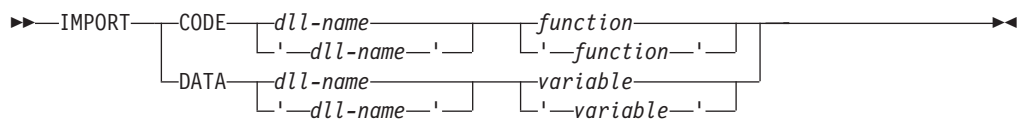
specify it as follows:

'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:



### 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 445.

### 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.

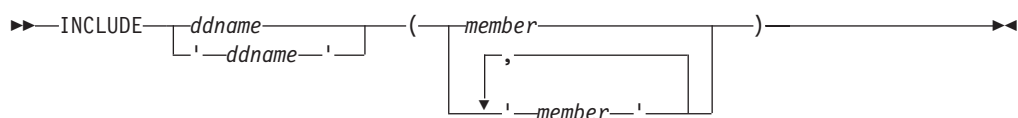
### function

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:



**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 OS/390 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.

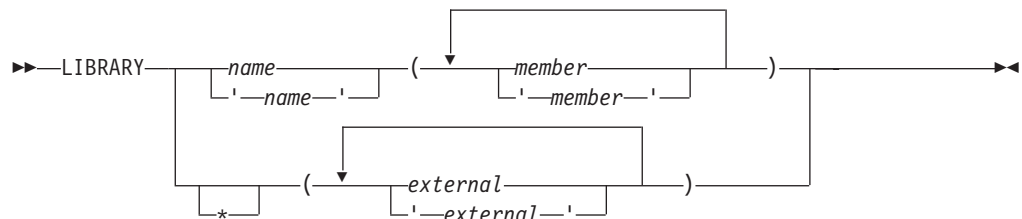
**Note:** The INCLUDE control statement is removed and not placed in the prelinker output object module; the system linkage editor does not see the INCLUDE control statement.

For more information on the linkage editor, refer to *DFSMS/MVS Program Management*.

## LIBRARY Control Statement

The LIBRARY control statement has the following syntax:

### NOOMVS



### OMVS



#### *name*

the name of a DD that defines a library, under OS/390. 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 OS/390, 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 SEARCH ➤➤
```

Diagram illustrating the syntax for the RENAME control statement in NOOMVS mode. The statement is enclosed in double chevrons (➤➤) at the beginning and end. It consists of the keyword RENAME, followed by a bracketed section containing the long name (with a continuation line below it: '—long name—'). This is followed by a bracketed section containing the short name, and finally a bracketed section containing the optional SEARCH parameter.

### OMVS

```
➤➤ RENAME long name short name ➤➤
```

Diagram illustrating the syntax for the RENAME control statement in OMVS mode. The statement is enclosed in double chevrons (➤➤) at the beginning and end. It consists of the keyword RENAME, followed by a bracketed section containing the long name (with a continuation line below it: '—long name—'), and finally a bracketed section containing the short 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 S-Names” on page 407 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 S-Names” on page 407.
  - 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 in OS/390 C/C++, see *OS/390 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 OS/390 system link procedures such as IEWL, you must first call the prelinker.

The OS/390 UNIX System Services features require that all OS/390 UNIX System Services C/C++ application programs be reentrant. If you are using the c89 utility, it automatically invokes the OS/390 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 OS/390 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 OS/390 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 OS/390 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 *OS/390 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 OS/390 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 *OS/390 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 *OS/390 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 *OS/390 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 *OS/390 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 OS/390 Language Environment. For more information on using the OS/390 Language Environment Prelinker, see “Appendix A. Prelinking and Linking OS/390 C/C++ Programs” on page 403.

---

### Prelinker Options

The following section describes the prelink options available in OS/390 C/C++ by using OS/390 Language Environment.

#### DLLNAME(dll-name)

DLLNAME specifies the DLL name that appears on generated IMPORT control statements, described in “IMPORT Control Statement” on page 438. 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

The MAP option specifies that the prelinker should generate a prelink listing. See “OS/390 Language Environment Prelinker Map” on page 418 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 process 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 0E 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 OS/390 C/C++. For example, in COBOL and assembler, all external names are in uppercase. So, if the names are coded in lowercase in the OS/390 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 *DFSMS/MVS Program Management* manuals.



---

## Appendix C. Diagnosing Problems

This appendix tells you how to diagnose failures in the OS/390 C/C++ compiler.

---

### 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 OS/390 C/C++ compiler, its utility components, or the OS/390 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 OS/390 C/C++-specific messages from the runtime environment.
  - CEE - for language-independent messages from the common execution environment (CEE) library component of OS/390 Language Environment.
  - IBM, PLI, IGZ - for language-specific messages from OS/390 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 OS/390 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 *OS/390 MVS System Messages* volumes which accompany the OS/390 operating system. For example, *OS/390 MVS System Messages, Vol 1 (ABA-ASA)*.

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 bottom of the object file. If your program is written in OS/390 C, there is always a comment about the status of the `OPTIMIZE` option, whether you specify it or not.

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 OS/390 Program Directory for more details.
8. Use the Debug Tool, dbx (for OS/390 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 *OS/390 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 IBM OS/390 C/C++ compiler, the following may cause problems:
  - The source code does not conform to 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 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 OS/390 C/C++ Programs” on page 403.

---

## 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 OS/390 Language Environment, for prelink-time and run time diagnosis and debugging errors you should use *OS/390 Language Environment Customization* and the *OS/390 Language Environment Debugging Guide and Run-Time Messages* and for bind time and link-time diagnosis refer to *DFSMS/MVS Program Management*.

After you identify the failure, you can write a small test case that re-creates the problem. See the file CBC.SCBCDOC (APAR), for details on constructing a test case from a failing program.

## 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 OS/390 C++ does not allow missing prototypes.

2. Compile your program with either the CHECKOUT 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 if you send source files from MVS to VM or from VM to MVS. This can also happen in a source file that was meant to be compiled with margins but was actually compiled without margins or different margins (available in OS/390 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 *OS/390 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++ . If the problem is corrected, contact IBM service and be prepared to supply the relevant source (PPONLY) and IPA object modules.
  - Perform a binary search for the invalid IPA object module. To do this, compile one half of your source files with NOIPA, 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 329.

## 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 413, or the *OS/390 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 *OS/390 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 OS/390 Language Environment, you can find diagnostics and debugging information in *DFSMS/MVS Program Management*.
2. If you have a link time error while working with common execution environment (CEE) library component of OS/390 Language Environment, you can find diagnostics and debugging information for link-time errors in *OS/390 Language Environment Customization* and *OS/390 Language Environment Debugging Guide and Run-Time Messages*.

## 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. If XREF is specified, the listing also contains reference, definition and modification information.
	For C++ compile and IPA Link, external symbol cross reference listing.
EXPMAC	Macro expansions with the original source.
LIST	Listing of the pseudoassembly listing produced by the compiler.
OFFSET	Offset addresses of functions in the listing.
PPONLY	Completely expanded OS/390 C or OS/390 C++ source code, by activating the preprocessor (PP) only. The output shows, for example, all the <code>#include</code> and <code>#define</code> directives.
SHOWINC	All included text in the listing.
SOURCE	Listing of the source file.

XREF	For C compile, cross reference listing with reference, definition, and modification information.  For C++ compile, 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.
GONUMBER	Generates line number information that corresponds to input source files.
FLAG	Specifies the minimum message severity level that you want returned from the compiler.
TEST	To get information about the contents of variables at the point of the error, and to enable the use of the Debug Tool.
CHECKOUT	Indication of possible programming errors.
INLINE	(C only) Inline Summary and Detailed Call Structure Reports. (Specify with the REPORT suboption.)
INLRPT	(C++) only Generates a report on status of functions that were inlined. The OPTIMIZE option must also be specified.
INFO	(C++ only) Indication of possible programming errors.
SRCMSG	Adds the corresponding source code lines to the diagnostic messages that are written to <i>stderr</i> .

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. *OS/390 Language Environment Customization and OS/390 Language Environment Debugging Guide and Run-Time Messages* describe diagnostics and debugging information for runtime errors when executing with OS/390 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 OS/390 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. *OS/390 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.

---

## 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 OS/390 Program Directory that is applicable to your environment.
2. Consult the PSP bucket as described on page 7 on page 450.

If you still cannot solve the problem, develop a keyword string and contact your IBM Support Center.

You may need to reinstall the OS/390 C/C++ product by using the procedure that is documented in the OS/390 Program Directory. This procedure is tested for each product release and successfully installs the product.



---

## Appendix D. IBM Supplied Cataloged Procedures and REXX EXECs

This appendix describes the REXX EXECs (TSO) and cataloged procedures that the OS/390 C/C++ compiler provides in conjunction with OS/390 Language Environment, to call the various OS/390 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 OS/390 Language Environment provides, and on the cataloged procedures that do not contain a compile step, see *OS/390 Language Environment Programming Guide*.

For a description of CXXBIND see "Chapter 12. Binding OS/390 C/C++ Programs" on page 289. For a description of CXXMOD see "Prelinking and Linking under TSO" on page 428. For a list of the old syntax REXX EXECs, see "Appendix I. Other OS/390 C Utilities" on page 603.

	Name	Task Description
REXX EXECs for OS/390 C and OS/390 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 OS/390 C and OS/390 C++	EDCDLLRN	Rename DLLs with the DLLRNAME utility
	EDCLIB	Maintain an object library
REXX EXECs for OS/390 C	CC	Compile (new syntax - recommended approach)
	CDSECT	Run DSECT utility
	GENXLT	Generate a translate table
	ICONV	Run the character conversion utility
	LOCALEDEF	Produce a locale object

	Name	Task Description
Cataloged Procedures for OS/390 C	CEEWG	Run
	CEEWL	Link
	CEEWLG	Link and run
	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
	EDCGNXL	Generate a translate table
	EDCICONV	Run the character conversion utility
	EDCLDEF	Produce a locale object
REXX EXECs for OS/390 C++	CXX	Compile under TSO
cataloged procedures for OS/390 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

## 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 OS/390 C/C++ compiler and OS/390 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 procedures other than EDCI or CBCI, the input OS/390 C/C++ source file name or PDS name of source files. For EDCI or CBCI, primary input (object records).  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	LE runtime (Go step) options and parameters: If two contradictory options are specified, the last is accepted and the first ignored.
CRUN	Compile step execution runtime parameters for the OS/390 C compiler.
IRUN	IPA Link step runtime parameters: for the OS/390 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 in OS/390 C, and EDCLIB in OS/390 C++.

Parameter	Description
MEMBER	Member of the library to contain the object module. An example is MEMBER='MYPROG'. In OS/390 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 *OS/390 C/C++ Programming Guide* for more information about the attributes that are used when opening different types of files.

Table 41. 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 <sup>1</sup>	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			
SYSMSGGS				X			X
SYSOUT	X	X		X			X
SYSPRINT	X			X	X	X	X
SYSUTx	X	X			X (SYSUT1)		
IPACNTL		X					

**Note:** <sup>1</sup> Optional data sets, if the compiler and runtime library are installed in the LPA or ELPA. To save resources (especially in OS/390 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 42. 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.



Table 42. 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	SYSLIB	<p>For a C++, C, or IPA compilation, the data set for OS/390 C/C++ system header files for a source program.</p> <p>SYSLIB must be a PDS (DSORG=P0) and RECFM=VS, V, VB, VBS, F, FB LRECL≤32760.</p> <p>For more information on searching system header files, see “SEARCH   NOSEARCH” on page 140.</p>
COMPILE	SYSLIN	<p>Data set for object module.</p> <p>One of the following:</p> <ul style="list-style-type: none"> <li>• RECFM=F or FS</li> <li>• RECFM=FB or FBS.</li> </ul> <p>It can be a PDS.</p>
COMPILE	SYSOUT	<p>Data set for displaying compiler error messages.</p> <p>LRECL=137, RECFM=VBA, BLKSIZE=882. (Defaults for SYSOUT=*).</p>
COMPILE	STEPLIB	<p>Data set for OS/390 C/C++ compiler runtime library modules.</p> <p>STEPLIB must be a PDS (DSORG=P0) with RECFM=U, BLKSIZE≤32760.</p>
COMPILE	SYSCPRT	<p>Output data set for compiler listing.</p> <p>LRECL=137, RECFM=VBA, BLKSIZE=882 (default for SYSOUT=*)</p>
COMPILE	SYSUT1 and SYSUT4	<p>Work data sets.</p> <p>LRECL=80 and RECFM=F or FB or FBS.</p>
COMPILE	SYSUT5, SYSUT6, SYSUT7, SYSUT8, and SYSUT14	<p>Work data sets.</p> <p>LRECL=3200, RECFM=FB, and BLKSIZE=3200*n (where n is an integer value).</p>
COMPILE	SYSUT9	<p>Work data set.</p> <p>LRECL=137, RECFM=VB, and BLKSIZE=137*n (where n is an integer value) in OS/390 C, or 882 in OS/390 C++.</p>
COMPILE	SYSUT10	<p>PPONLY output data set.</p> <p>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.</p>

Table 42. 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	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).
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 140.
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: <ul style="list-style-type: none"> <li>Object module libraries. These can be PDSs (DSORG=P0) or PDSEs, with attributes RECFM=F or RECFM=FB, and LRECL=80.</li> <li>Load module libraries. These must be PDSs (DSORG=P0) with attributes RECFM=U and BLKSIZE≤32760.</li> </ul> SYSLIB must be cataloged.
IPA Link	SYSLIN	Data set for object module.  One of the following: <ul style="list-style-type: none"> <li>RECFM=F or FS</li> <li>RECFM=FB or FBS</li> </ul> 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 OS/390 C/C++ compiler/runtime library modules.  STEPLIB must be a PDS (DSORG=P0) with RECFM=U, BLKSIZE≤32760.

Table 42. 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	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.
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	SYSPRINT	Data set for listing of binder diagnostic messages.  LRECL=137, RECFM=VBA, BLKSIZE=882. (Default attributes for SYSOUT=*)).
LKED	SYSLIB	Data set for OS/390 C/C++ autocall library.  SYSLIB must be a PDS (DSORG=P0) and have the attributes RECFM=U and BLKSIZE≤32760.
PLKED	STEPLIB	Data set containing prelink utility modules.  STEPLIB must be a PDS (DSORG=P0) 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.

Table 42. 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	SYSLIB	Data set for prelinkage automatic call library.  SYSLIB must be cataloged <i>and</i> LRECL=80 and RECFM=F or FB or FBS. DSORG=P0
PLKED	SYSMOD	Data set for output of the prelink utility  LRECL=80 and RECFM=F or FB or FBS.
PLKED	SYSMSG	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 OS/390 C/C++ autocal library.  SYSLIB must be a PDS (DSORG=P0) <i>and</i> have the attributes RECFM=U and BLKSIZE≤32760.
LKED	SYSLIN	Primary input data set for linkage editor  One of the following: <ul style="list-style-type: none"> <li>• RECFM=F or FS</li> <li>• RECFM=FB or FBS</li> </ul>
LKED	SYSLOAD	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: <ul style="list-style-type: none"> <li>• LRECL=121, and RECFM=FA</li> <li>• LRECL=121, RECFM=FBA, and BLKSIZE=121*n (where n is less than or equal to 40).</li> </ul>
LKED	SYSUT1	Work data set.  The data set attributes will be supplied by the linkage editor.

Table 42. 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	STEPLIB	Runtime libraries.  STEPLIB must be a PDS (DSORG=P0) <i>and</i> have the attributes RECFM=U and BLKSIZE≤32760.
GO	CEEDUMP	Data set for error messages generated by Language Environment Dump Services. CEEDUMP must be a sequential data set <i>and</i> 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 be 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 72. Example Compilation for OS/390 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'
```

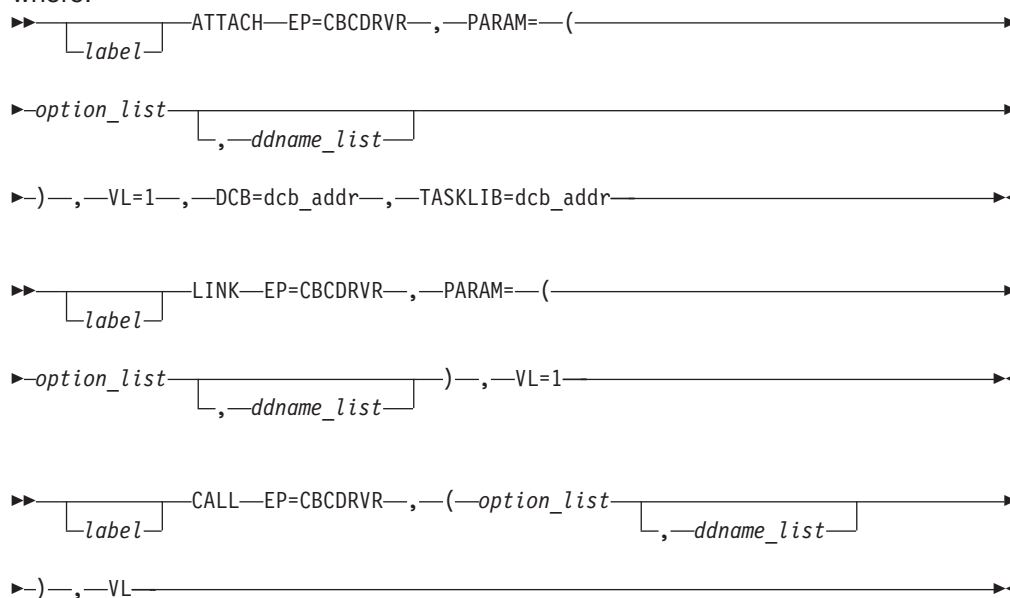
*Figure 73. Example Compilation for OS/390 C++ Using CBCC*

## Appendix E. Using Assembler Macros

To compile your C/C++ source program dynamically under OS/390, 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 *OS/390 Information Roadmap*.

The following is the syntax of each macro instruction:

where:



EP	Specifies the symbolic name of the OS/390 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 OS/390 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
- SYSMGS - 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.

<b>VL or VL=1</b>	Specifies that the sign bit is to be set to 1 in the last fullword of the address parameter.
<b>DCB</b>	Specifies the address of the control block for the partitioned data set that contains the compiler.
<b>TASKLIB</b>	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.



```

*****
*
* 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 74. 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 74. 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
/*   CBC3UAAQ which is stored in MYID.ATTACHDD.LOAD
/*
/*   This example uses the OS/390 Language Environment Library
/*-----
//G001001B EXEC PGM=CBC3UAAQ
//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=...
//SYSTEM 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
/*-----

```

Note that the sharing of resources between attached programs is not supported.

---

## CBC3UAAR

```
*****
*
* 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 76. Using the Assembler LINK Macro (Part 1 of 2)*

```

*
*   Constant and save area
*
*   This macro will compile for the OS/390 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 76. 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 OS/390 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=*
//SYSTEM 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
/*-----

```

---

## 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=CBCCRVR
          LR 15,0
          CALL (15),(OPTIONS,DDNAMES),VL
          L 13,4(,13)
          LM 14,12,12(13)
          SR 15,15
          BR 14
```

Figure 78. 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 78. 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 OS/390 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=...
//SYSTEM 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 79. JCL for the Assembler CALL Macro

---

## Appendix F. OS/390 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 43. Return Codes from Compilation of a OS/390 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 OS/390 Language Environment Release 3 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

---

**CBC1000**    **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)

---

**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 function'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 "\n" to continue the string on the next line. The "\n" 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 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 other 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 missing 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 "sobj.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 "sobj.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 "\n" to continue the string on the next line. The "\n" 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 "\n" to continue it on the next line. The "\n" 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 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 be 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 from &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 been 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 noline directive. (where &1 is a function name and type.)

**Explanation:** Only functions defined in this compilation can be used in pragma noline directive.

**User Response:** Remove the pragma noline directive 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 <someobj.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 preprocessor 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.

---

**CBC1811**     **64-bit portability: possible loss of digits through conversion of long type into int type.**

**Explanation:** A long type is assigned into an int type which may cause truncation in 64-bit mode.

**User Response:** Check the possible value ranges of the long type or change the assignment from an int to a long type.

---

**CBC1812**     **64-bit portability: possible change of result through conversion of int type into long type.**

**Explanation:** An int type is assigned into a long type which may cause unexpected results in 64-bit mode.

**User Response:** Check for possible sign extension of int type into long type.

---

**CBC1813**     **64-bit portability: possible truncation of pointer through conversion of pointer type into int type.**

**Explanation:** A pointer type is assigned into an int type leading to loss of the high-order bytes of the pointer in 64-bit mode.

**User Response:** Use a long type to hold a pointer type.

---

**CBC1814**     **64-bit portability: possible incorrect pointer through conversion of int type into pointer.**

**Explanation:** An int type is assigned into a pointer type leading to a possibly invalid address in 64-bit mode.

**User Response:** Use a long type to hold the address.

---

**CBC1815**     **64-bit portability: possible change of constant value through conversion into long type.**

**Explanation:** A constant is assigned into long type leading to possible change of value in 64-bit mode.

**User Response:** Check the possible value ranges of the constant when stored in a long type.

---

**CBC1816**     **64-bit portability: constant which selected unsigned long int in 32-bit mode may select long int in 64-bit mode.**

**Explanation:** A constant selected unsigned long int in 32-bit mode may fit a long int in 64-bit mode.

**User Response:** Check use of constant for possible change in usual arithmetic conversion rule as it propagates through expressions.

---

**CBC1817**     **64-bit portability: constant which will overflow in 32-bit mode may select unsigned long int or long int in 64-bit mode.**

**Explanation:** A constant is larger than `UINT_MAX` but smaller than `ULONGLONG_MAX` will overflow in 32-bit mode, but be acceptable in an unsigned long or signed long in 64-bit mode.

**User Response:** Make sure you intend this constant to be acceptable in 64-bit mode.

---

**CBC1818**     **Incompatible specifications for options -qarch and -q&1 (or environment variable OBJECT\_MODE)**

**Explanation:** The values specified for the -qarch and the -q32/64 options (or OBJECT MODE) are not compatible.

**User Response:** Change option values.

---

**CBC1819 Incompatible specifications for options -qtune and -q&1 (or environment variable OBJECT\_MODE)**

**Explanation:** The values specified for the -qtune and the -q32/64 options (or OBJECT\_MODE) are not compatible.

**User Response:** Change option values.

---

**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.

---

**CBC1821 64 bit code generation is not supported.**

**Explanation:** The C++ compiler does not support generation of 64 bit code.

**User Response:** Remove the -q64 option from the command line.

---

**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.

---

**CBC1828 Long type bitfields may change behaviour in future 64-bit mode. Long type bitfields currently default to int.**

**Explanation:** Long type bit fields are not currently supported in 64-bit mode. Future ABI may change.

**User Response:** Make sure that the bitfield is not larger than the length of an integer.

---

**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.

---

**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 preprocessor 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     Redclaration 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. Unsuffix 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". (where "&1" is a type.)**

**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. :p.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. :p.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. :p.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. :p.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. :p.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. :p.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.

---

**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. (where &1 is an identifier name. &2 is a line number. &3 is a file name.)**

**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. (where &1 is an identifier name.)**

**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      Redclaration 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**     **The variable '&1' is referenced before being initialized.**

**Explanation:** Because the variable has not been initialized, its value is undefined. The results of using an undefined variable are unpredictable.

**User Response:** Initialize the variable before its 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 \* 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 \* 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:** An 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".  
(where &1 is a C name or keyword)**

**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 and :href refid=prag370. of this book.

**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(locale name), the suboption 'locale name' 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 from &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 been 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. (where &1 is a function or type name)**

**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. (where &1 is a function or variable name)**

**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. (where &1 is a variable name)**

**Explanation:** Variables with NORENT option cannot be exported.

---

**CBC3683**     **"&1" will not be exported because it does not have external storage class. (where &1 is a variable or function name)**

**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 invalid 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.

---

**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. (where "&1" is the function specifier that was duplicated.)**

---

**CBC3713**    **Keyword "&1" is not allowed. (where "&1" is a keyword which is not allowed in this context.)**

---

**CBC3714**    **#include searching for file &1. (where The preprocessor is searching for the specified include file.)**

---

**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 is 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.

---

**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.

---

**CBC3807     Incompatible specifications for options -qarch and -q&1 (or environment variable OBJECT\_MODE)**

**Explanation:** The values specified for the -qarch and the -q32/64 options (or OBJECT MODE) are not compatible.

**User Response:** Change option values.

---

**CBC3808     Long type bitfields may change behaviour in future 64-bit mode. Long type bitfields currently default to int.**

**Explanation:** Long type bitfields currently default to int, but may change in future.

**User Response:** Try to use int type bitfields.

---

**CBC3809     Incompatible specifications for options -qtune and -q&1 (or environment variable OBJECT\_MODE)**

**Explanation:** The values specified for the -qtune and the -q32/64 options (or OBJECT MODE) are not compatible.

**User Response:** Change option values.

---

**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:** Specify the second option explicitly.

---

**CBC3812     Option FLOAT(BFP) may cause slow execution time when use 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 use 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.

---

**CBC5001     Internal compiler error at procedure &1.**

**Explanation:** An error occurred during compilation. See the C/370 Diagnosis Guide for a description of what to do.

---

**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 General message \*CHAR 100)**

**Explanation:** General error message.

**User Response:** General error message.

---

**CBC5004     Object cannot be declared as a pointer to a function.**

**User Response:** Remove or change the declaration.

---

**CBC5005     Parameter cannot be a pointer to a function.**

**Explanation:** The address of a function is being passed as a function parameter. Unpredictable results can occur depending on how this address is used.

**User Response:** If necessary, remove the function pointer parameter.

---

**CBC5006**     **A function call cannot be made using a function pointer.**

**User Response:** Remove the call.

---

**CBC5007**     **The reference to external name &1 cannot be resolved.**

**Explanation:** It is invalid to reference a function that is defined in another file, in an expression other than a function call.

**User Response:** Remove the invalid reference.

---

**CBC5008**     **Function name &1 does not have a length of four bytes.**

**Explanation:** Function having storage class extern must have four-character names.

**User Response:** Change the function, and all references, to have a four-character name or use #pragma map to map the name to a four-character name.

---

**CBC5009**     **Total program size cannot exceed 4096 bytes. Total program size is &1 bytes.**

**User Response:** Reorganize (split, delete, etc.) the source program so that the total size does not exceed 4096 bytes.

---

**CBC5010**     **Storage cannot be statically initialized to the address of the function &1.**

**Explanation:** It is invalid to statically initialize storage with the address of a function that is not defined in the current compilation.

**User Response:** Correct the invalid initialization.

---

**CBC5011**     **Variable cannot be defined externally.**

**Explanation:** It is invalid to declare a variable having the storage class extern.

**User Response:** Correct the invalid declaration.

---

**CBC5013**     **Static function &1 cannot have a #pragma linkage.**

**Explanation:** #pragma linkage cannot be applied to static functions.

**User Response:** Remove the #pragma linkage statement.

---

**CBC5014**     **Function &1 cannot have a #pragma linkage type N.**

**Explanation:** #pragma linkage type N cannot apply to defined C functions.

**User Response:** Remove the #pragma linkage statement.

---

**CBC5015**     **Parameter of type struct might cause stack space to overflow.**

**Explanation:** A function parameter of type struct adds to the size of stack and can potentially cause the stack to overflow.

**User Response:** Pass the structure by address using the address (&) operator to reduce stack size.

---

**CBC5016**     **Size of automatic storage cannot exceed 4096 bytes. Function &1 has &2 bytes of automatic storage.**

**Explanation:** The 4096 byte limit of automatic storage has been exceeded in function &1.

**User Response:** Reduce the number and size of the auto variables in the function.

---

**CBC5017**     **Total static size of &1 bytes exceeds the allowed maximum of 4096 bytes.**

**Explanation:** The 4096 byte limit of static storage has been exceeded.

**User Response:** Reduce the number and size of the static variables.

---

**CBC5018**     **'main' cannot be used as a function name.**

**Explanation:** The function 'main' is not supported in this environment.

**User Response:** Remove or rename the function 'main'.

---

**CBC5031**     **Unable to open &1.**

**User Response:** Ensure file exists.

---

**CBC5032**     **Unable to read &1. (where &1 file \*CHAR 100)**

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

---

**CBC5033**     **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.



---

**CBC5034      Read/write pointer initialization of read-only object &1 is not valid.**

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

---

**CBC5051      Function specified 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 specified is (or grows) too large to be inlined:**

**Explanation:** This occurs when a function is too large to be itself inlined into another function.

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

---

**CBC5053      Some calls to function specified 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 specified increased to over**

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

**User Response:** If feasible to do so, prevent inlining functions which have large auto storage on.

---

**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 of low severity has occurred.

**User Response:** Contact IBM support and/or compile with a different OPT level.

---

**CBC5104      Internal error while compiling function &1. &2. Compilation ended.**

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

**User Response:** Contact IBM support, prepared to quote the text of this message.

---

**CBC5105      Constant table overflow compiling function &1. Compilation ended.**

**Explanation:** 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 ended.**

**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 if desired.

---

**CBC5110 Loop too complex in function &1. Some optimizations not performed.**

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

**User Response:** None

---

**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 if desired.

---

**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 if desired.

---

**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 if desired.

---

**CBC5114 Internal error while compiling function &1. &2.**

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

**User Response:** Contact IBM support and/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.**

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

---

**CBC5120 Too many symbols in function &1; number of dictionary entries exceeds &2.**

**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.**

**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.

---

**CBC5124**    **Source line &1 creates a dependency on general purpose register 12.**

**Explanation:** The indicated source line has caused code to be generated which assumes that general purpose register 12 is loaded with the address of the LE run-time CAA. The code is using a function which is not supported in a stand-alone environment. &1 is the source line number. The code is generated but may cause unpredictable results.

**User Response:** Ensure that only functions which are supported in a stand-alone environment are used.

---

**CBC5125**    **User-defined prolog/epilog is of 0 size.**

**Explanation:** No code resulted from the assembly of a user-defined prolog or epilog. The code is generated but may cause unpredictable results.

**User Response:** Ensure that the prolog or epilog macro is coded correctly.

---

**CBC5126**    **An error occurred while building a user-defined prolog/epilog.**

**Explanation:** A prolog or epilog macro caused an error when being assembled by High Level Assembler. Processing for the prolog or epilog macro is terminated.

**User Response:** Refer to the message(s) produced by High Level Assembler for details. These can be found in the section of the compiler listing generated by the LIST option.

---

**CBC5127**    **Unsupported use of ORG statement in user-defined prolog/epilog.**

**Explanation:** A prolog or epilog macro contained an ORG statement which attempted to set the location counter to a position prior to the current location. Only ORGs with a value greater than or equal to the current location counter are supported. Processing for the prolog or epilog macro is terminated.

**User Response:** Modify the prolog or epilog macro to eliminate any ORG statements that attempt to reposition the location counter to a previous point in the code.

---

**CBC5128**    **Multiple CSECTs not supported in user-defined prolog/epilog.**

**Explanation:** A CSECT statement was found in a prolog or epilog macro with a label other than @@EDSASM. Processing for the prolog or epilog macro is terminated.

**User Response:** Change the label to @@EDSASM or remove the CSECT statement.

---

**CBC5129**    **Unsupported use of non-integral byte-sized data in user-defined prolog/epilog.**

**Explanation:** A bit field was not a multiple of 8 bits, resulting in non-integral byte-sized data. Processing for the prolog or epilog macro is terminated.

**User Response:** Only bit fields which result in an integral number of bytes (a multiple of 8 bits) are supported. Provide padding as necessary.

---

**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, or IMPORT.)**

**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 IMPORT 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, IMPORT, 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 or IMPORT.)**

**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, or IMPORT.)**

**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 :q.Object File Map:eq. 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 :q.Object File Map:eq. 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 :q.Object File Map:eq. 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 :q.Object File Map:eq. 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:

- 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 :q.Object Resolution Warnings:eq. 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 :q.Object Resolution Warnings:eq. 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 :q.Object Resolution Warnings:eq. 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 :q.Object Resolution Warnings:eq. 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.

---

**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 :q.Object Resolution Warnings:eq. 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 :q.Object Resolution Warnings:eq. 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 :q.Object Resolution Warnings:eq. 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 :q.Object Resolution Warnings:eq. 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.

---

**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.

---

**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.

---

**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.

**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.

---

**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: 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.

**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 G. Other Return Codes and Messages

See the *OS/390 Language Environment Debugging Guide and Run-Time Messages* 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)





---

## 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 *OS/390 Language Environment Debugging Guide and Run-Time Messages*.

---

### 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 44. 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:

**EDCnnnnns 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 systems 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 45. 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 *OS/390 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 46. 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. Other OS/390 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 80. Utilities for OS/390 C

For a description of CXXMOD see “Prelinking and Linking under TSO” on page 428.

---

### Using the Old Syntax for CC

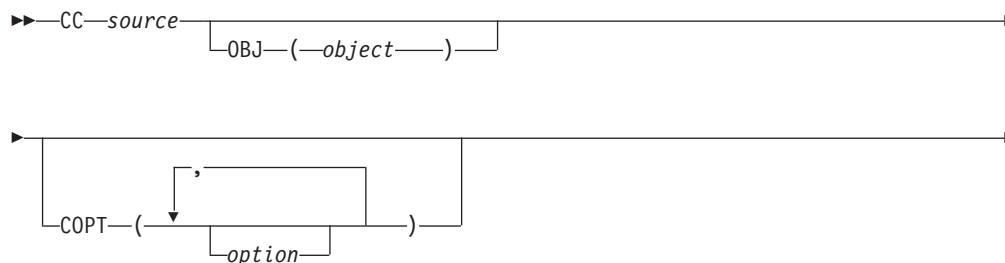
The CC command can now be invoked using a new syntax. At installation time, your system programmer can customize the CC EXEC to accept:

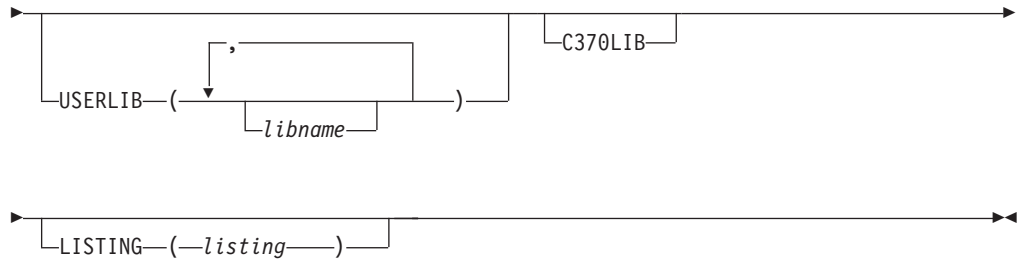
- Only the old syntax (the one supported by compilers prior to C/MVS Version 3 Release 2)
- Only the new syntax
- Both syntaxes

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 233. Refer to the OS/390 C/C++ Program Directory for more information about installation and customization.

The old syntax for the CC REXX EXEC is:





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:

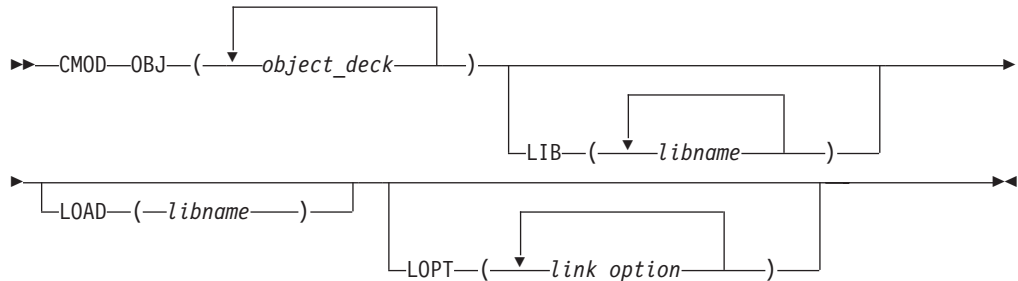
- When you are specifying a data set name, if the name is not enclosed in single quotation mark ('), your user prefix will be added to the beginning of the data set name. If the data set name is enclosed in quotation marks, it will be treated as a fully qualified name.
- When you need to use spaces, 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:



<b>OBJ</b>	Specifies the object decks that you want to link.
<b>LIB</b>	Specifies the libraries that are to be used to resolve external entries.
<b>LOAD</b>	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” on page 449. CMOD can also return the return code from LINK. See the appropriate book in your TSO library for more information on LINK.



---

## Appendix J. 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 85.

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 83 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 81. simple.c*

```
1  add some;
2  errors in the header file;
```

*Figure 82. 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 83. 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 83. Table 47 on page 608 describes the FILEID identifiers.

```
FILEID 0 1 0 10 ./simple.c
      A B C D E
```

Table 47. 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 Filend Field

The following is an example of the FILEEND field from the sample SYSEVENT file that is shown in Figure 83 on page 607. Table 48 describes the FILEEND identifiers.

```
FILEEND 0 1 6
      A B C
```

Table 48. 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 83 on page 607. Table 49 describes the ERROR identifiers.

```
ERROR 0 1 0 0 3 3 0 0 CBCMMM E 12 26 Undeclared identifier add.
      A B C D E F G H I       J K L M
```

Table 49. 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 49. 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.



---

# Glossary

This glossary defines terms and abbreviations that are used in this book. Included are terms and definitions from the following sources:

- *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, 1430 Broadway, New York, New York 10018. Such 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 have a direct object of an 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 OS/390 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) An array or a class object with no private or protected members, no constructors, no base classes, and no virtual functions. (4) In programming languages, a structured collection of data items that form a data type. *ISO-JTC1*.

**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 alphabetic characters 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 MVS, a program attribute that refers to the address length that a program is prepared to handle upon entry. In MVS, 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. Also called *parameter*. (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) An integral object, a bit field, or floating-point 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. *IBM.*

**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 OS/390 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 OS/390 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 prelinker or 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
- OS/390 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.** To combine one or more control sections or program modules into a single program module, resolving references between them, or 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 or OS/390 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 OS/390 UNIX application program from the OS/390 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 a OS/390 Language Environment-conforming high level language using the conventional OS/390 Language Environment-defined call interface, and usable by all programs sharing the OS/390 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 routines and subroutines.

**caller.** A routine that calls another routine.

**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 expression and a colon. When the selector evaluates the value of the constant expression, the statements following the case label are processed.

**cast expression.** A cast expression explicitly converts its operand to a specified arithmetic, scalar, or class type.

**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.** (1) A constant with a character value. *IBM.* (2) A string of any of the characters that can be represented, usually enclosed in apostrophes. *IBM.* (3) In some languages, a character enclosed in apostrophes. *IBM.*

**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. Classes may be defined hierarchically, allowing one class to be derived from another, and 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 of a class type that becomes a reserved word within its scope.

**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 1-byte code representing one of 256 potential characters. (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 of multiple collating elements. *X/Open*.

**collection.** (1) An abstract class without any ordering, element properties, or key properties. All abstract classes are derived from collection. (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 that contains two operands separated by a comma. Although the compiler evaluates both operands, the value of the expression is the value of the right operand. If the left operand produces a value, the compiler discards this value. Typically, the left operand of a comma expression is used 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.** A class that implements an abstract data type but does not allow polymorphism.

**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 OS/390 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 OS/390 C/C++ `signal()` function call) invoked by the OS/390 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 OS/390 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.

**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) In programming languages, 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 STOP. *ISO Draft*. (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.

**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 OS/390 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.

**DATABASE 2.** Pertaining to an IBM relational database.

**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 MVS, 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, inrapartition, and indirect destination. Extrapartition entries address data sets external to the CICS region. Inrapartition destination entries contain the information required to locate the queue in the inrapartition data set. Indirect destination entries contain the information required to locate the queue in the inrapartition 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 MVS, 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 statement that directs the preprocessor to replace an identifier or macro invocation with special code.

**define statement.** A preprocessor statement that causes the preprocessor to replace an identifier or macro call with specified code. *IBM.*

**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*.

**denormal.** Pertaining to a number with a value so close to 0 that its exponent cannot be represented normally. The exponent can be represented in a special way at the possible cost of a loss of significance.

**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:

- |          |   |
|----------|---|
| <b>0</b> | Undetached. An undetached thread keeps its resources after termination of the thread. |
| <b>1</b> | 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  $m \leq n$ , the difference contains that element zero times.

**digraph.** A combination of two keystrokes used to represent unavailable characters in a C++ source program. Digraphs are read as tokens during the preprocessor phase.

**directory.** A type of file containing the names and controlling information for other files or other directories. *IBM.*

**Direct-to-SOM (DTS).** (1) Term applied to the method by which the OS/390 C++ compiler converts existing C++ classes to SOM classes. (2) Term applied to a class that has been converted to SOM by the OS/390 C++ compiler.

**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.*

**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 bits 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.

**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 the Language Environment for MVS and VM, 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.** An operation that adds an element as the last element to a queue.

**entry point.** In assembler language, 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*.

**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 small positive integer that the system uses instead of the file name to identify an open file. *IBM*. (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 and classes 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 list of expressions enclosed in parentheses (the condition) and a statement (the action). Each expression in the parenthesized list is separated by a semicolon. You can omit any of the expressions, but you cannot omit the semicolons.

**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 OS/390 C/C++ application that does not use the services of the dynamic OS/390 C/C++ run-time library or of the Language Environment. Under OS/390 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 an optional storage class specifier, an optional type specifier, a function declarator, optional parameter declarations, and a block statement (the function body).

**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.

**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.

**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 bits 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'.

**hyperspace memory file.** An IBM file used under MVS 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 OS/390 C/C++ books that are listed in "IBM OS/390 C/C++ and Related Publications" on page 4 for information about implementation-defined behavior in the OS/390 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*.

**include statement.** In the C and C++ languages, a preprocessor statement that causes the preprocessor to replace the statement with the contents of a specified file. *IBM.*

**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*.

**inheritance.** A technique that allows the use of an existing class as the base for creating other classes.

**initial heap.** The OS/390 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*.

**job control.** A facility that allows users to selectively stop (suspend) the execution of a process and continue (resume) their execution at a later point.

The user typically employs this facility via the interactive interface jointly supplied by the terminal I/O driver and a command interpreter. *X/Open*. *ISO.1*.

## 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 IBM Language Environment for MVS and VM. 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 or OS/390 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 MVS, 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 OS/390 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.

**OS/390 UNIX System Services (OS/390 UNIX).** An element of the OS/390 operating system, (formerly known as OpenEdition). OS/390 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).

**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*.

**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*.

**PCH (precompiled header).** One or more headers that have already been compiled.

**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>         ^
<circumflex-accent>  ^
<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 consist 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.

**precompiled header.** See *PCH*.

**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 OS/390 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 form suitable for loading into main storage for execution. A program object is the output of the OS/390 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 MVS, 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 OS/390 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 through 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 Object Model (SOM).** Defines an IBM interface between programs, or between libraries and programs, so that an object's interface is separated from its implementation. SOM allows classes of objects to be defined in one programming language and used in another, and it allows libraries of such classes to be updated without requiring client code to be recompiled. *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 conversion.** Synonym for *boundary alignment*.

**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 bytes 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 an MVS 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.** (1) In virtual storage systems, the virtual storage assigned to a batched or terminal job, a system task, or a task initiated by a command. (2) In VSE, a subdivision of the virtual address area available to the user for the allocation of private, non-shared partitions.

**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*.

**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.

---

## Bibliography

This bibliography lists the publications for IBM products that are related to the OS/390 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 OS/390 C/C++ users. Refer to the *OS/390 Information Roadmap*, GC28-1727, for a complete list of publications belonging to the OS/390 product.

Related publications not listed in this section can be found on the IBM Online Library Omnibus Edition: MVS Collection CD-ROM (SK2T-0710), the *IBM Online Library Omnibus Edition: OS/390 Collection* CD-ROM (SK2T-6700), or on a tape available with OS/390.

---

### OS/390

- *OS/390 Printing Softcopy BOOKs*, S544-5354
- *OS/390 Introduction and Release Guide*, GC28-1725
- *OS/390 Planning for Installation*, GC28-1726
- *OS/390 Summary of Message Changes*, GC28-1499
- *OS/390 Information Roadmap*, GC28-1727

---

### VS COBOL II Release 4

- *General Information*, GC26-4042
- *Migration Guide for MVS and CMS*, GC26-3151
- *Installation and Customization for MVS*, SC26-4048
- *Application Programming Guide for MVS and CMS*, SC26-4045
- *Application Programming Language Reference*, GC26-4047
- *Application Programming Reference Summary*, SX26-3721
- *Application Programming Debugging*, SC26-4049
- *Application Programming Diagnosis Guide*, LY27-9523
- *Application Programming Diagnosis Reference*, LY27-9522

---

### COBOL FOR MVS & VM Release 2

- *Compiler and Run-Time Migration Guide*, GC26-4764
- *Programming Guide*, SC26-4767
- *Language Reference*, SC26-4769
- *Diagnosis Guide*, SC26-3138
- *Licensed Program Specifications*, GC26-4761
- *Installation and Customization under MVS*, SC26-4766

---

### COBOL for OS/390 & VM Version 2 Release 1

- *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 OS/390*, GC26-9045
- *Program Directory for VM*
- *Fact Sheet*, GC26-9048

---

## **PL/I for MVS & VM Release 1 Modification 1**

- *Language Reference*, SC26-3114
- *Compiler and Run-Time Migration Guide*, SC26-3118
- *Programming Guide*, SC26-3113
- *Compile-Time Messages and Codes*, SC26-3229
- *Reference Summary*, SX26-3821
- *Diagnosis Guide*, SC26-3149
- *Installation and Customization under MVS*, SC26-3119
- *Licensed Program Specifications*, GC26-3116

---

## **OS PL/I Version 2 Release 3**

- *Programming Guide*, SC26-4307
- *Programming: Language Reference*, SC26-4308
- *Programming: Messages and Codes*, SC26-4309

---

## **VS FORTRAN Version 2 Release 6**

- *Programming Reference*, SC26-4221
- *Programming Guide*, SC26-4222

---

## **CICS/ESA Version 4 Release 1**

- *Application Programming Reference*, SC33-1170
- *Application Programming Guide*, SC33-1169
- *Installation Guide*, SC33-1163
- *System Definition Guide*, SC33-1164
- *Resource Definition Guide*, SC33-1166
- *Messages and Codes*, SC33-1177

---

## **CICS Transaction Server for OS/390 Release 2**

- *Application Programming Guide*, SC33-1687
- *Application Programming Reference*, SC33-1688
- *System Programming Reference*, SC33-1689
- *Distributed Transaction Programming Guide*, SC33-1691
- *Front End Programming Interface User's Guide*, SC33-1692

---

## **DB2 Version 3 Release 1**

- *SQL Reference*, SC26-4890
- *Reference Summary*, SX26-3801
- *Command and Utility Reference*, SC26-4891
- *Application Programming and SQL Guide*, SC26-4889

---

## DB2 Version 4 Release 1

- *SQL Reference*, SC26-3270
- *Reference Summary*, SX26-3829
- *Command Reference*, SC26-3267
- *Application Programming and SQL Guide*, SC26-3266
- *Utility Guide and Reference*, SC26-3395

---

## DB2 Version 5 Release 1

- *Administration Guide*, SC26-8957
- *Application Programming and SQL Guide*, SC26-8958
- *Call Level Interface Guide and Reference*, SC26-8959
- *Command Reference*, SC26-8960
- *Data Sharing: Planning and Administration*, SC26-8961
- *Installation Guide*, GC26-8970
- *Messages and Codes*, GC26-8979
- *SQL Reference*, SC26-8966
- *Reference for Remote DRDA Requesters and Servers*, SC26-8964
- *Utility Guide and Reference*, SC26-8967

---

## IMS/ESA Version 4 Release 1

- *Application Programming: Design Guide*, SC26-3066
- *Application Programming: DL/I Calls*, SC26-3062
- *Application Programming: Data Communication*, SC26-3058
- *Application Programming: EXEC DL/I Commands*, SC26-3063

---

## IMS/ESA Version 5 Release 1

- *Application Programming: Design Guide*, SC26-8016
- *Application Programming: Transaction Manager*, SC26-8017
- *Application Programming: Database Manager*, SC26-8015
- *Application Programming: EXEC DL/I Commands for CICS and IMS*, SC26-8018

---

## IMS/ESA Version 6 Release 1

- *Application Programming: Design Guide*, SC26-8728
- *Application Programming: Transaction Manager*, SC26-8729
- *Application Programming: Database Manager*, SC26-8727
- *Application Programming: EXEC DL/I Commands for CICS and IMS*, SC26-8726

---

## QMF Version 3 Release 2

- *Introducing QMF*, GC26-4713
- *Using QMF*, SC26-8078
- *Developing QMF Applications*, SC26-4722
- *Reference*, SC26-4716
- *Managing QMF for MVS*, SC26-8218
- *Reference*, SC26-4716

- *Messages and Codes*, SC26-4834
- *Installing on MVS*, SC26-4719

---

## VSAM

- *MVS/ESA VSAM Catalog Administration: Access Method Services Reference*, SC26-4501
- *MVS/ESA VSAM Administration: Macro Instruction Reference*, SC26-4517
- *MVS/ESA VSAM Administration Guide for MVS/DFP*, SC26-4518
- *MVS/ESA Integrated Catalog Administration: Access Method Services Reference*, SC26-4500
- *DFSMS/MVS Access Method Services for VSAM*, SC26-4905
- *MVS/DFP Access Method Services for VSAM Catalogs*, SC26-4570
- *MVS/Extended Architecture VSAM Catalog Administration: Access Method Services Reference (Data Facility Product, Version 2)*, GC26-4136

# INDEX

## Special Characters

\_ALL\_SOURCE feature test macro 245  
\_MSE\_PROTOS feature test macro 246  
\_OE\_SOCKETS feature test macro 246  
\_OPEN\_DEFAULT feature test macro 245  
\_OPEN\_SOURCE feature test macro 245  
\_OPEN\_SYS feature test macro 245  
\_OPEN\_SYS\_IPC\_EXTENSIONS feature test macro 245  
\_OPEN\_SYS\_PTY\_EXTENSIONS feature test macro 245  
\_OPEN\_THREADS feature test macro 245  
\_POSIX\_C\_SOURCE feature test macro 245  
\_POSIX\_SOURCE feature test macro 244  
\_POSIX1\_SOURCE feature test macro 244  
\_XOPEN\_SOURCE\_EXTENDED feature test macro 245  
\_XOPEN\_SOURCE feature test macro 245

## A

abbreviated compiler options 59, 62  
Abstract Code Unit (ACU) 101  
ACU (Abstract Code Unit) 101  
AGGREGATE compiler option 70, 453  
aggregate layout 453  
alias  
    hidden 207  
ALIAS compiler option 71  
ALIASES binder option 207  
allocation  
    standard files with BPXBATCH 397  
allocation, standard files with BPXBATCH 397  
AMODE 340  
    restrictions 340  
ANSIALIAS compiler option 72  
ar utility  
    creating archive libraries 395  
    maintaining program objects 395  
ARCHITECTURE compiler option 73  
archive libraries  
    ar utility 395  
    creating 395  
    displaying the object files in 395  
    file naming convention for c89 use 395  
ARGPARSE compiler option 74  
argv (argument vector)  
    under TSO 338  
argv, under TSO 338  
assembler  
    generation of C structures 378  
    macros 467  
ATTACH assembler macro 467  
ATTRIBUTE compiler option 75, 453  
attributes, for DD statements 460  
AUTO|NOAUTO prelinker option 445

automatic library call  
    CALL binder option 207  
    library search processing 313  
    processing 312  
automatic library call processing  
    input to linkage editor 410  
    prelinking and 423  
    SYSLIB dataset 409

## B

binder  
    compatibility level 208  
    DLLs, creating and loading 208  
    map 209, 210  
    options file 209  
    reusability 209  
    uppercase mapping of symbol names 210  
binder options  
    ALIASES 207  
    CALL 207  
    CASE 207  
    COMPAT 208  
    DYNAM 208  
    LET 208  
    LIST 209  
    MAP 209  
    OPTION 209  
    REUS 209  
    UPCASE 210  
    XREF 210  
BITF0XL DSECT utility option 372  
BLKSIZE DSECT utility option 378  
BPARM JCL parameter 459  
BPXBATCH program  
    invoking from OS/390 batch 341  
    invoking from TSO/E 341  
    running an executable HFS file 340  
    syntax 397

## C

C++ class libraries 417  
C370LIB  
    directory 351  
    EXEC 352  
c89 utility  
    compiling and binding application programs 242  
    compiling source and object files 240  
    invoked through the make utility 243  
    linkage editor options 434  
    run by the make utility 240  
CALL  
    assembler macro 467  
    command 337  
    command, under TSO 337  
CALL binder option 207



- CALLBACKANY 84
- CASE binder option 207
- cataloged procedures
  - for binding 299
  - for compiling, prelinking, linking and running 423
  - for compiling, prelinking and linking 423
  - for prelinking, linking and running 423
  - for prelinking and linking 423
  - for specifying prelinker and linkage editor options 424
  - specifying runtime options 337
  - supplied by IBM
    - CBCB 299, 457
    - CBCBG 457
    - CBCCB 299, 457
    - CBCCBG 299, 457
    - CBCCL 424
    - CBCCLG 424
    - CBCI 457
    - CBCL 424
    - CBCLG 424
    - CEEWG 457
    - CEEWL 457
    - CEEWLG 457
  - data sets used by 460
  - EDCB 299
  - EDCC 457
  - EDCCB 299, 457
  - EDCCBG 299, 457
  - EDCCL 457
  - EDCCLGB 457
  - EDCCLIB 351, 457
  - EDCCPLG 457
  - EDCCSECT 457
  - EDCGNXLT 388
  - EDCI 457
  - EDCICONV 385
  - EDCLDEF 389
  - EDCLIB 351, 457
  - EDCPL 457
- CBC message prefix 449
- CBCB 299
- CBCCB 299
- CBCCBG 299
- CBCCL cataloged procedure 424
- CBCCLG cataloged procedure 424
- CBCL cataloged procedure 424
- CBCLG cataloged procedure 424
- CC REXX EXEC
  - C370LIB parameter 353
  - new syntax 233
  - old syntax 603
  - using under TSO 236
  - using with HFS 235
- CDSECT EXEC 382
- CEE message prefix 449
- CEESTART
  - CSECT 412
  - START compiler option 152
- character
  - trigraph representation 451
- character (*continued*)
  - unprintable 451
- characters
  - converting from one code set to another 387
- CHECKOUT compiler option 76, 451, 454
- class libraries
  - compiling with 257
  - input to the prelinker 423
- class names used with CXXFILT 365
- CLASSNAME option of CXXFILT utility 367
- CMOD REXX EXEC, syntax 604
- code set conversion utilities
  - genxlt
    - OS/390 Batch 388
    - TSO 388
    - usage 385
  - iconv
    - OS/390 Batch 385
    - TSO 386
    - usage 385
- COMMENT DSECT utility option 373
- COMPAT binder option 208
- compile-time error 450
- compiler
  - c89 utility interface to 240
  - error messages 90, 475
  - input 223, 231
    - valid input/output file types 227
  - listing
    - include file option (SHOWINC) 145
    - list inlined functions (INLRPT) 102
    - object module option (LIST) 110
    - OS/390 C++ cross reference listing 191
    - OS/390 C++ error messages 191
    - OS/390 C++ external symbol cross reference 193
    - OS/390 C++ external symbol dictionary 193
    - OS/390 C++ heading information 190
    - OS/390 C++ includes section 191
    - OS/390 C++ inline report 192
    - OS/390 C++ object code 193
    - OS/390 C++ prolog 190
    - OS/390 C++ pseudo assembly listing 193
    - OS/390 C++ source program 191
    - OS/390 C cross reference listing 181
    - OS/390 C error messages 181
    - OS/390 C external symbol cross reference 183
    - OS/390 C external symbol dictionary 183
    - OS/390 C heading information 180
    - OS/390 C includes section 181
    - OS/390 C inline report 182
    - OS/390 C object code 183
    - OS/390 C prolog 181
    - OS/390 C pseudo assembly listing 183
    - OS/390 C source program 181
    - OS/390 C storage offset listing 183
    - OS/390 C structure and union maps 181
    - source program option (SOURCE) 148
  - object module optimization 131
  - options to produce debug information
    - AGGREGATE 453



## compiler (*continued*)

- ATTRIBUTE 453
- CHECKOUT 451, 454
- EXPMAC 453
- FLAG 454
- GONUMBER 454
- INFO 454
- INLINE 454
- INLRPT 454
- LIST 453
- MARGINS 451
- NOMARGINS 451
- NOOPTIMIZE 451
- NOSEQUENCE 451
- OFFSET 453
- OPTIMIZE 451
- PPONLY 451, 453
- SEQUENCE 451
- SHOWINC 453
- SOURCE 453
- SRCMSG 454
- TEST 454
- XREF 454

## output

- create listing file 226
- create object module 226
- create preprocessor output 226
- create template instantiation output 226
- using compiler options to specify 224
- using DD statements to specify 232
- valid input/output file types 227

## return codes 475

## compiler options

- #pragma options 58
- abbreviations 59, 62
- AGGREGATE | NOAGGREGATE 70
- ALIAS | NOALIAS 71
- ANSIALIAS | NOANSIALIAS 72
- ARCHITECTURE 73
- ARGPARSE | NOARGPARSE 74
- ATTRIBUTE | NOATTRIBUTE 75
- CHECKOUT | NOCHECKOUT 76
- CONVLIT | NOCONVLIT 78
- CSECT | NOCSECT 79
- DECK | NODECK 169
- defaults 59, 62
- DEFINE 82
- DIGRAPH | NODIGRAPH 82
- DLL | NODLL 84
- EVENTS | NOEVENTS 85
- EXECOPS | NOEXECOPS 86
- EXH | NOEXH 87
- EXPMAC | NOEXPMAC 88
- EXPORTALL | NOEXPORTALL 88
- FASTT | NOFASTT 89
- FLAG | NOFLAG 90
- FLOAT 91
- GENPCH | NOGENPCH 95
- GONUMBER | NOGONUMBER 96
- HALT 97
- HWOPTS | NOHWOPTS 170

## compiler options (*continued*)

- INFO | NOINFO 98
- INLINE | NOINLINE 99
- INLRPT | NOINLRPT 102
- IPA | NOIPA 103
- IPA considerations 56
- LANGLVL 107
- LIBANSI | NOLIBANSI 110
- LIST | NOLIST 110
- LOCALE | NOLOCALE 112
- LONGNAME | NOLONGNAME 114
- LSEARCH | NOLSEARCH 115
- MARGINS | NOMARGINS 121
- MAXMEM | NOMAXMEM 123
- MEMORY | NOMEMORY 124
- NESTINC | NONESTINC 125
- OBJECT | NOBJECT 125
- OE | NOOE 127
- OFFSET | NOOFFSET 128
- OMVS | NOOMVS 129
- OPTFILE | NOOPTFILE 129
- OPTIMIZE | NOOPTIMIZE 131
- overriding defaults 55
- PHASEID 133
- PLIST 134
- PORT | NOPORT 134
- PPONLY | NOPPONLY 136
- pragma options 58
- REDIR | NOREDIR 138
- RENT | NORENT 139
- ROUND 140
- SEARCH | NOSEARCH 140
- SEQUENCE | NOSEQUENCE 143
- SERVICE | NOSERVICE 142
- SHOWINC | NOSHOWNINC 145
- SOM | NOSOM 145
- SOMEINIT | NOSOMEINIT 146
- SOMGS | NOSOMGS 146
- SOMRO | NOSOMRO 147
- SOMVOLATTR | NOSOMVOLATTR 148
- SOURCE | NOSOURCE 148
- specifying under TSO 236
- SPILL | SPILL 150
- SRCMSG | NOSRCMSG 151
- SSCOMM | NOSSCOMM 151
- START | NOSTART 152
- STRICT | NOSTRICT 153
- SYSLIB 170
- SYSPATH 171
- TARGET 153
- TEMPINC | NOTEMPINC 156
- TERMINAL | NOTERMINAL 157
- TEST | NOTEST 158
- TUNE | NOTUNE 161
- UNDEFINE 163
- UPCONV | NOUPCONV 163
- USEPCH | NOUSEPCH 164
- USERLIB 172
- USERPATH 173
- using GENP and USEP together 263
- WSIZEOF | NOWSIZEOF 165

- compiler options (*continued*)
  - XREF | NOXREF 166
  - XSOMINC | NOXSOMINC 167
- compiling
  - dynamically with OS/390 macro instructions 467
  - TSO, under 233
  - using c89 and c++ to compile and bind 242
  - using cataloged procedures supplied by IBM 228
  - using make to compile and bind 243
- compiling and binding using c89 and c++ 242
- concatenation
  - multiple libraries 232
- concatenation, multiple libraries 232
- continuation character
  - prelinker control statements 437
- control statements
  - AUTOCALL, binder 211
  - ENTRY, binder 211
  - IMPORT, binder 212
  - IMPORT, prelinker 438
  - INCLUDE 427
  - INCLUDE, binder 212
  - INCLUDE, prelinker 438
  - LIBRARY 427
  - LIBRARY, binder 213
  - LIBRARY, prelinker 439
  - linkage editor 426
  - NAME, binder 214
  - processing 437
  - RENAME, binder 214
  - RENAME, prelinker 440
- convert
  - characters from one code set to another 387
  - source definitions for locale categories 390
- Convlit 78
- CONVLIT compiler option 78
- CPARM JCL parameter 459
- CPLINK REXX EXEC
  - example 432
  - syntax 431
- cross reference listing 454
- cross reference table
  - creating with XREF compiler option 166
  - OS/390 C++ listing 191
  - OS/390 C listing 181
- CSECT (control section)
  - CEESTART 412
  - compiler option 79
  - pragma 407
- customizing locales 388
- CXX REXX EXEC
  - syntax 233
  - using under TSO 236
  - using with HFS 235
- CXXBIND REXX EXEC 305
- CXXFILT utility
  - class names 365
  - error messages 599, 600
  - input under OS/390 batch 367
  - input under TSO 368
  - options 366

- CXXFILT utility (*continued*)
  - OS/390 batch 367
  - overview 365
  - PROC for OS/390 367
  - regular names 365
  - return codes. 600
  - special names 365
  - termination 369
  - termination under OS/390 batch 368
  - TSO 368
  - unknown names 367
- CXXMOD REXX EXEC
  - keyword parameters
    - LIB 430
    - LIST 430
    - LOAD 430
    - LOPT 429
    - OBJ 429
    - PLIB 429
    - PMAP 430
    - PMOD 430
    - POPT 429
  - syntax 428

## D

- data sets
  - concatenating 232
  - for linking 408
  - for prelinking 404
  - supported attributes 460
  - usage 460
  - user prefixes 37, 43
- data types, preserving unsignedness 163
- DD statement
  - for linkage editor data sets 408
  - for prelinker data sets 404
- ddname
  - alternative 467
  - defaults 460
- debugging
  - error traceback (GONUMBER compiler option) 96
  - errors 76, 85
  - SERVICE compiler option 142
  - SRCMSG compiler option 151
  - TEST compiler option 158
- DECK compiler option 169
- default
  - compiler options 59, 62
  - output file names 111
  - overriding compiler option 55
- define
  - local environments 390
- DEFINE compiler option 82
- definition side-deck 412, 413
- DEFSUB DSECT utility option 373
- digraphs, DIGRAPH compiler option 82
- disk search sequence
  - LSEARCH compiler option 115
  - SEARCH compiler option 140

- DLLRNAM utility
  - return codes. 599
- DLLRNAME utility 357
  - input 358
  - output 358
  - under OS/390 batch 360
  - under TSO 361
- DLLs (Dynamic Link Libraries)
  - binding 208
  - building 413
  - definition side-deck 417
  - DLL compiler option 84
  - DLLNAME() prelinker option 413
  - DLLRNAME utility 357
  - EXPORTALL compiler option 88
  - IMPORT control statement 413
  - NAME control statement 413
  - prelinking 404
  - prelinking a DLL 412
  - prelinking a DLL application 413
  - redistributing 357
  - renaming 357
- DMS message prefix 449
- doublebyte characters, converting 387
- DSECT utility
  - BITF0XL option 372
  - BLKSIZE option 378
  - COMMENT option 373
  - DEFSUB option 373
  - EQUATE option 373
  - error messages 597
  - HDRSKIP option 375
  - INDENT option 376
  - LOCALE option 376
  - LOWERCASE option 376
  - LRECL option 378
  - OS/390 batch 381
  - OUTPUT option 378
  - PPCOND option 377
  - RECFM option 378
  - return codes 597
  - SECT option 371
  - SEQUENCE option 377
  - structure produced 378
  - TSO 382
  - UNNAMED option 378
- DUP prelinker option 445
- DYNAM binder option 208

## E

- EDC message prefix 449
- EDCB 299
- EDCCB 299, 300
- EDCCBG 299
- EDCCLIB cataloged procedure 351
- EDCDSECT cataloged procedure 381
- EDCGNXLTL cataloged procedure 388
- EDCICONV cataloged procedure 385
- EDCLDEF cataloged procedure 389
- EDCLDEF CLIST 390

- EDCLIB cataloged procedure 351
- EDCnnnn messages 475
- efficiency, object module optimization 131
- ENTRY linkage editor control statement 412
- environment
  - defining local 390
- EQA message prefix 449
- EQUATE DSECT utility option
  - BIT suboption 374
  - BITL suboption 374
  - DEF suboption 375
- ER prelinker option 445
- error
  - compile-time 450
  - determining source of 449
  - link time 453
  - messages
    - compiler 475
    - CXXFILT utility 600
    - directing to your terminal 157
    - DLLRNAME utility 599
    - DSECT utility 597
  - re-creating 450
  - runtime 453
- escape sequence 451
- escaping special characters 57, 230, 235
- EVENTS compiler option 85
- example
  - cbc3uaam 35
  - cbc3uaan 36
  - cbc3uaap 469
  - cbc3uaaq 470
  - cbc3uaar 471
  - cbc3uaas 472
  - cbc3uaat 473
  - cbc3uaau 474
  - cbc3ubrc 41
  - cbc3ubrh.h 40
  - cbc3uncl 52
  - clb3aitr.c 47
  - clb3aitr.h 47
  - clb3alst.c 46
  - clb3alst.h 46
  - clb3amax.c 48
  - clb3amax.h 47
  - clb3amin.c 48
  - clb3amin.h 48
  - clb3astr.h 49
  - clb3atmp.cxx 50
- examples
  - machine-readable 9
  - naming of 9
  - softcopy 9
- Examples
  - assembler macro 469
  - compile, link and run 43, 51
  - OS/390 C++ source 39
  - OS/390 C source 35
  - sample program 39
  - sample template program 45

- exception handling
  - compiler error message severity levels 90
  - compiler return codes 475
  - linkage editor 411
- EXEC
  - JCL statement
    - GPARM parameter 337
    - specifying runtime options 336
  - statement
    - invoking linkage editor 425
    - invoking prelinker 425
  - supplied by IBM
    - CDSECT 382
    - DLLRNAME 457
    - GENXLT 388
    - ICONV 385
- executable
  - files
    - invoking OS/390 load modules from the shell 340
    - placing OS/390 load modules in the HFS 340
    - running 340
    - running, under OS/390 batch 335
- EXH compiler option 87
- EXPMAC compiler option 88, 453
- EXPORTALL compiler option 88
- external
  - entry points 71
  - names
    - long name support 114
    - prelinker 403, 404
  - references
    - resolving 433
    - unresolved 445
  - variables
    - exporting 88
    - importing 88

## F

- FASTTEMPINC compiler option 89
- feature test macro 244
- files
  - names
    - generated default 111
    - include files 247
    - user prefixes 37, 43
  - searching paths 115, 140
- FLAG compiler option 90, 454
- FLOAT compiler option 91
- foreground compilation
  - panels in ISPF 236
- functions
  - code set conversion 385
  - exporting 88
  - importing 88
  - linking 411

## G

- GENPCH compiler option 95

- genxlt utility
  - CLIST 388
  - OS/390 Batch 388
  - TSO 388
  - usage 385
- get and set methods 146
- GONUMBER compiler option 96, 454
- GPARM
  - JCL parameter 460
  - parameter of EXEC statement 337

## H

- HALT compiler option 97
- HDRSKIP DSECT utility option 375
- header files
  - system 170, 171, 232
  - user 172
  - user-defined 173
- heading information
  - for IPA Link listings 201
  - for OS/390 C++ listings 190
  - for OS/390 C listings 180
- HFS (Hierarchical File System)
  - placing OS/390 load modules 340
- HWOPTS compiler option 170

## I

- IBM message prefix 449
- iconv shell command 387
- iconv utility
  - CLIST 386
  - OS/390 Batch 385
  - TSO 386
  - usage 385
- IGZ message prefix 449
- implicit mode 145
- IMPORT statement
  - syntax description 438
- IMS
  - PLIST compiler option 134
- INCLUDE control statement
  - for prelinking and linking 427
  - linkage editor and 410
  - OS/390 C/C++ prelinker and 438
  - syntax description 438
- include files
  - naming 247
  - nested 125
  - preprocessor directive
    - syntax 246
  - record format 247
  - SHOWINC compiler option 145
  - system files and libraries
    - OPTFILE compiler option 129
    - SEARCH compiler option 140
    - using 246
  - user files and libraries
    - using 246
- INDENT DSECT utility option 376

- INFILE REXX EXEC parameter 459
- INFO compiler option 98, 454
- inline
  - OS/390 C++ report 192
  - OS/390 C report 182
  - report for IPA inliner 202
- INLINE compiler option 454
  - description 99
- INLRPT compiler option 102, 454
- input
  - compiler 223, 231
  - linkage editor 409
  - prelinker 404, 405, 406
  - record sequence numbers 143
- installation
  - problems 455
  - PTFs 450
- IPA
  - invoking from the c89 utility 242
  - IPA Compile step
    - flow of processing 221
  - IPA compiler option 103
  - IPA Link step
    - automatic library call processing 273
    - compiler options map listing section 202
    - control file 277
    - error source 451
    - external symbol cross reference listing section 204
    - external symbol dictionary listing section 204
    - flow of processing 222
    - global symbols map listing section 202
    - IMPORT IPA Link control statement 276
    - INCLUDE IPA Link control statement 277
    - input 271
    - IPA inliner listing section 202
    - IPA Link control statements 275
    - LIBRARY IPA Link control statement 277
    - library routine considerations 274
    - listing heading information 201
    - listing message summary 205
    - listing messages section 205
    - listing output 282
    - listing overview 174, 183, 193
    - listing prolog 201
    - object file map listing section 201
    - object module output 282
    - object record formats 275
    - options, specifying under OS/390 batch 285
    - output 281
    - overview 267
    - partition map listing section 204
    - primary input 271
    - pseudo assembly listing section 204
    - region size, specifying under OS/390 batch 285
    - running in the OS/390 UNIX environment 286
    - running under OS/390 batch 283
    - secondary input 272
    - secondary input, specifying under OS/390 batch 285
    - source file map listing section 202

- IPA (*continued*)
  - IPA Link step (*continued*)
    - storage offset listing section 204
    - uppercase name resolution 273
    - using CBCI 284
    - using DD statements for the standard data sets 268
    - using DLLs 274
    - using EDCI 284
    - using the c89 utility with 287
    - using your own JCL for 286
  - object modules under IPA 226
  - overview 221
  - using catalogued procedures 229
- IPACNTL data set 460, 462
- IPARM JCL parameter 459
- IRUN JCL parameter 459
- ISPF (Interactive System Productivity Facility)
  - batch compile panels 238
  - foreground compile panels 236
  - starting the compiler with 236

## J

- JCL (Job Control Language)
  - C comments 151
  - description 231
  - ENTRY control statement 412
  - specifying prelinker and linkage editor options 425, 426

## L

- LANGLVL compiler option 107
- LET binder option 208
- LIB parameter CXXMOD EXEC 430
- LIBANSI compiler option 110
- library
  - archive
    - creating 395
    - displaying the object files in 395
    - file naming convention for c89 use 395
    - use by application programs 395
  - OS/390 Language Environment
    - components 411
    - required to run the compiler 223
    - runtime 223
  - search sequence
    - with LSEARCH compiler option 115
    - with SEARCH compiler option 140
- LIBRARY control statement
  - linkage editor and 410
  - prelinker and 427, 439
  - using with linkage editor 427
- LIBRARY JCL parameter 460
- LINK
  - assembler macro 467
  - command
    - input 433
    - LOAD operand 434
    - syntax 433

- link time error 453
- linkage editor
  - creating a load module under OS/390 batch 424
  - errors 411
  - function of 417
  - INCLUDE statement and 410
  - input to 409, 418
  - LIBRARY statement and 410
  - options
    - MAP|NOMAP 405
    - specifying 424
  - output 409, 410, 418
  - requesting options with c89 434
  - using c89 and c++ to compile and bindt 242
  - using make to compile and bind 243
  - using under TSO 428
- linking 423
  - IBM-supplied class libraries 423
  - multiple object modules 412
- LIST binder option 209
- LIST compiler option 110, 453
- LIST parameter CXXMOD EXEC 430
- listings
  - all included text 453
  - binder map 209, 210
  - cross reference 454
  - from linkage editor 409
  - from prelinker 405, 418
  - include file option (SHOWINC) 145
  - IPA Compile step, using 174, 183
  - IPA Link step, using 193
  - IPA Link step compiler options map 202
  - IPA Link step external symbol cross reference 204
  - IPA Link step external symbol dictionary 204
  - IPA Link step global symbols map 202
  - IPA Link step heading information 201
  - IPA Link step inliner 202
  - IPA Link step message summary 205
  - IPA Link step messages 205
  - IPA Link step object file map 201
  - IPA Link step partition map 204
  - IPA Link step prolog 201
  - IPA Link step pseudo assembly 204
  - IPA Link step source file map 202
  - IPA Link step storage offset 204
  - message summary, OS/390 C 182
  - message summary, OS/390 C++ 192
  - object code 453
  - object library utility map 351
  - object module option (LIST) 110
  - OS/390 C++ cross reference table 191
  - OS/390 C++ external symbol cross reference 193
  - OS/390 C++ external symbol dictionary 193
  - OS/390 C++ includes section 191
  - OS/390 C++ messages 191
  - OS/390 C++ object code 193
  - OS/390 C++ pseudo assembly listing 193
  - OS/390 C++ source program 191
  - OS/390 C, using 174
  - OS/390 C cross reference table 181
  - OS/390 C external symbol cross reference 183
- listings (*continued*)
  - OS/390 C external symbol dictionary 183
  - OS/390 C includes section 181
  - OS/390 C messages 181
  - OS/390 C object code 183
  - OS/390 C pseudo assembly listing 183
  - OS/390 C source program 181
  - OS/390 C structure and union maps 181
  - source file 453
  - using OS/390 C++ 183
- load library
  - storing object modules 434
- load module
  - creating 408
  - inputs for 418
- LOAD parameter CXXMOD EXEC 430
- local environment, defining 390
- local variables 150
- locale
  - converting source definitions for categories 390
  - customizing 388
  - definition file 388
  - DSECT utility option 376
  - object 389
- LOCALE
  - compiler options 112
- localedef shell command 390
- localedef utility
  - OS/390 batch 389
  - TSO 390
- logical
  - string assist (LSA) 170
- logical string assist (LSA) 170
- long names
  - definition of 403
  - LIBRARY control statement and 439
  - mapping to short names 407
  - RENAME control statement and 440
  - resolving undefined 423
  - support 114
  - unresolved 423
  - UPCASE prelink option and 445
- LONGNAME compiler option 114
- LOPT parameter CXXMOD EXEC 429
- LOWERCASE DSECT utility option 376
- LPARM parameter 424
- LPARM REXX EXEC parameter 459
- LRECL (logical record length) parameter
  - DSECT utility option 378
- LRECL DSECT utility option 378
- LSEARCH compiler option 115

## M

- macro
  - assembler
    - ATTACH 467
    - CALL 467
    - compiling OS/390 C/C++ programs with 467
    - LINK 467
    - expanded in source listing 88



- macro (*continued*)
  - expansion 453
  - feature test 244
- maintaining
  - objects in an archive library 395
  - programs through makefiles 396
  - programs with make using c89 243
- make utility
  - compiling and binding application programs 243
  - compiling source and object files 240
  - creating makefiles 396
  - maintaining OS/390 C/C++ application programs 396
- makefiles
  - creating 396
  - maintaining application programs 396
- mangled name filter utility 365
- map
  - load module 410
  - pragma 407
  - prelinker 405, 406, 418
- MAP binder option 209
- MAP prelinker option 405, 418, 445
- mapping
  - long names to short names
    - rules for 407
  - of load modules 425
- MARGINS compiler option 121, 451
- MAXMEM compiler option 123
- MEMBER JCL parameter 460
- memory
  - files, compiler work-files 124
  - MAXMEM compiler option 123
  - MEMORY compiler option 124
  - MEMORY prelinker option 445
- message prefixes
  - CBC 449
  - CEE 449
  - DMS 449
  - EDC 449
  - EQA 449
  - IBM 449
  - IGZ 449
  - others 449
  - PLI 449
- messages
  - compiler, list of 475
  - directing to your terminal 157
  - generate warning 98
  - on IPA Link step listings 205
  - on OS/390 C++ compiler listings 191
  - on OS/390 C compiler listings 181
  - specifying severity of 90
- mismatches, type 451
- MVS (Multiple Virtual System)
  - batch environment
    - running shell scripts and OS/390 C/C++ applications 397

## N

- NAME control statement 405, 410
- natural reentrancy
  - generating 139
  - linking 453
- NCAL prelinker option 445
- NESTINC compiler option 125
- NOAGGREGATE compiler option 70
- NOALIAS compiler option 71
- NOANSIALIAS compiler option 72
- NOARGPARSE compiler option 74
- NOATTRIBUTE compiler option 75
- NOCALLBACKANY 84
- NOCHECKOUT compiler option 76
- NOCLASSNAME option of CXXFILT utility 367
- NOCSECT compiler option 79
- NODECK compiler option 169
- NODIGRAPH compiler option 82
- NODLL compiler option 84
- NODUP prelinker option 445
- NOER prelinker option 445
- NOEVENTS compiler option 85
- NOEXECOPS compiler option 86
- NOEXPMAC compiler option 88
- NOEXPORTALL compiler option 88
- NOFASTTEMPINC compiler option 89
- NOFLAG compiler option 90
- NOGENPCH compiler option 95
- NOGONUMBER compiler option 96
- NOHWOPTS compiler option 170
- NOINFO compiler option 98
- NOINLINE compiler option 99
- NOINLRPT compiler option 102
- NOIPA compiler option 103
- NOLIBANSI compiler option 110
- NOLIST compiler option 110
- NOLOCALE compiler option 112
- NOLONGNAME compiler option 114
- NOLSEARCH compiler option 115
- NOMAP prelinker option 445
- NOMARGINS compiler option 121, 451
- NOMAXMEM compiler option 123
- NOMEMORY compiler option 124
- NOMEMORY prelinker option 445
- NONCAL prelinker option 445
- NONESTINC compiler option 125
- NOOBJECT compiler option 125
- NOOE compiler option 127
- NOOFFSET compiler option 128
- NOOMVS compiler option 129
- NOOPTFILE compiler option 129
- NOOPTIMIZE compiler option 131, 451
- NOPONLY compiler option 136
- NOREDIRE compiler option 138
- NOREGULARNAME option of CXXFILT utility 366
- NORENT compiler option 139
- NOSEARCH compiler option 140
- NOSEQUENCE compiler option 143, 451
- NOSERVICE compiler option 142
- NOSHOWINC compiler option 145
- NOSIDEBYSIDE option of CXXFILT utility 366

- NOSOM compiler option 145
- NOSOMEINIT compiler option 146
- NOSOMGS compiler option 146
- NOSOMRO compiler option 147
- NOSOMVOLATTR compiler option 148
- NOSOURCE compiler option 148
- NOSPECIALNAME option of CXXFILT utility 367
- NOSPILL compiler option 150
- NOSRCMSG compiler option 151
- NOSSCOMM compiler option 151
- NOSTART compiler option 152
- NOSTRICT compiler option 153
- NOSYMMAP option of CXXFILT utility 366
- NOSYSPATH compiler option 171
- NOTEMPINC compiler option 156
- NOTERMINAL compiler option 157
- NOTEST compiler option 158
- NOUPCASE prelinker option 445
- NOUPCONV compiler option 163
- NOUSEPCH compiler option 164
- NOWIDTH option of CXXFILT utility 366
- NOWSIZEOF compiler option 165
- NOXREF compiler option 166
- NOXSOMINC compiler option 167

## O

- OBJ parameter for CXXMOD EXEC 429

### object

- code 221

#### library

- adding object modules 351
- deleting object modules 351
- listing the contents 351
- OS/390 batch 351
- TSO 353

#### module

- additional object modules as input 410
- creating 427
- DECK compiler option 169
- DLL compiler option 84
- EXPORTALL compiler option 88
- link-editing multiple modules 412
- LIST compiler option 110
- OBJECT compiler option 125
- optimization 131
- OS/390 C++ object listing 193
- OS/390 C object listing 183
- storing in a load library 434
- TARGET compiler option 153

### OBJECT

- compiler option 125
- JCL parameter 460

- object code, listing 453

### Object Library Utility

- example under OS/390 batch 351
- long name support 351
- map
  - heading 354
  - member heading 354
  - symbol information 354

### Object Library Utility *(continued)*

#### map *(continued)*

- user comments 354

- OE compiler option 127

- OFFSET compiler option 128, 453

- OGET utility 241, 340

- OGETX utility 340

### OMVS

- compiler option 129

- OE compiler option 127

- OPARM JCL parameter 460

- OPEN\_MSGQ\_EXT feature test macro 246

- OPTFILE compiler option 129

### optimization

- object module 131

- OPTIMIZE compiler option 131

- storage requirements 131

- TUNE compiler option 161

- OPTIMIZE compiler option 131, 451

- OPTION binder option 209

### options

- compiler 59

- CXXFILT syntax 365

- linkage editor 410

- NOSYMMAP, for CXXFILT 366

- runtime 217

- SYMMAP, for CLASSNAME 367

- SYMMAP, for CXXFILT 366

- SYMMAP, for NOCLASSNAME 367

- SYMMAP, for NOREGULARNAME 366

- SYMMAP, for NOSIDEBYSIDE 366

- SYMMAP, for NOSPECIALNAME 367

- SYMMAP, for NOWIDTH 366

- SYMMAP, for REGULARNAME 366

- SYMMAP, for SIDEBYSIDE 366

- SYMMAP, for SPECIALNAME 367

- SYMMAP, for WIDTH 366

- OPUTX utility 340

### OS/390 batch

- compile, ISPF panels 238

- compiling under 228

- link-editing 427

- running shell scripts and OS/390 C/C++ applications 397

- running your program 335

- OS/390 Program Directory 455

### OS/390 UNIX

- compiling and binding using c89 240

- compiling and binding using c89 and c++ 242

- compiling and binding using make 243

- maintaining objects in an archive library 395

- maintaining through makefiles 396

- OE compiler option 127

- OMVS compiler option 129

- placing OS/390 load modules in the HFS 340

- OUTFILE REXX EXEC parameter 459

### output

- from the linkage editor 409, 410

- from the prelinker 405

- OUTPUT DSECT utility option 378



## P

- PARM parameter 425
- passing
  - arguments 217
- passing arguments 217
- PHASEID compiler option 133
- PLI message prefix 449
- PLIB parameter CXXMOD EXEC 429
- PLIST compiler option 134
- PMAP parameter CXXMOD EXEC 430
- PMOD parameter CXXMOD EXEC 430
- POPT parameter CXXMOD EXEC 429
- PORT compiler option 134
- PPARM
  - JCL parameter 460
  - parameter 424
- PPCOND DSECT utility option 377
- PPONLY compiler option 136, 451, 453
- pragmas
  - csect 407
  - map 407
  - options 58
  - runopts 217
- precompiled headers
  - creating 95
  - GENP and USEP compiler options 263
  - initial sequence of headers
    - determining 259
    - example of varying 260
    - long 265
    - matching 262
    - reusing 263
    - short 265
    - terminating 260
    - using alternate 264
  - maintain current 263
  - organizing source files
    - a PCH file for each member of the directory 266
    - common header file 266
    - global PCH file 266
    - hints and tips 265
  - overview 259
  - restrictions 264
  - reusing 164
  - usage 259
- prelinker
  - building and using DLLs 413
  - error source 452
  - function of 417
  - functions of 403
  - IBM-supplied class libraries 423
  - IMPORT statement and 438
  - INCLUDE statement and 438
  - input 404, 405, 406, 417
  - LIBRARY statement and 439
  - load modules 452
  - map 405, 418
  - mapping long names to short names 407
  - messages from 405
  - options
    - MAP|NOMAP 418

- prelinker (*continued*)
  - options (*continued*)
    - specifying 424
  - output from 405, 417, 432
  - overview 403
  - RENAME statement and 440
  - resolving undefined symbols 423
  - under OS/390 batch 426
  - usage 403
- preprocessor, diagnostic information 453
- preprocessor directives
  - effects of PPOONLY compiler option 136
  - include 246
- preventive service planning (PSP) bucket 450, 455
- primary data set
  - specifying input to the compiler 224
  - specifying input to the linkage editor 410
- primary input
  - compiler 224
  - linkage editor 410
  - to the IPA Link step 269, 271
  - to the linkage editor 409
  - to the prelinker 404
- processing a C program
  - OS/390 C sample program, under OS/390 Batch 37
  - OS/390 C sample program, under TSO 37
- programming errors 76
- PSP (preventive service planning) bucket 450, 455
- PTF (Program Temporary Fix)
  - error solution 450
  - error source 450
  - installation 450

## R

- RECFM DSECT utility option 378
- record
  - margins 121
- record margins 121
- REDIR compiler option 138
- reentrancy
  - linking 452
  - RENT compiler option 139
- reentrant code
  - linking 452
  - RENT compiler option 139
- regular names used with CXXFILT 365
- REGULARNAME option of CXXFILT utility 366
- Release order 147
- RENAME control statement
  - mapping long names to short names 407
  - syntax 440
- RENT compiler option
  - syntax 139
- return
  - codes
    - compiler 475
    - CXXFILT utility 600
    - DLLRNAME utility 599
    - DSECT utility 597
    - severity 475

- return codes
  - compiler 475
  - CXXFILT utility 600
  - DLLRNAME utility 599
  - DSECT utility 597
  - severity 475
- REUS binder option 209
- REXX EXECs
  - supplied by IBM
    - C370LIB 457
    - CC, new syntax 457
    - CC, old syntax 603
    - CDSECT 457
    - CMOD 603, 604
    - CPLINK 431
    - CXXBIND 457
    - CXXMOD 457
    - EDCLDEF 390
    - GENXLT 388, 457
    - ICONV 386, 457
    - LOCALEDEF 457
- ROUND compiler option 140
- running programs
  - OS/390 batch
    - BPXBATCH 340
    - example 336
    - with EXEC JCL statement 335
  - OS/390 UNIX application 339
  - TSO
    - example 337
    - specifying runtime options 338
    - with CALL TSO command 337
- runtime
  - errors 453
  - options
    - in the EXEC statement 336
    - recognize at runtime 86
    - specifying 217
    - under OS/390 batch 335
    - under OS/390 UNIX 339
  - specifying runtime environment 153
- secondary data set
  - libraries 232
  - secondary input to the linkage editor 410
- secondary input
  - compiler 224, 232
  - linkage editor 410
  - to the IPA Link step 269, 272
  - to the linkage editor 409
  - to the prelinker 405
- SECT DSECT utility option 371
- SEQUENCE compiler option 143, 451
- SEQUENCE DSECT utility option 377
- sequence numbers on input records 143
- SERVICE compiler option 142
- severity, compiler return codes. 475
- shell
  - compiling and link-editing within
    - using the c89 utility 240
  - invoking load modules 340
  - using BPXBATCH to run commands or scripts 397
- short names
  - automatic library call processing 423
  - definition of 403
  - mapping 407
  - unresolved 423
- SHOWINC compiler option 145, 453
- SIDEBYSIDE option of CXXFILT utility 366
- singlebyte characters
  - converting 387
- SOM compiler options
  - SOM | NOSOM 145
  - SOMEINIT | NOSOMEINIT 146
  - SOMGS | NOSOMGS 146
  - SOMRO | NOSOMRO 147
  - SOMVOLATTR | NOSOMVOLATTR 148
  - XSOMINC | NOXSOMINC 167
- source
  - file listing 453
  - program
    - comment (SSCOMM compiler option) 151
    - compiler listing options 145, 148
    - file names in include files 247
    - generating reentrant code 139
    - input data set 224
    - margins 121
    - SEQUENCE compiler option 143
- source code
  - compiling using c89 239
  - OS/390 C++ example program 39
  - OS/390 C sample program 35
- SOURCE compiler option 148, 453
- source definitions
  - converting for locale categories 390
- special characters, escaping 57, 230, 235
- special names used with CXXFILT 365
- SPECIALNAME option of CXXFILT utility 367
- spill area
  - changing the size of 150
  - definition of 150
  - pragma 150
- SPILL compiler option 150

## S

- sample program
  - OS/390 C++ source 39
  - OS/390 C source 35
  - processing OS/390 C under OS/390 Batch 37
  - processing OS/390 C under TSO 37
- SCEECPP library 429
- SCEELKED library
  - prelinker and 418, 430
- SEARCH compiler option 140
- Search option
  - using to compile OS/390 C++ code 257
  - using to compile OS/390 C code 257
- search sequence
  - library files 335
  - system include files 141
  - user include files 115

- SRCMSG compiler option 151, 454
- SSCOMM compiler option 151
- standard
  - files, allocation for BPXBATCH 397
- standard files, allocation for BPXBATCH 397
- standards
  - ANSI compiler option 107
  - LIBANSI compiler option 110
- START compiler option 152
- statement
  - failure in 454
  - switch 454
- STEPLIB
  - data set 460, 461, 462
  - ddname 404
  - prelinker 405
- storage
  - optimization 131
- STRICT compiler option 153
- structure and union maps, OS/390 C compiler
  - listing 181
- stub routines
  - contents of 411
  - in OS/390 Language Environment 411
- switch statement 454
- SYMMAP option of CXXFILT utility 366
- syntax diagrams, how to read 11
- SYSCPRT data set 226, 460, 461, 463
- SYSDEFSD data set
  - description 463
  - prelinker and 404, 405
- SYSEVENT data set
  - description of 462
- SYSIN data set for stdin
  - description of 460, 462
  - primary input to prelinker 404, 417
  - primary input to the compiler 231
  - usage 460
- SYSLIB compiler option 170
- SYSLIB data set
  - description of 461, 462
  - IPA Link step and 272
  - linkage editor and 408, 409, 418
  - prelinker and 404, 405, 417
  - secondary input to linkage editor 410
  - specifying 232
  - usage 460
- SYSLIN data set
  - description of 461, 462
  - IPA Link step and 270
  - linkage editor and 408, 409, 417
  - primary input to linkage editor 410
  - usage 460
  - with OBJECT compiler option 226
- SYSMOD data set 408, 409, 418, 460
- SYSMOD data set 404, 405, 418, 460
- SYSMSGs data set 404, 405, 460
- SYSOUT data set
  - description of 461, 462, 465
  - prelinker and 404, 405
  - usage 460

- SYPATH compiler option 171, 224
- SYSPRINT data set
  - linkage editor and 408, 409
  - prelinker and 404
  - usage 460
- SYPUNCH data set
  - with DECK compiler option 226
- system
  - files and libraries 129, 140
  - programmer
    - establishing library access 233, 337
- system-defined header files 170, 171
- system header files 232
- SYSUT1 data set 408, 409, 460, 461
- SYSUT4-10 data sets 461

## T

- TARGET compiler option 153
- TEMPINC compiler option 156
- templates
  - create template instantiation output 226
  - program example 45
- TERMINAL compiler option 157
- test case, creating 450
- TEST compiler option 158, 454
- TEXT deck 450
- trigraph 451
- TSO (Time Sharing Option)
  - compiling under 233
  - LINK command 434
- TUNE compiler option 161
- type conversion, preserving unsignedness 163
- type conversions 163
- type mismatches 451

## U

- UNDEFINE compiler option 163
- unknown names input to CXXFILT utility 367
- UNNAMED DSECT utility option 378
- unprintable character 451
- unsignedness preservation, type conversion 163
- UPCASE binder option 210
- UPCASE prelinker option 445
- UPCONV compiler option 163
- USEPCH compiler option 164
- user
  - comments, object library utility map 354
  - include files
    - LSEARCH compiler option 115
    - SEARCH compiler option 140
    - specifying with #include directive 246
  - prefix 37, 43
- user-defined header files 172, 173
- USERLIB compiler option 172
- USERPATH compiler option 173, 224
- utilities
  - CXXFILT 365, 600
  - DLLRNAME 599
  - DSECT 597

utilities (*continued*)

- mangled name filter 365
- OS/390 C 457
- OS/390 C++ 457
- OS/390 C, old syntax 603
- Symbol is obsolete - use ouss or oux 395

## W

- WIDTH option of CXXFILT utility 366
- work data sets 460
- writable static
  - object library 351
  - prelinker and 406
  - relative offsets 403
- WSIZEOF compiler option 165

## X

- XREF binder option 210
- XREF compiler option 166, 454
- XSOMINC compiler option 167

---

## Readers' Comments — We'd Like to Hear from You

OS/390  
C/C++  
User's Guide

Publication No. SC09-2361-04

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

---

Name

---

Address

---

Company or Organization

---

Phone No.



Cut or Fold  
Along Line

Fold and Tape

Please do not staple

Fold and Tape

PLACE  
POSTAGE  
STAMP  
HERE

IBM Canada Ltd. Laboratory  
Information Development  
2G/345/1150/TOR  
1150 EGLINTON AVENUE EAST  
NORTH YORK ONTARIO CANADA  
M3C 1H7

Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold  
Along Line





Printed in the United States of America

SC09-2361-04

