

z/OS



Language Environment Debugging Guide

z/OS



Language Environment Debugging Guide

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 417.

Seventh Edition, September 2005

This is a major revision of GA22-7560-05.

This edition applies to Language Environment in z/OS Version 1 Release 7 (5694-A01), Version 1, Release 7 of z/OS.e™ (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405

FAX (Other Countries):

Your International Access Code +1+845+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)

Internet e-mail: mhvrfs@us.ibm.com

World Wide Web: www.ibm.com/servers/eserver/zseries/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this document
- 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 1991, 2005. All rights reserved.

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

Contents

Figures.	ix
Tables.	xiii
About this document	xv
Using your documentation.	xvi
How to read syntax diagrams	xvii
Symbols	xvii
Syntax items	xvii
Syntax examples.	xviii
This Debugging Guide	xix
Where to find more information	xix
Using LookAt to look up message explanations.	xix
Using IBM Health Checker for z/OS	xx
Information updates on the web.	xx
Summary of Changes	xxi
Part 1. Introduction to debugging in Language Environment	1
Chapter 1. Preparing your routine for debugging	3
Setting compiler options	3
XL C and XL C++ compiler options	3
COBOL compiler options.	6
Fortran compiler options	7
PL/I compiler options	8
VisualAge PL/I compiler options	9
Using Language Environment run-time options	9
Determining run-time options in effect	10
Using the CLER CICS transaction to display and set run-time options.	12
Controlling storage allocation.	12
Stack storage statistics	18
Heap storage statistics	20
HeapPools storage statistics	21
Modifying condition handling behavior	22
Language Environment callable services	22
Language Environment run-time options	22
Customizing condition handlers	24
Invoking the assembler user exit	25
Establishing enclave termination behavior for unhandled conditions	26
Using messages in your routine.	26
C/C++	27
COBOL	27
Fortran	27
PL/I	27
Using condition information	27
Using the feedback code parameter	27
Using the symbolic feedback code.	29
Chapter 2. Classifying errors	31
Identifying problems in routines	31
Language Environment module names	31
Common errors in routines	31

Interpreting run-time messages	32
Message prefix	33
Message number	33
Severity code	33
Message text	34
Understanding abend codes	34
User abends	34
System abends	34
Chapter 3. Using Language Environment debugging facilities	35
Debug tool	35
Language Environment dump service, CEE3DMP	35
Generating a Language Environment dump with CEE3DMP	35
Generating a Language Environment dump with TERMTHDACT	39
Generating a Language Environment dump with language-specific functions	43
Understanding the Language Environment dump	43
Debugging with specific sections of the Language Environment dump.	60
Multiple enclave dumps.	73
Generating a system dump	75
Steps for generating a system dump in a batch run-time environment.	75
Steps for generating a system dump in an IMS run-time environment	76
Steps for generating a system dump in a CICS run-time environment	76
Steps for generating a Language Environment U4039 abend	77
Steps for generating a system dump in a z/OS UNIX shell	77
Formatting and analyzing system dumps	78
Preparing to use the Language Environment support for IPCS	78
Language Environment IPCS Verbexit – LEDATA	78
Format	79
Parameters	79
Understanding the Language Environment IPCS Verbexit LEDATA output	81
Understanding the HEAP LEDATA output	95
Diagnosing heap fragmentation problems.	101
Understanding the HEAPPOOLS trace LEDATA output.	101
Understanding the C/C++-specific LEDATA output	103
C/C++-specific sections of the LEDATA output	108
Understanding the COBOL-specific LEDATA output	109
COBOL-specific sections of the LEDATA Output	111
Formatting individual control blocks	112
Requesting a Language Environment trace for debugging.	114
Locating the trace dump	115
Using the Language Environment trace table format in a dump report	116
Understanding the trace table entry (TTE)	116
Sample dump for the trace table entry	123

Part 2. Debugging language-specific routines 125

Chapter 4. Debugging C/C++ routines	127
Debugging C/C++ input/output programs	127
Using the __amrc and __amrc2 structures	127
__last_op values.	129
Displaying an error message with the perror() function	133
Using __errno2() to diagnose application problems	133
Using C/C++ listings	134
Generating C/C++ listings and maps	134
Finding variables.	137
Generating a Language Environment dump of a C/C++ routine.	145

cdump()	145
csnap()	146
ctrace()	146
Sample C routine that calls cdump()	146
Sample C++ routine that generates a Language Environment dump	148
Sample Language Environment dump with C/C++-specific information	150
Finding C/C++ information in a Language Environment dump	156
Sample Language Environment dump with XPLINK-specific information	161
Finding XPLINK information in a Language Environment dump	166
C/C++ contents of the Language Environment trace tables	167
Debugging examples of C/C++ routines	172
Divide-by-zero error	172
Calling a nonexistent non-XPLINK function	176
Calling a nonexistent XPLINK function	179
Handling dumps written to the z/OS UNIX file system	183
Multithreading consideration	184
Understanding C/C++ heap information in storage reports	184
Language Environment storage report with HeapPools statistics	185
C function __uheapreport() storage report	186
MEMCHECK VHM memory leak analysis tool	187
Invoking MEMCHECK VHM	187
MEMCHECK VHM environment variables	188
MEMCHECK VHM report sample scenario	189
MEMCHECK VHM report examples	190
Chapter 5. Debugging COBOL programs	195
Determining the source of error	195
Tracing program logic	195
Finding input/output errors	195
Handling input/output errors	196
Validating data (class test)	196
Assessing switch problems	196
Generating information about procedures	196
Using COBOL listings	199
Generating a Language Environment dump of a COBOL program	199
COBOL program that calls another COBOL program	199
COBOL program that calls the Language Environment CEE3DMP callable service	200
Sample Language Environment dump with COBOL-specific information	201
Finding COBOL information in a dump	203
Debugging example COBOL programs	207
Subscript range error	207
Calling a nonexistent subroutine	210
Divide-by-zero error	213
Chapter 6. Debugging FORTRAN routines	219
Determining the source of errors in FORTRAN routines	219
Identifying run-time errors	219
Using FORTRAN compiler listings	221
Generating a Language Environment dump of a FORTRAN routine	221
DUMP/PDUMP subroutines	222
CDUMP/CPDUMP subroutines	223
SDUMP subroutine	224
Finding FORTRAN information in a Language Environment dump	227
Understanding the Language Environment traceback table	228
Examples of debugging FORTRAN routines	229

Calling a nonexistent routine	229
Divide-by-zero error.	231
Chapter 7. Debugging PL/I routines	235
Determining the source of errors in PL/I routines	235
Logic errors in the source routine.	235
Invalid use of PL/I	235
Unforeseen errors	236
Invalid input data.	236
Compiler or run-time routine malfunction	236
System malfunction.	236
Unidentified routine malfunction	236
Storage overlay problems	237
Using PL/I compiler listings	238
Generating PL/I listings and maps	238
Finding information in PL/I listings	239
Generating a Language Environment dump of a PL/I routine	245
PLIDUMP syntax and options	245
PLIDUMP usage notes	246
Finding PL/I information in a dump	247
Traceback	247
Control blocks for active routines.	249
Control blocks associated with the thread.	251
PL/I contents of the Language Environment trace table	253
Debugging example of PL/I routines	253
Subscript range error	253
Calling a nonexistent subroutine	256
Divide-by-zero error.	258
Chapter 8. Debugging under CICS	263
Accessing debugging information.	263
Locating Language Environment run-time messages	263
Locating the Language Environment traceback.	263
Locating the Language Environment dump	264
Using CICS transaction dump	264
Using CICS register and program status word contents	265
Using Language Environment abend and reason codes	265
Using Language Environment return codes to CICS.	265
Activating Language Environment feature trace records under CICS.	266
Ensuring transaction rollback	268
Finding data when Language Environment returns a nonzero return code	268
Finding data when Language Environment abends internally	268
Finding data when Language Environment abends from an EXEC CICS command	269
Displaying and modifying run-time options with the CLER transaction	269

Part 3. Debugging Language Environment AMODE 64 applications 271

Chapter 9. Preparing your AMODE 64 application for debugging	273
Setting compiler options	273
XL C and XL C++ compiler options for AMODE 64 applications	273
Using Language Environment run-time options.	273
Determining run-time options in effect	274
Controlling storage allocation	275
HeapPools storage statistics	277
Modifying exception handling behavior.	277

Language Environment application program interfaces (API).	278
Language Environment run-time options	278
Customizing exception handlers	279
Using condition information	279
Using the feedback code parameter.	279
Using the symbolic feedback code	281
Chapter 10. Classifying AMODE 64 application errors	283
Identifying problems in routines	283
Language Environment module names	283
Common errors in routines	283
Interpreting run-time messages	284
Message prefix	285
Message number	285
Severity code	285
Message text	285
Understanding abend codes	285
User abends	286
System abends	286
Chapter 11. Using Language Environment AMODE 64 debugging facilities	287
Debugging tools	287
Language Environment dumps	287
Generating a Language Environment dump with TERMTHDACT	287
Generating a Language Environment dump with language-specific functions	289
Understanding the Language Environment dump	290
Generating a system dump	306
Steps for Generating a system dump in a z/OS UNIX shell	307
Formatting and analyzing system dumps	308
Preparing to use the Language Environment support for IPCS	308
Language Environment IPCS Verbexit – LEDATA.	308
Format	309
Parameters.	309
Understanding the Language Environment IPCS Verbexit LEDATA output	311
Understanding the HEAP LEDATA output.	326
Diagnosing heap fragmentation problems.	338
Understanding the HEAPPOOLS trace LEDATA output.	338
Understanding the C/C++-specific LEDATA output	341
C/C++-specific sections of the LEDATA output	348
Understanding the AUTH LEDATA output.	349
Sections of the AUTH LEDATA Verbexit formatted output	355
Formatting individual control blocks	357
Requesting a Language Environment trace for debugging	359
Locating the trace dump	360
Using the Language Environment trace table format in a dump report	360
Understanding the trace table entry (TTE)	361
Sample dump for the trace table entry	367
Chapter 12. Debugging AMODE 64 C/C++ routines	369
Debugging C/C++ input/output programs	369
Using the __amrc and __amrc2 structures	369
__last_op values.	371
Displaying an error message with the perror() function	375
Using __errno2() to diagnose application problems	375
Using C/C++ listings	376
Generating C/C++ listings and maps	376

I	Finding variables.	378
	Generating a Language Environment dump of a C/C++ routine.	381
	cdump()	381
	csnap()	382
	ctrace()	382
	Sample C routine that calls cdump()	382
	Sample C++ routine that generates a Language Environment dump	384
	Sample Language Environment dump with C/C++-specific information	386
	C/C++ contents of the Language Environment trace tables	390
	Debugging examples of C/C++ routines	392
	Divide-by-zero error.	392
	Calling a nonexistent function	398
	Handling dumps written to the z/OS UNIX file system	403
	Multithreading consideration	404
	Understanding C/C++ heap information in storage reports	404
	Language Environment storage report with HeapPools statistics	405
	C function __uheapreport() storage report	406
	Appendix A. Diagnosing problems with Language Environment	407
	Diagnosis checklist	407
	Locating the name of the failing routine for a non-XPLINK application	408
	Searching the IBM Software Support Database	411
	Preparing documentation for an Authorized Program Analysis Report (APAR)	411
	Appendix B. Accessibility	415
	Using assistive technologies	415
	Keyboard navigation of the user interface.	415
	z/OS information	415
	Notices	417
	Programming Interface Information	419
	Trademarks.	419
	Bibliography	421
	Language Products Publications	421
	Related Publications	422
	Softcopy Publications	423
	Index	425

Figures

I	1. Options report example produced by run-time option RPTOPTS(ON).	11
	2. Storage report produced by run-time option RPTSTG(ON)	13
	3. Storage report produced by RPTSTG(ON) with XPLINK	15
	4. Language Environment condition token	29
	5. The C program CELSAMP	44
	6. The C DLL CELDLL.	47
	7. Example dump using CEE3DMP	48
	8. Upward-growing (non-XPLINK) stack frame format	61
	9. Downward-growing (XPLINK) stack frame format	62
	10. Common anchor area	63
	11. Condition information block	70
	12. Machine state information block	72
	13. Language Environment dump of multiple enclaves	74
	14. Example of formatted output from LEDATA Verbexit	82
	15. Example formatted detailed heap segment report from LEDATA Verbexit	96
	16. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit	101
	17. Example formatted C/C++ output from LEDATA Verbexit	104
	18. Example formatted COBOL output from LEDATA Verbexit	110
	19. The CAA formatted by the CBFORMAT IPCS command	113
	20. Format of the trace table entry	116
	21. Trace table in dump output.	124
	22. __amrc structure	128
	23. __amrc2 structure	128
	24. Example of a routine using perror()	133
	25. Example of a routine using __errno2()	134
	26. Example of a routine using _EDC_ADD_ERRNO2	134
	27. Sample output of a routine using _EDC_ADD_ERRNO2	134
	28. Writable static map produced by prelinker	139
	29. Location of RENT static variable in storage.	140
	30. Writable static map produced by prelinker	140
	31. Location of NORENT static variable in storage	141
	32. Example code for parameter variable	142
	33. Example code for parameter variable	142
	34. Partial storage offset listing	143
	35. Example code for structure variable	143
	36. Example of aggregate map	144
	37. Writable static map produced by prelinker	144
	38. Example C routine using cdump() to generate a dump	147
	39. Fetched module for C routine.	148
	40. Example C++ routine with protection exception generating a dump	149
	41. Template file STACK.C	149
	42. Header file STACK.H	150
	43. Example dump from sample C routine	151
	44. Memory file control block	159
	45. Registers on entry to CEE3DMP	160
	46. Parameters, registers, and variables for active routines	160
	47. Condition information for active routines	161
	48. Sample XPLINK-compiled program (tranmain) which calls a NOXPLINK-compiled program	162
	49. Sample NOXPLINK-compiled program (trandll) which calls an XPLINK-compiled program	163
	50. Example dump of calling between XPLINK and non-XPLINK programs	164
	51. Trace table with C/C++ trace table entry types 1 thru 4	169
	52. Trace table with XPLINK trace table entries 5 and 6.	171
	53. C routine with a divide-by-zero error	172

54. Sections of the dump from example C/C++ routine	173
55. Pseudo assembly listing.	174
56. C/C++ CAA information in dump.	175
57. Writable static map	175
58. Enclave storage section of dump	176
59. C/C++ example of calling a nonexistent subroutine	176
60. Sections of the dump from example C routine.	177
61. Pseudo assembly listing.	178
62. Writable static map	178
63. Enclave control blocks and storage sections in dump	178
64. C/C++ example of calling a nonexistent subroutine	179
65. Sections of the dump from example C routine.	180
66. Pseudo assembly listing.	182
67. Writable static map	182
68. Enclave control blocks and storage sections in dump	183
69. IPCS panel for entering data set information	184
70. Storage report generated by __uheapreport()	187
71. Trace report generated by MEMCHECK VHM.	191
72. Heap Leak Report generated by MEMCHECK VHM	193
73. Example of using the WITH DEBUGGING MODE clause	198
74. COBOL program COBDUMP1 calling COBDUMP2	200
75. COBOL program COBDUMP2 calling the Language Environment dump service CEE3DMP	201
76. Sections of the Language Environment dump called from COBDUMP2	202
77. Control block information for active COBOL routines	204
78. Storage for active COBOL programs	205
79. Enclave-level data for COBOL programs.	206
80. Process-level control blocks for COBOL programs	207
81. COBOL example of moving a value outside an array range.	207
82. Sections of Language Environment dump for COBOLX	208
83. COBOL listing for COBOLX	210
84. COBOL example of calling a nonexistent subroutine	210
85. Sections of Language Environment dump for COBOLY	211
86. COBOL Listing for COBOLY	212
87. Parameters, registers, and variables for active routines section of dump for COBOLY	213
88. Main COBOL program, COBOL subroutine, and assembler routine	214
89. Sections of Language Environment dump for program COBOLZ1	215
90. COBOL listing for COBOLZ2	216
91. Listing for ASSEMZ3	216
92. Variables section of Language Environment dump for COBOLZ2.	217
93. Listing for COBOLZ2	217
94. Variables section of Language Environment dump for COBOLZ1.	217
95. Example program that calls SDUMP	226
96. Language Environment dump generated using SDUMP	227
97. Sections of the Language Environment dump	228
98. Example of calling a nonexistent routine.	229
99. Sections of the Language Environment dump resulting from a call to a nonexistent routine	230
100. FORTRAN routine with a divide-by-zero error.	231
101. Language Environment dump from divide-by-zero FORTRAN example	232
102. PL/I routine compiled with LIST and MAP	239
103. Compiler-generated listings from example PL/I routine	240
104. Traceback section of dump	248
105. Task traceback section	249
106. Control blocks for active routines section of the dump.	250
107. Control blocks associated with the thread section of the dump	252
108. Example of moving a value outside an array range.	254
109. Sections of the Language Environment dump	255

110. Example of calling a nonexistent subroutine	256
111. Sections of the Language Environment dump	257
112. PL/I routine with a divide-by-zero error	258
113. Variables from routine SAMPLE	258
114. Object code listing from example PL/I routine	259
115. Language Environment dump from example PL/I routine	259
116. Language Environment traceback written to the transient data queue	264
117. CICS trace output in the ABBREV format.	266
118. CICS trace output in the FULL format.	267
I 119. 64-bit options report	274
120. 64-bit storage report	276
121. Language Environment condition token	280
122. The C program CELQSAMP	291
123. The C DLL CELQDLL	293
124. Example dump using CEE3DMP	294
125. Example of formatted output from LEDATA Verbexit	312
126. Example formatted detailed heap segment report from LEDATA Verbexit	327
127. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit	339
128. Example formatted C/C++ output from LEDATA Verbexit	342
I 129. Example of formatted AUTH output from LEDATA Verbexit	350
130. The CAA formatted by the CBFORMAT IPCS command	358
131. Format of the trace table entry	361
132. Trace table in dump output.	368
133. __amrc structure	370
134. __amrc2 structure	370
135. Example of a routine using perror()	375
136. Example of a routine using __errno2()	376
137. Example of a routine using _EDC_ADD_ERRNO2	376
138. Sample output of a routine using _EDC_ADD_ERRNO2	376
I 139. Example code for structure variables	381
I 140. Example of aggregate map	381
141. Example C routine using cdump() to generate a dump	383
142. Fetched module for C routine.	384
143. Example C++ routine with protection exception generating a dump	385
144. Template file STACK.C	385
145. Header file STACK.H	386
146. Example dump from sample C routine	387
147. Trace table with XPLINK trace table entries 5 and 6.	391
148. C routine with a divide-by-zero error	393
149. Sections of the dump from example C/C++ routine	394
150. Pseudo assembly listing.	395
151. C/C++ CAA information in dump.	397
152. Writable static map	397
153. IPCS storage display of the writeable static area.	397
154. C/C++ example of calling a nonexistent subroutine	398
155. Sections of the dump from example C routine.	399
156. Pseudo assembly listing.	401
157. Writable static map	403
158. IPCS storage display of the writeable static area.	403
159. IPCS panel for entering data set information	404
160. Storage report generated by __uheapreport()	406
161. C PPA1	410
162. Nonconforming entry point type with sample dump	410

Tables

1.	How to Use z/OS Language Environment Publications	xvi
2.	Syntax examples	xviii
3.	Common error symptoms, possible causes, and programmer responses	32
4.	CEE3DMP options	36
5.	TERMTHDACT suboptions, level of information, and destinations	39
6.	Condition handling of 0Cx abends	42
7.	List of CAA fields	64
8.	Language Environment Control blocks which can be individually formatted	113
9.	LE=1 entry records	117
10.	LE=2 entry records	118
11.	Format of the mutex/CV/latch records	122
12.	LE=8 entry records	123
13.	__last_op values and diagnosis information	129
14.	Contents of listing and associated compiler options	135
15.	XL C compiler listings	136
16.	XL C++ compiler listings	136
17.	XL C/C++ IPA link step listings	137
18.	Finding the WSA base address	138
19.	Compiler-generated COBOL listings and their contents	199
20.	Compiler-generated FORTRAN listings and their contents	221
21.	Compiler-generated PL/I listings and their contents	238
22.	Typical comments in a PL/I static storage listing	241
23.	Comments in a PL/I object code listing	243
24.	PL/I mnemonics	244
25.	Finding data when Language Environment returns a nonzero return code	268
26.	Finding data when Language Environment abends internally	268
27.	Finding data when Language Environment abends from an EXEC CICS command	269
28.	Common error symptoms, possible causes, and programmer responses	284
29.	TERMTHDACT suboptions, level of information, and destinations	287
30.	Language Environment control blocks which can be individually formatted	358
31.	Preinitialized Environments for Authorized Programs control blocks which can be individually formatted	359
32.	LE=1 entry records	362
33.	LE=2 entry records	362
34.	Format of the mutex/CV/latch records	366
35.	LE=8 entry records	367
36.	__last_op values and diagnosis information	372
37.	Contents of listing and associated compiler options	377
38.	XL C compiler listings	378
39.	XL C++ compiler listings	378
40.	XL C/C++ IPA link step listings	378
41.	Problem resolution documentation requirements	412

About this document

This document supports z/OS (5694–A01) and z/OS.e™ (5655–G52).

IBM z/OS Language Environment (also called Language Environment) provides common services and language-specific routines in a single run-time environment for C, C++, COBOL, Fortran (z/OS only; no support for z/OS UNIX System Services or CICS®), PL/I, and assembler applications. It offers consistent and predictable results for language applications, independent of the language in which they are written.

Language Environment is the prerequisite run-time environment for applications generated with the following IBM compiler products:

- z/OS XL C/C++ (feature of z/OS)
- C/C++ for MVS/ESA™
- C/C++ for z/VM
- AD/Cycle® C/370™
- VisualAge for Java, Enterprise Edition for OS/390
- Enterprise COBOL for z/OS
- COBOL for OS/390 & VM
- COBOL for MVS & VM (formerly COBOL/370)
- Enterprise PL/I for z/OS
- PL/I for MVS & VM (formerly PL/I MVS™ & VM)
- VS FORTRAN and FORTRAN IV (in compatibility mode)

Restrictions: The following restrictions apply to z/OS.e:

- The following compilers are not licensed for use on z/OS.e:
 - COBOL
 - PL/I
 - Fortran
- The following subsystems are not licensed for use on z/OS.e:
 - CICS
 - IMS™
- Execution of applications written in the following languages is not functionally supported on z/OS.e:
 - COBOL (except for precompiled COBOL DB2® stored procedures and other precompiled COBOL applications using the Language Environment preinitialization interface)
 - Fortran
- The following are not functional and/or not licensed for use on z/OS.e:
 - Language Environment Library Routine Retention (LRR)
 - Language Environment compatibility preinitialization for C and PL/I
- Customers are not permitted to use lower levels of Language Environment on z/OS.e.

Language Environment supports, but is not required for, an interactive debug tool for debugging applications in your native z/OS environment. The IBM interactive Debug Tool is available with the latest releases of the COBOL, PL/I, and C/C++ compiler products.

Debug Tool is also available as a standalone product. Debug Tool Utilities and Advanced Functions is also available. For more information, see www.ibm.com/software/awdtools/debugtool/

Language Environment supports, but is not required for, VS FORTRAN Version 2 compiled code (z/OS only).

Language Environment consists of the common execution library (CEL) and the run-time libraries for C/C++, COBOL, Fortran, and PL/I.

For more information on VisualAge for Java, Enterprise Edition for OS/390, program number 5655-JAV, see the product documentation.

Using your documentation

The publications provided with Language Environment are designed to help you:

- Manage the run-time environment for applications generated with a Language Environment-conforming compiler.
- Write applications that use the Language Environment callable services.
- Develop interlanguage communication applications.
- Customize Language Environment.
- Debug problems in applications that run with Language Environment.
- Migrate your high-level language applications to Language Environment.

Language programming information is provided in the supported high-level language programming manuals, which provide language definition, library function syntax and semantics, and programming guidance information.

Each publication helps you perform different tasks, some of which are listed in Table 1. All books are available in printable (PDF) and BookManager softcopy formats. For a complete list of publications that you may need, see “Bibliography” on page 421.

Table 1. How to Use z/OS Language Environment Publications

To ...	Use ...
Evaluate Language Environment	<i>z/OS Language Environment Concepts Guide</i>
Plan for Language Environment	<i>z/OS Language Environment Concepts Guide</i> <i>z/OS Language Environment Run-Time Application Migration Guide</i>
Install Language Environment	<i>z/OS Program Directory</i>
Customize Language Environment	<i>z/OS Language Environment Customization</i>
Understand Language Environment program models and concepts	<i>z/OS Language Environment Concepts Guide</i> <i>z/OS Language Environment Programming Guide</i> <i>z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode</i>
Find syntax for Language Environment run-time options and callable services	<i>z/OS Language Environment Programming Reference</i>
Develop applications that run with Language Environment	<i>z/OS Language Environment Programming Guide and your language programming guide</i>

Table 1. How to Use z/OS Language Environment Publications (continued)

To ...	Use ...
Debug applications that run with Language Environment, diagnose problems with Language Environment	<i>z/OS Language Environment Debugging Guide</i>
Get details on run-time messages	<i>z/OS Language Environment Run-Time Messages</i>
Develop interlanguage communication (ILC) applications	<i>z/OS Language Environment Writing Interlanguage Communication Applications</i> and your language programming guide
Migrate applications to Language Environment	<i>z/OS Language Environment Run-Time Application Migration Guide</i> and the migration guide for each Language Environment-enabled language

How to read syntax diagrams

This section describes how to read syntax diagrams. It defines syntax diagram symbols, items that may be contained within the diagrams (keywords, variables, delimiters, operators, fragment references, operands) and provides syntax examples that contain these items.

Syntax diagrams pictorially display the order and parts (options and arguments) that comprise a command statement. They are read from left to right and from top to bottom, following the main path of the horizontal line.

Symbols

The following symbols may be displayed in syntax diagrams:

Symbol	Definition
▶▶—	Indicates the beginning of the syntax diagram.
—▶	Indicates that the syntax diagram is continued to the next line.
▶—	Indicates that the syntax is continued from the previous line.
—▶▶	Indicates the end of the syntax diagram.

Syntax items

Syntax diagrams contain many different items. Syntax items include:

- Keywords - a command name or any other literal information.
- Variables - variables are italicized, appear in lowercase, and represent the name of values you can supply.
- Delimiters - delimiters indicate the start or end of keywords, variables, or operators. For example, a left parenthesis is a delimiter.
- Operators - operators include add (+), subtract (-), multiply (*), divide (/), equal (=), and other mathematical operations that may need to be performed.
- Fragment references - a part of a syntax diagram, separated from the diagram to show greater detail.
- Separators - a separator separates keywords, variables or operators. For example, a comma (,) is a separator.

Note: If a syntax diagram shows a character that is not alphanumeric (for example, parentheses, periods, commas, equal signs, a blank space), enter the character as part of the syntax.

Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

Item type	Definition
Required	Required items are displayed on the main path of the horizontal line.
Optional	Optional items are displayed below the main path of the horizontal line.
Default	Default items are displayed above the main path of the horizontal line.

Syntax examples

The following table provides syntax examples.

Table 2. Syntax examples

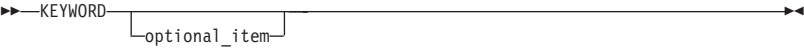
Item	Syntax example
Required item.	
Required items appear on the main path of the horizontal line. You must specify these items.	
Required choice.	
A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.	
Optional item.	
Optional items appear below the main path of the horizontal line.	
Optional choice.	
An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.	
Default.	
Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items.	
Variable.	
Variables appear in lowercase italics. They represent names or values.	

Table 2. Syntax examples (continued)

Item	Syntax example
Repeatable item.	
An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.	
A character within the arrow means you must separate repeated items with that character.	
An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.	
Fragment.	
The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.	fragment:

This Debugging Guide

z/OS Language Environment Debugging Guide provides assistance with detecting and locating programming errors that occur during run time under Language Environment. It can help you establish a debugging process to analyze data and narrow the scope and location of where an error might have occurred. You can read about how to prepare a routine for debugging, how to classify errors, and how to use the debugging facilities Language Environment provides. Also included are chapters on debugging HLL-specific routines and routines that run under CICS. Debugging for AMODE 64 applications is covered in separate chapters, corresponding to the topics and contents provided above.

This book is for application programmers interested in techniques for debugging run-time programs. To use this book, you should be familiar with:

- The Language Environment product
- Appropriate languages that use the compilers listed above
- Program storage concepts

Where to find more information

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

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from these locations to find IBM message explanations for z/OS® elements and features, z/VM®, VSE/ESA™, and Clusters for AIX® and Linux™:

- The Internet. You can access IBM message explanations directly from the LookAt Web site at <http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/>.
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e systems to access IBM message explanations using LookAt from a TSO/E command line (for example: TSO/E prompt, ISPF, or z/OS UNIX® System Services).
- Your Microsoft® Windows® workstation. You can install LookAt directly from the *z/OS Collection* (SK3T-4269) or the *z/OS and Software Products DVD Collection* (SK3T4271) and use it from the resulting Windows graphical user interface (GUI). The command prompt (also known as the DOS > command line) version can still be used from the directory in which you install the Windows version of LookAt.
- Your wireless handheld device. You can use the LookAt Mobile Edition from <http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookatm.html> with a handheld device that has wireless access and an Internet browser (for example: Internet Explorer for Pocket PCs, Blazer or Eudora for Palm OS, or Opera for Linux handheld devices).

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from:

- A CD-ROM in the *z/OS Collection* (SK3T-4269).
- The *z/OS and Software Products DVD Collection* (SK3T4271).
- The LookAt Web site (click **Download** and then select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

Using IBM Health Checker for z/OS

IBM Health Checker for z/OS is a z/OS component that installations can use to gather information about their system environment and system parameters to help identify potential configuration problems before they impact availability or cause outages. Individual products, z/OS components, or ISV software can provide checks that take advantage of the IBM Health Checker for z/OS framework. This book refers to checks or messages associated with this component.

For additional information about checks and about IBM Health Checker for z/OS, see *IBM Health Checker for z/OS: User's Guide*. z/OS V1R4, V1R5, and V1R6 users can obtain the IBM Health Checker for z/OS from the z/OS Downloads page at <http://www.ibm.com/servers/eserver/zseries/zos/downloads/>.

SDSF also provides functions to simplify the management of checks. See *z/OS SDSF Operation and Customization* for additional information.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS and z/OS.e, see the online document at:

publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS

This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

Summary of Changes

Summary of Changes for GA22-7560-06 z/OS Version 1 Release 7

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-05, which supported z/OS Version 1 Release 6.

The following summarizes the changes to that information.

New Information

- Run-time options parmlib and DD statement
- Updates for debugging AMODE 64 applications
- An appendix with z/OS product accessibility information has been added.

References to OpenEdition have been replaced with z/OS UNIX System Services, or z/OS UNIX.

This document also includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this document – for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

Summary of Changes for GA22-7560-05 z/OS Version 1 Release 6

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-04, which supported z/OS Version 1 Release 5.

The following summarizes the changes to that information.

New Information

- New HeapPools Trace option HPT for LEDATA Verbexit.
- Additional trace table entries.
- Chapters 9 through 12 contain information for debugging AMODE 64 applications.

Summary of Changes for GA22-7560-04 z/OS Version 1 Release 5

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-03, which supported z/OS Version 1 Release 4.

This document also includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

**Summary of Changes
for GA22-7560-03
z/OS Version 1 Release 4**

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-02, which supported z/OS Version 1 Release 3.

The following summarizes the changes to that information.

New Information

- Information is added to indicate this document supports z/OS.e.
- Additional Common Anchor Area (CAA) fields have been added. For more information, see Table 7 on page 64.

Changed Information

- The Debugging C/C++ Routines chapter has been updated. For more information, see “Finding variables” on page 137.

This document also includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

**Summary of Changes
for GA22-7560-02
z/OS Version 1 Release 3**

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-01, which supported z/OS Version 1 Release 2.

The following summarizes the changes to that information.

New Information

- Alternative Vendor Heap Manager (VHM) support has been documented.
- An appendix with z/OS product accessibility information has been added.

This document also includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

**Summary of Changes
for GA22-7560-01
z/OS Version 1 Release 2**

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-00, which supported z/OS Version 1 Release 1.

The following summarizes the changes to that information.

New Information

- A new trace function is available which allows the user to format individual control blocks in a dump, rather than having to create the full LEDATA output which contains many formatted control blocks. See “Formatting individual control blocks” on page 112.
- “Understanding the trace table entry (TTE)” on page 116 has been enhanced to provide significant additional information about the content and meaning of trace table entries.
- The HEAPCHK run-time option has a new suboption (*call-level*) which will aid in identifying the cause of heap storage leaks. See “Diagnosing storage leak problems” on page 100.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Part 1. Introduction to debugging in Language Environment

This part provides information about options and features you can use to prepare your routine for debugging. It describes some common errors that occur in routines and provides methods of generating dumps to help you get the information you need to debug your routine.

Chapter 1. Preparing your routine for debugging

This chapter describes options and features that you can use to prepare your routine for debugging. The following topics are covered:

- Compiler options for C, C++, COBOL, Fortran, and PL/I
- Language Environment run-time options
- Use of storage in routines
- Options for modifying condition handling
- Assembler user exits
- Enclave termination behavior
- User-created messages
- Language Environment feedback codes and condition tokens

Setting compiler options

The following sections discuss language-specific compiler options important to debugging routines in Language Environment. These sections cover only the compiler options that are important to debugging. For a complete list of compiler options, refer to the appropriate HLL publications.

The use of some compiler options (such as TEST) can affect the performance of your routine. You must set these options before you compile. In some cases, you might need to remove the option and recompile your routine before delivering your application.

XL C and XL C++ compiler options

When using XL C, set the TEST(ALL) suboption, which is equivalent to TEST(LINE,BLOCK,PATH,SYM,HOOK). For XL C++, the option TEST is equivalent to TEST(HOOK). Following is a list of TEST suboptions that you can use to simplify run-time debugging.

ALL	Sets all of the TEST suboptions.
BLOCK	Generates symbol information for nested blocks.
HOOK	Generates all possible hooks. For details on this suboption, see <i>z/OS XL C/C++ User's Guide</i> .
LINE	Generates line number hooks and allows a debugging tool to generate a symbolic dump.
PATH	Generates hooks at all path points; for example, hooks are inserted at if-then-else points before a function call and after a function call.
SYM	Generates symbol table information and enables Language Environment to generate a dump at run time.

When you specify SYM, you also get the value and type of variables displayed in the Local Variables section of the dump. For example, if in block 4 the variable x is a signed integer of 12, and in block 2 the variable x is a signed integer of 1, the following output appears in the Local Variables section of the dump:

```
%BLOCK4:>x    signed int      12
%BLOCK2:>x    signed int       1
```

If a nonzero optimization level is used, variables do not appear in the dump.

You can use these C/C++ compiler options to make run-time debugging easier:

AGGREGATE (C)	Specifies that a layout for struct and union type variables appear in the listing.
ATTRIBUTE (C++)	For XL C++ compile, cross reference listing with attribute information. If XREF is specified, the listing also contains reference, definition and modification information.
CHECKOUT (C)	Provides informational messages indicating possible programming errors.
EVENTS	Produces an events file that contains error information and source file statistics.
EXPMAC	Macro expansions with the original source.
FLAG	Specifies the minimum severity level that is tolerated.
GONUMBER	Generates line number tables corresponding to the input source file. This option is turned on when the TEST option is used. This option is needed to show statement numbers in dump output.
INFO (C++)	Indication of possible programming errors.
INLINE	Inline Summary and Detailed Call Structure Reports. (Specify with the REPORT sub-option).
INLRPT	Generates a report on status of functions that were inlined. The OPTIMIZE option must also be specified.
LIST	Listing of the pseudo-assembly listing produced by the compiler.
OFFSET	Displays the offset addresses relative to the entry point of each function.
PHASEID	Causes each compiler module (phase) to issue an informational message which identifies the compiler phase module name, product identifier, and build level.
PPONLY	Completely expanded z/OS XL C, or z/OS XL C++ source code, by activating the preprocessor (PP) only. The output shows, for example, all the "#include" and "#define" directives.
SERVICE	Places a string in the object module, which is displayed in the traceback if the application fails abnormally.
SHOWINC	All included text in the listing.
SOURCE	Includes source input statements and diagnostic messages in the listing.
TERMINAL	Directs all error messages from the compiler to the terminal. If not specified, this is the default.
TEST	Generates information for debugging interface. This also generates symbol tables needed for symbolic variables in the dump.
XPLINK (BACKCHAIN)	Generates a prolog that saves redundant information in the calling function's stack frame.
XPLINK (STOREARGS)	Generates code to store arguments that are normally passed in registers, into the argument area.
XREF	For XL C compile, cross reference listing with reference, definition, and modification information. For XL C++ compile, cross reference listing with reference, definition, and modification information. If you specify ATTRIBUTE, the listing also contains attribute information.

For a detailed explanation of these options, see *z/OS XL C/C++ User's Guide*.

IPA compile step sub-options

You can use the following IPA compile sub-options to prepare your program for run-time debugging:

ATTRIBUTE | NOATTRIBUTE

Indicates whether the compiler generates information in the IPA object file that will be used in the IPA Link step 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 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.

The default is IPA(NOATTRIBUTE). The abbreviations are IPA(ATTRINOATTR). 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.

The default is IPA(NOOGONUMBER). The abbreviations are IPA(GONUMINOOGONUM). If you specify the GONUMBER or LIST option, it overrides the IPA(NOOGONUMBER) 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 a Pseudo-Assembly listing to be generated during IPA Compile step code generation.

The default is IPA(NOLIST). The abbreviations are IPA(LISINOLIS). If you specify the GONUMBER or LIST option, it overrides the IPA(NOLIST) option.

XREF | NOXREF

Indicates whether the compiler generates information 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.

The default is IPA(NOXREF). The abbreviations are IPA(XRINOXR). If you specify the ATTR or XREF option, it overrides the IPA(NOXREF) option.

IPA link step sub-options

You can use these IPA Link Step sub-options to prepare your program for run-time debugging:

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).

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).

Refer to the Inter-procedural Analysis chapter in the *z/OS XL C/C++ Programming Guide* for an overview and more details about Inter-procedural Analysis.

COBOL compiler options

When using COBOL, set the SYM suboption of the TEST compiler option. The SYM suboption of TEST causes the compiler to add debugging information into the object program to resolve user names in the routine and to generate a symbolic dump of the DATA DIVISION. With this suboption specified, statement numbers will also be used in the dump output along with offset values.

To simplify debugging, use the NOOPTIMIZE compiler option. Program optimization can change the location of parameters and instructions in the dump output.

You can use the following COBOL compiler options to prepare your program for run-time debugging:

LIST	Produces a listing of the assembler expansion of your source code and global tables, literal pools, information about working storage, and size of routine's working storage.
MAP	Produces lists of items in data division including a data division map, global tables, literal pools, a nested program structure map, and attributes.
OFFSET	Produces a condensed PROCEDURE DIVISION listing containing line numbers, statement references, and location of the first instruction generated for each statement.
OUTDD	Specifies the destination of DISPLAY statement messages.
SOURCE	Produces a listing of your source program with any statements embedded by PROCESS, COPY, or BASIS statements.
TEST	Produces object code that can run with a debugging tool, or adds information to the object program to produce formatted dumps. With or without any suboptions, this option forces the OBJECT option. When specified with any of the hook-location suboption values except NONE, this option forces the NOOPTIMIZE option. SYM suboption includes statement numbers in the Language Environment dump and produces a symbolic dump.

VBREF	Produces a cross-reference of all verb types used in the source program and a summary of how many times each verb is used.
XREF	Creates a sorted cross-reference listing.

For more detail on these options and functions, see *Enterprise COBOL for z/OS Programming Guide* or *COBOL for OS/390 & VM Programming Guide*

Fortran compiler options

You can use these Fortran compiler options to prepare your program for run-time debugging:

FIPS	Specifies whether standard language flagging is to be performed. This is valuable if you want to write a program conforming to FORTRAN 77.
FLAG	Specifies the level of diagnostic messages to be written. I (Information), E (Error), W (Warning), or S (Severe). You can also use FLAG to suppress messages that are below a specified level. This is useful if you want to suppress information messages, for example.
GOSTMT	Specifies that statement numbers are included in the run-time messages and in the Language Environment dump.
ICA	Specifies whether intercompilation analysis is to be performed, specifies the files containing intercompilation analysis information to be used or updated, and controls output from intercompilation analysis. Specify ICA when you have a group of programs and subprograms that you want to process together and you need to know if there are any conflicting external references, mismatched commons, and so on.
LIST	Specifies whether the object module list is to be written. The LIST option lets you see the pseudo-assembly language code that is similar to what is actually generated.
MAP	Specifies whether a table of source program variable names, named constants, and statement labels and their displacements is to be produced.
OPTIMIZE	Specifies the optimizing level to be used during compilation. If you are debugging your program, it is advisable to use NOOPTIMIZE.
SDUMP	Specifies whether dump information is to be generated.
SOURCE	Specifies whether a source listing is to be produced.
SRCFLG	Controls insertion of error messages in the source listing. SRCFLG allows you to view error messages after the initial line of each source statement that caused the error, rather than at the end of the listing.
SXM™	Formats SREF or MAP listing output to a 72-character width.
SYM	Invokes the production of SYM cards in the object text file. SYM cards contain location information for variables within a Fortran program.
TERMINAL	Specifies whether error messages and compiler diagnostics are to be written on the SYSTERM data set and whether a summary of messages for all the compilations is to be written at the end of the listing.
TEST	Specifies whether to override any optimization level above OPTIMIZE(0). This option adds run-time overhead.
TRMFLG	Specifies whether to display the initial line of source statements in error and their associated error messages at the terminal.
XREF	Creates a cross-reference listing.
VECTOR	Specifies whether to invoke the vectorization process. A vectorization report provides detailed information about the vectorization process.

For more detail on these options and functions, see *VS FORTRAN Version 2 Programming Guide for CMS and MVS* or *VS FORTRAN Version 2 Language and Library Reference*.

PL/I compiler options

When using PL/I, specify the TEST compiler option to control the level of testing capability that are generated as part of the object code. Suboptions of the TEST option such as SYM, BLOCK, STMT, and PATH control the location of test hooks and specify whether or not a symbol table is generated. For more information about TEST, its suboptions, and the placement of test hooks, see *PL/I for MVS & VM Programming Guide*.

To simplify debugging and decrease compile time, set optimization to NOOPTIMIZE or OPTIMIZE(0). Higher optimization levels can change the location where parameters and instructions appear in the dump output.

You can use these compiler options to prepare PL/I routines for debugging:

AGGREGATE	Specifies that a layout for arrays and major structures appears in the listing.
ESD	Includes the external symbol dictionary in the listing.
GONUMBER / GOSTMT	Tells the compiler to produce additional information specifying that line numbers from the source routine can be included in run-time messages and in the Language Environment dump.
INTERRUPT	Specifies that users can establish an ATTENTION ON-unit that gains control when an attention interrupt occurs.
LIST	Produces a listing of the assembler expansion of source code and global tables, literal pools, information about working storage, and size of routine's working storage.
LMESSAGE	Tells the compiler to produce run-time messages in a long form. If the cause of a run-time malfunction is a programmer's understanding of language semantics, specifying LMESSAGE could better explain warnings or other information generated by the compiler.
MAP	Tells the compiler to produce tables showing how the variables are mapped in the static internal control section and in the stack frames, thus enabling static internal and automatic variables to be found in the Language Environment dump. If LIST is also specified, the MAP option also produces tables showing constants, control blocks, and INITIAL variable values.
OFFSET	Specifies that the compiler prints a table of statement or line numbers for each procedure with their offset addresses relative to the primary entry point of the procedure.
SOURCE	Specifies that the compiler includes a listing of the source routine in the listing.
STORAGE	Includes a table of the main storage requirements for the object module in the listing.
TERMINAL	Specifies what parts of the compiler listing produced during compilation are directed to the terminal.
TEST	Specifies the level of testing capability that is generated as part of the object code. TEST also controls the location of test hooks and whether or not the symbol table is generated. Because the TEST option increases the size of the object code and can affect performance, limit the number and placement of hooks.
XREF and ATTRIBUTES	Creates a sorted cross-reference listing with attributes.

For more detail on PL/I compiler options, see *PL/I for MVS & VM Programming Guide*.

VisualAge PL/I compiler options

The following VisualAge PL/I compiler options can be specified when preparing your VisualAge PL/I routines for debugging:

AGGREGATE	Specifies that a layout for arrays and major structures appears in the listing.
GONUMBER / GOSTMT	Tells the compiler to produce additional information specifying that line numbers from the source routine can be included in run-time messages and in the Language Environment dump.
INTERRUPT	Specifies that users can establish an ATTENTION ON-unit that gains control when an attention interrupt occurs.
LIST	Produces a listing of the assembler expansion of source code and global tables, literal pools, information about working storage, and size of routine's working storage.
OFFSET	Displays the offset addresses relative to the entry point of each function.
SOURCE	Specifies that the compiler includes a listing of the source routine in the listing.
STORAGE	Includes a table of the main storage requirements for the object module in the listing.
TEST	Specifies the level of testing capability that is generated as part of the object code. TEST also controls the location of test hooks and whether or not the symbol table is generated. Because the TEST option increases the size of the object code and can affect performance, limit the number and placement of hooks.
XREF and ATTRIBUTES	Creates a sorted cross-reference listing with attributes.

For further details on the VisualAge PL/I compiler options, see *VisualAge PL/I for OS/390 Programming Guide*.

Using Language Environment run-time options

There are several run-time options that affect debugging in Language Environment. The TEST run-time option, for example, can be used with a debugging tool to specify the level of control the debugging tool has when the routine being initialized is started. The ABPERC, CHECK, DEPTHCONDLMT, ERRCOUNT, HEAPCHK, INTERRUPT, TERMTHDACT, TRACE, TRAP, and USRHDLR options affect condition handling. The ABTERMENC option affects how an application ends (that is, with an abend or with a return code and reason code) when an unhandled condition of severity 2 or greater occurs.

The following Language Environment run-time options affect debugging:

ABPERC	Specifies that the indicated abend code bypasses the condition handler.
ABTERMENC	Specifies enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater.
CHECK	Determines whether run-time checking is performed.
NODEBUG	Controls the COBOL USE FOR DEBUGGING declarative.
DEPTHCONDLMT	Specifies the limit for the depth of nested synchronous conditions in user-written condition handlers. (Asynchronous signals do not affect DEPTHCONDLMT.)
ERRCOUNT	Specifies the number of synchronous conditions of severity 2 or greater tolerated. (Asynchronous signals do not affect ERRCOUNT.)
HEAPCHK	Determines whether additional heap check tests are performed.

INFOMSGFILTER	Filters user specified informational messages from the MSGFILE. Note: Affects only those messages generated by Language Environment® and any routine that calls CEEMSG. Other routines that write to the message file, such as CEEMOUT, do not have a filtering option.
INTERRUPT	Causes Language Environment to recognize attention interrupts.
MSGFILE	Specifies the ddname of the Language Environment message file.
MSGQ	Specifies the number of instance specific information (ISI) blocks that are allocated on a per-thread basis for use by an application. Located within the Language Environment condition token is an ISI token. The ISI contains information used by the condition manager to identify and react to a specific occurrence of a condition.
PROFILE	Controls the use of an optional profiler tool, which collects performance data for the running application. When this option is in effect, the profiler is loaded and the debugger cannot be loaded. If the TEST option is in effect when PROFILE is specified, the profiler tool will not be loaded.
RPTOPTS	Causes a report to be produced which contains the run-time options in effect. See “Determining run-time options in effect” below.
RPTSTG	Generates a report of the storage used by an enclave. See “Controlling storage allocation” on page 12.
STORAGE	Specifies that Language Environment initializes all heap and stack storage to a user-specified value.
TERMTHDACT	Controls response when an enclave terminates due to an unhandled condition of severity 2 or greater.
TEST	Specifies the conditions under which a debugging tool assumes control.
TRACE	Activates Language Environment run-time library tracing and controls the size of the trace table, the type of trace, and whether the trace table should be dumped unconditionally upon termination of the application.
TRAP	When TRAP is set to ON, Language Environment traps routine interrupts and abends, and optionally prints trace information or invokes a user-written condition handling routine. With TRAP set to OFF, the operating system handles all interrupts and abends. You should generally set TRAP to ON, or your run-time results can be unpredictable.
USRHDLR	Specifies the behavior of two user-written condition handlers. The first handler specified will be registered at stack frame 0. The second handler specified will be registered before any other user-written condition handlers, once the handler is enabled by a condition.
XUFLOW	Specifies whether or not an exponent underflow causes a routine interrupt.

For a more detailed discussion of these run-time options, see *z/OS Language Environment Programming Reference* .

Determining run-time options in effect

The run-time options in effect at the time the routine is run can affect routine behavior. Use RPTOPTS(ON) to generate an options report in the Language Environment message file when your routine terminates. The options report lists run-time options, and indicates where they were set.

Figure 1 on page 11 shows a sample options report.

Options Report for Enclave main 10/13/04 3:25:50 PM
Language Environment V01 R07.00

LAST WHERE SET	OPTION
Installation default	ABPERC(NONE)
Installation default	ABTERMENC(ABEND)
Installation default	NOAIXBLD
Installation default	ALL31(ON)
Installation default	ANYHEAP(16384,8192,ANYWHERE,FREE)
Installation default	NOAUTOTASK
Installation default	BELOWHEAP(8192,4096,FREE)
Installation default	CBLOPTS(ON)
Installation default	CBLPSHPOP(ON)
Installation default	CBLQDA(OFF)
Installation default	CHECK(ON)
Installation default	COUNTRY(US)
Installation default	NODEBUG
Installation default	DEPTHCONDLMT(10)
Installation default	ENVAR("")
PARMLIB(CEEPRM01)	ENVAR("A1A=Test 1","B1B=Test 2")
Installation default	ERRCOUNT(0)
Installation default	ERRUNIT(6)
Installation default	FILEHIST
Installation default	FILETAG(NOAUTOCVT,NOAUTOTAG)
Default setting	NOFLOW
PARMLIB(CEEPRMPV)	HEAP(4194304,5242880,ANYWHERE,KEEP,8192,4096)
Installation default	HEAPCHK(OFF,1,0,0,0)
Invocation command	HEAPPOLLS(OFF,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,10,0,10,0,10,0,10,0,10)
Installation default	INFMSGFILTER(OFF,,,))
Installation default	INQPCOPN
Installation default	INTERRUPT(OFF)
Installation default	LIBSTACK(4096,4096,FREE)
Installation default	MSGFILE(SYSOUT,FBA,121,0,NOENQ)
Installation default	MSGQ(15)
Installation default	NATLANG(ENU)
Ignored	NONONIPSTACK(See THREADSTACK)
Installation default	OCSTATUS
Installation default	NOPC
Installation default	PLITASKCOUNT(20)
Invocation command	POSIX(ON)
PARMLIB(CEEPRM01)	PROFILE(OFF,"XXX")
Installation default	PRTUNIT(6)
Installation default	PUNUNIT(7)
Installation default	RDRUNIT(5)
Installation default	RECPAD(OFF)
DD:CEEOPTS	RPTOPTS(ON)
SETCEE command	RPTSTG(ON)
Installation default	NORTEREUS
Installation default	NOSIMVRD
Installation default	STACK(131072,131072,ANYWHERE,KEEP,524288,131072)
PARMLIB(CEEPRMPV)	STORAGE(AA,BB,NONE,0)
Installation default	TERMTHDACT(TRACE,,96)
Installation default	NOTEST(ALL,"*","PROMPT","INSPREF")
PARMLIB(CEEPRMPV)	THREADHEAP(8192,10240,ANYWHERE,KEEP)
Installation default	THREADSTACK(OFF,4096,4096,ANYWHERE,KEEP,131072,131072)
Installation default	TRACE(OFF,4096,DUMP,LE=0)
Installation default	TRAP(ON,SPIE)
Installation default	UPSI(00000000)
Installation default	NOUSRHDRLR(,)
Installation default	VCTRSAVE(OFF)
Installation default	XPLINK(OFF)
Installation default	XUFLOW(AUTO)

Figure 1. Options report example produced by run-time option RPTOPTS(ON)

Using the CLER CICS transaction to display and set run-time options

The CICS transaction (CLER) allows you to display all the current Language Environment run-time options for a region, and to also have the capability to modify a subset of these options.

The following run-time options can be modified with the CICS CLER transaction:

- TRAP(ONIOFF)
- TERMTHDACT(QUIETIMSGITRACEIDUMPIUAONLYIUATRACEIUADUMPIUAIMM)
- RPTOPTS(ONIOFF)
- RPTSTG(ONIOFF)
- ALL31(ONIOFF)
- CBLPSHPOP(ONIOFF)

For more information about the CICS CLER transaction, see “Displaying and modifying run-time options with the CLER transaction” on page 269.

Controlling storage allocation

The following run-time options control storage allocation:

- STACK
- THREADSTACK
- LIBSTACK
- THREADHEAP
- HEAP
- ANYHEAP
- BELOWHEAP
- STORAGE
- HEAPPOOLS

z/OS Language Environment Programming Guide provides useful tips to assist with the tuning process. Appropriate tuning is necessary to avoid performance problems.

To generate a report of the storage a routine (or more specifically, an enclave) used during its run, specify the RPTSTG(ON) run-time option. The storage report, generated during enclave termination provides statistics that can help you understand how space is being consumed as the enclave runs. If storage management tuning is desired, the statistics can help you set the corresponding storage-related run-time options for future runs. The output is written to the Language Environment message file.

Neither the storage report nor the corresponding run-time options include the storage that Language Environment acquires during early initialization, before run-time options processing, and before the start of space management monitoring. In addition, Language Environment does not report alternative Vendor Heap Manager activity.

Figure 2 on page 13 and Figure 3 on page 15 show sample storage reports. The sections that follow these reports describe the contents of the reports.

Storage Report for Enclave main 03/18/04 5:17:20 PM
Language Environment V01 R06.00

```
STACK statistics:
  Initial size:                4096
  Increment size:              4096
  Maximum used by all concurrent threads: 7488
  Largest used by any thread:  7488
  Number of segments allocated: 2
  Number of segments freed:    0
THREADSTACK statistics:
  Initial size:                4096
  Increment size:              4096
  Maximum used by all concurrent threads: 3352
  Largest used by any thread:  3352
  Number of segments allocated: 6
  Number of segments freed:    0
LIBSTACK statistics:
  Initial size:                4096
  Increment size:              4096
  Maximum used by all concurrent threads: 0
  Largest used by any thread:    0
  Number of segments allocated: 0
  Number of segments freed:    0
THREADHEAP statistics:
  Initial size:                4096
  Increment size:              4096
  Maximum used by all concurrent threads: 0
  Largest used by any thread:    0
  Successful Get Heap requests:  0
  Successful Free Heap requests: 0
  Number of segments allocated: 0
  Number of segments freed:    0
HEAP statistics:
  Initial size:                49152
  Increment size:              16384
  Total heap storage used (sugg. initial size): 29112
  Successful Get Heap requests:  251
  Successful Free Heap requests:  218
  Number of segments allocated:  1
  Number of segments freed:    0
HEAP24 statistics:
  Initial size:                8192
  Increment size:              4096
  Total heap storage used (sugg. initial size): 0
  Successful Get Heap requests:  0
  Successful Free Heap requests:  0
  Number of segments allocated:  0
  Number of segments freed:    0
ANYHEAP statistics:
  Initial size:                32768
  Increment size:              16384
  Total heap storage used (sugg. initial size): 104696
  Successful Get Heap requests:  28
  Successful Free Heap requests:  15
  Number of segments allocated:  6
  Number of segments freed:    5
```

Figure 2. Storage report produced by run-time option RPTSTG(ON) (Part 1 of 2)

BELOWHEAP statistics:	
Initial size:	8192
Increment size:	8192
Total heap storage used (sugg. initial size):	0
Successful Get Heap requests:	0
Successful Free Heap requests:	0
Number of segments allocated:	0
Number of segments freed:	0
Additional Heap statistics:	
Successful Create Heap requests:	1
Successful Discard Heap requests:	1
Total heap storage used:	4912
Successful Get Heap requests:	3
Successful Free Heap requests:	3
Number of segments allocated:	2
Number of segments freed:	2
Largest number of threads concurrently active:	2
End of Storage Report	

Figure 2. Storage report produced by run-time option RPTSTG(ON) (Part 2 of 2)

Storage Report for Enclave main 03/18/04 5:18:13 PM
Language Environment V01 R06.00

```
STACK statistics:
  Initial size: 4096
  Increment size: 4096
  Maximum used by all concurrent threads: 4256
  Largest used by any thread: 4256
  Number of segments allocated: 2
  Number of segments freed: 0
THREADSTACK statistics:
  Initial size: 4096
  Increment size: 4096
  Maximum used by all concurrent threads: 11688
  Largest used by any thread: 2584
  Number of segments allocated: 6
  Number of segments freed: 0
XPLINK STACK statistics:
  Initial size: 524288
  Increment size: 131072
  Largest used by any thread: 4480
  Number of segments allocated: 1
  Number of segments freed: 0
XPLINK THREADSTACK statistics:
  Initial size: 131072
  Increment size: 131072
  Largest used by any thread: 3072
  Number of segments allocated: 6
  Number of segments freed: 0
LIBSTACK statistics:
  Initial size: 4096
  Increment size: 4096
  Maximum used by all concurrent threads: 0
  Largest used by any thread: 0
  Number of segments allocated: 0
  Number of segments freed: 0
THREADHEAP statistics:
  Initial size: 4096
  Increment size: 4096
  Maximum used by all concurrent threads: 0
  Largest used by any thread: 0
  Successful Get Heap requests: 0
  Successful Free Heap requests: 0
  Number of segments allocated: 0
  Number of segments freed: 0
HEAP statistics:
  Initial size: 32768
  Increment size: 32768
  Total heap storage used (sugg. initial size): 59056
  Successful Get Heap requests: 29
  Successful Free Heap requests: 13
  Number of segments allocated: 2
  Number of segments freed: 0
```

Figure 3. Storage report produced by RPTSTG(ON) with XPLINK (Part 1 of 4)

```

HEAP24 statistics:
  Initial size:                        8192
  Increment size:                     4096
  Total heap storage used (sugg. initial size): 0
  Successful Get Heap requests:        0
  Successful Free Heap requests:        0
  Number of segments allocated:        0
  Number of segments freed:            0
ANYHEAP statistics:
  Initial size:                       16384
  Increment size:                     8192
  Total heap storage used (sugg. initial size): 1024672
  Successful Get Heap requests:        33
  Successful Free Heap requests:        15
  Number of segments allocated:        10
  Number of segments freed:            8
BELOWHEAP statistics:
  Initial size:                       8192
  Increment size:                     4096
  Total heap storage used (sugg. initial size): 0
  Successful Get Heap requests:        0
  Successful Free Heap requests:        0
  Number of segments allocated:        0
  Number of segments freed:            0
Additional Heap statistics:
  Successful Create Heap requests:      1
  Successful Discard Heap requests:      1
  Total heap storage used:              4912
  Successful Get Heap requests:         3
  Successful Free Heap requests:         3
  Number of segments allocated:         2
  Number of segments freed:             2
HeapPools Statistics:
  Pool 1 size: 8
    Successful Get Heap requests: 1- 8 8
  Pool 2 size: 32
    Successful Get Heap requests: 9- 16 3
    Successful Get Heap requests: 17- 24 5
    Successful Get Heap requests: 25- 32 3
  Pool 3 size: 128
    Successful Get Heap requests: 33- 40 3
    Successful Get Heap requests: 41- 48 3
    Successful Get Heap requests: 49- 56 11
    Successful Get Heap requests: 57- 64 4
    Successful Get Heap requests: 65- 72 3
    Successful Get Heap requests: 73- 80 4
    Successful Get Heap requests: 81- 88 5
    Successful Get Heap requests: 89- 96 4
    Successful Get Heap requests: 97- 104 4
    Successful Get Heap requests: 113- 120 5
    Successful Get Heap requests: 121- 128 4

```

Figure 3. Storage report produced by RPTSTG(ON) with XPLINK (Part 2 of 4)

```

Pool 4 size: 256
Successful Get Heap requests: 129- 136      6
Successful Get Heap requests: 137- 144      3
Successful Get Heap requests: 145- 152      4
Successful Get Heap requests: 153- 160      2
Successful Get Heap requests: 161- 168      8
Successful Get Heap requests: 169- 176      5
Successful Get Heap requests: 177- 184      4
Successful Get Heap requests: 185- 192      6
Successful Get Heap requests: 193- 200      3
Successful Get Heap requests: 201- 208      4
Successful Get Heap requests: 209- 216      2
Successful Get Heap requests: 217- 224      3
Successful Get Heap requests: 225- 232      4
Successful Get Heap requests: 233- 240      2
Successful Get Heap requests: 241- 248      2
Successful Get Heap requests: 249- 256      1
Pool 5 size: 1024
Successful Get Heap requests: 257- 264      4
Successful Get Heap requests: 265- 272      1
Successful Get Heap requests: 273- 280      3
Successful Get Heap requests: 281- 288      2
Successful Get Heap requests: 289- 296      2
Successful Get Heap requests: 305- 312      6
Successful Get Heap requests: 313- 320      5
Successful Get Heap requests: 321- 328      4
Successful Get Heap requests: 329- 336      2
Successful Get Heap requests: 337- 344      3
Successful Get Heap requests: 353- 360      2
Successful Get Heap requests: 361- 368      4
Successful Get Heap requests: 369- 376      5
Successful Get Heap requests: 377- 384      2
Successful Get Heap requests: 385- 392      2
Successful Get Heap requests: 393- 400      2
Successful Get Heap requests: 401- 408      5
Successful Get Heap requests: 409- 416      3
Successful Get Heap requests: 417- 424      2
Successful Get Heap requests: 425- 432      1
Successful Get Heap requests: 433- 440      2
Successful Get Heap requests: 441- 448      4
Successful Get Heap requests: 457- 464      1
Successful Get Heap requests: 465- 472      1
Successful Get Heap requests: 473- 480      2
Successful Get Heap requests: 481- 488      1
Successful Get Heap requests: 489- 496      2
Successful Get Heap requests: 497- 504      5
Successful Get Heap requests: 505- 512      2
Successful Get Heap requests: 545- 552      1
Successful Get Heap requests: 577- 584      1
Successful Get Heap requests: 641- 648      2
Successful Get Heap requests: 825- 832      1
Successful Get Heap requests: 913- 920      1

```

Figure 3. Storage report produced by RPTSTG(ON) with XPLINK (Part 3 of 4)

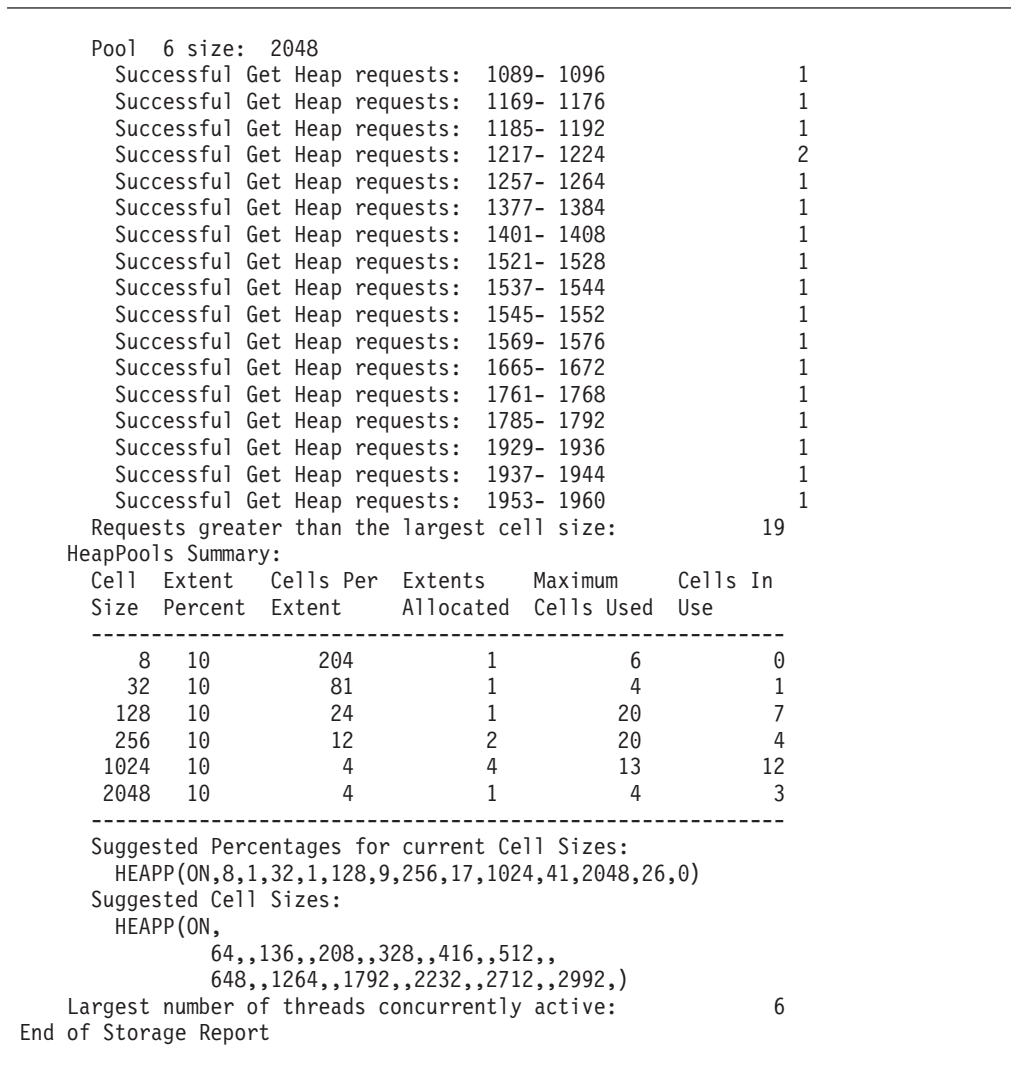


Figure 3. Storage report produced by RPTSTG(ON) with XPLINK (Part 4 of 4)

The statistics for initial and incremental allocations of storage types that have a corresponding run-time option differ from the run-time option settings when their values have been rounded up by the implementation, or when allocations larger than the amounts specified were required during execution. All of the following are rounded up to an integral number of double-words:

- Initial STACK allocations
- Initial allocations of THREADSTACK
- Initial allocations of all types of heap
- Incremental allocations of all types of stack and heap

The run-time options should be tuned appropriately to avoid performance problems. Refer to *z/OS Language Environment Programming Guide* for tips on tuning.

Stack storage statistics

Language Environment stack storage is managed at the thread level—each thread has its own stack-type resources.

STACK, THREADSTACK, LIBSTACK, and IPT STACK statistics for the upward-growing stack

- Initial size — the actual size of the initial segment assigned to each thread. (If a pthread-attributes-table is provided on the invocation of pthread-create, then the stack size specified in the pthread-attributes-table will take precedence over the STACK run-time option.)
- Increment size — the size of each incremental segment acquired, as determined by the increment portion of the corresponding run-time option.
- Maximum used by all concurrent threads — the maximum amount allocated in total at any one time by all concurrently executing threads.
- Largest used by any thread — the largest amount allocated ever by any single thread.
- Number of segments (or increments) allocated — the number of incremental segments allocated by all threads.
- Number of segments freed — the number of incremental segments freed by all threads.

The number of incremental segments freed could be less than the number allocated if any of the segments were not freed individually during the life of the thread, but rather were freed implicitly in the course of thread termination.

XPLINK statistics — XPLINK STACK and XPLINK THREADSTACK statistics for the downward-growing stack

These sections of the storage report only apply if XPLINK is in effect.

- Initial size — the actual size of the initial segment assigned to each thread.
- Increment size — the size of each incremental segment acquired, as determined by the increment portion of the corresponding run-time option.
- Maximum used by all concurrent threads — the maximum amount allocated in total at any one time by all concurrently executing threads.
- Largest used by any thread — the largest amount allocated ever by any single thread.
- Number of segments allocated — the number of incremental segments allocated by all threads.
- Number of segments freed — the number of incremental segments freed by all threads.

The number of incremental segments freed could be less than the number allocated if any of the segments were not freed individually during the life of the thread, but rather were freed implicitly in the course of thread termination.

Determining the applicable threads

If the application is not a multithreading or PL/I multitasking application, then the STACK statistics are for the one and only thread that executed, and the THREADSTACK statistics are all zero.

If the application is a multithreading or PL/I multitasking application, and THREADSTACK(ON) was specified, then the STACK statistics are for the initial thread (the IPT), and the THREADSTACK statistics are for the other threads. However, if THREADSTACK(OFF) was specified, then the STACK statistics are for all of the threads, initial and other.

Allocating stack storage

Another type of stack, called the reserve stack, is allocated for each thread and used to handle out-of-storage conditions. Its size is controlled by the 4th subparameter of the STORAGE run-time option, but its usage is neither tracked nor reported in the storage report.

In a single-threaded environment, Language Environment allocates the initial segments for STACK, LIBSTACK and reserve stack using GETMAIN. The LIBSTACK initial segment allocation is deferred until first use, except when STACK(„BELOW,,) is in effect. The reserve stack is allocated with STACK. In a multi-threaded POSIX(ON) environment, allocation of stack storage for the initial processing thread (IPT) is the same as the single-threaded environment. For threads other than the IPT, the initial STACK (or THREADSTACK) segment and reserve stack is allocated from ANYHEAP or BELOWHEAP, according to the STACK (or THREADSTACK) location. The initial LIBSTACK segment allocation is again deferred until first use, except when STACK(„BELOW,,) or THREADSTACK(ON,,,BELOW,,) is in effect. When a STACK, THREADSTACK, or LIBSTACK overflow occurs on any thread, Language Environment obtains the new segment using GETMAIN. The reserve stack does not tolerate overflow.

Heap storage statistics

Language Environment heap storage, other than what is explicitly defined using THREADHEAP, is managed at the enclave level—each enclave has its own heap-type resources, which are shared by the threads that execute within the enclave. Heap storage defined using THREADHEAP is controlled at the thread level.

THREADHEAP statistics

- Initial size—the default initial allocation, as specified by the corresponding run-time option. Please note that for HEAP24, the initial size is the value of the initsz24 of the HEAP option.
- Increment size—the minimum incremental allocation, as specified by the corresponding run-time option. Please note that for HEAP24, the increment size is the value of the incrsz24 of the HEAP option.
- Maximum used by all concurrent threads—the maximum total amount used by all concurrent threads at any one time.
- Largest used by any thread—the largest amount used by any single thread.
- Successful Get Heap requests—the number of Get Heap requests.
- Successful Free Heap requests—the number of Free Heap requests.
- Number of segments allocated—the number of incremental segments allocated.
- Number of segments freed—the number of incremental segments individually freed.

These statistics, in all cases, specify totals for the whole enclave. For THREADHEAP, they indicate the total across all threads in the enclave.

HEAP, HEAP24, ANYHEAP, and BELOWHEAP statistics

- Initial size—the default initial allocation, as specified by the corresponding run-time option. Please note that for HEAP24, the initial size is the value of the initsz24 of the HEAP option.
- Increment size—the minimum incremental allocation, as specified by the corresponding run-time option. Please note that for HEAP24, the increment size is the value of the incrsz24 of the HEAP option.

- Total heap storage used—the largest total amount used by the enclave at any one time.
- Successful Get Heap requests—the number of Get Heap requests.
- Successful Free Heap requests—the number of Free Heap requests.
- Number of segments allocated—the number of incremental segments allocated.
- Number of segments freed—the number of incremental segments individually freed.

The number of Free Heap requests could be less than the number of Get Heap requests if the pieces of heap storage acquired by individual Get Heap requests were not freed individually, but rather were freed implicitly in the course of enclave termination.

The number of incremental segments individually freed could be less than the number allocated if the segments were not freed individually, but rather were freed implicitly in the course of enclave termination.

These statistics, in all cases, specify totals for the whole enclave. For THREADHEAP, they indicate the total across all threads in the enclave.

Additional heap statistics

Besides the fixed types of heap, additional types of heap can be created, each with its own heap ID. You can create and discard these additional types of heap by using the CEECRHP

- Successful Create Heap requests—the number of successful Create Heap requests.
- Successful Discard Heap requests—the number of successful Discard Heap requests.

The number of Discard Heap requests could be less than the number of Create Heap requests if the special heaps allocated by individual Create Heap requests were not freed individually, but rather were freed implicitly in the course of enclave termination.

- Total heap storage used—the largest total amount used by the enclave at any one time.
- Successful Get Heap requests—the number of Get Heap requests.
- Successful Free Heap requests—the number of Free Heap requests.
- Number of segments allocated—the number of incremental segments allocated.
- Number of segments freed—the number of incremental segments individually freed.

HeapPools storage statistics

The HEAPPOOLS run-time option (for C/C++ applications only) controls usage of the heap pools storage algorithm at the enclave level. The heap pools algorithm allows for the definition of one to twelve heap pools, each consisting of a number of storage cells of a specified length. For further details regarding HeapPools storage statistics in the storage report, see “HeapPools storage statistics” on page 185.

Modifying condition handling behavior

Setting the condition handling behavior of your routine affects the response that occurs when the routine encounters an error.

You can modify condition handling behavior in the following ways:

- Callable services
- Run-time options
- User-written condition handlers
- POSIX functions (used to specifically set signal actions and signal masks)

Language Environment callable services

You can use these callable services to modify condition handling:

CEE3ABD	Terminates an enclave using an abend.
CEEMRCE	Moves the resume cursor to an explicit location where resumption is to occur after a condition has been handled.
CEEMRCR	Moves the resume cursor relative to the current position of the handle cursor.
CEE3CIB	Returns a pointer to a condition information block (CIB) associated with a given condition token. The CIB contains detailed information about the condition.
CEE3GRO	Returns the offset of the location within the most current Language Environment-conforming routine where a condition occurred.
CEE3SPM	Specifies the settings of the routine mask. The routine mask controls: • Fixed overflow • Decimal overflow • Exponent underflow • Significance
	You can use CEE3SPM to modify Language Environment hardware conditions. Because such modifications can affect the behavior of your routine, however, you should be careful when doing so.
CEE3SRP	Sets a resume point within user application code to resume from a Language Environment user condition handler.

Fortran programs cannot directly call Language Environment callable services. For more information about callable services, see *z/OS Language Environment Programming Reference*. For more information about how to invoke callable services from Fortran, see *Language Environment Fortran Run-Time Migration Guide*.

Language Environment run-time options

These Language Environment run-time options can affect your routine's condition handling behavior:

ABPERC	Specifies a system- or user-specified abend code that percolates without further action while the Language Environment condition handler is enabled. Normal condition handling activities are performed for everything except the specified abend code. System abends are specified as Shhh, where hhh is a hexadecimal system abend code. User abends are specified as Udddd, where dddd is a decimal user abend code. Any other 4-character EBCDIC string, such as NONE, that is not of the form Shhh can also be specified as a user-specified abend code. You can specify only one abend code with this option. This option assumes the use of TRAP(ON). ABPERC is not supported in CICS. Language Environment ignores ABPERC(0Cx). No abend is percolated and Language Environment condition handling semantics are in effect.
CHECK	Specifies that checking errors within an application are detected. The Language Environment-conforming languages can define error checking differently.
DEPTHCONDLMT	Limits the extent to which synchronous conditions can be nested in a user-written condition handler. (Asynchronous signals do not affect DEPTHCONDLMT .) For example, if you specify 5, the initial condition and four nested conditions are processed. If the limit is exceeded, the application terminates with abend code 4091 and reason code 21 (X'15').
ERRCOUNT	Specifies the number of synchronous conditions of severity 2, 3, and 4 that are tolerated before the enclave terminates abnormally. (Asynchronous signals do not affect ERRCOUNT .) If you specify 0 an unlimited number of conditions is tolerated.
INTERRUPT	Causes attentions recognized by the host operating system to be passed to and recognized by Language Environment after the environment has been initialized.
TERMTHDACT	Sets the level of information that is produced when a condition of severity 2 or greater remains unhandled within the enclave. There are five possible parameter settings for different levels of information: • QUIET for no information • MSG for message only • TRACE for message and a traceback • DUMP for message, traceback, and Language Environment dump • UAONLY for message and a system dump of the user address space • UATRACE for message, Language Environment dump with traceback information only, and a system dump of the user address space • UADUMP for message, traceback, Language Environment dump, and system dump • UAIMM for a system dump of the user address space of the original abend or program interrupt prior to the Language Environment condition manager processing the condition.

TRAP(ON)	Fully enables the Language Environment condition handler. This causes the Language Environment condition handler to intercept error conditions and routine interrupts. When TRAP(ON, NOSPIE) is specified, Language Environment handles all program interrupts and abends through an ESTAE. Use this feature when you do not want Language Environment to issue an ESPIE macro. During normal operation, you should use TRAP(ON) when running your applications.
TRAP(OFF)	Disables the Language Environment condition handler from handling abends and program checks/interrupts. ESPIE is not issued with TRAP(OFF), it is still possible to invoke the condition handler through the CEESGL callable service and pass conditions to registered user-written condition handlers. Specify TRAP(OFF) when you do not want Language Environment to issue an ESTAE or an ESPIE. When TRAP(OFF), TRAP(OFF,SPIE), or TRAP(OFF,NOSPIE) is specified and either a program interrupt or abend occurs, the user exit for termination is ignored. TRAP(OFF) can cause several unexpected side effects. For further information, see the TRAP run-time option in <i>z/OS Language Environment Programming Reference</i>.
USRHDLR	Specifies the behavior of two user-written condition handlers. The first handler specified will be registered at stack frame 0. The second handler specified will be registered before any other user-written condition handlers, once the handler is enabled by a condition. When you specify USRHDLR(lastname,supername), lastname gets control at stack frame 0. Supername will get control first, before any user-written condition handlers but after supername has gone through the enablement phase, when a condition occurs.
XUFLOW	Specifies whether an exponent underflow causes a routine interrupt.

Customizing condition handlers

User-written condition handlers permit you to customize condition handling for certain conditions. You can register a user-written condition handler for the current stack frame by using the CEEHDLR callable service. You can use the Language Environment USRHDLR run-time option to register a user-written condition handler for stack frame 0. You can also use USRHDLR to register a user-written condition handler before any other user condition handlers.

When the Language Environment condition manager encounters the condition, it requests that the condition handler associated with the current stack frame handle the condition. If the condition is not handled, the Language Environment condition manager percolates the condition to the next (earlier) stack frame, and so forth to earlier stack frames until the condition has been handled. Conditions that remain unhandled after the first (earliest) stack frame has been reached are presented to the Language Environment condition handler. One of the following Language Environment default actions is then taken, depending on the severity of the condition:

- Resume
- Percolate
- Promote

- Fix-up and resume

For more information about user-written condition handlers and the Language Environment condition manager, see *z/OS Language Environment Programming Guide*.

Invoking the assembler user exit

For debugging purposes, the CEEBXITA assembler user exit can be invoked during:

- Enclave initialization
- Enclave termination
- Process termination

The functions of the CEEBXITA user exit depend on when the user exit is invoked and whether it is application-specific or installation-wide. Application-specific user exits must be linked with the application load module and run only when that application runs. Installation-wide user exits must be linked with the Language Environment initialization/termination library routines and run with all Language Environment library routines. Because an application-specific user exit has priority over any installation-wide user exit, you can customize a user exit for a particular application without affecting the user exit for any other applications.

At enclave initialization, the CEEBXITA user exit runs prior to the enclave establishment. Thus you can modify the environment in which your application runs in the following ways:

- Specify run-time options
- Allocate data sets/files in the user exit
- List abend codes to be passed to the operating system
- Check the values of routine arguments

At enclave termination, the CEEBXITA user exit runs prior to the termination activity. Thus, you can request an abend and perform specified actions based on received return and reason codes. (This does not apply when Language Environment terminates with an abend.)

At process termination, the CEEBXITA user exit runs after the enclave termination activity completes. Thus you can request an abend and deallocate files.

The assembler user exit must have an entry point of CEEBXITA, must be reentrant, and must be capable of running in AMODE(ANY) and RMODE(ANY).

You can use the assembler user exit to establish enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater in the following ways:

- If you do not request an abend in the assembler user exit for the enclave termination call, Language Environment honors the setting of the ABTERMENC option to determine how to end the enclave.
- If you request an abend in the assembler user exit for the enclave termination call, Language Environment issues an abend to end the enclave.

For more information on the assembler user exit, see *z/OS Language Environment Programming Guide*.

Establishing enclave termination behavior for unhandled conditions

To establish enclave termination behavior when an unhandled condition of severity 2 or greater occurs, use one of the following methods:

- The assembler user exit (see “Invoking the assembler user exit” on page 25 and *z/OS Language Environment Programming Guide*)
- POSIX signal default action (see *z/OS Language Environment Programming Guide*)
- The ABTERMENC run-time option (discussed below)

The ABTERMENC run-time option sets the enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater.

If you specify the IBM-supplied default suboption ABEND, Language Environment issues an abend to end the enclave regardless of the setting of the CEEAUE_ABND flag. Additionally, the assembler user exit can alter the abend code, abend reason code, abend dump attribute, and the abend task/step attribute. For more information on using ABTERMENC, see *z/OS Language Environment Programming Reference*, and for more information on the assembler user exit, see *z/OS Language Environment Programming Guide*.

If you specify the RETCODE suboption, Language Environment uses the CEEAUE_ABND flag value set by the assembler user exit (which is called for enclave termination) to determine whether or not to issue an abend to end the enclave when an unhandled condition of severity 2 or greater occurs.

Using messages in your routine

You can create messages and use them in your routine to indicate the status and progress of the routine during run time, and to display variable values. The process of creating messages and using them requires that you create a message source file, and convert the source file into loadable code for use in your routine.

You can use the Language Environment callable service CEEMOUT to direct user-created message output to the Language Environment message file. To direct the message output to another destination, use the Language Environment MSGFILE run-time option to specify the ddname of the file.

When multiple Language Environment environments are running in the same address space and the same MSGFILE ddname is specified, writing contention can occur. To avoid contention, use the MSGFILE suboption ENQ. ENQ tells Language Environment to perform serialization around writes to the MSGFILE ddname specified which eliminates writing contention. Writing contention can also be eliminated by specifying unique MSGFILE ddnames.

Each Language Environment-conforming language also provides ways to display both user-created and run-time messages. (For an explanation of Language Environment run-time messages, see “Interpreting run-time messages” on page 32.)

The following sections discuss how to create messages in each of the HLLs. For a more detailed explanation of how to create messages and use them in C, C++, COBOL, Fortran, or PL/I routines, see *z/OS Language Environment Programming Guide*.

C/C++

For C/C++ routines, output from the `printf` function is directed to `stdout`, which is associated with `SYSPRINT`. All C/C++ run-time messages and `perror()` messages are directed to `stderr`. `stderr` corresponds to the `ddname` associated with the Language Environment `MSGFILE` run-time option. The destination of the `printf` function output can be changed by using the redirection `1>&2` at routine invocation to redirect `stdout` to the `stderr` destination. Both streams can be controlled by the `MSGFILE` run-time option.

COBOL

For COBOL programs, you can use the `DISPLAY` statement to display messages. Output from the `DISPLAY` statement is directed to `SYSOUT`. `SYSOUT` is the IBM-supplied default for the Language Environment message file. The `OUTDD` compiler option can be used to change the destination of the `DISPLAY` messages.

Fortran

For Fortran programs, run-time messages, output written to the print unit, and other output (such as output from the `SDUMP` callable service) are directed to the file specified by the `MSGFILE` run-time option. If the print unit is different than the error message unit (`PRTUNIT` and `ERRUNIT` run-time options have different values), however, output from the `PRINT` statement won't be directed to the Language Environment message file.

PL/I

Under PL/I, run-time messages are directed to the file specified in the Language Environment `MSGFILE` run-time option, instead of the PL/I `SYSPRINT STREAM PRINT` file. User-specified output is still directed to the PL/I `SYSPRINT STREAM PRINT` file. To direct this output to the Language Environment `MSGFILE` file, specify the run-time option `MSGFILE(SYSPRINT)`.

Using condition information

If a condition that might require attention occurs while an application is running, Language Environment builds a condition token. The condition token contains 12 bytes (96 bits) of information about the condition that Language Environment or your routines can use to respond appropriately. Each condition is associated with a single Language Environment run-time message.

You can use this condition information in two primary ways:

- To specify the feedback code parameter when calling Language Environment services (see “Using the feedback code parameter”).
- To code a symbolic feedback code in a user-written condition handler (see “Using the symbolic feedback code” on page 29).

Using the feedback code parameter

The feedback code is an optional parameter of the Language Environment callable services. (For COBOL/370 programs, you must provide the **fc** parameter in each call to a Language Environment callable service. For C/C++, Enterprise COBOL for z/OS, COBOL for OS/390 & VM, COBOL for MVS & VM, and PL/I routines, this parameter is optional. For more information about **fc** and condition tokens, see *z/OS Language Environment Programming Guide*.)

When you provide the feedback code (**fc**) parameter, the callable service in which the condition occurs sets the feedback code to a specific value called a condition token.

The condition token does not apply to asynchronous signals. For a discussion of the distinctions between synchronous signals and asynchronous signals with POSIX(ON), see *z/OS Language Environment Programming Guide*.

When you do not provide the **fc** parameter, any nonzero condition is signaled and processed by Language Environment condition handling routines. If you have registered a user-written condition handler, Language Environment passes control to the handler, which determines the next action to take. If the condition remains unhandled, Language Environment writes a message to the Language Environment message file. The message is the translation of the condition token into English (or another supported national language).

Language Environment provides callable services that can be used to convert condition tokens to routine variables, messages, or signaled conditions. The following table lists these callable services and their functions.

CEEMSG	Transforms the condition token into a message and writes the message to the message file.
CEEMGET	Transforms the condition token into a message and stores the message in a buffer.
CEEDCOD	Decodes the condition token; that is, separates it into distinct user-supplied variables. Also, if a language does not support structures, CEEDCOD provides direct access to the token.
CEESGL	Signals the condition. This passes control to any registered user-written condition handlers. If a user-written condition handler does not exist, or the condition is not handled, Language Environment by default writes the corresponding message to the message file and terminates the routine for severity 2 or higher. For severity 0 and 1, Language Environment continues without writing a message. COBOL, however, issues severity 1 messages before continuing. CEESGL can signal a POSIX condition. For details, see <i>z/OS Language Environment Programming Guide</i> .

There are two types of condition tokens. Case 1 condition tokens contain condition information, including the Language Environment message number. All Language Environment callable services and most application routines use case 1 condition tokens. Case 2 condition tokens contain condition information and a user-specified class and cause code. Application routines, user-written condition handlers, assembler user exits, and some operating systems can use case 2 condition tokens.

0 - 31	32 - 33	34 - 36	37 - 39	40 - 63	64 - 95
Condition_ID	Case Number	Severity Number	Control Code	Facility_ID	ISI

For Case 1 condition tokens,
Condition_ID is:

0 - 15	16 - 31
Severity Number	Message Number

For Case 2 condition tokens,
Condition_ID is:

0 - 15	16 - 31
Class Code	Cause Code

A symbolic feedback code represents the first 8 bytes of a condition token. It contains the Condition_ID, Case Number, Severity Number, Control Code, and Facility_ID, whose bit offsets are indicated.

Figure 4. Language Environment condition token

For example, in the condition token: X'0003032D 59C3C5C5 00000000'

- X'0003' is severity.
- X'032D' is message number 813.
- X'59' are hexadecimal flags for case, severity, and control.
- X'C3C5C5' is the CEE facility ID.
- X'00000000' is the ISI. (In this case, no ISI was provided.)

If a Language Environment traceback or dump is generated while a condition token is being processed or when a condition exists, Language Environment writes the run-time message to the condition section of the traceback or dump. If a condition is detected when a callable service is invoked without a feedback code, the condition token is passed to the Language Environment condition manager. The condition manager polls active condition handlers for a response. If a condition of severity 0 or 1 remains unhandled, Language Environment resumes without issuing a message. Language Environment does issue messages, however, for COBOL severity 1 conditions. For unhandled conditions of severity 2 or greater, Language Environment issues a message and terminates. For a list of Language Environment run-time messages and corrective information, see *z/OS Language Environment Run-Time Messages*.

If a second condition is raised while Language Environment is attempting to handle a condition, the message CEE0374C CONDITION = <message no.> is displayed using a write-to-operator (WTO). The message number in the CEE0374C message indicates the original condition that was being handled when the second condition was raised. This can happen when a critical error is signaled (for example, when internal control blocks are damaged).

If the output for this error message appears several times in sequence, the conditions appear in order of occurrence. Correcting the earliest condition can cause your application to run successfully.

Using the symbolic feedback code

The symbolic feedback code represents the first 8 bytes of a 12-byte condition token. You can think of the symbolic feedback code as the nickname for a condition. As such, the symbolic feedback code can be used in user-written condition handlers to screen for a given condition, even if it occurs at different locations in an application.

For more details on symbolic feedback codes, see *z/OS Language Environment Programming Guide*.

Chapter 2. Classifying errors

This chapter describes errors that commonly occur in Language Environment routines. It also explains how to use run-time messages and abend codes to obtain information about errors in your routine.

Identifying problems in routines

The following sections describe how you can identify errors in Language Environment routines. Included are common error symptoms and solutions.

Language Environment module names

You can identify Language Environment-supplied module elements by any of the following three-character prefixes:

- CEE (Language Environment)
- EDC (C/C++)
- FOR (Fortran)
- IBM (PL/I)
- IGZ (COBOL)

Module elements or text files with other prefixes are not part of the Language Environment product.

Common errors in routines

These common errors have simple solutions:

- If you do not have enough virtual storage, increase your region size or decrease your storage usage (stack size) by using the storage-related run-time options and callable services. (See “Controlling storage allocation” on page 12 for information about using storage in routines.)
- If you do not have enough disk space, increase your disk allocation.
- If executable files are not available, check your executable library to ensure that they are defined. For example, check your STEPLIB or JOBLIB definitions.

If your error is not caused by any of the items listed above, examine your routine or routines for changes since the last successful run. If there have been changes, review these changes for errors that might be causing the problem. One way to isolate the problem is to branch around or comment out recent changes and rerun the routine. If the run is successful, the error can be narrowed to the scope of the changes.

Duplicate names shared between Fortran routines and C library routines can produce unexpected results. Language Environment provides several cataloged procedures to properly resolve duplicate names. For more information on how to avoid name conflicts, see *z/OS Language Environment Programming Guide*.

Changes in optimization levels, addressing modes, and input/output file formats can also cause unanticipated problems in your routine.

In most cases, generated condition tokens or run-time messages point to the nature of the error. The run-time messages offer the most efficient corrective action. To help you analyze errors and determine the most useful method to fix the problem, Table 3 on page 32 lists common error symptoms, possible causes, and programmer responses.

Table 3. Common error symptoms, possible causes, and programmer responses

Error Symptom	Possible Cause	Programmer Response
Numbered run-time message appears	Condition raised in routine	For any messages you receive, read the Programmer Response. For information about message structure, see “Interpreting run-time messages” below.
User abend code < 4000	a) A non-Language Environment abend occurred b) The assembler user exit requested an abend for an unhandled condition of severity ≥ 2	See the Language Environment abend codes in <i>z/OS Language Environment Run-Time Messages</i> . Check for a subsystem-generated abend or a user-specified abend.
User abend code ≥ 4000	a) Language Environment detected an error and could not proceed b) An unhandled software-raised condition occurred and ABTERMENC(ABEND) was in effect c) The assembler user exit requested an abend for an unhandled condition of severity 4	For any abends you receive, read the appropriate explanation listed in the abend codes section of <i>z/OS Language Environment Run-Time Messages</i> .
System abend with TRAP(OFF)	Cause depends on type of malfunction	Respond appropriately. Refer to the messages and codes book of the operating system.
System abend with TRAP(ON)	System-detected error	Refer to the messages and codes book of the operating system.
No response (wait/loop)	Application logic failure	Check routine logic. Ensure ERRCOUNT and DEPTHCONDLMT run-time options are set to a nonzero value.
Unexpected message (message received was not from most recent service)	Condition caused by something related to current service	Generate a traceback using CEE3DMP.
Incorrect output	Incorrect file definitions, storage overlay, incorrect routine mask setting, references to uninitialized variables, data input errors, or application routine logic error	Correct the appropriate parameters.
No output	Incorrect ddname, file definitions, or message file setting	Correct the appropriate parameters.
Nonzero return code from enclave	Unhandled condition of severity 2, 3, or 4, or the return code was issued by the application routine	Check the Language Environment message file for run-time message.
Unexpected output	Conflicting library module names	Refer to the name conflict resolution steps outlined in <i>z/OS Language Environment Programming Guide</i> .

Interpreting run-time messages

The first step in debugging your routine is to look up any run-time messages. To find run-time messages, check the message file:

- On z/OS, run-time messages are written by default to ddname SYSOUT. If SYSOUT is not specified, then the messages are written to SYSOUT=*.
- On CICS, the run-time messages are written to the CESE transient data QUEUE.

The default message file ddname can be changed by using the MSGFILE run-time option. For information about displaying run-time messages for C/C++, COBOL, Fortran, or PL/I routines, see *z/OS Language Environment Programming Guide*.

Run-time messages provide users with additional information about a condition, and possible solutions for any errors that occurred. They can be issued by Language Environment common routines or language-specific run-time routines and contain a message prefix, message number, severity code, and descriptive text.

In the following example Language Environment message:

CEE3206S The system detected a specification exception.

- The message prefix is CEE.
- The message number is 3206.
- The severity code is S.
- The message text is “The system detected a specification exception”.

Language Environment messages can appear even though you made no explicit calls to Language Environment services. C/C++, COBOL, and PL/I run-time library routines commonly use the Language Environment services. This is why you can see Language Environment messages even when the application routine does not directly call common run-time services.

Message prefix

The message prefix indicates the Language Environment component that generated the message. The message prefix is the first three characters of the message number and is also the facility ID in the condition token. See the following table for more information about Language Environment run-time messages.

Message Prefix	Language Environment Component
CEE	Common run time
EDC	C/C++ run time
FOR	Fortran run time
IBM	PL/I run time
IGZ	COBOL run time

The messages for the various components can be found in *z/OS Language Environment Run-Time Messages*.

Message number

The message number is the 4-digit number following the message prefix. Leading zeros are inserted if the message number is less than four digits. It identifies the condition raised and references additional condition and programmer response information.

Severity code

The severity code is the letter following the message number and indicates the level of attention called for by the condition. Messages with severity of I are informational messages and do not usually require any corrective action. In general, if more than one run-time message appears, the first noninformational message indicates the problem. For a complete list of severity codes, severity values, condition information, and default actions, see *z/OS Language Environment Programming Guide*.

Message text

The message text provides a brief explanation of the condition.

Understanding abend codes

Under Language Environment, abnormal terminations generate abend codes. There are two types of abend codes: 1) user (Language Environment and user-specified) abends and 2) system abends. User abends follow the format of *Udddd*, where *dddd* is a decimal user abend code. System abends follow the format of *Shhh*, where *hhh* is a hexadecimal abend code. Language Environment abend codes are usually in the range of 4000 to 4095. However, some subsystem abend codes can also fall in this range. User-specified abends use the range of 0 to 3999.

Example abend codes are:

User (Language Environment) abend code:U4041
User-specified abend code:U0005
System abend code:S80A

The Language Environment callable service CEE3ABD terminates your application with an abend. You can set the `clean-up` parameter value to determine how the abend is processed and how Language Environment handles the raised condition. For more information about CEE3ABD and `clean-up`, see *z/OS Language Environment Programming Reference*.

You can specify the ABTERMENC run-time option to determine what action is taken when an unhandled condition of severity 2 or greater occurs. For more information on ABTERMENC, see “Establishing enclave termination behavior for unhandled conditions” on page 26, as well as *z/OS Language Environment Programming Reference*.

User abends

If you receive a Language Environment abend code, see *z/OS Language Environment Run-Time Messages* for a list of abend codes, error descriptions, and programmer responses.

System abends

If you receive a system abend code, look up the code and the corresponding information in the publications for the system you are using.

When a system abend occurs, the operating system can generate a system dump. System dumps are written to ddname SYSMDUMP, SYSABEND, or SYSUDUMP. System dumps show the memory state at the time of the condition. See “Generating a system dump” on page 75 for more information about system dumps.

Chapter 3. Using Language Environment debugging facilities

This chapter describes methods of debugging routines in Language Environment. Currently, most problems in Language Environment and member language routines can be determined through the use of a debugging tool or through information provided in the Language Environment dump.

Debug tool

Debug tools are designed to help you detect errors early in your routine. IBM offers Debug Tool, a comprehensive compile, edit, and debug product that is provided with the C/C++, Enterprise COBOL for z/OS, COBOL for OS/390 & VM, COBOL for MVS & VM, PL/I for MVS & VM, VisualAge PL/I, and VisualAge for Java compiler products.

You can use the IBM Debug Tool to examine, monitor, and control how your routines run, and debug your routines interactively or in batch mode. Debug Tool also provides facilities for setting breakpoints and altering the contents and values of variables. Language Environment run-time options can be used with Debug Tool to debug or analyze your routine. Refer to the Debug Tool publications for a detailed explanation of how to invoke and run Debug Tool.

You can also use **dbx** to debug Language Environment applications, including C/C++ programs. *z/OS UNIX System Services Command Reference* has information on **dbx** subcommands, while *z/OS UNIX System Services Programming Tools* contains usage information.

Language Environment dump service, CEE3DMP

The following sections provide information about using the Language Environment dump service, and describe the contents of the Language Environment dump.

Below is a list of methods to invoke the Language Environment dump service:

- CEE3DMP callable service (non-64-bit only)
- TERMTHDACT run-time option
- HLL-specific functions

Generating a Language Environment dump with CEE3DMP

For non-64-bit, the CEE3DMP callable service generates a dump of the run-time environment for Language Environment and the member language libraries at the point of the CEE3DMP call. You can call CEE3DMP directly from an application routine.

Depending on the CEE3DMP options you specify, the dump can contain information about conditions, tracebacks, variables, control blocks, stack and heap storage, file status and attributes, and language-specific information.

All output from CEE3DMP is written to the default ddname CEEDUMP. CEEDUMP, by default, sends the output to the SDSF output queue. You can direct the output from the CEEDUMP to a specific sysout class by using the environment variable, `_CEE_DMPTARG=SYSOUT(x)`, where x is the output class.

Under z/OS UNIX, if the application is running in an address-space created as a result of a `fork()`, `spawn()`, `spawnp()`, `vfork()`, or one of the `exec` family of functions, then the CEE3DMP is placed in the HFS in one of the following directories in the specified order:

1. the directory found in environment variable `_CEE_DMPTARG`, if found
2. the current working directory, if this is not the root directory (`/`), and the directory is writable
3. the directory found in environment variable `TMPDIR` (an environment variable that indicates the location of a temporary directory if it is not `/tmp`)
4. the `/tmp` directory.

The syntax for CEE3DMP is:

Syntax

```
►► CEE3DMP (—title—, —options—, —fc—) ►►
```

title

An 80-byte fixed-length character string that contains a title that is printed at the top of each page of the dump.

options

A 255-byte fixed-length character string that contains options describing the type, format, and destination of dump information. The options are declared as a string of keywords separated by blanks or commas. Some options also have suboptions that follow the option keyword, and are contained in parentheses. The last option declaration is honored if there is a conflict between it and any preceding options.

fc (output)

A 12-byte feedback token code that indicates the result of a call to CEE3DMP. If specified as an argument, feedback information, in the form of a condition token, is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Following is a list of CEE3DMP options and related information:

Table 4. CEE3DMP options

Dump Options	Abbreviation	Action Taken
ENCLAVE(ALL)	ENCL	Dumps all enclaves associated with the current process. (In ILC applications in which a C/C++ routine calls another member language routine, and that routine in turn calls CEE3DMP, traceback information for the C/C++ routine is not provided in the dump.) This is the default setting for ENCLAVE.
ENCLAVE(CURRENT)	ENCL(CUR)	Dumps the current enclave.
ENCLAVE(n)	ENCL(n)	Dumps a fixed number of enclaves, indicated by <i>n</i> .

Table 4. CEE3DMP options (continued)

Dump Options	Abbreviation	Action Taken
THREAD(ALL)	THR(ALL)	Dumps all threads in this enclave (including in a PL/I multitasking environment).
THREAD(CURRENT)	THR(CUR)	Dumps the current thread in this enclave.
TRACEBACK	TRACE	Includes a traceback of all active routines. The traceback shows transfer of control from either calls or exceptions. Calls include PL/I transfers of control from BEGIN-END blocks or ON-units.
NOTRACEBACK	NOTRACE	Does not include a traceback of all active routines.
FILES	FILE	Includes attributes of all open files. File control blocks are included when the BLOCKS option is also specified. File buffers are included when the STORAGE option is specified.
NOFILES	NOFILE	Does not include file attributes.
VARIABLES	VAR	Includes a symbolic dump of all variables, arguments, and registers.
NOVARIABLES	NOVAR	Does not include variables, arguments, and registers.
BLOCKS	BLOCK	Dumps control blocks from Language Environment and member language libraries. Global control blocks, as well as control blocks associated with routines on the call chain, are printed. Control blocks are printed for the routine that called CEE3DMP. The dump proceeds up the call chain for the number of routines specified by the STACKFRAME option (see below). Control blocks for files are also dumped if the FILES option was specified. See the FILES option above for more information. If the TRACE run-time option is set to ON, the trace table is dumped if BLOCKS is specified. If the Heap Storage Diagnostics report is requested via the HEAPCHK run-time option, the report is displayed when BLOCKS is specified.
NOBLOCKS	NOBLOCK	Does not include control blocks.
STORAGE	STOR	Dumps the storage used by the routine. The number of routines dumped is controlled by the STACKFRAME option.
NOSTORAGE	NOSTOR	Suppresses storage dumps.
STACKFRAME(ALL)	SF(ALL)	Dumps all stack frames from the call chain. This is the default setting for STACKFRAME.

Table 4. CEE3DMP options (continued)

Dump Options	Abbreviation	Action Taken
STACKFRAME(n)	SF(n)	Dumps a fixed number of stack frames, indicated by <i>n</i> , from the call chain. The specific information dumped for each stack frame depends on the VARIABLES, BLOCK, and STORAGE options declarations. The first stack frame dumped is the caller of CEE3DMP, followed by its caller, and proceeding backward up the call chain.
PAGESIZE(n)	PAGE(n)	Specifies the number of lines on each page of the dump.
FNAME(s)	FNAME(s)	Specifies the ddname of the file to which the dump is written.
CONDITION	COND	Dumps condition information for each condition active on the call chain.
NOCONDITION	NOCOND	For each condition active on the call chain, does not dump condition information.
ENTRY	ENT	Includes a description of the program unit that called CEE3DMP and the registers on entry to CEE3DMP.
NOENTRY	NOENT	Does not include a description of the program unit that called CEE3DMP or registers on entry to CEE3DMP.
GENOPTS	GENO	Generate a run-time options report in the dump output. This will be the default if an unhandled condition occurs, and a CEEDUMP is generated due to the setting of the TERMTHDACT run-time option setting.
NOGENOPTS	NOGENO	Do not generate a run-time options report in the dump output. NOGENOPTS is the default for user-called dumps.
REGSTOR(reg_stor_amount)	REGST(reg_stor_amount)	Controls the amount of storage to be dumped around registers. Default is 96 bytes. Specify REGSTOR(0) if no storage around registers is required.
Note: On CICS, only ENCLAVE(CURRENT) and ENCLAVE(1) settings are supported.		

The IBM-supplied default settings for CEE3DMP are:

```
ENCLAVE(ALL) TRACEBACK
THREAD(CURRENT) FILES VARIABLES NOBLOCKS NOSTORAGE
STACKFRAME(ALL) PAGESIZE(60) FNAME(CEEDUMP)
CONDITION ENTRY NOGENOPTS REGSTOR(96)
```

For additional information about the CEE3DMP callable service and dump options, see *z/OS Language Environment Programming Reference*. For an example of a Language Environment dump, see “Understanding the Language Environment dump” on page 43.

Generating a Language Environment dump with TERMTHDACT

The TERMTHDACT run-time option produces a dump during program checks, abnormal terminations, or calls to the CEESGL service. You must use TERMTHDACT(DUMP) in conjunction with TRAP(ON) to generate a Language Environment dump.

You can use TERMTHDACT to produce a traceback, Language Environment dump, or user address space dump when a thread ends abnormally because of an unhandled condition of severity 2 or greater. If this is the last thread in the process, the enclave goes away. A thread terminating in a non-POSIX environment is analogous to an enclave terminating because Language Environment Version 1 supports only single threads. For information on enclave termination, see *z/OS Language Environment Programming Guide*.

The TERMTHDACT suboptions QUIET, MSG, TRACE, DUMP, UAONLY, UATRACE, UADUMP, and UAIMM control the level of information available. Following are the suboptions, the levels of information produced, and the destination of each.

Table 5. TERMTHDACT suboptions, level of information, and destinations

Suboption	Level of Information	Destination
QUIET	No information	No destination.
MSG	Message	Terminal or ddname specified in MSGFILE run-time option.
TRACE	Message and Language Environment dump containing only a traceback	Message goes to terminal or ddname specified in MSGFILE run-time option. Traceback goes to CEEDUMP file.
DUMP	Message and complete Language Environment dump	Message goes to terminal or ddname specified in MSGFILE run-time option. Language Environment dump goes to CEEDUMP file.
UAONLY	SYSMDUMP, SYSABEND dump, or SYSUDUMP depending on the DD card used in the JCL in z/OS. In CICS, a transaction dump is created. In non-CICS you will get a system dump of your user address space if the appropriate DD statement is used. Note: A Language Environment dump is not generated.	Language Environment generates a U4039 abend which allows a system dump of the user address space to be generated. For z/OS, the system dump is written to the ddname specified; for CICS the transaction dump goes to DFHDMPA or the DFHDMPB data set.
UATRACE	Message, Language Environment dump containing only a traceback, and a system dump of the user address space	Message goes to terminal or ddname specified in MSGFILE run-time option. Traceback goes to CEEDUMP file. Language Environment generates a U4039 abend which allows a system dump of the user address space to be generated. For z/OS, the system dump is written to the ddname specified; for CICS the transaction dump goes to DFHDMPA or the DFHDMPB data set.

Table 5. TERMTHDACT suboptions, level of information, and destinations (continued)

Suboption	Level of Information	Destination
UADUMP	Message, Language Environment dump, and SYSMDUMP, SYSABEND dump, or SYSUDUMP depending on the DD card used in the JCL in z/OS. In CICS, a transaction dump is created.	Message goes to terminal or ddname specified in MSGFILE run-time option. Language Environment dump goes to CEEDUMP file. Language Environment generates a U4039 abend which allows a system dump of the user address space to be generated. For z/OS, the system dump is written to the ddname specified; for CICS the transaction dump goes to DFHDMPA or the DFHDMPB data set.
UAIMM	Language Environment generates a system dump of the original abend/program interrupt of the user address space. In CICS, a transaction dump is created. In non-CICS you will get a system dump of your user address space if the appropriate DD statement is used. After the dump is taken by the operating system, Language Environment condition manager continues processing. Note: Under CICS, UAIMM yields UAONLY behavior. Under non-CICS, TRAP(ON,NOSPIE) must be in effect. When TRAP(ON,SPIE) is in effect, UAIMM yields UAONLY behavior. For software raised conditions or signals, UAIMM behaves the same as UAONLY.	Message goes to terminal or ddname specified in MSGFILE run-time option. User address space dump goes to ddname specified for z/OS; or a CICS transaction dump goes to the DFHDMPA or DFHDMPB data set.

The TRACE and UATRACE suboptions of TERMTHDACT use these dump options:

- ENCLAVE(ALL)
- NOENTRY
- CONDITION
- TRACEBACK
- THREAD(ALL)
- NOBLOCKS
- NOSTORAGE
- VARIABLES
- FILES
- STACKFRAME(ALL)
- PAGESIZE(60)
- FNAME(CEEDUMP)
- GENOPTS
- REGSTOR(96)

The DUMP and UADUMP suboptions of TERMTHDACT use these dump options:

- ENCLAVE(ALL)
- NOENTRY
- CONDITION
- TRACEBACK
- THREAD(CURRENT)

- BLOCKS
- STORAGE
- VARIABLES
- FILES
- STACKFRAME(ALL)
- PAGESIZE(60)
- FNAME(CEEDUMP)
- GENOPTS
- REGSTOR(96)

Although you can modify CEE3DMP options, you cannot change options for a traceback or dump produced by TERMTHDACT.

Considerations for setting TERMTHDACT options

The output of TERMTHDACT may vary depending upon which languages and subsystems are processing the request. This section describes the considerations associated with issuing the TERMTHDACT suboptions.

- COBOL Considerations
 - The following TERMTHDACT suboptions for COBOL are recommended, UAONLY, UATRACE, and UADUMP. A system dump will always be generated when one of these suboptions is specified.
- PL/I Considerations
 - After a normal return from a PL/I ERROR ON-unit, or from a PL/I FINISH ON-unit, Language Environment considers the condition unhandled. If a GOTO is not performed and the resume cursor is not moved, then the thread terminates. The TERMTHDACT setting guides the amount of information that is produced, so the message is not presented twice.
- PL/I MTF Considerations
 - TERMTHDACT applies to a task that terminates abnormally due to an unhandled condition of severity 2 or higher that is percolated beyond the initial routine's stack frame. All active subtasks that were created from the incurring task will terminate abnormally, but the enclave will continue to run.
- z/OS UNIX Considerations
 - The TERMTHDACT option applies when a thread terminates abnormally. Abnormal termination of a single thread causes termination of the entire enclave. If an unhandled condition of severity 2 or higher percolates beyond the first routine's stack frame the enclave terminates abnormally.
 - If an enclave terminates due to a POSIX default signal action, then TERMTHDACT applies to conditions that result from software signals, program checks, or abends.
 - If running under a shell and Language Environment generates a system dump, then a storage dump is generated to a file based on the kernel environment variable, _BPXK_MDUMP.
- CICS Considerations
 - TERMTHDACT output is written either to a transient data queue named CESE, or to the CICS transaction dump, depending on the setting of the CESEICISDDDS suboption of the TERMTHDACT run-time option. Table 6 on page 42 shows the behavior of CESEICISDDDS when they are used with the other suboptions of TERMTHDACT.
 - Since Language Environment does not own the ESTAE, the suboption UAIMM will be treated as UAONLY.
 - All associated Language Environment dumps will be suppressed if termination processing is the result of an EXEC CICS ABEND with NODUMP.

Table 6. Condition handling of 0Cx abends

Options	TERMTHDACT(X,CESE,)	TERMTHDACT(X,CICSDDS,)
QUIET	No output.	No output.
MSG	Message written to CESE queue or MSGFILE.	Message written to CESE queue or MSGFILE.
TRACE	The traceback is written to the CESE queue, followed by U4038 abend with nodump option.	- Language Environment will write traceback, variables, COBOL working storage, C writeable static. The member handlers will be invoked to provide the desired output to the new transaction server queue (which CICS will read and write to CICS transaction dump later). - U4039 abend to force CICS transaction dump followed by U4038 abend with nodump option. - Message to CESE or MSGFILE.
DUMP	CEEDUMP to CESE queue followed by U4038 abend with nodump option.	- CEEDUMP to new transaction server queue which CICS will read and write to CICS transaction dump later. - U4039 abend to force CICS transaction dump followed by U4038 abend with nodump option. - Message to CESE or MSGFILE.
UATRACE	U4039 abend with traceback to CESE queue followed by U4038 abend with nodump option.	- Language Environment will write traceback, variables, COBOL working storage, C writeable statics. The member handlers will be invoked to provide the desired output to the new transaction server queue (which CICS will read and write to CICS transaction dump later). - U4039 abend to force CICS transaction dump followed by U4038 abend with nodump option. - Message to CESE or MSGFILE.
UADUMP	U4039 abend with CEEDUMP to CESE queue followed by U4038 abend with nodump option.	- CEEDUMP to new transaction server queue which CICS will read and write to CICS transaction dump later. - U4039 abend to force CICS transaction dump followed by U4038 abend with nodump option. - Message to CESE or MSGFILE.
UAONLY	U4039 abend followed by U4038 abend with nodump option.	- U4039 abend followed by U4038 abend with nodump option. - No CEEDUMP information is generated. - Same as CESE.
UAIMM	U4039 abend followed by U4038 abend with nodump option.	- U4039 abend followed by U4038 abend with nodump option. - No CEEDUMP information is generated. - Same as CESE.

Note: Program checks and other abends will cause CICS to produce a CICS transaction dump.

For more information about the TERMTHDACT run-time option, see *z/OS Language Environment Programming Reference*.

Generating a Language Environment dump with language-specific functions

In addition to the CEE3DMP callable service and the TERMTHDACT run-time option, you can use language-specific routines such as C functions, the Fortran SDUMP service, and the PL/I PLIDUMP service to generate a dump.

C/C++ routines can use the functions `cdump()`, `csnap()`, and `ctrace()` to produce a Language Environment dump. All three functions call the CEE3DMP callable service, and each function includes an options string consisting of different CEE3DMP options that you can use to control the information contained in the dump. For more information on these functions, see “Generating a Language Environment dump of a C/C++ routine” on page 145.

Fortran programs can call SDUMP, DUMP/PDUMP, or CDUMP/CPDUMP to generate a Language Environment dump. CEE3DMP cannot be called directly from a Fortran program. For more information on these functions, see “Generating a Language Environment dump of a FORTRAN routine” on page 221.

PL/I routines can call PLIDUMP instead of CEE3DMP to produce a dump. PLIDUMP includes options that you can specify to obtain a variety of information in the dump. For a detailed explanation about PLIDUMP, see “Generating a Language Environment dump of a PL/I routine” on page 245.

Understanding the Language Environment dump

The Language Environment dump service generates output of data and storage from the Language Environment run-time environment on an enclave basis. This output contains the information needed to debug most basic routine errors.

Figure 7 on page 48 illustrates a dump for enclave main. The example assumes full use of the CEE3DMP dump options. Ellipses are used to summarize some sections of the dump and information regarding unhandled conditions may not be present at all. Sections of the dump are numbered to correspond with the descriptions given in “Sections of the Language Environment dump” on page 55.

The CEE3DMP was generated by the C program CELSAMP shown in Figure 5 on page 44. CELSAMP uses the DLL CELDLL shown in Figure 6 on page 47.

```

#pragma options(SERVICE("1.1.c"),NOOPT,TEST,GONUMBER)
#pragma runopts(TERMTHDACT(UADUMP),POSIX(ON))
#define _OPEN_THREADS
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <dll.h>
#include <signal.h>
#include <leawi.h>
#include <ceedcct.h>

pthread_mutex_t      mut;
pthread_t            thread[2];
int                 threads_joined = 0;
char *               t1 = "Thread 1";
char *               t2 = "Thread 2";
/*****
/*  thread_cleanup: condition handler to clean up threads
*****/
void thread_cleanup(_FEEDBACK *cond,_INT4 *input_token,
                   _INT4 *result, _FEEDBACK *new_cond) {

    /* values for handling the conditions */
#define percolate    20
    printf(">>> Thread_CleanUp: Msg # is %d\n",cond->tok_msgno);
    if (!threads_joined) {
        printf(">>> Thread_CleanUp: Unlocking mutex\n");
        pthread_mutex_unlock(&mut);
        printf(">>> Thread_CleanUp: Joining threads\n");
        if (pthread_join(thread[0],NULL) == -1 )
            perror("Join of Thread #1 failed");
        if (pthread_join(thread[1],NULL) == -1 )
            perror("Join of Thread #2 failed");
        threads_joined = 1;
    }
    *result = percolate;
    printf(">>> Thread_CleanUp: Percolating condition\n");
}
/*****
/*  thread_func: Invoked via pthread_create.
*****/
void *thread_func(void *parm)
{
    printf(">>> Thread_func: %s locking mutex\n",parm);
    pthread_mutex_lock(&mut);
    pthread_mutex_unlock(&mut);
    printf(">>> Thread_func: %s exiting\n",parm);
    pthread_exit(NULL);
}

```

Figure 5. The C program CELSAMP (Part 1 of 3)

```

/*****
/* Start of Main function.
*****/
main()
{
    dllhandle *      handle;
    int              i = 0;
    FILE*            fp1;
    FILE*            fp2;
    _FEEDBACK         fc;
    _INT4             token;
    _ENTRY            pgmptr;

    printf("Init MUTEX...\n");
    if (pthread_mutex_init(&mut, NULL) == -1) {
        perror("Init of mut failed");
        exit(101);
    }

    printf("Lock Mutex Lock...\n");
    if (pthread_mutex_lock(&mut) == -1) {
        perror("Lock of mut failed");
        exit(102);
    }

    printf("Create 1st thread...\n");
    if (pthread_create(&thread[0], NULL, thread_func, (void *)t1) == -1) {
        perror("Could not create thread #1");
        exit(103);
    }
    printf("Create 2nd thread...\n");
    if (pthread_create(&thread[1], NULL, thread_func, (void *)t2) == -1) {
        perror("Could not create thread #2");
        exit(104);
    }
    printf("Register thread cleanup condition handler...\n");
    pgmptr.address = (_POINTER)thread_cleanup;
    pgmptr.nesting = NULL;
    token = 1;
    CEEHDLR (&pgmptr, &token, &fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf( "CEEHDLR failed with message number %d\n", fc.tok_msgno);
        exit(105);
    }
    printf("Load DLL...\n");
    handle = dllload("CELDLL");
    if (handle == NULL) {
        perror("Could not load DLL CELDLL");
        exit(106);
    }

    printf("Query DLL...\n");
    pgmptr.address = (_POINTER)dllqueryfn(handle, "dump_n_perc");
    if (pgmptr.address == NULL) {
        perror("Could not find dump_n_perc");
        exit(107);
    }
}

```

Figure 5. The C program CELSAMP (Part 2 of 3)

```

printf("Register condition handler...\n");
pgmptr.nesting = NULL;
token = 2;
CEEHDLR (&pgmptr, &token, &fc);
if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf( "CEEHDLR failed with message number %d\n",fc.tok_msgno);
    exit(108);
}

printf("Write to some files...\n");
fp1 = fopen("myfile.data", "w");
if (!fp1) {
    perror("Could not open myfile.data for write");
    exit(109);
}

fprintf(fp1, "record 1\n");
fprintf(fp1, "record 2\n");
fprintf(fp1, "record 3\n");

fp2 = fopen("memory.data", "wb,type=memory");
if (!fp2) {
    perror("Could not open memory.data for write");
    exit(112);
}

fprintf(fp2, "some data");
fprintf(fp2, "some more data");
fprintf(fp2, "even more data");

printf("Divide by zero...\n");
i = 1/i;
printf("Error -- Should not get here\n");
exit(110);
}

```

Figure 5. The C program CELSAMP (Part 3 of 3)

```

/* DLL containing Condition Handler that takes dump and percolates */
#pragma options(SERVICE("1.3.a"),GONUMBER,TEST,NOOPT)
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceedcct.h>
char wsa_array[10] = { 'C','E','L','D','L','L',' ',' ','W','S','A'};
#define OPT_STR "THREAD(ALL) BLOCKS STORAGE"
#define TITLE_STR "Sample dump produced by calling CEE3DMP"

void dump_n_perc(_FEEDBACK *cond, _INT4 *input_token,
                _INT4 *result, _FEEDBACK *new_cond) {

    /* values for handling the conditions */
    #define percolate 20

    _CHAR80 title;
    _CHAR255 options;
    _FEEDBACK fc;

    printf(">>> dump_n_perc: Msg # is %d\n",cond->tok_msgno);

    /* check if the DIVIDE-BY-ZERO message (0C9) */
    if (cond->tok_msgno == 3209) {
        memset(options,' ',sizeof(options));
        memcpy(options,OPT_STR,sizeof(OPT_STR)-1);

        memset(title,' ',sizeof(title));
        memcpy(title,TITLE_STR,sizeof(TITLE_STR)-1);

        printf(">>> dump_n_perc: Taking dump\n");
        CEE3DMP(title,options,&fc);
        if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
            printf("CEE3DMP failed with msgno %d\n",fc.tok_msgno);
            exit(299);
        }
    }
    *result = percolate;
    printf(">>> dump_n_perc: Percolating condition\n");
}

```

Figure 6. The C DLL CELDLL

For easy reference, the sections of the following dump are numbered to correspond with the descriptions in “Sections of the Language Environment dump” on page 55.

```

[1]CEE3DMP V1 R3.0: Sample dump produced by calling CEE3DMP                                08/15/01 1:05:29 PM                Page:    1
[2]CEE3DMP called by program unit POSIX.CRTL.C(CELDLL) (entry point dump_n_perc) at statement 33 (offset +0000010C).

[3]Registers on Entry to CEE3DMP:

PM..... 0100
GPR0..... 2471CEB0  GPR1..... 00024A00  GPR2..... 2471CEB0  GPR3..... A4700B06
GPR4..... 00024A00  GPR5..... 00000010  GPR6..... 000000F0  GPR7..... 000A2794
GPR8..... 2471CEC0  GPR9..... 2471CFA0  GPR10..... 000A26C2  GPR11..... 2479A6E8
GPR12..... 00015920  GPR13..... 00024800  GPR14..... 800180E2  GPR15..... A4759658
FPR0..... 4DB035F6  D8F87B96  FPR2..... 00000000  00000000
FPR4..... 00000000  00000000  FPR6..... 00000000  00000000

GPREG STORAGE:
Storage around GPR0 (2471CEB0)
-0020 2471CE90 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 2471CEB0 C3C5D3C4 D3D340E6 E2C10000 00000000 6E6E6E40 84A49497 6D956D97 8599837A |CELDLL WSA.....>>> dump_n_perc:|
+0020 2471CED0 40D4A287 407B4089 A2406C84 15000000 E3C8D9C5 C1C44DC1 D3D35D40 C2D3D6C3 |Msg # is %d....THREAD(ALL) BLOC|

Storage around GPR1 (00024A00)
-0020 000249E0 40404040 40404040 40404040 40404040 40404040 40404040 40404054 |.....|
+0000 00024A00 000248B0 00024900 000248A0 00024A40 00004A48 00024800 00024E70 A475A51C |.....+.u.v.|
+0020 00024A20 2475C480 04000000 00024A9C 259B1028 00000000 0004E544 0004EF94 2478E6D8 |..D.....V....m..WQ|

:

Storage around GPR15(24759658)
-0020 24759638 F1F9F9F8 F0F3F0F9 F1F1F4F0 F0F0F0F1 F0F9F0F0 0001F000 00000750 00000000 |19980309114000010900..0....&...|
+0000 24759658 47F0F014 00C3C5C5 00000460 00002DC8 47F0F001 90ECD00C 18BF41A0 BFFF4190 |.00..CEE.....H.00.....|
+0020 24759678 AFFF5800 9E225810 D04C1E01 5500C00C 47D0B03C 58F0C2BC 05EF181F 5000104C |.....<.....0B.....&.<|

[4]Information for enclave main

[5]Information for thread 24ABED8000000000

Registers on Entry to CEE3DMP:
PM..... 0100
GPR0..... 2471CEB0  GPR1..... 00024A00  GPR2..... 2471CEB0  GPR3..... A4700B06
GPR4..... 00024A00  GPR5..... 00000010  GPR6..... 000000F0  GPR7..... 000A2794
GPR8..... 2471CEC0  GPR9..... 2471CFA0  GPR10..... 000A26C2  GPR11..... 2479A6E8
GPR12..... 00015920  GPR13..... 00024800  GPR14..... 800180E2  GPR15..... A4759658
FPR0..... 4DB035F6  D8F87B96  FPR2..... 00000000  00000000
FPR4..... 00000000  00000000  FPR6..... 00000000  00000000

GPREG STORAGE:
Storage around GPR0 (2471CEB0)
-0020 2471CE90 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 2471CEB0 C3C5D3C4 D3D340E6 E2C10000 00000000 6E6E6E40 84A49497 6D956D97 8599837A |CELDLL WSA.....>>> dump_n_perc:|
+0020 2471CED0 40D4A287 407B4089 A2406C84 15000000 E3C8D9C5 C1C44DC1 D3D35D40 C2D3D6C3 |Msg # is %d....THREAD(ALL) BLOC|

:

[6]Traceback:
DSA Addr  Program Unit  PU Addr  PU Offset  Entry  E Addr  E Offset  Statement  Load Mod  Service  Status
00024800  POSIX.CRTL.C(CELDLL)
24700AB8  +0000010C  dump_n_perc  24700AB8  +0000010C  33  CELDLL  1.3.a  Call
00024748  2479A848  -00000106  CEEPGTFN  2479A6E8  +0000005A  CEEPLPKA  Call
000A2098  CEEHDSR  247469A8  +00001460  CEEHDSR  247469A8  +00001460  CEEPLPKA  Call
000241E0  POSIX.CRTL.C(CELSAMP)
24702178  +0000088E  main  24702178  +0000088E  141  CELSAMP  1.1.c  Exception
000240C8  2491595E  -248D16F6  EDCZMINV  2491595E  -248D16F6  CEEEV003  Call
00024018  CEEBBEXT  00007D20  +0000013C  CEEBBEXT  00007D20  +0000013C  CEEBINIT  Call

```

Figure 7. Example dump using CEE3DMP (Part 1 of 8)

```

[7]Condition Information for Active Routines
Condition Information for POSIX.CRTL.C(CELSAMP) (DSA address 000241E0)
CIB Address: 000A26A8
Current Condition:
CEE3209S The system detected a fixed-point divide exception.
Location:
Program Unit: POSIX.CRTL.C(CELSAMP)
Program Unit:Entry:      main Statement: 141 Offset: +0000088E

Machine State:
ILC..... 0004      Interruption Code..... 0009
PSW..... 078D2400 A4702A0A
GPR0..... 000242B8 GPR1..... 000242A8 GPR2..... A4915A12 GPR3..... A47021C6
GPR4..... 80007E04 GPR5..... 25993870 GPR6..... 259938E8 GPR7..... 25993CC8
GPR8..... 00000000 GPR9..... 00000001 GPR10..... A4915952 GPR11..... 80007D20
GPR12..... 00015920 GPR13..... 000241E0 GPR14..... A47029F6 GPR15..... 00000012
Storage dump near condition, beginning at location: 247029F6
+000000 247029F6 4400C1C4 4400C1AC 41800001 8E800020 5D80D09C 5090D09C 4400C1AC 417063F8 |..AD..A.....)....&.....A....8|

[8]Parameters, Registers, and Variables for Active Routines:
dump_n_perc (DSA address 00024800):
Saved Registers:
GPR0..... 2471CEB0 GPR1..... 00024A00 GPR2..... 2471CEB0 GPR3..... A4700B06
GPR4..... 00024A00 GPR5..... 00000010 GPR6..... 000000F0 GPR7..... 000A2794
GPR8..... 2471CEC0 GPR9..... 2471CFA0 GPR10..... 000A26C2 GPR11..... 2479A6E8
GPR12..... 00015920 GPR13..... 00024800 GPR14..... 800180E2 GPR15..... A4759658
GPREG STORAGE:
Storage around GPR0 (2471CEB0)
-0020 2471CE90 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

:

main (DSA address 000241E0):
Saved Registers:
GPR0..... 000242B8 GPR1..... 000242A8 GPR2..... A4915A12 GPR3..... A47021C6
GPR4..... 80007E04 GPR5..... 25993870 GPR6..... 259938E8 GPR7..... 25993CC8
GPR8..... 00000000 GPR9..... 00000001 GPR10..... A4915952 GPR11..... 80007D20
GPR12..... 00015920 GPR13..... 000241E0 GPR14..... A47029F6 GPR15..... 00000012
GPREG STORAGE:
Storage around GPR0 (000242B8)
-0020 00024298 259ADB90 00000000 00000000 00000000 25993CC8 25993CB8 00000003 259938F1 |.....r.H.r.....r.1|

:

Local Variables:

:

[9]Control Blocks for Active Routines:
DSA for dump_n_perc: 00024800
+000000 FLAGS.... 1004      member... FFF0      BKC..... 00024748 FWC..... 00024A10 R14..... 800180E2
+000010 R15..... A4759658 R0..... 2471CEB0 R1..... 00024A00 R2..... 2471CEB0 R3..... A4700B06
+000024 R4..... 00024A00 R5..... 00000010 R6..... 000000F0 R7..... 000A2794 R8..... 2471CEC0
+000038 R9..... 2471CFA0 R10..... 000A26C2 R11..... 2479A6E8 R12..... 00015920 reserved. 000163D0
+00004C NAB..... 00024A10 PNAB..... 2474D3FF reserved. 2474C400 00015920 000248B4 000248A0
+000064 reserved. 24717AE4 reserved. 000248DC MODE..... A4700BC6 reserved. 00024890 000A24C0
+000078 reserved. 24717AE8 reserved. 00024890
DSA for CEEPGTFN: 00024748
+000000 FLAGS.... 1000      member... 3B40      BKC..... 000A2098 FWC..... 00024928 R14..... A479A744
+000010 R15..... 24700AB8 R0..... 2471CEB0 R1..... 247178F8 R2..... 00044220 R3..... 259ADB90
+000024 R4..... 000A26A8 R5..... 00000002 R6..... 247178D0 R7..... 000A3097 R8..... 247499A5
+000038 R9..... 25993870 R10..... 247479A7 R11..... A474EAC0 R12..... 00015920 reserved. 000163D0
+00004C NAB..... 00024800 PNAB..... 800118E0 reserved. 00000000 00000000 259A0408 00000009
+000064 reserved. 25993C9D reserved. 259A0408 MODE..... A474C756 reserved. 259A03F0 24719C7C
+000078 reserved. 2471A1B0 reserved. A4A81C38
DSA for CEEHDSP: 000A2098
+000000 FLAGS.... 0808      member... CEE1      BKC..... 000241E0 FWC..... 00024748 R14..... A4747E0A
+000010 R15..... 2479A6E8 R0..... 259ADB90 R1..... 247178F8 R2..... 00044220 R3..... 00014558
+000024 R4..... 000A26A8 R5..... 00000002 R6..... 247178D0 R7..... 000A3097 R8..... 247499A5
+000038 R9..... 247489A6 R10..... 247479A7 R11..... A47469A8 R12..... 00015920 reserved. 000163D0
+00004C NAB..... 00024748 PNAB..... 000A2098 reserved. 00000000 00000000 00000000 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE..... A4746F9A reserved. 00000000 00000000
+000078 reserved. 00000000 reserved. 00000000

```

Figure 7. Example dump using CEE3DMP (Part 2 of 8)

```

DSA for main: 000241E0
+000000  FLAG5.... 1000      member... 0000      BKC..... 000240C8  FWC..... 000242B8  R14..... A47029F6
+000010  R15..... 2489F3D8    R0..... 000242B8  R1..... 000242A8  R2..... A4915A12  R3..... A47021C6
+000024  R4..... 80007E04    R5..... 25993870  R6..... 259938E8  R7..... 25993CC8  R8..... 00000001
+000038  R9..... 80000000    R10..... A4915952  R11..... 80007D20  R12..... 00015920  reserved. 000163D0
+00004C  NAB..... 000242B8    PNAB..... 00017038  reserved. 00014D30  247C16F0  00015920  00000000
+000064  reserved. 00024328  reserved. 01000000  MODE..... A470271C  reserved. 00000000  00000000
+000078  reserved. 00000000  reserved. 00000000

CIB for main: 000A26A8
+000000 000A26A8 C3C9C240 00000000 00000000 010C0004 00000000 00000000 00030C89 59C3C5C5 | CIB .....i.CEE|
+000020 000A26C8 00000001 000A27B4 00030C89 59C3C5C5 00000001 00000000 000241E0 A47ECFC8 | .....i.CEE.....u=.H|
+000040 000A26E8 00000000 000241E0 24702A0A 24717D50 00000003 00000000 00000000 00000000 | .....&.....|
+000060 000A2708 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|
+000080 000A2728 - +00009F 000A2747 same as above | .....|
+0000A0 000A2748 00000000 00000000 00000000 00000000 40250000 940C9000 00000009 00000000 | .....m.....|
+0000C0 000A2768 00000000 247031E0 000241E0 000241E0 24702A06 00000000 00000000 00000001 | .....|
+0000E0 000A2788 00000002 00000003 00000002 00000014 00000002 00000000 00000000 00000000 | .....|
+000100 000A27A8 00000008 24717E9C 00000000 E9D4C3C8 00000000 000242B8 000242A8 A4915A12 | .....=.....ZMCH.....yuj!|

:

[10]Storage for Active Routines:
DSA frame: 00024800
+000000 00024800 1004FFF0 00024748 00024A10 800180E2 A4759658 2471CEB0 00024A00 2471CEB0 | ...0.....Su.o.....|
+000020 00024820 A4700B06 00024A00 00000010 000000F0 000A2794 2471CEC0 2471CFA0 000A26C2 | u.....0...m.....B|
+000040 00024840 2479A6E8 00015920 000163D0 00024A10 247403FF 2474C400 00015920 000248B4 | ..wY.....L...D...|
+000060 00024860 000248A0 24717AE4 000248DC A4700BC6 00024890 000A24C0 24717AE8 00024890 | .....U...u..F.....Y...|
+000080 00024880 000248B4 000248DC 00000001 A4A8DCF6 000A3430 000A3421 247178D0 0000088E | .....uy.6.....|
+0000A0 000248A0 00000003 00015920 000A26A8 00000000 E2819497 93854084 A4949740 97999684 | .....y....Sample dump prod|
+0000C0 000248C0 A4838584 4082A840 83819393 89958740 C3C5C5F3 C4D4D740 40404040 40404040 | uced by calling CEE3DMP|
+0000E0 000248E0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 | THREAD(ALL) BLOCKS STORAGE|
+000100 00024900 E3C8D9C5 C1C44DC1 D3D35D40 C2D3D6C3 D2E240E2 E3D6D9C1 C7C54040 40404040 | |
+000120 00024920 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 | |
+000140 00024940 - +0001DF 000249DF same as above | |
+0001E0 000249E0 40404040 40404040 40404040 40404040 40404040 40404040 40404054 | .....|.|
+000200 00024A00 000248B0 00024900 000248A0 00024A40 00004A48 00024800 00024E70 A475A51C | .....+.u.v..|

DSA frame: 00024748
+000000 00024748 10003B40 000A2098 00024928 A479A744 24700AB8 2471CEB0 247178F8 00044220 | ... ..q...u.x.....8...|
+000020 00024768 259ADB90 000A26A8 00000002 247178D0 000A3097 247499A5 25993870 247479A7 | .....y.....p..rv.r....x|
+000040 00024788 A474EAC0 00015920 000163D0 00024800 800118E0 00000000 00000000 259A0408 | u.....|
+000060 000247A8 00000009 25993C9D 259A0408 A474C756 259A03F0 24719C7C 2471A1B0 A4A81C38 | .....r.....u.G.....0...@...uy..|
+000080 000247C8 000247F0 00024802 00024800 000247FC 000247F8 00024802 000247F0 00024802 | ..0.....8.....0....|
+0000A0 000247E8 00024800 000A3110 2491595E 00000000 00024018 00024018 1004FFF0 00024748 | .....j;.....0....|

DSA frame: 000241E0
+000000 000241E0 10000000 000240C8 000242B8 A47029F6 2489F3D8 000242B8 000242A8 A4915A12 | ..... H...u..6.i3Q.....yuj!|
+000020 00024200 A47021C6 80007E04 25993870 259938E8 25993C8C 00000001 80000000 A4915952 | u..F..=.r...r.Y.r.H.....uj..|
+000040 00024220 80007D20 00015920 000163D0 000242B8 00017038 00014D30 247C16F0 00015920 | ..'|.....(.@.0...|
+000060 00024240 00000000 00024328 01000000 A470271C 00000000 00000000 00000000 00000000 | .....u.....|
+000080 00024260 00000000 A47C1792 04000000 F97FC14F 00000001 F97FC14F 259AD3DE 00000000 | .....u@.k...9"A|...9"A|...L....|
+0000A0 00024280 2599F1EC 259A03F4 00000000 00000000 00000000 00000002 259ADB90 00000000 | ..r1...4.....|
+0000C0 000242A0 00000000 00000000 25993CC8 25993CB8 00000003 259938F1 00000100 24716FB0 | .....r.H.r.....r.1.....?|

:

[11]Control Blocks Associated with the Thread:
CAA: 00015920
+000000 00015920 00000800 00000000 00024000 00044000 00000000 00000000 00000000 00000000 | .....|
+000020 00015940 00000000 00000000 00016610 00000000 00000000 00000000 00000000 00000000 | .....|
+000040 00015960 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|
+000060 00015980 00000000 00000000 00000000 00000000 00000000 80012690 00000000 00000000 | .....|
+000080 000159A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|
+0000A0 000159C0 - +00011F 00015A3F same as above | .....|
+000120 00015A40 24716B40 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .., .....|
+000140 00015A60 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|
+000160 00015A80 - +00017F 00015A9F same as above | .....|
+000180 00015AA0 00000000 00000000 00000000 00000000 00000000 50C0D064 05C058C0 | .....&.....|
+0001A0 00015AC0 C00605CC 00008546 0700C198 0700C198 0700C198 0700C198 0700C198 0700C198 | .....e...Aq..Aq..Aq..Aq..Aq..Aq..|
+0001C0 00015AE0 0700C198 0700C198 0700C198 0700C198 0700C198 0700C198 0700C198 0700C198 | ..Aq..Aq..Aq..Aq..Aq..Aq..Aq..|

:

```

Figure 7. Example dump using CEE3DMP (Part 3 of 8)

```

Thread Synchronization Queue Element (SQEL): 247181F8
+000000 247181F8 00000000 00000000 00000000 00000000 2599EF90 00000007 0004E5F8 00000000 | .....r.....V8....
+000020 24718218 00015920 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
DUMMY DSA: 000161C8
+000000 FLAGS.... 0000      member... 0000      BKC..... 00005F80 FWC..... 00024018 R14..... A47048C8
+000010 R15..... 80007D20 R0..... 7D00002B R1..... 25993D08 R2..... 00000000 R3..... 00000000
+000024 R4..... 00000000 R5..... 00000000 R6..... 00000000 R7..... 00017038 R8..... 24703F60
+000038 R9..... 009DB428 R10..... 00000000 R11..... A47047F2 R12..... 00015920 reserved. 000163D0
+00004C NAB..... 00024018 PNAB..... 00024018 reserved. 00000000 00000000 00000000 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE..... 00000000 reserved. 00000000 00000000
+000078 reserved. 00000000 reserved. 00000000

:
:

[5]Information for thread 24ABF69000000001

Registers on Entry to CEE3DMP:
PM..... 0100
GPR0..... 247181F8 GPR1..... 000569FC GPR2..... 0004E164 GPR3..... 00054958
GPR4..... 24A84D80 GPR5..... 2599EF9A GPR6..... 259938BC GPR7..... 2599EF90
:
:

[6]Traceback:
DSA Addr Program Unit PU Addr PU Offset Entry E Addr E Offset Statement Load Mod Service Status
00056978 CEEOPML2 24A84558 +000004BE CEEOPML2 24A84558 +000004BE CEEOLVD Call
00056728 2489A658 +0000160A EDCOWRP2 2489B31C +00000946 CEEEV003 Call
00056680 POSIX.CRTL.C(CELSAMP)
24701F00 +000000A4 thread_func 24701F00 +000000A4 45 CELSAMP 1.1.c Call
7F83E5F0 POSIX.CRTL.C(CELSAMP)
24702178 -246F57F6 main 24702178 -246F57F6 CELSAMP 1.1.c Call

[8]Parameters, Registers, and Variables for Active Routines:
CEEOPML2 (DSA address 00056978):
Saved Registers:
GPR0..... 247181F8 GPR1..... 000569FC GPR2..... 0004E164 GPR3..... 00054958
GPR4..... 24A84D80 GPR5..... 2599EF9A GPR6..... 259938BC GPR7..... 2599EF90
:
:

thread_func (DSA address 00056680):
Parameters:
parm void * 0x259938E8
Saved Registers:
GPR0..... 00056728 GPR1..... 0005671C GPR2..... 24A92528 GPR3..... A4701F4E
GPR4..... 00000000 GPR5..... 25993870 GPR6..... 259938E8 GPR7..... 259938BC
:
:

[9]Control Blocks for Active Routines:
DSA for CEEOPML2: 00056978
+000000 FLAGS.... 0000      member... 0007      BKC..... 00056728 FWC..... 00056A78 R14..... A4A84A18
+000010 R15..... A4A85B18 R0..... 247181F8 R1..... 000569FC R2..... 0004E164 R3..... 00054958
+000024 R4..... 24A84D80 R5..... 2599EF9A R6..... 259938BC R7..... 2599EF90 R8..... 25991027
:
:

[10]Storage for Active Routines:
DSA frame: 00056728
+000000 00056728 10000000 00056680 00056978 8001E862 A4A84558 247181F8 0007B958 24A92528 | .....Y.uy....a8....Z..
+000020 00056748 A4701F4E 00000000 25993870 259938E8 259938BC 25991027 00056210 8000D0A7 | u..+.....r...Y.r...r.....x
:
:

[11]Control Blocks Associated with the Thread:
CAA: 00055748
+000000 00055748 00000800 00000000 00056668 00076668 00000000 00000000 00000000 00000000 | .....
+000020 00055768 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
:
:

```

Figure 7. Example dump using CEE3DMP (Part 4 of 8)

[12] Enclave Control Blocks:

```

EDB: 000148B0
+000000 000148B0 C3C5C5C5 C4C24040 C7000001 000157E0 00014F00 00000000 00000000 00000000 | CEEEDB G.....|.....
+000020 000148D0 00014D78 00014DA8 00017038 00014558 00000000 80013808 000149D0 00008000 | ..(..(y.....
+000040 000148F0 00000000 00000000 00005F80 00000000 00000000 0001E038 24716738 259938B8 | .....~.....r..
+000060 00014910 0000CF00 0004E000 2471C02C 00000000 000166E0 00000000 247E71A0 00015920 | .....=.....
+000080 00014930 80000000 0000CFC4 00000000 00000000 00000003 00000000 00005FD0 009DB038 | .....D.....~.....

+0000A0 00014950 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000003 | .....
MEML: 000157E0
+000000 000157E0 00000000 00000000 247288A8 00000000 00000000 00000000 247288A8 00000000 | .....hy.....hy...
+000020 00015800 00000000 00000000 247288A8 00000000 2471A5A4 00000000 A47ECFC8 00000000 | .....hy.....vu...u=.H...
+000040 00015820 00000000 00000000 247288A8 00000000 00000000 00000000 247288A8 00000000 | .....hy.....hy...
+000060 00015840 - +00011F 000158FF same as above
Mutex and Condition Variable Blocks (MCVB+MHT+CHT): 0004E018
+000000 0004E018 00008F50 0004E044 000003F8 00001FC0 00000000 259940C0 0004E444 000000F8 | ...&.....8.....r ...U...8
+000020 0004E038 000007C0 00000000 259940D8 00000000 25994020 00000000 25993FF8 00000000 | .....r Q.....r...r.8...
+000040 0004E058 25993FB8 00000000 25993F78 00000000 00000000 00000000 00000000 00000000 | .r.....r.....
+000060 0004E078 25994070 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .r .....

:

Thread Synchronization Enclave Latch Table (EPALT): 0004E544
+000000 0004E544 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000020 0004E564 - +00009F 0004E5E3 same as above
+0000A0 0004E5E4 00000000 00000000 00000000 00000000 00000000 DB8E7E08 247181F8 0004E850 | .....=...a8..Y&
+0000C0 0004E604 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+0000E0 0004E624 - +00029F 0004E7E3 same as above

:

Thread Synchronization Trace Block (OTRB): 0004E000
+000000 0004E000 00046000 00000004 000007FF 00046000 3E008000 BE008000 00008F50 0004E044 | ..-.....-.....&....
Thread Synchronization Trace Table (OTRTBL): 00046000
+000000 00046000 0000D4E7 40C9D540 259938BC 00000000 0001D4E7 40C14040 259938BC 00000000 | ..MX IN .r.....MX A .r.....
+000020 00046020 0002D4E7 40E64040 259938BC 00000002 0003D4E7 40E64040 259938BC 00000000 | ..MX W .r.....MX W .r.....
+000040 00046040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000060 00046060 - +003FFF 00049FFF same as above

DLL Information:
WSA Addr Module Addr Thread ID Use Count Name
2471CEB0 247006F0 24ABED8000000000 00000001 CELDLL
HEAPCHK Option Control Block (HCOP): 25993028
+000000 25993028 C8C3D6D7 00000024 00000001 00000000 00000000 259BD028 2599304C 00000000 | HCOP.....r.<....
+000020 25993048 00000000 C8C3C6E3 00000200 00000000 00000000 00000000 00000000 00000000 | ....HCFT.....
HEAPCHK Element Table (HCEL) for Heapid 259B24B4 :
Header: 259BD028
+000000 259BD028 C8C3C5D3 259AF028 00000000 259B24B4 000001F4 00000007 00000007 00000000 | HCEL..0.....4.....
Address Seg Addr Length Address Seg Addr Length
Table: 259BD048
+000000 259BD048 259BC020 259BC000 00000050 00000000 259BC070 259BC000 00000020 00000000 | .....&.....
+000020 259BD068 259BC090 259BC000 00000018 00000000 259BC0A8 259BC000 00000088 00000000 | .....y.....h...
+000040 259BD088 259BC130 259BC000 00000050 00000000 259BC180 259BC000 00000020 00000000 | ..A.....&.....A.....
+000060 259BD0A8 259BF020 259BF000 000003C8 00000000 00000000 00000000 00000000 00000000 | ..0...0...H.....
HEAPCHK Element Table (HCEL) for Heapid 259ADC14 :
Header: 259AF028
+000000 259AF028 C8C3C5D3 2599D028 259BD028 259ADC14 000001F4 00000001 00000001 00000000 | HCEL.r.....4.....
Address Seg Addr Length Address Seg Addr Length
Table: 259AF048
+000000 259AF048 259AE020 259AE000 000001C8 00000000 00000000 00000000 00000000 00000000 | .....H.....
HEAPCHK Element Table (HCEL) for Heapid 00000000 :
Header: 2599D028
+000000 2599D028 C8C3C5D3 00000000 259AF028 00000000 000001F4 00000004 00000004 00000000 | HCEL.....0.....4.....
Address Seg Addr Length Address Seg Addr Length
Table: 2599D048
+000000 2599D048 25995020 25995000 00000038 00000000 25995058 25995000 00000038 00000000 | .r&..r&.....r&..r&.....
+000020 2599D068 25995090 25995000 00000010 00000000 259950A0 25995000 00000010 00000000 | .r&..r&.....r&..r&.....

```

Figure 7. Example dump using CEE3DMP (Part 5 of 8)

Heap Storage Diagnostics						
Stg Addr	ID	Length	Entry	E Addr	E Offset	Load Mod
25358020	25321564	000002A8	CEEV#GTS	05AB26F8	+00000000	CEEPLPKA
			CEEVGST	05ABC840	+00000072	CEEPLPKA
			CEEVGTHP	05ABC0C8	+000000FA	CEEPLPKA
			CEEOSIGG	05A72700	+00001D48	CEEPLPKA
			CEEV#GH	05ABA688	-FFF3E3FC	CEEPLPKA
			func3	25308A70	+00000078	*PATHNAM
			func2	25308BA0	+00000076	*PATHNAM
			func1	25308C78	+00000076	*PATHNAM
			main	25308D50	+00000076	*PATHNAM
			EDCZMINV	05DF29FE	-FA214608	CEEV003
25342038	00000000	00000120	CEEV#GH	05ABA688	+00000000	CEEPLPKA
			setlocale	05DABB18	+000000FC	CEEV003
			tzset	05D12108	+00000594	CEEV003
			_cinit	05C1F8F0	+00002C18	CEEV003
			CEEZINV	05AE60B0	+00000C76	CEEPLPKA
			main	25308D50	-DB85A804	*PATHNAM

Language Environment Trace Table:

Most recent trace entry is at displacement: 002980

Displacement	Trace Entry in Hexadecimal					Trace Entry in EBCDIC
+000000	Time 20.32.18.430976	Date 2001.08.26	Thread ID...	24ABED8000000000		
+000010	Member ID.... 03	Flags..... 00004B	Entry Type.....	00000001		
+000018	94818995 40404040	40404040 40404040	40404040 40404040	40404040 40404040	main	
+000038	60606E4D F0F8F55D	40979989 95A3864D	5D404040 40404040	40404040 40404040	-->(085) printf()	
+000058	40404040 40404040	40404040 40404040	40404040 40404040	40404040 40404040		
+000078	40404040 40404040					
+000080	Time 20.32.18.448404	Date 2001.08.26	Thread ID...	24ABED8000000000		
+000090	Member ID.... 03	Flags..... 00004B	Entry Type.....	00000002		
+000098	4C60604D F0F8F55D	40D9F1F5 7EF0F0F0	F0F0F0F0 C540C5D9	D9D5D67E F0F0F0F0	<--(085) R15=0000000E ERRNO=0000	
+0000B8	F0F0F0F0 00000000	00000000 00000000	00000000 00000000	00000000 00000000	0000.....	
+0000D8	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000		
+0000F8	00000000 00000000					
:						
+002900	Time 20.32.18.718260	Date 2001.08.26	Thread ID...	24ABED8000000000		
+002910	Member ID.... 03	Flags..... 00004B	Entry Type.....	00000001		
+002918	84A49497 6D956D97	85998340 40404040	40404040 40404040	40404040 40404040	dump_n_perc	
+002938	60606E4D F0F8F55D	40979989 95A3864D	5D404040 40404040	40404040 40404040	-->(085) printf()	
+002958	40404040 40404040	40404040 40404040	40404040 40404040	40404040 40404040		
+002978	40404040 40404040					
+002980	Time 20.32.18.718278	Date 2001.08.26	Thread ID...	24ABED8000000000		
+002990	Member ID.... 03	Flags..... 00004B	Entry Type.....	00000002		
+002998	4C60604D F0F8F55D	40D9F1F5 7EF0F0F0	F0F0F0F0 F040C5D9	D9D5D67E F0F0F0F0	<--(085) R15=0000000E ERRNO=0000	
+0029B8	F0F0F0F0 00000000	00000000 00000000	00000000 00000000	00000000 00000000	0000.....	
+0029D8	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000		
+0029F8	00000000 00000000					

[13]Enclave Storage:

Initial (User) Heap										
+000000	25995000	C8C1D5C3	00014D48	00014D48	00000000	25995000	259950B0	00008000	00007F50	HANC..((.....r&..r&....."&
+000020	25995020	25995000	00000038	C3C4D3D3	00000000	40000000	00000000	24700F98	24703F70	.r&....CDLL.... q....
+000040	25995040	25993870	00000490	00000000	00000000	00000000	00000000	25995000	00000038	.r.....r&.....
+000060	25995060	C3C4D3D3	25995028	80000000	00000000	247006F0	24700770	2471CEB0	00000150	CDLL.r&......0.....&
+000080	25995080	00000000	00000000	00000000	00000000	25995000	00000010	259ADB88	00000000	r&.....
+0000A0	259950A0	25995000	00000010	259ADBE0	00000000	00000000	00000000	00000000	00000000	.r&.....
+0000C0	259950C0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
+0000E0	259950E0	- +007FFF	2599CFFF			same as above				

Figure 7. Example dump using CEE3DMP (Part 6 of 8)

```

LE/370 Anywhere Heap                                : 24A91000
+000000 24A91000 C8C1D5C3 25993000 00014D78 00014D78 24A91000 00000000 00F00028 00000000 HANC.r....(...(.z.....0.....
+000020 24A91020 24A91000 00F00008 B035F6D8 B2C00081 24ABED80 00000000 03000000 00000001 .z...0....6Q...a.....
+000040 24A91040 94818995 40404040 40404040 40404040 40404040 40404040 40404040 40404040 main
+000060 24A91060 60606E4D F0F8F55D 40979989 95A3864D 5D404040 40404040 40404040 40404040 -->(085) printf()
:

LE/370 Below Heap                                    : 00044000
+000000 00044000 C8C1D5C3 00054000 00014DA8 00014DA8 80044000 00044388 00002000 00001C78 HANC.. ... (y.. (y.. ....h.....
+000020 00044020 00044000 00000048 C8C4D3E2 00000000 00044220 00000040 00010000 00000001 .. ....HDLS.....
+000040 00044040 000241E0 24701038 00000000 00000000 00000000 00000000 00000000 00000000 .....
+000060 00044060 00000000 00000000 00044000 00000128 07000700 05E0900F E0A641DE 002258C0 .....W.....
:

Additional Heap, heapid = 259B24B4                    : 259BC000
+000000 259BC000 C8C1D5C3 259BF000 259B24B4 259B24B4 259BC000 259BC1A0 000003E8 00000248 HANC..0.....A....Y....
+000020 259BC020 259BC000 00000050 00000000 24700C8C 24700AB8 259BC0B0 2471ABD4 4E801000 .....&.....M+....
+000040 259BC040 00000003 259BC098 00020000 259BC078 00000000 48C34DC3 C5D3C4D3 00000000 .....q.....C(CELDL...
+000060 259BC060 00000000 24700CA0 00000000 00000000 259BC000 00000020 0014D7D6 E2C9E74B .....POSIX.
+000080 259BC080 C3D9E3D3 4BC34DC3 C5D3C4D3 D35D0000 259BC000 00000018 00000003 00000130 CRTL.C(CELDLL).....
:

WSA for Program Object(s)
WSA: 2471CEB0
+000000 2471CEB0 C3C5D3C4 D3D340E6 E2C10000 00000000 6E6E6E40 84A49497 6D956D97 8599837A CELDLL WSA.....>>> dump_n_perc:
+000020 2471CED0 40D4A287 407B4089 A2406C84 15000000 E3C8D9C5 C1C44DC1 D3D35D40 C2D3D6C3 Msg # is %d....THREAD(ALL) BLOC
+000040 2471CEF0 D2E240E2 E3D6D9C1 C7C50000 00000000 E2819497 93854084 A4949740 97999684 KS STORAGE.....Sample dump prod
+000060 2471CF10 A4838584 4082A840 83819393 89958740 C3C5C5F3 C4D4D700 6E6E6E40 84A49497 uced by calling CEE3DMP.>>> dump
+000080 2471CF30 6D956D97 8599837A 40E38192 89958740 84A49497 15000000 00000000 00000000 _n_perc: Taking dump.....
+0000A0 2471CF50 00000000 00000000 C3C5C5F3 C4D4D740 86818993 858440A6 89A38840 94A28795 .....CEE3DMP failed with msgn
+0000C0 2471CF70 96406C84 15000000 6E6E6E40 84A49497 6D956D97 8599837A 40D78599 83969381 o %d....>>> dump_n_perc: Percola
+0000E0 2471CF90 A3899587 40839695 8489A389 96951500 180F58F0 F01007FF 24700A70 2471CEB0 ting condition.....00.....
+000100 2471CFB0 24700A80 00000000 247006F0 2471CEB0 180F58F0 F01007FF 247008A8 2471CEB0 .....0.....00.....y....
+000120 2471CFD0 24700A80 00000000 247006F0 2471CEB0 180F58F0 F01007FF 24700898 2471CEB0 .....0.....00.....q....
+000140 2471CFF0 24700A80 00000000 247006F0 2471CEB0 00000000 00000000 00000000 00000000 .....0.....
:

[14]Run-Time Options Report:
LAST WHERE SET      OPTION
-----
Installation default ABPERC (NONE)
Installation default ABTERMENC (ABEND)
Installation default NOAIXBLD
Installation default ALL31 (ON)
Assembler user exit ANYHEAP (32768,16384,ANYWHERE,FREE)
Installation default NOAUTOTASK
Assembler user exit BELOWHEAP (8192,8192,FREE)
Installation default CBLLOPTS (ON)
Installation default CBLPSHPOP (ON)
Installation default CBLQDA (OFF)
Installation default CHECK (ON)
Installation default COUNTRY (US)
Installation default NODEBUG
Installation default DEPTHCONDLMT (10)
Installation default ENVAR (")
Installation default ERRCOUNT (0)
Installation default ERRUNIT (6)
Installation default FILEHIST
Installation default FILETAG (NOAUTOCVT,NOAUTOTAG)
Default setting NOFLOW
Assembler user exit HEAP (49152,16384,ANYWHERE,KEEP,8192,4096)
Installation default HEAPCHK (OFF,1,0,0)
Installation default HEAPPOOLS (OFF,8,10,32,10,128,10,256,10,1024,10,2048,10)
Installation default INFMSGFILTER (OFF,,,)
Installation default INQPCOPN
Installation default INTERRUPT (OFF)
Installation default LIBRARY (SYSCEE)
Installation default LIBSTACK (4096,4096,FREE)
Installation default MSGFILE (SYSOUT,FBA,121,0,NOENQ)
Installation default MSGQ (15)
Installation default NATLANG (ENU)
Ignored NONONIPSTACK (See THREADSTACK)
Installation default OCSTATUS
Installation default NOPC

```

Figure 7. Example dump using CEE3DMP (Part 7 of 8)


```

Installation default      NOPC
Invocation command       POSIX(ON)
Installation default      PROFILE(OFF,"")
Installation default      PRTUNIT(6)
Installation default      PUNUNIT(7)
Installation default      RDRUNIT(5)
Installation default      RECPAD(OFF)
Invocation command       RPTOPTS(ON)
Invocation command       RPTSTG(ON)
Installation default      NORTEREUS
Installation default      RTLS(OFF)
Installation default      NOSIMVRD
Programmer default       STACK(4096,4096,ANYWHERE,KEEP,524288,131072)
Installation default      STORAGE(NONE,NONE,NONE,0)
Installation default      TERMTHDACT(TRACE,,96)
Installation default      NOTEST(ALL,"*","PROMPT","INSPREF")
Installation default      THREADHEAP(4096,4096,ANYWHERE,KEEP)
Installation default      THREADSTACK(OFF,4096,4096,ANYWHERE,KEEP,131072,131072)
Installation default      TRACE(OFF,4096,DUMP,LE=0)
Installation default      TRAP(ON,SPIE)
Installation default      UPSI(00000000)
Installation default      NOUSRHDLR(,)
Installation default      VCTRSVE(OFF)
Installation default      VERSION()
Installation default      XPLINK(OFF)
Installation default      XUFLOW(AUTO)

```

[15] Process Control Blocks:

```

PCB: 00014558
+000000 00014558 C3C5C5D7 C3C24040 03030298 00000000 00000000 00000000 00014788 247E8CD8 | CEEPCB ...q.....h.=.Q
+000020 00014578 247E2D68 247E7540 247E7068 2470A938 00013918 00000000 00000000 000148B0 | .==.=. ....Z.....
+000040 00014598 247E7390 7E000000 00000000 000122D4 00000000 00000000 00000000 00000000 | .==.=. ....M.....

MEML: 00014788
+000000 00014788 00000000 00000000 247288A8 00000000 00000000 00000000 247288A8 00000000 | .....hy.....hy....
+000020 000147A8 00000000 00000000 247288A8 00000000 24719004 00000000 A47ECFC8 00000000 | .....hy.....u=.H....
+000040 000147C8 00000000 00000000 247288A8 00000000 00000000 00000000 247288A8 00000000 | .....hy.....hy....
+000060 000147E8 - +00011F 000148A7 same as above

Thread Synchronization Process Latch Table (PPALT): 0004EF44
+000000 0004EF44 DB8E7E08 247181F8 0004E9E0 00000000 00000000 00000000 00000000 00000000 | ..=.a8..Z.....
+000020 0004EF64 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000040 0004EF84 00000000 00000000 00000000 00000000 DB8E7E08 247181F8 0004EA44 00000000 | .....=.a8.....
+000060 0004EFA4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000080 0004EFC4 - +0009FF 0004F943 same as above

```

:

Figure 7. Example dump using CEE3DMP (Part 8 of 8)

Sections of the Language Environment dump

The sections of the dump listed here appear independently of the Language Environment-conforming languages used. Each conforming language adds language-specific storage and file information to the dump.

For a detailed explanation of language-specific dump output:

- For C/C++ routines, see “Finding C/C++ information in a Language Environment dump” on page 156.
- For COBOL routines, see “Finding COBOL information in a dump” on page 203.
- For Fortran routines, see “Finding FORTRAN information in a Language Environment dump” on page 227.
- For PL/I routines, see “Finding PL/I information in a dump” on page 247.

[1] Page Heading

The page heading section appears on the top of each page of the dump and contains:

- CEE3DMP identifier
- Title

For dumps generated as a result of an unhandled condition, the title is “Condition processing resulted in the Unhandled condition.”
- Product abbreviation of Language Environment
- Version number
- Release number
- Date
- Time
- Page number

[2] Caller Program Unit and Offset

This information identifies the routine name and offset in the calling routine of the call to the dump service.

[3] Registers on Entry to CEE3DMP

This section of the dump shows data at the time of the call to the dump service.

- Program mask

The program mask contains the bits for the fixed-point overflow mask, decimal overflow mask, exponent underflow mask, and significance mask.
- General purpose registers (GPRs) 0–15

On entry to CEE3DMP, the GPRs contain:

GPR 0	Working register
GPR 1	Pointer to the argument list
GPR 2–11	Working registers
GPR 12	Address of CAA
GPR 13	Pointer to caller’s stack frame
GPR 14	Address of next instruction to run if the ALL31 run-time option is set to ON
GPR 15	Entry point of CEE3DMP
- Floating point registers (FPRs) 0, 2, 4, 6
- Storage pointed to by General Purpose Registers

Treating the contents of each register as an address, 32 bytes before and 64 bytes after the address are shown.

[4] - [14] Enclave Information

These sections show information that is specific to an enclave. When multiple enclaves are dumped, these sections will appear for each enclave.

[4] Enclave Identifier

This statement names the enclave for which information in the dump is provided. If multiple enclaves exist, the dump service generates data and storage information for the most current enclave, followed by previous enclaves in a last-in-first-out (LIFO) order. For more information about dumps for multiple enclaves, see “Multiple enclave dumps” on page 73.

[5] - [11] Thread Information

These sections show information that is specific to a thread. When multiple threads are dumped, these sections will appear for each thread.

[5] Information for thread

This section shows the system identifier for the thread. Each thread has a unique identifier.

[6] Traceback

In a multithread case, the traceback reflects only the current thread. For all active routines, the traceback section shows:

- Stack frame (DSA) address
- Program unit

The primary entry point of the external procedure. For COBOL programs, this is the PROGRAM-ID name. For C, Fortran, and PL/I routines, this is the compile unit name. For Language Environment-conforming assemblers, this is either the EPNAME = value on the CEEPPA macro, or a fully qualified path name.

- Program unit address
- Program unit offset

The offset of the last instruction to run in the routine. If the offset is a negative number, zero, or a very large positive number, the routine associated with the offset probably did not allocate a save area or the routine could have been called using SVC-assisted linkage. Adding the program unit address to the offset gives you the location of the current instruction in the routine. This offset is from the starting address of the routine.

- Entry

For COBOL, Fortran, PL/I, and VisualAge PL/I routines, this is the entry point name. For C/C++ routines, this is the function name. If a function name or entry point was not specified for a particular routine, then the string `*** NoName ***` will appear.

- Entry point address
- Entry point offset
- Load module
- Service level

The latest service level applied to the compile unit (for example, for IBM products, it would be the PTF number).

- Statement number

The last statement to run in the routine. The statement number appears only if your routine was compiled with the options required to generate statement numbers.

- Status

The reason Language Environment left the program or routine. The status can be either call or exception.

[7] Condition Information for Active Routines

This section displays the following information for all conditions currently active on the call chain:

- Statement showing failing routine and stack frame address of routine
- Condition information block (CIB) address
- Current condition, in the form of a Language Environment message for the condition raised or a Language Environment abend code, if the condition was caused by an abend

- Location
For the failing routine, this is the program unit, entry routine, statement number, and offset.
- Machine state, which shows:
 - Instruction length counter (ILC)
 - Interruption code
 - *Program status word (PSW)*
 - Contents of GPRs 0–15
 - Storage dump near condition (2 hex-bytes of storage near the PSW)
 These values are the current values at the time the condition was raised.

[8] Parameters, Registers, and Variables for Active Routines

For each active routine, this section shows:

- Routine name and stack frame address
- Arguments
For C/C++ and Fortran, arguments are shown here rather than with the local variables. For COBOL, arguments are shown as part of local variables. PL/I arguments are not displayed in the Language Environment dump.
- Saved registers
This lists the contents of GPRs 0–15 at the time the routine transferred control.
- Storage pointed to by the saved registers
Treating the saved contents of each register as an address, 32 bytes before and 64 bytes after the address shown.
- Local variables
This section displays the local variables and arguments for the routine. This section also shows the variable type. Variables are displayed only if the symbol tables are available. To generate a symbol table and display variables, use the following compile options:
 - For C, use TEST(SYM).
 - For C++, use TEST.
 - For VS COBOL II, use FDUMP.
 - For COBOL/370, use TEST(SYM).
 - For COBOL for OS/390 & VM, use TEST(SYM).
 - For Enterprise COBOL for z/OS, use TEST(SYM)
 - For Fortran, use SDUMP.
 - For PL/I, arguments and variables are not displayed.

[9] Control Blocks for Active Routines

For each active routine controlled by the STACKFRAME option, this section lists contents of related control blocks. The Language Environment-conforming language determines which language-specific control blocks appear. The possible control blocks are:

- Stack frame
- Condition information block
- Language-specific control blocks

[10] Storage for Active Routines

This displays local storage for each active routine. The storage is dumped in hexadecimal, with EBCDIC translations on the right side of the page. There can be other information, depending on the language used. For C/C++ routines, this is the

stack frame storage. For COBOL programs, this is language-specific information, WORKING-STORAGE, and LOCAL-STORAGE.

[11] Control Blocks Associated with the Thread

This section lists the contents of the Language Environment common anchor area (CAA), thread synchronization queue element (SQEL) and dummy stack frame. Other language-specific control blocks can appear in this section.

[12] Enclave Control Blocks

This section lists the contents of the Language Environment enclave data block (EDB) and enclave member list (MEML). The information presented may vary depending on which run-time options are set.

- If the POSIX run-time option is set to ON, this section lists the contents of the mutex and condition variable control blocks, the enclave level latch table, and the thread synchronization trace block and trace table.
- If DLLs have been loaded, this section shows information for each DLL including the DLL name, load address, use count, writeable static area (WSA) address, and the thread id of the thread that loaded the DLL.
- If the HEAPCHK run-time option is set to ON, this section shows the contents of the HEAPCHK options control block (HCOP) and the HEAPCHK element tables (HCEL). A HEAPCHK element table contains the location and length of all allocated storage elements for a heap in the order that they were allocated.
- When the *call-level* suboption of the HEAPCHK run-time option is set, any unfreed storage, which would indicate a storage leak, would be displayed in this area. The traceback could then be used to identify the program which did not free the storage.
- If the TRACE run-time option is set to ON, this section shows the contents of the Language Environment trace table.

Other language-specific control blocks can appear in this section.

[13] Enclave Storage

This section shows the Language Environment heap storage. For C/C++ and PL/I routines, heap storage is the dynamically allocated storage. For COBOL programs, it is the storage used for WORKING-STORAGE data items. This section also shows the writeable static area (WSA) storage for program objects. Other language-specific storage can appear in this section.

[14] Run-Time Options Report

This section lists the Language Environment run-time options in effect when the routine was executed.

[15] Process Control Blocks

This section lists the contents for the Language Environment process control block (PCB), process member list (MEML), and if the POSIX run-time option is set to ON, the process level latch table. Other language-specific control blocks can appear in this section.

Debugging with specific sections of the Language Environment dump

The following sections describe how you can use particular blocks of the dump to help you debug errors.

The tracebacks, condition information, and data values section

The CEE3DMP call with dump options TRACEBACK, CONDITION, and VARIABLES generates output that contains a traceback, information about any conditions, and a list of arguments, registers, and variables.

The traceback, condition, and variable information provided in the Language Environment dump can help you determine the location and context of the error without any additional information. The traceback section includes a sequential list for all active routines and the routine name, statement number, and offset where the exception occurred. The condition information section displays a message describing the condition and the address of the condition information block. The arguments, registers, and variables section shows the values of your arrays, structures, arguments, and data during the sequence of calls in your application. Static data values do not appear. Single quotes indicate character fields.

These sections of the dump are shown in Figure 7 on page 48.

The upward-growing (non-XPLINK) stack frame section

The stack frame, also called dynamic save area (DSA), for each active routine is listed in the full dump.

A stack frame chain is associated with each thread in the run-time environment and is acquired every time a separately compiled procedure or block is entered. A stack frame is also allocated for each call to a Language Environment service. All stack frames are back-chained with a stopping stack frame (also called a dummy DSA) as the first stack frame on the stack. Register 13 addresses the recently active stack frame or a standard register save area (RSA). The standard save area back chain must be initialized, and it holds the address of the previous save area. Not all Language Environment-conforming compilers set the forward chain; thus, it cannot be guaranteed in all instances. Calling routines establish the member-defined fields.

When a routine makes a call, registers 0–15 contain the following values:

- R1 is a pointer to parameter list or 0 if no parameter list passed.
- R0, R2–R11 is unreferenced by Language Environment. Caller's values are passed transparently.
- R12 is the pointer to the CAA if entry to an external routine.
- R13 is the pointer to caller's stack frame.
- R14 is the return address.
- R15 is the address of the called entry point.

With an optimization level other than 0, C/C++ routines save only the registers used during the running of the current routine. Non-Language Environment RSAs can be in the save area chain. The length of the save area and the saved register contents do not always conform to Language Environment conventions. For a detailed description of stack frames Language Environment storage management, see *z/OS Language Environment Programming Guide*. Figure 8 on page 61 shows the format of the upward-growing stack frame.

Note: The *Member-defined* fields are reserved for the specific higher level language.

00	Flags	Member-defined
04	CEEDSABACK - Standard Save Area Back Chain	
08	CEEDSAFWD - Standard Save Area Forward Chain	
0C	CEEDSASAVE - GPRs 14, 15, 0-12	
~		
48	Member-defined	
4C	CEEDSANAB - Current Next Available Byte (NAB) in Stack	
50	CEEDSAPNAB - End of Prolog NAB	
54	Member-defined	
58	Member-defined	
5C	Member-defined	
60	Member-defined	
64	Reserved for Debugging	
68	Member-defined	
6C	CEESAMODE - Return Address of the Module That Caused the Last Mode Switch	
70	Member-defined	
74	Member-defined	
78	Reserved for Future Condition Handling	
7C	Reserved for Future Use	

Figure 8. Upward-growing (non-XPLINK) stack frame format

The downward-growing (XPLINK) stack frame section

Figure 9 on page 62 shows the format of the downward-growing stack frame.

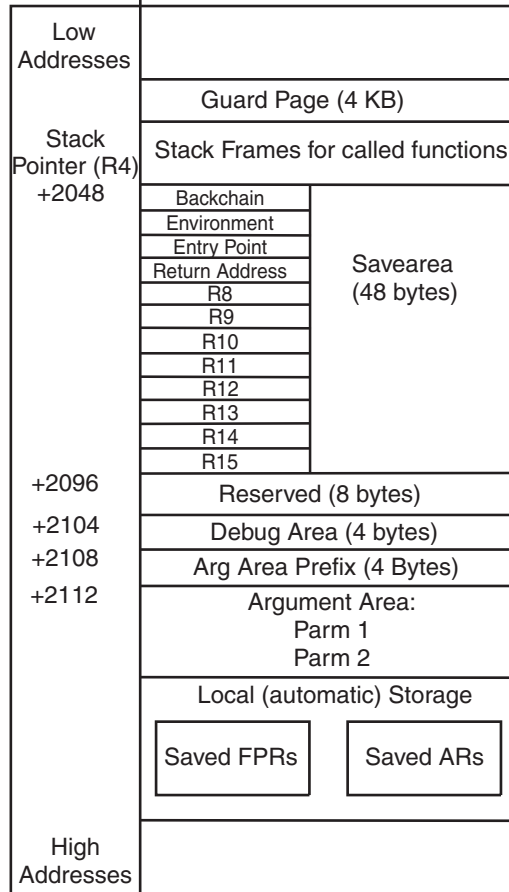


Figure 9. Downward-growing (XPLINK) stack frame format

For detailed information about the downward-growing stack, register conventions and parameter passing conventions, see *z/OS Language Environment Programming Guide*.

The Common Anchor Area

Each thread is represented by a common anchor area (CAA), which is the central communication area for Language Environment. All thread- and enclave-related resources are anchored, provided for, or can be obtained through the CAA. The CAA is generated during thread initialization and deleted during thread termination. When calling Language Environment-conforming routines, register 12 points to the address of the CAA.

Use CAA fields as described. Do not modify fields and do not use routine addresses as entry points, except as specified. Fields marked 'Reserved' exist for migration of specific languages, or internal use by Language Environment. Language Environment defines their location in the CAA, but not their use. Do not use or reference them except as specified by the language that defines them.

Figure 10 on page 63 shows the format of the Language Environment CAA.

-18	CEECAAEYE CL8'EECAA '			
-0C - -01	Reserved			
000000	CEECAAFLAG0	Reserved	CEECAALANGP	Reserved
000004	Reserved			
000008	CEECAABOS - Start of Current Storage Segment			
00000C	CEECAAEOS - End of Current Storage Segment			
000010	Reserved - 10 thru 43			
000044	CEECAATORC - POSIX Thread-Level Return Code			
	Reserved - 48 thru 73			
000074	CEECAATOVF - Addr of Stack Overflow Routine			
	Reserved - 78 thru 11F			
000120	CEECAAAATTN - Addr of CEL Attention Handler			
000124	Reserved - 124 thru 15B			
00015C	CEECAAHLEXIT - Flag for User Hook Exit			
~~				
0001A8	CEECAATHOOKS - Execute Hooks - 18 4-Byte Hooks			
~~				
0001F0	Reserved - 1F0 thru 2AB			
0002AC	CEECAASYSTM	CEECAHRDWR	CEECAASBSYS	CEECAAFLAG2
0002B0	CEECAALEVEL CAA Level ID	CEECAA_PM	Reserved	
0002B4	CEECAAGETLS - Addr of CEL Library Stack Mgr			
0002B8	CEECAACELV - Addr of CEL LIBVEC			
0002BC	CEECAAGETS - Addr of CEL Get Stack Stg Rtn			
0002C0	CEECAALBOS - Start of Library Stack Stg Seg			
0002C4	CEECAALEOS - End of Library Stack Stg Seg			
0002C8	CEECAALNAB - Next Available Byte of Lib Stg			
00024C	CEECAADMC - Addr of ESPIE Shunt Routine			
0002D0	CEECAAACD - Reserved			
0002D4	CEECAAARS - Reserved			
0002D8	CEECAAERR - Addr of the Current Condition Information Block			
0002DC	CEECAAGETSEX - Addr of CEL Stack Stg Extender			
0002E0	CEECAADDSA - Addr of the Dummy DSA			
0002E4	CEECAASECTSZ - Vector Section Size			

0002E8	CEECAAPARTSUM - Vector Partial Sum Number	
0002EC	CEECAASSEXPNT - Log of Vector Section Size	
0002F0	CEECAAEDB - Addr of the EDB	
0002F4	CEECAAPCB - Addr of the PCB	
0002F8	CEECAAIEPTR - Addr of the CAA Eyecatcher	
0002FC	CEECAAPTR - Addr of this CAA	
000300	CEECAAGETS1 - Stack Overflow for Non-DSA Save Area	
000304	CEECAASHAB - Reserved	
000308	CEECAAPRGCK - Program Interrupt Code for CAADMC	
00030C	CEECAAF1AG1	Reserved
000310	CEECAAURC - Thread Level Return Code	
000314	CEECAAESS - End of Current User Stack	
000318	CEECAALESS - End of Current Library Stack	
00031C	CEECAAOGETS - Overflow from User Stack	
000320	CEECAAOGETLS - Overflow from Library Stack	
000324	CEECAAPICIB - Addr of the Preinit Compatibility Control Block	
000328	CEECAARSRV2 - Reserved	
00032C	CEECAAGOSMR - Reserved	Reserved
000330	CEECAALEOV - Addr of z/OS UNIX MVS Library Vector	
000334	CEECAA_SIGSCTR - SIGSAFE Counter	
000338	CEECAA_SIGSFLG - SIGSAFE Flags	
00033C	CEECAATHDID - Thread ID	
~~		
000344	CEECAA_DCARENT - Reserved	
000348	CEECAA_DANCHOR - Reserved	
00034C	CEECAA_CTOC - Reserved	
~~		
000354	CEECAACICRSRN - CICS Reason Code	
000358	CEECAAMEMBR - Addr of Thread Member List	
00035C	CEECAA_SIGNAL-STATUS - of Terminating Thread	

Figure 10. Common anchor area (Part 2 of 2)

Table 7 contains a list of CAA fields:

Table 7. List of CAA fields

CAA Field	Explanation								
CEECAAF1AG0	CAA flag bits. The bits are defined as follows: <table> <tr> <th>Bit</th><th>Description</th></tr> <tr> <td>0–5</td><td>Reserved</td></tr> <tr> <td>6</td><td>CEECAAXHDL. A flag used by the condition handler. If the flag is set to 1, the application requires immediate return/percolation to the system on any interrupt or condition handler event.</td></tr> <tr> <td>7</td><td>Reserved</td></tr> </table>	Bit	Description	0–5	Reserved	6	CEECAAXHDL. A flag used by the condition handler. If the flag is set to 1, the application requires immediate return/percolation to the system on any interrupt or condition handler event.	7	Reserved
Bit	Description								
0–5	Reserved								
6	CEECAAXHDL. A flag used by the condition handler. If the flag is set to 1, the application requires immediate return/percolation to the system on any interrupt or condition handler event.								
7	Reserved								

Table 7. List of CAA fields (continued)

CAA Field	Explanation								
CEECAALANGP	PL/I language compatibility flags external to Language Environment. The bits are defined as follows: <table> <tr> <th>Bit</th><th>Description</th></tr> <tr> <td>0–3</td><td>Reserved</td></tr> <tr> <td>4</td><td>CEECAATHFN. A flag set by PL/I to indicate a PL/I FINISH ON-unit is active. If the flag is set to 1, no PL/I FINISH ON-unit is active. If the flag is set to 0, a PL/I FINISH ON-unit could be active.</td></tr> <tr> <td>5–7</td><td>Reserved</td></tr> </table>	Bit	Description	0–3	Reserved	4	CEECAATHFN. A flag set by PL/I to indicate a PL/I FINISH ON-unit is active. If the flag is set to 1, no PL/I FINISH ON-unit is active. If the flag is set to 0, a PL/I FINISH ON-unit could be active.	5–7	Reserved
Bit	Description								
0–3	Reserved								
4	CEECAATHFN. A flag set by PL/I to indicate a PL/I FINISH ON-unit is active. If the flag is set to 1, no PL/I FINISH ON-unit is active. If the flag is set to 0, a PL/I FINISH ON-unit could be active.								
5–7	Reserved								
CEECAABOS	Start of the current storage segment. This field is initially set during thread initialization. It indicates the start of the current stack storage segment. It is altered when the current stack storage segment is changed.								
CEECAAEOS	This field is used to determine if a stack overflow routine must be called when allocating storage from the user stack. Normally, the value of this field will represent the end of the current user stack segment. However, its value can also be zero to force the call of a stack overflow routine for every allocation of storage from the user stack. This field is used by function prologs that do not use FASTLINK linkage conventions.								
CEECAATORC	Thread level return code. The thread level return code set by CEESRC callable service.								
CEECAATOVF	Address of stack overflow routine.								
CEECAAATTN	Address of the Language Environment attention handling routine. The address of the Language Environment attention handling routine supports common run-time environment's polling code convention for attention processing.								
CEECAHLLEXIT	Address of the Exit List Control Block set by the HLL user exit CEEBINT.								
CEECAAHOOKS	Hook area. This is the start of 18 fullword execute hooks. Language Environment initializes each fullword to X'07000000'. The hooks can be altered to support various debugging hook mechanisms.								
CEECAASYSTM	Underlying operating system. The value indicates the operating system supporting the active environment. <table> <tr> <th>Value</th><th>Operating System</th></tr> <tr> <td>0</td><td>Undefined. This value should not appear after Language Environment is initialized.</td></tr> <tr> <td>1</td><td>Unsupported</td></tr> <tr> <td>3</td><td>z/OS</td></tr> </table>	Value	Operating System	0	Undefined. This value should not appear after Language Environment is initialized.	1	Unsupported	3	z/OS
Value	Operating System								
0	Undefined. This value should not appear after Language Environment is initialized.								
1	Unsupported								
3	z/OS								

Table 7. List of CAA fields (continued)

CAA Field	Explanation																		
CEECAHRDWR	Underlying hardware. This value indicates the type of hardware on which the routine is running. <table> <tr> <th>Value</th><th>Hardware</th></tr> <tr> <td>0</td><td>Undefined. This value should not appear after Language Environment is initialized.</td></tr> <tr> <td>1</td><td>Unsupported</td></tr> <tr> <td>2</td><td>System/370™, non-XA</td></tr> <tr> <td>3</td><td>System/370, XA</td></tr> <tr> <td>4</td><td>System/370, ESA</td></tr> </table>	Value	Hardware	0	Undefined. This value should not appear after Language Environment is initialized.	1	Unsupported	2	System/370™, non-XA	3	System/370, XA	4	System/370, ESA						
Value	Hardware																		
0	Undefined. This value should not appear after Language Environment is initialized.																		
1	Unsupported																		
2	System/370™, non-XA																		
3	System/370, XA																		
4	System/370, ESA																		
CEECAASBSYS	Underlying subsystem. This value indicates the subsystem (if any) on which the routine is running. <table> <tr> <th>Value</th><th>Subsystem</th></tr> <tr> <td>0</td><td>Undefined. This value should not occur after Language Environment is initialized.</td></tr> <tr> <td>1</td><td>Unsupported</td></tr> <tr> <td>2</td><td>None. The routine is not running under a Language Environment-recognized subsystem.</td></tr> <tr> <td>3</td><td>TSO</td></tr> <tr> <td>4</td><td>IMS</td></tr> <tr> <td>5</td><td>CICS</td></tr> </table>	Value	Subsystem	0	Undefined. This value should not occur after Language Environment is initialized.	1	Unsupported	2	None. The routine is not running under a Language Environment-recognized subsystem.	3	TSO	4	IMS	5	CICS				
Value	Subsystem																		
0	Undefined. This value should not occur after Language Environment is initialized.																		
1	Unsupported																		
2	None. The routine is not running under a Language Environment-recognized subsystem.																		
3	TSO																		
4	IMS																		
5	CICS																		
CEECAFLAG2	CAA Flag 2. <table> <tr> <th>Bit</th><th>Description</th></tr> <tr> <td>0</td><td>Bimodal addressing is available.</td></tr> <tr> <td>1</td><td>Vector hardware is available.</td></tr> <tr> <td>2</td><td>Thread terminating.</td></tr> <tr> <td>3</td><td>Initial thread</td></tr> <tr> <td>4</td><td>Library trace is active. The TRACE run-time option was set.</td></tr> <tr> <td>5</td><td>Reserved</td></tr> <tr> <td>6</td><td>CEECAA_ENQ_Wait_Interruptible. Thread is in an enqueue wait.</td></tr> <tr> <td>7</td><td>Reserved</td></tr> </table>	Bit	Description	0	Bimodal addressing is available.	1	Vector hardware is available.	2	Thread terminating.	3	Initial thread	4	Library trace is active. The TRACE run-time option was set.	5	Reserved	6	CEECAA_ENQ_Wait_Interruptible. Thread is in an enqueue wait.	7	Reserved
Bit	Description																		
0	Bimodal addressing is available.																		
1	Vector hardware is available.																		
2	Thread terminating.																		
3	Initial thread																		
4	Library trace is active. The TRACE run-time option was set.																		
5	Reserved																		
6	CEECAA_ENQ_Wait_Interruptible. Thread is in an enqueue wait.																		
7	Reserved																		
CEECAALEVEL	Language Environment level identifier. This contains a unique value that identifies each release of Language Environment. This number is incremented for each new release of Language Environment.																		
CEECAA_PM	Image of current program mask.																		
CEECAAGETLS	Address of stack overflow for library routines.																		
CEECAACELV	Address of the Language Environment library vector. This field is used to locate dynamically loaded Language Environment routines.																		

Table 7. List of CAA fields (continued)

CAA Field	Explanation
CEECAAGETS	Address of the Language Environment prolog stack overflow routine. The address of the Language Environment get stack storage routine is included in prolog code for fast reference.
CEECAALBOS	Start of the library stack storage segment. This field is initially set during thread initialization. It indicates the start of the library stack storage segment. It is altered when the library stack storage segment is changed.
CEECAALEOS	This field is used to determine if a stack overflow routine must be called when allocating storage from the library stack. Normally, the value of this field will represent the end of the current library stack segment. However, its value can also be zero to force the call of a stack overflow routine for every allocation of storage from the library stack. This field is used by function prologs that do not use FASTLINK linkage conventions.
CEECAALNAB	Next available library stack storage byte. This contains the address of the next available byte of storage on the library stack. It is modified when library stack storage is obtained or released.
CEECAADMC	Language Environment shunt routine address. Its value is initially set to 0 during thread initialization. If it is nonzero, this is the address of a routine used in specialized exception processing.
CEECAAACD	Most recent CAASHAB abend code.
CEEAAABCODE	Most recent abend completion code.
CEECAAARS	Most recent CAASHAB reason code.
CEECAAARSNCODE	Most recent abend reason code.
CEECAAERR	Address of the current condition information block. After completion of initialization, this always points to a condition information block. During exception processing, the current condition information block contains information about the current exception being processed. Otherwise, it indicates no exception being processed.
CEECAAGETSEX	Address of the user stack extender routine. This routine is called to extend the current stack frame in the user stack. Its address is in the CEECAA for performance reasons.
CEECAADDSA	Address of the Language Environment dummy DSA. This address determines whether a stack frame is the dummy DSA, also known as the zeroth DSA.
CEECAASECTSIZ	Vector section size. This field is used by the vector math services.
CEECAAPARTSUM	Vector partial sum number. This field is used by the vector math services.
CEECAASSEXPNT	Log of the vector section size. This field is used by the vector math services.
CEECAAEDB	Address of the Language Environment EDB. This field points to the encompassing EDB.
CEECAAPCB	Address of the Language Environment PCB. This field points to the encompassing PCB.

Table 7. List of CAA fields (continued)

CAA Field	Explanation						
CEECAAIEYEPTR	Address of the CAA eye catcher. The CAA eye catcher is CEECAA. This field can be used for validation of the CAA.						
CEECAAPTR	Address of the CAA. This field points to the CAA itself and can be used in validation of the CAA.						
CEECAAGETS1	Non-DSA stack overflow. This field is the address of a stack overflow routine, which cannot guarantee that the current register 13 is pointing at a stack frame. Register 13 must point, at a minimum, to a save area.						
CEECAASHAB	ABEND shunt routine. Its value is initially set to zero during thread initialization. If it is nonzero, this is the address of a routine used in specialized exception processing for ABENDs that are intercepted in the ESTAE exit.						
CEECAAPRGCK	Routine interrupt code for CEECAADMC. If CEECAADMC is nonzero, and a routine interrupt occurs, this field is set to the routine interrupt code and control is passed to the address in CEECAAMDC.						
CEECAAFLAG1	CAA flag bits. The bits are defined as follows: <table> <tr> <th>Bit</th><th>Description</th></tr> <tr> <td>0</td><td>CEECAASORT. A call to DFSORT™ is active.</td></tr> <tr> <td>1–7</td><td>Reserved</td></tr> </table>	Bit	Description	0	CEECAASORT. A call to DFSORT™ is active.	1–7	Reserved
Bit	Description						
0	CEECAASORT. A call to DFSORT™ is active.						
1–7	Reserved						
CEECAAURC	Thread level return code. This is the common place for members to set the return codes for subroutine-to-subroutine return code processing.						
CEECAAESS	This field is used to determine if a stack overflow routine must be called when allocating storage from the user stack. Normally, the value of this field will represent the end of the current user stack segment. However, its value can also be zero to force the call of a stack overflow routine for every allocation of storage from the user stack. This field is used by function prologs that use FASTLINK linkage conventions.						
CEECAALESS	This field is used to determine if a stack overflow routine must be called when allocating storage from the library stack. Normally, the value of this field will represent the end of the current library stack segment. However, its value can also be zero to force the call of a stack overflow routine for every allocation of storage from the library stack. This field is used by function prologs that use FASTLINK linkage conventions.						
CEECAOGETS	Overflow from user stack allocations.						
CEECAOGETLS	Overflow from library stack allocations.						
CEECAARSRV1	Reserved.						
CEECAAPICIB	Address of the preinitialization compatibility control block.						
CEECAOGETSX	User DSA exit from OPLINK.						
CEECAARSRV2	Reserved.						
CEECAAGOSMR	Go some more—Used CEEHTRAV multiple.						
CEECAALEOV	This field is the address of the Language Environment library vector for z/OS UNIX support.						

Table 7. List of CAA fields (continued)

CAA Field	Explanation												
CEECAA_SIGSCTR	SIGSAFE counter.												
CEECAA_SIGSFLG	SIGSAFE flags. SIGSAFE flags indicate the signal safety of the library. <table> <tr> <th>Bit</th><th>Description</th></tr> <tr> <td>0</td><td>CEECAA_SIGPUTBACK. The signal cannot be delivered, therefore the signal is put back to the kernel.</td></tr> <tr> <td>1</td><td>CEECAA_SA_RESTART. Indicates that a signal registered with the SA_RESTART flag interrupted the last kernel call, and the signal catcher returned.</td></tr> <tr> <td>2</td><td>Reserved</td></tr> <tr> <td>3</td><td>CEECAA_SIGSAFE. It is safe to deliver the signal, while in library code.</td></tr> <tr> <td>4</td><td>CEECAA_CANCELSAFE. It is safe to deliver the cancel signal, while in library code.</td></tr> </table>	Bit	Description	0	CEECAA_SIGPUTBACK. The signal cannot be delivered, therefore the signal is put back to the kernel.	1	CEECAA_SA_RESTART. Indicates that a signal registered with the SA_RESTART flag interrupted the last kernel call, and the signal catcher returned.	2	Reserved	3	CEECAA_SIGSAFE. It is safe to deliver the signal, while in library code.	4	CEECAA_CANCELSAFE. It is safe to deliver the cancel signal, while in library code.
Bit	Description												
0	CEECAA_SIGPUTBACK. The signal cannot be delivered, therefore the signal is put back to the kernel.												
1	CEECAA_SA_RESTART. Indicates that a signal registered with the SA_RESTART flag interrupted the last kernel call, and the signal catcher returned.												
2	Reserved												
3	CEECAA_SIGSAFE. It is safe to deliver the signal, while in library code.												
4	CEECAA_CANCELSAFE. It is safe to deliver the cancel signal, while in library code.												
CEECAATHDID	Thread id. This field is the thread identifier.												
CEECAA_DCARENT	DCE's read/write static external anchor.												
CEECAA_DANCHOR	DCE's per-thread anchor.												
CEECAA_CTOC	TOC anchor for CRENT.												
CEECAACICSRSN	CICS reason code from member language.												
CEECAAMEMBR	Address of thread-level member list.												
CEECAA_SIGNAL_STATUS	Signal status of the terminating thread member list.												

The condition information block

The Language Environment condition manager creates a condition information block (CIB) for each condition encountered in the Language Environment environment. The CIB holds data required by the condition handling facilities and pointers to locations of other data. The address of the current CIB is located in the CAA.

For COBOL, Fortran, and PL/I applications, Language Environment provides macros (in the SCEESAMP data set) that map the CIB. For C/C++ applications, the macros are in `leawi.h`.

Figure 11 on page 70 shows the condition information block.

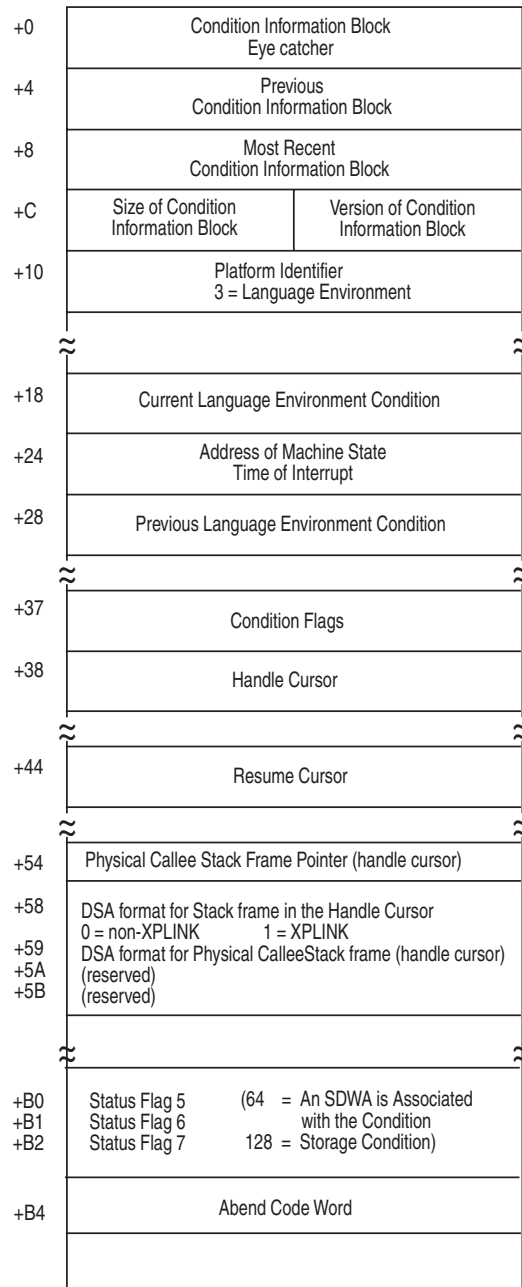


Figure 11. Condition information block (Part 1 of 2)

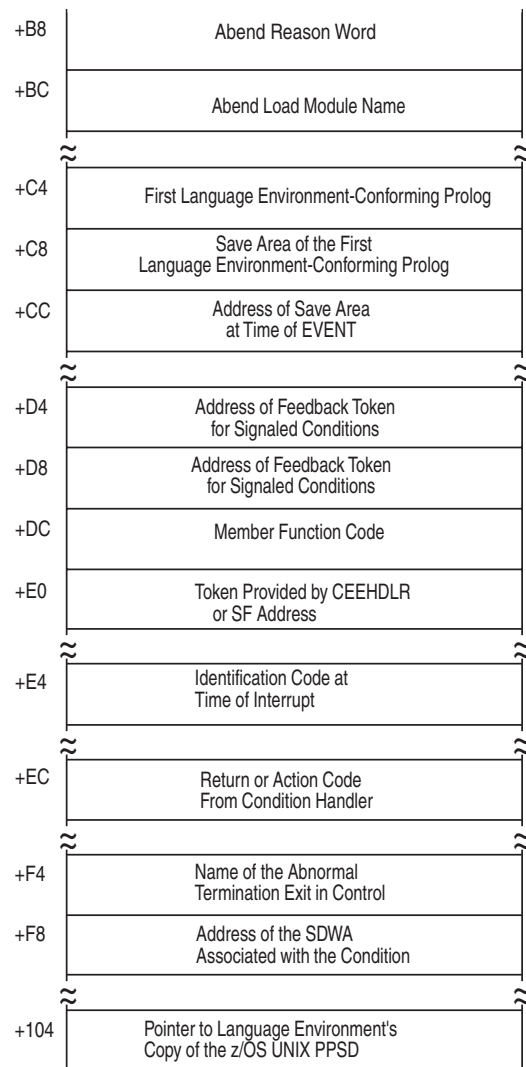


Figure 11. Condition information block (Part 2 of 2)

The flags for Condition Flag 4:

- 2** The resume cursor has been moved
- 4** Message service has processed the condition
- 8** The resume cursor has been moved explicitly

The flags for Status Flag 5, Language Environment events:

- 1** Caused by an attention interrupt
- 2** Caused by a signaled condition
- 4** Caused by a promoted condition
- 8** Caused by a condition management raised TIU
- 32** Caused by a condition signaled via CEEOKILL ¹
- 64** Caused by a program check
- 128** Caused by an abend

The flags for Status Flag 6, Language Environment actions:

1. The signaled-via-CEEOKILL flag is always set with the signaled flag; thus, a signaled condition can have a value of either 2 or 34. (The value is 2 if the signaled condition does not come through CEEOKILL. If it comes through CEEOKILL, its value is 2+32=34.)

- 2 Doing stack frame zero scan
- 4 H-cursor pointing to owning SF
- 8 Enable only pass (no condition pass)
- 16 MRC type 1
- 32 Resume allowed
- 64 Math service condition
- 128 Abend reason code valid

The language-specific function codes for the CIB:

- X'1' For condition procedure
- X'2' For enablement
- X'3' For stack frame zero conditions

Using the machine state information block

The Language Environment machine state information block contains condition information pertaining to the hardware state at the time of the error. Figure 12 shows the machine state information block.

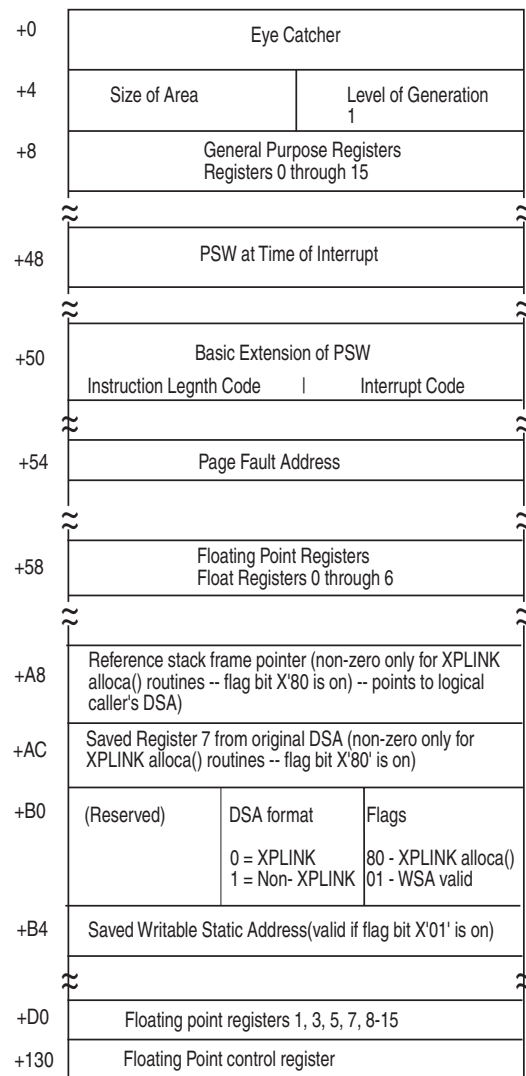


Figure 12. Machine state information block

Multiple enclave dumps

If multiple enclaves are used, the dump service generates data and storage information for the most current enclave and moves up the chain of enclaves to the starting enclave in a LIFO order. For example, if two enclaves are used, the dump service first generates output for the most current enclave. Then the service creates output for the previous enclave. A thread terminating in a non-POSIX environment is analogous to an enclave terminating because Language Environment Version 1 supports only single threads.

Figure 13 on page 74 illustrates the information available in the Language Environment dump and the order of information for multiple enclaves.

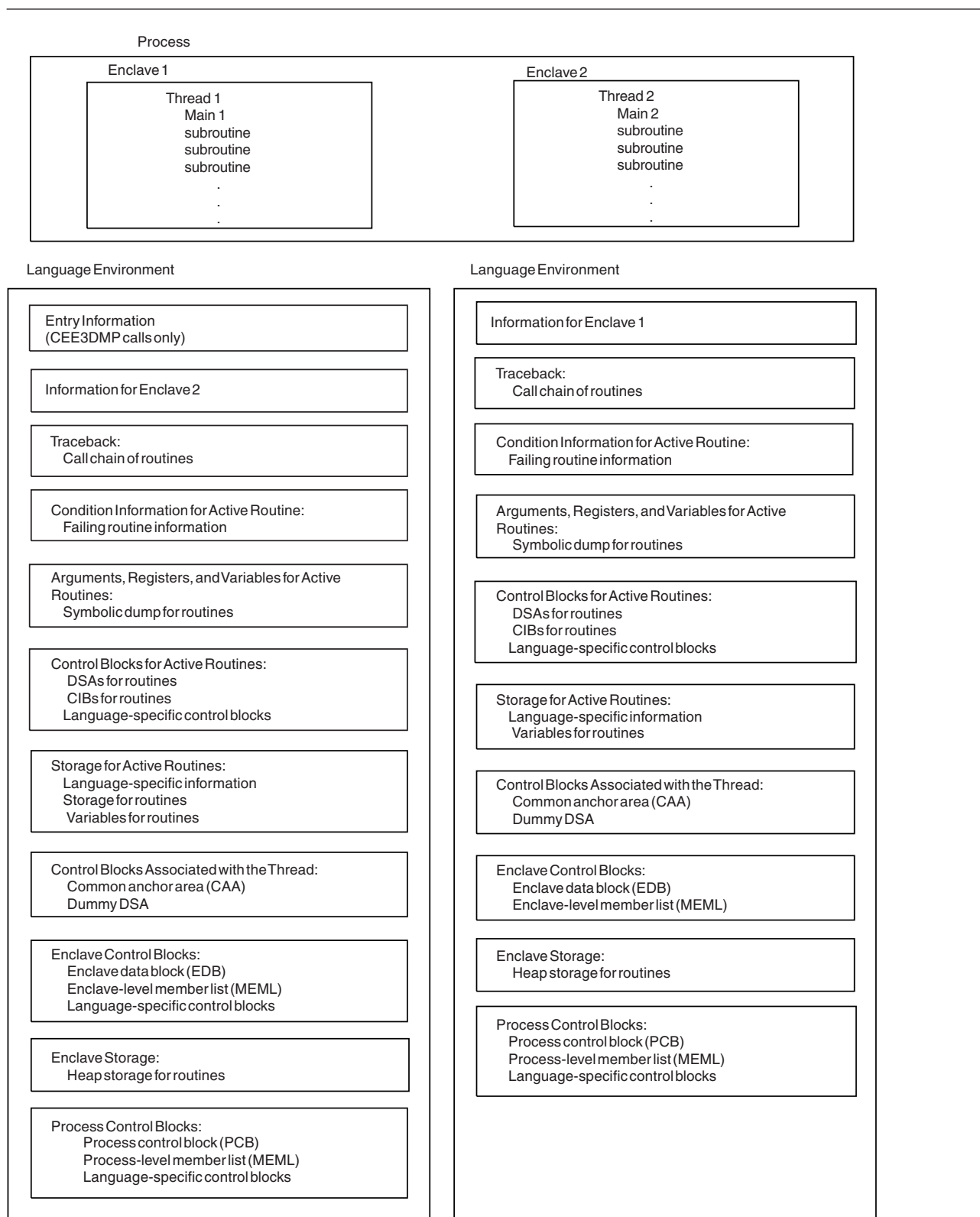


Figure 13. Language Environment dump of multiple enclaves

Generating a system dump

A system dump contains the storage information needed to diagnose errors. You can use Language Environment to generate a system dump through any of the following methods:

TERMTHDACT(UAONLY, UATRACE, or UADUMP)

You can use these run-time options, with TRAP(ON), to generate a system dump if an unhandled condition of severity 2 or greater occurs. For further details regarding the level of dump information produced by each of the TERMTHDACT suboptions, see “Generating a Language Environment dump with TERMTHDACT” on page 39.

TRAP(ON,NOSPIE) TERMTHDACT(UAImm)

TRAP(ON,NOSPIE) TERMTHDACT(UAImm) generates a system dump of the user address space of the original abend or program interrupt prior to the Language Environment condition manager processing the condition.

ABPERC(abcode)

The ABPERC run-time option specifies one abend code that is exempt from the Language Environment condition handler. The Language Environment condition handler percolates the specified abend code to the operating system. The operating system handles the abend and generates a system dump.

ABPERC is ignored under CICS.

Abend Codes in Initialization Assembler User Exit

Abend codes listed in the initialization assembler user exit are passed to the operating system. The operating system can then generate a system dump.

CEE3ABD

You can use the CEE3ABD callable service to cause the operating system to handle an abend.

Refer to system or subsystem documentation for detailed system dump information.

The method for generating a system dump varies for each of the Language Environment run-time environments. The following sections describe the recommended steps needed to generate a system dump in a batch, IMS, CICS, and z/OS UNIX shell run-time environments. Other methods may exist, but these are the recommended steps for generating a system dump.

For details on setting Language Environment run-time options, see *z/OS Language Environment Programming Guide*.

Steps for generating a system dump in a batch run-time environment

Perform the following steps to generate a system dump in a batch run-time environment:

1. Specify run-time options TERMTHDACT(UAONLY, UADUMP, UATRACE, or UAImm), and TRAP(ON). If you specify the suboption UAImm then you must set

TRAP(ON,NOSPIE). The TERMTHDACT suboption determines the level of detail of the Language Environment formatted dump. For further details on the TERMTHDACT suboptions, see “Generating a Language Environment dump with TERMTHDACT” on page 39.

2. Include a SYSMDUMP DD card with the desired data set name and DCB information:

LRECL=4160, BLKSIZE=4160, and RECFM=FBS.

3. Rerun the program.

When you are done, you have a generated system dump in a batch run-time environment.

Steps for generating a system dump in an IMS run-time environment

Perform the following steps to generate a system dump in an IMS run-time environment:

1. Specify run-time options TERMTHDACT(UAONLY, UADUMP, UATRACE, or UAIMM), ABTERM(ABEND), and TRAP(ON). If you specify the suboption UAIMM, then you must set TRAP(ON,NOSPIE). The TERMTHDACT suboption determines the level of detail of the Language Environment formatted dump. For further details on the TERMTHDACT suboptions, see “Generating a Language Environment dump with TERMTHDACT” on page 39.

Restriction: In an IMS environment, you can only use CEEUOPT, CEEDOPT, or CEEROPT to change run-time options. CEEUOPT cannot be used by OS/VS COBOL or non-Language Environment assembler.

2. Include a SYSMDUMP DD card with the desired data set name and DCB information:

LRECL=4160, BLKSIZE=4160, and RECFM=FBS.

3. Rerun the program.

When you are done, you have a generated system dump in an IMS run-time environment.

Steps for generating a system dump in a CICS run-time environment

Before you begin: Under CICS, a system dump provides the most useful information for diagnosing problems. However, if you have a Language Environment U4038 abend, CICS will not generate a system dump. In order to generate diagnostic information for a CICS run-time environment with a Language Environment U4038 abend, you must create a Language Environment U4039 abend. For instructions on how to create a Language Environment U4039 abend, see “Steps for generating a Language Environment U4039 abend” on page 77.

Perform the following steps to generate a system dump in a CICS run-time environment:

1. Specify run-time options TERMTHDACT(UAONLY, UADUMP, or UATRACE), ABTERM(ABEND), and TRAP(ON). The TERMTHDACT suboption determines the level of detail of the Language Environment formatted dump. For further details on the TERMTHDACT suboptions, see “Generating a Language Environment dump with TERMTHDACT” on page 39.

2. Update the transaction dump table with the CICS supplied CEMT command:

CEMT SET TRD(40XX) SYS ADD

Result: You will see CEMT output.

Example:

```
STATUS: RESULTS - OVERTYPE TO MODIFY
Trd(4088) Sys   Loc Max( 999 ) Cur(0000)
```

3. Rerun the program.

When you are done, you have a generated system dump in a CICS run-time environment.

Steps for generating a Language Environment U4039 abend

If you have a Language Environment U4038 abend, CICS will not generate a system dump. In order to generate diagnostic information, you must create a Language Environment U4039 abend by performing the following steps:

1. Specify DUMP=YES in CICS DFHSIT.
2. Relink your program by including CEEUOPT.
Restriction:CEEUOPT cannot be used by OS/VS COBOL or non-Language Environment assembler.
3. Take CEECOPT from SCEESAMP and modify the Language Environment run-time options TERMTHDACT(UAONLY, UATRACE, or UADUMP), ABTERM(ABEND), and TRAP(ON).
Result:By setting these run-time options, a Language Environment U4039 abend occurs which generates a system dump.
4. Rerun the program.

Note: In the CICS run-time environment, the TERMTHDACT suboption UA IMM is processed the same as UAONLY.

Steps for generating a system dump in a z/OS UNIX shell

Perform the following steps to generate a system dump from a z/OS UNIX shell:

1. Specify where to write the system dump
 - To write the system dump to a z/OS data set, issue the command:
`export _BPXK_MDUMP=filename`

where *filename* is a fully qualified data set name with DCB information: LRECL=4160, BLKSIZE=4160, and RECFM=FBS.

Example:

```
export _BPXK_MDUMP=h1q.mydump
```

- To write the system dump to an HFS file, issue the command:
`export _BPXK_MDUMP=filename`

where *filename* is a fully qualified HFS filename.

Example:

```
export _BPXK_MDUMP=/tmp/mydump.dmp
```

2. Specify Language Environment run-time options:
`export _CEE_RUNOPTS="termthdact(suboption)"`

where *suboption* = UAONLY, UADUMP, UATRACE, or UA IMM. If UA IMM is set, TRAP(ON,NOSPIE) must also be set. The TERMTHDACT suboption determines the level of detail of the Language Environment formatted dump. For further details regarding the TERMTHDACT suboptions, see “Generating a Language Environment dump with TERMTHDACT” on page 39.

3. Rerun the program.

When you are done, the system dump is written to the data set name or HFS file name specified.

For additional BPXK_MDUMP information see *z/OS UNIX System Services Command Reference*.

Note: You can also specify the signal SIGDUMP on the kill command to generate a system dump of the user address space. For more information regarding the SIGDUMP signal, see *z/OS UNIX System Services Command Reference*.

Formatting and analyzing system dumps

You can use the interactive problem control system (IPCS) to format and analyze system dumps. Language Environment provides an IPCS Verbexit LEDATA that can be used to format Language Environment control blocks.

For more information on using IPCS, refer to *z/OS MVS IPCS User's Guide*.

Preparing to use the Language Environment support for IPCS

Guidelines: Use the following guidelines before you use IPCS to format Language Environment control blocks:

- Ensure that your IPCS job can find the CEEIPCSP member.

IPCS provides an exit control table with imbed statements to enable other products to supply exit control information. The IPCS default table, BLSCECT, normally in the SYS1.PARMLIB library, has the following entry for Language Environment:

```
IMBED MEMBER(CEEIPCSP) ENVIRONMENT(IPCS)
```

The Language Environment-supplied CEEIPCSP member, installed in the SYS1.PARMLIB library, contains the Language Environment-specific entries for the IPCS exit control table.

- Provide an IPCSPARM DD statement to specify the libraries containing the IPCS control tables.

Example:

```
//IPCS Parm DD DSN=SYS1.PARMLIB,DISP=SHR
```

- Ensure that your IPCS job can find the Language Environment-supplied ANALYZE exit routines installed in the SYS1.MIGLIB library.
- To aid in debugging system or address space hang situations, Language Environment mutexes, latches and condition variables can be displayed if the CEEIPCSP member you are using is updated to identify the Language Environment ANALYZE exit, by including the following statement:

```
EXIT EP(CEEEANLZ) ANALYZE
```

Language Environment IPCS Verbexit – LEDATA

Use the LEDATA Verbexit to format data for Language Environment. This Verbexit provides information about the following topics:

- A summary of Language Environment at the time of the dump
- Run-time Options
- Storage Management Control Blocks
- Condition Management Control Blocks
- Message Handler Control Blocks
- C/C++ Control Blocks
- COBOL Control Blocks

Format

Syntax

```
VERBEXIT LEDATA [ 'parameter[,parameter]...' ]
```

Report Type Parameters:

```
[ SUM ]  
[ HEAP | STACK | SM ]  
[ HPT(value) ]  
[ CM ]  
[ MH ]  
[ CEEDUMP ]  
[ ALL ]
```

Data Selection Parameters:

```
[ DETAIL | EXCEPTION ]
```

Control Block Selection Parameters:

```
[ CAA(caa-address) ]  
[ DSA(dsa-address) ]  
[ TCB(tcb-address) ]  
[ ASID(address-space-id) ]  
[ NTHREADS(value) ]
```

Parameters

Report type parameters

Use these parameters to select the type of report. You can specify as many reports as you wish. If you omit these parameters, the default is SUMMARY.

SUMmary

Requests a summary of the Language Environment at the time of the dump.

The following information is included:

- TCB address
- Address Space Identifier
- Language Environment Release
- Active members
- Formatted CAA, PCB, RCB, EDB and PMCB
- Run-time Options in effect

HEAP | STACK | SM

HEAP

Requests a report on Storage Management control blocks pertaining to HEAP storage, as well as a detailed report on heap segments. The detailed report includes information about the free storage tree in the heap segment, and information about each allocated storage element.

Note: Language Environment does not provide support for alternative Vendor Heap Manager (VHM) data.

STACK

Requests a report on Storage Management control blocks pertaining to STACK storage.

SM

Requests a report on Storage Management control blocks. This is the same as specifying both HEAP and STACK.

HPT(value)

Requests that the heappools trace (if available) be formatted. If the value is 0 or *, the trace for every heappools poolid is formatted. If the value is a single number (1-12), the trace for the specific heappools poolid is formatted.

CM

Requests a report on Condition Management control blocks.

MH

Requests a report on Message Handler control blocks.

CEEDump

Requests a CEEDUMP-like report. Currently this includes the traceback, the Language Environment trace, and thread synchronization control blocks at process, enclave and thread levels.

ALL

Requests all above reports, as well as C/C++ and COBOL reports.

Data selection parameters

Data selection parameters limit the scope of the data in the report. If no data selection parameter is selected, the default is DETAIL.

DETail

Requests formatting all control blocks for the selected components. Only significant fields in each control block are formatted.

Note: For the Heap and Storage Management Reports, the DETAIL parameter will provide a detailed heap segment report for each heap segment in the dump. The detailed heap segment report includes information on the free storage tree in the heap segments, and all allocated storage elements. This report will also identify problems detected in the heap management data structures. For more information about the Heap Reports, see “Understanding the HEAP LEDATA output” on page 95.

EXCeption

Requests validating all control blocks for the selected components. Output is only produced naming the control block and its address for the first control block in a chain that is invalid. Validation consists of control block header verification at the very least.

Note: For the Summary, CEEDUMP, C/C++, and COBOL reports, the EXCEPTION parameter has not been implemented. For these reports, DETAIL output is always produced.

Control block selection parameters

Use these parameters to select the CAA and DSA control blocks used as the starting points for formatting.

CAA(caa-address)

specifies the address of the CAA. If not specified, the CAA address is obtained from the TCB.

DSA(dsa-address)

specifies the address of the DSA. If not specified, the DSA address is assumed to be the register 13 value for the TCB.

TCB(tcb-address)

specifies the address of the TCB. If not specified, the TCB address of the current TCB from the CVT is used.

ASID(address-space-id)

specifies the hexadecimal address space id. If not specified, the IPCS default address space id is used. This parameter is not needed when the dump only has one address space.

NTHREADS(value)

specifies the number of TCBs for which the traceback will be displayed. If NTHREADS is not specified, *value* will default to (1). If *value* is specified as asterisk (*), all TCBs will be displayed.

Understanding the Language Environment IPCS Verbexit LEDATA output

The Language Environment IPCS Verbexit LEDATA generates formatted output of the Language Environment run-time environment control blocks from a system dump. Figure 14 on page 82 illustrates the output produced when the LEDATA Verbexit is invoked with the ALL parameter. The system dump being formatted was obtained by specifying the TERMTHDACT(UADUMP) run-time option when running the program CELSAMP in Figure 5 on page 44. “Sections of the Language Environment LEDATA Verbexit formatted output” on page 92 describes the information contained in the formatted output. Ellipses are used to summarize some sections of the dump.

For easy reference, the sections of the following dump are numbered to correspond with the descriptions in “Sections of the Language Environment LEDATA Verbexit formatted output” on page 92.

```

IP VERBEXIT LEDATA 'ALL CAA(060F0200) DSA(07402968)'
*****
LANGUAGE ENVIRONMENT DATA
*****
Language Environment Productt 04 V01 R03.00

[1]TCB: 006E7378          LE Level: 0D          ASID: 001F

[2]Active Members: C/C++

[3]+000000 FLAG0:00 LANGP:08 BO S:00022000 EOS:00000000
+000044 TORC:00000000 TOVF:800136D8 ATTN:063186E8
+00015C HLLEXIT:00000000 HOOK:50C0D064 05C058C0 C00605CC
+0001A4 DIMA:00008B58 ALLOC:0700C3C8 STATE:0700C3C8
+0001B0 ENTRY:0700C3C8 EXIT:0700C3C8 MEXIT:0700C3C8
+0001BC LABEL:0700C3C8 BCALL:0700C3C8 ACALL:0700C3C8
+0001C8 DO:0700C3C8 IFTRUE:0700C3C8 IFFALSE:0700C3C8
+0001D4 WHEN:0700C3C8 OTHER:0700C3C8 CGOTO:0700C3C8
+0001F0 CGENE:0631D7A4 CRENT:067C68F8 CTHD:0631BDFC
+000210 EDCV:8661466C CEDB:0631CD54 EDCOV:0660F154
+000258 TCASRV_USERWORD:00000000 TCASRV_WORKAREA:06318038
+000260 TCASRV_GETMAIN:00000000 TCASRV_FREEMAIN:00000000
+000268 TCASRV_LOAD:8000E018 TCASRV_DELETE:8000DF38
+000270 TCASRV_EXCEPTION:00000000 TCASRV_ATTENTION:00000000
+000278 TCASRV_MESSAGE:00000000 LWS:00017630 SAVR:0656BA3A
+0002AC SYSTM:03 HRDWR:03 SBSYS:02 FLAG2:B0 LEVEL:0D
+0002B1 PM:04 GETLS:00011318 CELV:00018038 GETS:00011408
+0002C0 LBOS:00021000 LEOS:00000000 LNAB:00021018
+0002CC DMC:00000000 ABCODE:00000000 RSNCODE:00000000
+0002D8 ERR:00023308 GETSX:000129E0 DDSA:00017428
+0002E4 SECTSI:00000000 PARTSUM:00000000
+0002EC SSEXPNT:00000000 EDB:000159D0 PCB:00015560
+0002F8 EYEPR:00016AA8 PTR:00016AC0 GETS1:00012AD8
+000304 SHAB:00000000 PRGCK:00000004 FLAG1:00 URC:00000000
+000314 ESS:00000000 LESS:00000000 OGETS:00013200
+000320 OGETLS:00000000 PICICB:00000000 GETSX:00000000 GOSMR:0000
+000330 LEOV:067C5038 SIGSCTR:00000000 SIGSFLG:00000000
+00033C THDID:06769920 00000000 DCRENT:00000000
+000348 DANCHOR:00000000 CTOC:00000000 RCB:00014918
+000354 CICSRSN:00000000 MEMBR:000174C8
+00035C SIGNAL_STATUS:00000008 HCOM_REG7:00000000
+000364 STACKFLOOR:7FFFFFFF HPGETS:00000000 EDCHPXV:00000000
+000370 FOR1:00000000 FOR2:00000000 THREADHEADID:00017304
+00037C SYS_RTNCODE:00000000 SYS_RSNCODE:00000000 GETFN:0646A3A0
+000390 SIGNGPTR:00016E54 SIGNG:00000001 FORDBG:00000000
+00039C AB_STATUS:00 STACKDIRECTION:00 AB_GR0:00000000
+0003A4 AB_ICD1:00000000 AB_ABCC:00000000 AB_CRC:00000000
+0003B0 GT5:0000FC18 LERN5N1:00000000 HERP:063CF7C8
+0003BC USTKBOS:00000000 USTKEOS:00000000
+0003C4 USERRTN:00000000 UDHOOK:A7F4FEE8 A7F401A0
+0003D0 HPXV_B:064DB218 HPXV_M:064DC7C0 HPXV_L:064E1170
+0003DC HPXV_O:064E1260 SMCB:000171F0 ERRCM:063186A0
+000430 MIB_PTR:00000000 STV:00 A_ISA:00000000
+00043C ISA_SIZE:00000000 PTATPTR:00000000 SIGSSDSA1:00
+000445 SIGSSDSA2:00 STACKUNSTABLE:00 STACK_FLAG:00
+000448 SQLADDR:0631A618 VBA:00000000 TCS:068ECE78
+00045C THDSTATUS:00000000 TICB_PTR:063199C8
+0004A4 FWD_CHAIN:00016AC0 BKWD_CHAIN:00016AC0

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 1 of 11)

```

[4]CEEPCB: 00015560
+000000 PCB EYE:CEEPCB SYSTM:03 HRDWR:03 SBSYS:02 FLAG2:98
+00000C DBGEH:00000000 DMBR:00015790 ZL0D:064D3D20
+000020 ZDEL:064CB928 ZGETST:064D1AB0 ZFREEST:064D15E0
+00002C LVTL:0630B978 RCB:00014918 SYSEIB:00000000
+000038 PSL:00000000 PSA:000159D0 PSRA:064D1908
+000044 OMVS_LEVEL:7F000000 PCB_CHAIN:00000000
+00004C PCB_VSSFE:00013304 PCB_PRFEH:00000000
+000084 LPKA_LODTYP:00000003 IMS:00000000 ABENDCODE:00000000
+000090 REASON:00000000 F3456:000080C2 MEML:00015778
+00009C MEMBR:00015790 PCB_EYE:00000000 PCB_BKC:00005F78
+0000A8 PCB_FWC:00000000 PCB_R14:863002BE
+0000B0 PCB_R15:00006EA8 PCB_R0:7D000016 PCB_R1:00005FE8
+0000BC PCB_R2:06300080 PCB_R3:00000000 PCB_R4:00000000
+0000C8 PCB_R5:00000000 PCB_R6:00000000 PCB_R7:00000000
+0000D4 PCB_R8:063040B0 PCB_R9:006E7710 PCB_R10:00000000
+0000E0 PCB_R11:863001F2 PCB_R12:00000000 CELV24:00018038
+0000EC CELV31:0630ED20 SLDR:8000E108 SECTSIZE:00000000
+0000F8 PARTSUM:00000000 SSEXPT:00000000 BMPS:06321FC8
+000104 BMPE:0639A708 BLEHL:0630B450 BCMXB:00014B08 BSTV:02
+000111 PM_BYTE:00 INI AMODE:00 FLAGS1:28 ISA:06306000
+000118 ISA_SIZ:00018A5C SRV_CNT:00000000
+000120 SRV_UWORD:00000000 WORKAR:00000000 LOAD:0000E6C0
+00012C DELETE:0000E338 GETSTOR:00010430 FREESTOR:0000FF30
+000138 EXCEPT:00000000 ATTN:00000000 MSGS:00000000
+000144 ABEND:000080D8 MSGOU:0000B2B8 GLAT:0642A090
+000150 RLAT:06452B40 ELAT:06423398 IPTQ:064634B0
+00015C IENV:06462680 DBG_LODTYP:FFFFFFFF DUMMY_STK:06306008
+000168 DUMMY_LIB:00014000 DUMMY_CAA:0630A010
+000170 TST_LVL:FFFFFFFF GETCAA:00014D38 SETCAA:00014D40
+00017C LLTPTR:0630AD30 AUE:00000000 RC:00000000
+000188 REASON:00000000 RC_MOD:00000000 AUE_UWORD:00000000
+000194 FB_TOKEN:..... EOV:067C5038 PPA:068D1F44
+0001A8 PPA_SIZ:00000A00 BELOW:00014000 BELOW_LEN:00003940
+0001B4 PICB:06315228 UTL1:06315238 ZINA:864D26A0
+0001C0 ZINB:000138D8 XPLINKFLAGS:00 FLAGS5:80
+0001D4 LANGINIT:00000001 00000000 00000000 00000000 0000
+0001E8 NUMINIT:00000001 LASTINIT:00000003
+0001F0 LANGREUSE:00000000 00000000 00000000 00000000 0000
+000202 REUSEMEMS:00000000 00000000 00000000 00000000 0000

CEEMEML: 00015790
+000000 MEMLDEF:..... EXIT:063A9C58 LLVTL:00000000

[5]CEERCB: 00014918
+000000 EYE:CEERCB SYSTM:03 HRDWR:03 SBSYS:02 FLAGS:80
+000014 DMBR:0630AC08 ZL0D:064D5550 ZDEL:064CD008
+000020 ZGETST:064D1AB0 ZFREEST:064D15E0 VERSION_ID:03020A00

[6]CEEEDB: 000159D0
+000000 EYE:CEEEDB FLAG1:D7 BIPM:00 BPM:00
+00000B CREATOR_ID:01 MEMBR:00016980 OPTCB:00016058
+000014 URC:00000000 RSNCD:00000000 DBGEH:00000000
+000020 BANHP:00015E98 BBEHP:00015EC8 BCELV:00018038
+00002C PCB:00015560 ELIST:00000000 PL_ASTRPTR:00014808
+000038 DEFPLPTR:00015AF0 CXIT_PAGE:00000000
+000040 DEBUG_TERMID:00000000 PARENT:00000000 R13_PARENT:00005F78
+000054 LEOV:067C5038 ENVAR:063182E0 ENVIRON:00015A28
+000060 CEEOSIGR:0000DAF8 OTRB:068D1000 PSA31:0631EA5C
+00006C PSL31:00000000 PSA24:00017940 PSL24:00000000
+000078 PSRA:064D1710 CAACHAIN:00016AC0 FLAG1A:90
+000084 CEEOSGR1:0000DE66 MEMBERCOMPAT1:00
+000090 THREADSACTIVE:00000001 CURMSGFILEDCBPTR:00014B88
+000098 CEEINT_INPUT_R1:00005FE8 LAST_RBADDR:006E7A50
+0000A0 LAST_RBCNT:00000001

CEEMEML: 00016980
+000000 MEMLDEF:..... EXIT:063A9C58 LLVTL:00000000

[7]PMCB: 06318008
+000000 EYE:PMCB PREV$:00000000 NEXT$:00000000
+000010 LVT_CURR$:00018038 LLT_CURR$:068D47F8 FLAGS:A0000000

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 2 of 11)

```

[8]
Language Environment Run-Time Options in effect.
LAST WHERE SET      Override  OPTIONS
*****
INSTALLATION DEFAULT OVR      ABPERC (NONE)
INSTALLATION DEFAULT OVR      ABTERMENC (ABEND)
INSTALLATION DEFAULT OVR      NOAIXBLD
INSTALLATION DEFAULT OVR      ALL31 (ON)
INSTALLATION DEFAULT OVR      ANYHEAP (00016384,00008192,ANY ,FREE)
INSTALLATION DEFAULT OVR      NOAUTOTASK
INSTALLATION DEFAULT OVR      BELOWHEAP (00008192,00004096,FREE)
INSTALLATION DEFAULT OVR      CBLOPTS (ON)
INSTALLATION DEFAULT OVR      CBLPSHPOP (ON)
INSTALLATION DEFAULT OVR      CBLQDA (OFF)
INSTALLATION DEFAULT OVR      CHECK (ON)
INSTALLATION DEFAULT OVR      COUNTRY (US)
INSTALLATION DEFAULT OVR      NODEBUG
INSTALLATION DEFAULT OVR      DEPTHCONDLMT (00000010)
INSTALLATION DEFAULT OVR      ENVAR ("")
INSTALLATION DEFAULT OVR      ERRCOUNT (00000000)
INSTALLATION DEFAULT OVR      ERRUNIT (00000006)
INSTALLATION DEFAULT OVR      FILEHIST
INSTALLATION DEFAULT OVR      FILETAG (NOAUTOCVT,NOAUTOTAG)
DEFAULT SETTING      OVR      NOFLOW
INSTALLATION DEFAULT OVR      HEAP (00032768,00032768,ANY ,
                                KEEP,00008192,00004096)
PROGRAMMER DEFAULT   OVR      HEAPCHK (ON,00000001,00000000,00000000)
INSTALLATION DEFAULT OVR      HEAPPOLLS (OFF,
                                00000008,00000010,
                                00000032,00000010,
                                00000128,00000010,
                                00000256,00000010,
                                00001024,00000010,
                                00002048,00000010)
INSTALLATION DEFAULT OVR      INFOMSGFILTER (OFF)
INSTALLATION DEFAULT OVR      INQPCOPN
INSTALLATION DEFAULT OVR      INTERRUPT (OFF)
INSTALLATION DEFAULT OVR      LIBRARY (SYSCEE)
INSTALLATION DEFAULT OVR      LIBSTACK (00004096,00004096,FREE)
INSTALLATION DEFAULT OVR      MSGFILE (SYSOUT ,FBA ,00000121,00000000,
                                NOENQ)
INSTALLATION DEFAULT OVR      MSGQ (00000015)
INSTALLATION DEFAULT OVR      NATLANG (ENU)
IGNORED              OVR      NONONIPSTACK (See THREADSTACK)
INSTALLATION DEFAULT OVR      OCSTATUS
INSTALLATION DEFAULT OVR      NOPC
INSTALLATION DEFAULT OVR      PLITASKCOUNT (00000020)
PROGRAMMER DEFAULT   OVR      POSIX (ON)
INSTALLATION DEFAULT OVR      PROFILE (OFF,"")
INSTALLATION DEFAULT OVR      PRTUNIT (00000006)
INSTALLATION DEFAULT OVR      PUNUNIT (00000007)
INSTALLATION DEFAULT OVR      RDRUNIT (00000005)
INSTALLATION DEFAULT OVR      RECPAD (OFF)
INSTALLATION DEFAULT OVR      RPTOPTS (OFF)
PROGRAMMER DEFAULT   OVR      RPTSTG (ON)
INSTALLATION DEFAULT OVR      NORTEREUS
INSTALLATION DEFAULT OVR      RTLS (OFF)
INSTALLATION DEFAULT OVR      NOSIMVRD
INSTALLATION DEFAULT OVR      STACK (00131072,00131072,ANY ,KEEP,
                                00524288,00131072)
INSTALLATION DEFAULT OVR      STORAGE (NONE,NONE,NONE,00008192)
PROGRAMMER DEFAULT   OVR      TERMTHDACT (UADUMP,,00000096)
INSTALLATION DEFAULT OVR      NOTEST (ALL,*,PROMPT,INSPREF)
INSTALLATION DEFAULT OVR      THREADHEAP (00004096,00004096,ANY ,KEEP)
INSTALLATION DEFAULT OVR      THREADSTACK (OFF,00004096,00004096,ANY ,FREE,
                                00131072,00131072)
PROGRAMMER DEFAULT   OVR      TRACE (ON,01048576,NODUMP,LE=00000001)
INSTALLATION DEFAULT OVR      TRAP (ON,SPIE)
INSTALLATION DEFAULT OVR      UPSI (00000000)
INSTALLATION DEFAULT OVR      NOUSRHDLR ()
INSTALLATION DEFAULT OVR      VCTRSVAVE (OFF)
INSTALLATION DEFAULT OVR      VERSION ()
INSTALLATION DEFAULT OVR      XPLINK (OFF)
INSTALLATION DEFAULT OVR      XUFLOW (AUTO)
*****

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 3 of 11)

[9]Heap Storage Control Blocks

```

ENSM: 00015E50
+000000 EYE_CATCHER:ENSM ST_HEAP_ALLOC_FLAG:00000000
+000008 ST_HEAP_ALLOC_VAL:00000000 ST_HEAP_FREE_FLAG:00000000
+000010 ST_HEAP_FREE_VAL:00000000 REPORT_STORAGE:00015F2C
+000018 UHEAP:C8D7C3C2 068D5000 068D5000 00008000 00008000 00002000 00001000 00000000 00
+000048 AHEAP:C8D7C3C2 067C7000 068F5000 00004000 00002000 00002000 00001000 00000000 00
+000078 BHEAP:C8D7C3C2 00042000 00042000 00002000 00001000 00002000 00001000 80000000 00
+0000A8 ENSM_ADDL_HEAPS:068F0200

```

```

STSB: 00015F2C
+000000 EYE_CATCHER:STSB CRHP_REQ:00000002 DSHP_REQ:00000001
+00000C IPT_INIT_SIZE:00020000 NONIPT_INIT_SIZE:00020000
+000014 IPT_INCR_SIZE:00020000 NONIPT_INCR_SIZE:00020000
+00001C THEAP_MAX_STOR:00000000

```

Enclave Level Stack Statistics

```

SKSB: 00015FC4
+000000 MAX_ALLOC:00008CF8 CURR_ALLOC:00003460
+000008 LARGEST:00008CF8 GETMAINS:00000001
+000010 FREEMAINS:00000000

```

```

SKSB: 00015FEC
+000000 MAX_ALLOC:00001978 CURR_ALLOC:00000000
+000008 LARGEST:00000DB8 GETMAINS:00000002
+000010 FREEMAINS:00000000

```

```

SKSB: 00015FD8
+000000 MAX_ALLOC:00000330 CURR_ALLOC:00000330
+000008 LARGEST:00000330 GETMAINS:00000001
+000010 FREEMAINS:00000000

```

User Heap Control Blocks

```

HPCB: 00015E68
+000000 EYE_CATCHER:HPCB FIRST:068D5000 LAST:068D5000

```

```

HPSB: 00015F4C
+000000 BYTES_ALLOC:00000C38 CURR_ALLOC:00000C38
+000008 GET_REQ:00000007 FREE_REQ:00000000
+000010 GETMAINS:00000001 FREEMAINS:00000000

```

```

HPSB: 00016000
+000000 BYTES_ALLOC:00000000 CURR_ALLOC:00000000
+000008 GET_REQ:00000000 FREE_REQ:00000000
+000010 GETMAINS:00000000 FREEMAINS:00000000

```

```

HANC: 068D5000
+000000 EYE_CATCHER:HANC NEXT:00015E68 PREV:00015E68
+00000C HEAPID:00000000 SEG_ADDR:068D5000 ROOT_ADDR:068D5C38
+000018 SEG_LEN:00008000 ROOT_LEN:000073C8

```

This is the last heap segment in the current heap.

Free Storage Tree for Heap Segment 068D5000

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	068D5C38	000073C8	00000000	00000000	00000000	00000000	00000000

Map of Heap Segment 068D5000

To display entire segment: IP LIST 068D5000 LEN(X'00008000') ASID(X'001F')

```

068D5020: Allocated storage element, length=00000110. To display: IP LIST 068D5020 LEN(X'00000110') ASID(X'001F')
068D5028: 068D5138 068D5320 068D5350 068D539A 068D53D7 068D5414 068D5451 068D548E |.....).....P.....|

```

```

068D5130: Allocated storage element, length=00000828. To display: IP LIST 068D5130 LEN(X'00000828') ASID(X'001F')
068D5138: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 4 of 11)

[9]Heap Storage Control Blocks

```
068D5958: Allocated storage element, length=00000250. To display: IP LIST 068D5958 LEN(X'00000250') ASID(X'001F')
068D5960: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

068D5BA8: Allocated storage element, length=00000038. To display: IP LIST 068D5BA8 LEN(X'00000038') ASID(X'001F')
068D5BB0: C3C4D3D3 00000000 C0000000 00000000 06300000 063000C0 067C68F8 00000704 |CDLL.....@.8....|

068D5BE0: Allocated storage element, length=00000038. To display: IP LIST 068D5BE0 LEN(X'00000038') ASID(X'001F')
068D5BE8: C3C4D3D3 068D5BB0 80000000 00000000 068ED000 068ED0C8 0631EE80 0000017A |CDLL..$......H.....|

068D5C18: Allocated storage element, length=00000010. To display: IP LIST 068D5C18 LEN(X'00000010') ASID(X'001F')
068D5C20: 068F0180 00000000 |.....|

068D5C28: Allocated storage element, length=00000010. To display: IP LIST 068D5C28 LEN(X'00000010') ASID(X'001F')
068D5C30: 068F01C0 00000000 |.....|

068D5C38: Free storage element, length=000073C8. To display: IP LIST 068D5C38 LEN(X'000073C8') ASID(X'001F')
```

Summary of analysis for Heap Segment 068D5000:
Amounts of identified storage: Free:000073C8 Allocated:00000C18 Total:00007FE0
Number of identified areas : Free: 1 Allocated: 7 Total: 8
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
This is the last heap segment in the current heap.

[10]Stack Storage Control Blocks

```
SMCB: 000171F0
+000000 EYE_CATCHER:SMCB US_EYE_CATCHER:USTK USFIRST:00022000
+00000C USLAST:00022000 USBOS:00022000 USEOS:00042000
+000018 USNAB:000253C0 USINITSZ:00020000 USINCRSZ:00020000
+000024 USANYBELOW:80000000 USKEEFPREE:00000000 USPOOL:80000002
+000030 USPREALLOC:00000001 US_BYTES_ALLOC:00008CF8
+000038 US_CURR_ALLOC:00003460 US_GETMAINS:00000000
+000040 US_FREEMAINS:00000000 US_OPLINK:00 LS_THIS_IS:LSTK
+00004C LSFIRST:00021000 LSLAST:00021000 LSBOS:00021000
+000058 LSEOS:00022000 LS_NAB:00021018 LSINITSZ:00001000
+000064 LSINCRSZ:00001000 LSANYBELOW:80000000
+00006C LSKEEFPREE:00000001 LSPPOOL:80000001 LSPREALLOC:00000001
+000078 LS_BYTES_ALLOC:00000330 LS_CURR_ALLOC:00000330
+000080 LS_GETMAINS:00000000 LS_FREEMAINS:00000000 LS_OPLINK:00
+00008C RSBOS:0001F000 RSEOS:00021000 RSIZE:00002000
+000098 RACTIVE:00000000 SA_REG00:00025460
+0000A0 SA_REG01:000253C0 SA_REG02:00023308
+0000A8 SA_REG03:00000003 SA_REG04:00016058
+0000B0 SA_REG05:00000010 SA_REG06:000230D8
+0000B8 SA_REG07:00023C9F SA_REG08:063CECCD
+0000C0 SA_REG09:063CDCCE SA_REG10:063CCCCF
+0000C8 SA_REG11:063D8838 SA_REG12:00016AC0
+0000D0 SA_REG13:00022CA0 SA_REG14:863D886A
+0000D8 SA_REG15:00000000
+0000DC SAVEREG_XINIT:00000000 00000000 00000000 00000000
+0000EC CEEVGTSI:000114F8 ST_DSA_ALLOC_FLAG:00000000
+0000F4 ST_DSA_ALLOC_VAL:00000000 ALLOCSEG:00000000
+0000FC BELOW16M_FLAG:00000000 LOCAL_ALLOC:FFFFFFF0
+00010C LOCAL_GETMAINS:00000000 LOCAL_FREEMAINS:00000000
+00015C MOREFLAGS:00000000 DS_THIS_IS:... DSFIRST:00017350
+000168 DSLAST:00017350 DSBOS:00017350 DSINITSZ:00000000
+00017C DSINCRSZ:00000000 DSGUARDSZ:00000000
+000184 DSKEEFPREE:00000000 DSPPOOL:00000000 DSPREALLOC:00000000
+000190 DS_BYTES_ALLOC:00000000 DS_CURR_ALLOC:00000000
+000198 DS_GETMAINS:00000000 DS_FREEMAINS:00000000
+0001A0 DS_FLAGS:00000000
```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 5 of 11)

```

DSA backchain

    DSA: 000253C0

+000000  FLAGS:0000  MEMD:0000  BKC:00022CA0  FWC:000254C0
+00000C  R14:864C1E3A  R15:064C3F88  R0:000082A8
+000018  R1:00025454  R2:00017428  R3:0002547C
+000024  R4:00025470  R5:00022018  R6:00000000
+000030  R7:00025480  R8:00000001  R9:FFFFFFFC
+00003C  R10:063CE334  R11:064C1C38  R12:00016AC0
+000048  LWS:00017630  NAB:00025460  PNAB:00000000
+000064  RENT:00000000  CILC:00000000  MODE:863D04CE
+000078  RMR:00000000

Contents of DSA at location 000253C0:

+00000000 00000000 00022CA0 000254C0 864C1E3A 064C3F88 000082A8 00025454 00017428 | .....f<...<.h..by.....
+00000020 0002547C 00025470 00022018 00000000 00025480 00000001 FFFFFFFC 063CE334 | ..@.....T.
+00000040 064C1C38 00016AC0 00017630 00025460 00000000 00000000 00000000 00000000 | .<.....
+00000060 00000000 00000000 00000000 863D04CE 00000000 00000000 00000000 00000000 | .....f.....
+00000080 000230D8 063CF18F 000230F4 0002310C 00023110 00025484 00025474 00025478 | ...Q..1...4.....d.....

    DSA: 00022CA0
+000000  FLAGS:0000  MEMD:CEE1  BKC:000221F8  FWC:000253C0
+00000C  R14:863CE4C4  R15:063D8838  R0:06319A64
+000018  R1:000230B0  R2:00023308  R3:00000003
+000024  R4:00016058  R5:00000010  R6:000230D8
+000030  R7:00023C9F  R8:063CECCD  R9:063CDCE
+00003C  R10:063CCCF  R11:863CBDD0  R12:00016AC0
+000048  LWS:00017630  NAB:000253C0  PNAB:00000012
+000064  RENT:00022BF8  CILC:00000259  MODE:863CD462
+000078  RMR:06484AE8

Contents of DSA at location 00022CA0:

+00000000 0000CEE1 000221F8 000253C0 863CE4C4 063D8838 06319A64 000230B0 00023308 | .....8....f.UD..h.....
+00000020 00000003 00016058 00000010 000230D8 00023C9F 063CECCD 063CDCE 063CCCF | .....-.....Q.....
+00000040 863CBDD0 00016AC0 00017630 000253C0 00000012 867C5882 867C5896 86448940 | f.....f@.bf@.of.i
+00000060 FFFFFFFF 00022BF8 00000259 863CD462 00017630 00022D80 06484AE8 00000000 | .....8....f.M.....Y.....
+00000080 863CC09A 00000000 00024218 000230B0 000159D0 063186A0 00000000 00013304 | f.....f.....
+000000A0 00023C9F 063CECCD 063CDCE 063CCCF 863CBDD0 00016AC0 06319D54 06319A64 | .....f.....
+000000C0 863CC62C 000221F8 06319A64 00023194 00000002 063186A0 00000000 00000001 | f.F...8.....m.....f.....
+000000E0 063199D0 00023C9F 063CECCD 063CDCE 063CCCF 863CBDD0 00016AC0 06319A64 | ..r.....f.....

To display entire DSA: IP LIST 00022CA0 LEN(X'00002720') ASID(X'001F')

    DSA: 000221F8
+000000  FLAGS:1000  MEMD:0000  BKC:000220E0  FWC:000222D8
+00000C  R14:863026BA  R15:06615024  R0:067C68F8
+000018  R1:00022290  R2:067C6CC0  R3:06301BCA
+000024  R4:068E0414  R5:067C6B38  R6:000222B2
+000030  R7:000222BC  R8:000222C0  R9:80000000
+00003C  R10:8669199A  R11:800082A8  R12:00016AC0
+000048  LWS:00017630  NAB:000222D8  PNAB:065D28D2
+000064  RENT:00015E68  CILC:000159D0  MODE:863022E8
+000078  RMR:00017630

Contents of DSA at location 000221F8:

+00000000 10000000 000220E0 000222D8 863026BA 06615024 067C68F8 00022290 067C6CC0 | .....Qf.../&..@.8....@%.
+00000020 06301BCA 068E0414 067C6B38 000222B2 000222BC 000222C0 80000000 8669199A | .....@,.....f...
+00000040 800082A8 00016AC0 00017630 000222D8 065D28D2 067C68F8 067C68F8 000159D0 | ..by.....Q.)..K.@.8.@.8....
+00000060 00015E50 00015E68 000159D0 863022E8 8649AA90 00016AC0 00017630 000222C8 | .;&...;...f..Yf.....H
+00000080 00000000 00000004 04000000 F97FC14F 00000001 F97FC14F 067C6CC0 067C6C88 | .....9"A]....9"A]..@%..@%h
+000000A0 00000003 067C6BA8 068ECE56 00000000 068DF1EC 068E0414 00000000 00000000 | .....@,y.....1.....
+000000C0 00000000 00000002 068ECE50 00000000 00000000 00000000 067C68F8 00000000 | .....@.8....

:

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 6 of 11)

[11]Condition Management Control Blocks
User Stack Control Blocks

```
STKH: 00022000
+000000 EYE_CATCHER:STKU NEXT:000171F4 PREV:000171F4
+00000C SEGMENT_LEN:00020000
```

Library Stack Control Blocks

```
STKH: 00021000
+000000 EYE_CATCHER:STKL NEXT:00017238 PREV:00017238
+00000C SEGMENT_LEN:00001000
```

```
HCOM: 063186A0
+000000 PICA_AREA:00000000 00000000 EYES:HCOM CAA_PTR1:00016AC0
+000014 CVDTCB:9B FLAG:60F04000 EXIT_STK:068F0028
+000020 RSM_PTR:00000000 HDLL_STK:068ECF08
+000028 SRP_TOKEN:00000000 CSTK:00042028 CIBH:00023818
```

```
CIBH: 00023818
+000000 EYE:CIBH BACK:063199D0 FRWD:00000000
+000010 PTR_CIB:00000000 FLAG1:00 ERROR_LOCATION_FLAGS:00
+000018 HDLQ:00000000 STATE:00000000 PRM_DESC:00000000
+000024 PRM_PREFIX:00000000
+000028 PRM_LIST:00000000 00000000 00000000 00000000
+000038 PARM_DESC:00000000 PARM_PREFIX:00000000
+000040 PARM_LIST:00000000 00000000 00000000 00000000 FUN:00000000
+000054 CIB_SIZ:0000 CIB_VER:0000 FLG_5:00 FLG_6:00
+00005A FLG_7:00 FLG_8:00 FLG_1:00 FLG_2:00 FLG_3:00
+00005F FLG_4:00 ABCD:00000000 ABRC:00000000
+000068 OLD_COND_64:00000000 00000000 OLD_MIB:00000000
+000074 COND_64:00000000 00000000 MIB:00000000 PL:00000000
+000084 SV2:00000000 SV1:00000000 INT:00000000
+000090 MID:00000000 HDL_SF:00000000 HDL_EPT:00000000
+00009C HDL_RST:00000000 RSM_SF:00000000 RSM_POINT:00000000
+0000A8 RSM_MACHINE:00000000 COND_DEFAULT:00000000
+0000B4 Q_DATA_TOKEN:00000000 FDBK:00000000 ABNAME:.....
Machine State
+000348 MCH_EYE:....
+000350 MCH_GPR00:00000000 MCH_GPR01:00000000
+000358 MCH_GPR02:00000000 MCH_GPR03:00000000
+000360 MCH_GPR04:00000000 MCH_GPR05:00000000
+000368 MCH_GPR06:00000000 MCH_GPR07:00000000
+000370 MCH_GPR08:00000000 MCH_GPR09:00000000
+000378 MCH_GPR10:00000000 MCH_GPR11:00000000
+000380 MCH_GPR12:00000000 MCH_GPR13:00000000
+000388 MCH_GPR14:00000000 MCH_GPR15:00000000
+000390 MCH_PSW:00000000 00000000 MCH_ILC:0000 MCH_IC1:00
+000398 MCH_IC2:00 MCH_PFT:00000000 MCH_FLT_0:00000000 00000000
+0003A8 MCH_FLT_2:00000000 00000000 MCH_FLT_4:00000000 00000000
+0003B8 MCH_FLT_6:00000000 00000000 MCH_EXT:00000000
+000418 MCH_FLT_1:00000000 00000000 MCH_FLT_3:00000000 00000000
+000428 MCH_FLT_5:00000000 00000000 MCH_FLT_7:00000000 00000000
+000438 MCH_FLT_8:00000000 00000000 MCH_FLT_9:00000000 00000000
+000448 MCH_FLT_10:00000000 00000000
+000450 MCH_FLT_11:00000000 00000000
+000458 MCH_FLT_12:00000000 00000000
+000460 MCH_FLT_13:00000000 00000000
+000468 MCH_FLT_14:00000000 00000000
+000470 MCH_FLT_15:00000000 00000000 MCH_FPC:00000000
+00047C MCH_APF_FLAGS:00

+0009E0 ABCC:00000000 HRC:00000000 RSM_SF_FMT:00
+0009E9 RSM_PH_CALLEE_FMT:00 SV1_FMT:00 RSM_PH_CALLEE:00000000
+0009F0 INT_FCN_EP:00000000 HDL_SF_FMT:00 HDL_PH_CALLEE_FMT:00
+0009F6 SV2_FMT:00 HDL_PH_CALLEE:00000000
```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 7 of 11)

```

CIBH: 063199D0
+000000 EYE:CIBH BACK:00000000 FRWD:00023818
+000010 PTR_CIB:00023308 FLAG1:C5 ERROR_LOCATION_FLAGS:1F
+000018 HDLQ:00000000 STATE:00000000 PRM_DESC:00000000
+000024 PRM_PREFIX:00000000
+000028 PRM_LIST:00023320 000233E8 000233F4 0631A11C
+000038 PARM_DESC:00000000 PARM_PREFIX:00000000
+000040 PARM_LIST:000233E4 00023308 000233F4 0631A11C FUN:00000067
+000054 CIB_SIZ:010C CIB_VER:0004 FLG_5:48 FLG_6:23
+00005A FLG_7:00 FLG_8:00 FLG_1:00 FLG_2:00 FLG_3:00
+00005F FLG_4:05 ABCD:940C9000 ABRC:00000009
+000068 OLD_COND_64:00030C89 59C3C5C5 OLD_MIB:00000001
+000074 COND_64:00030C89 59C3C5C5 MIB:00000001 PL:06301B38
+000084 SV2:000221F8 SV1:000221F8 INT:063026CE
+000090 MID:00000003 HDL_SF:00017428 HDL_EPT:063A9C58
+00009C HDL_RST:00000000 RSM_SF:000221F8 RSM_POINT:063026D0
+0000A8 RSM_MACHINE:06319F18 COND_DEFAULT:00000003
+0000B4 Q_DATA_TOKEN:06319B08 FDBK:00000000 ABNAME:.....
Machine State
Machine State
+000348 MCH_EYE:ZMCH
+000350 MCH_GPR00:00000000 MCH_GPR01:00022290
+000358 MCH_GPR02:067C6CC0 MCH_GPR03:06301BCA
+000360 MCH_GPR04:068E0414 MCH_GPR05:067C6B38
+000368 MCH_GPR06:00000000 MCH_GPR07:00000001
+000370 MCH_GPR08:000222C0 MCH_GPR09:80000000
+000378 MCH_GPR10:8669199A MCH_GPR11:800082A8
+000380 MCH_GPR12:00016AC0 MCH_GPR13:000221F8
+000388 MCH_GPR14:863026BA MCH_GPR15:00000012
+000390 MCH_PSW:078D2400 863026D0 MCH_ILC:0002 MCH_IC1:00
+00039B MCH_IC2:09 MCH_PFT:00000000 MCH_FLT_0:4DB3EDBF DAC99794
+0003A8 MCH_FLT_2:00000000 00000000 MCH_FLT_4:00000000 00000000
+0003B8 MCH_FLT_6:00000000 00000000 MCH_EXT:00000000
+000418 MCH_FLT_1:00000000 00000000 MCH_FLT_3:00000000 00000000
+000428 MCH_FLT_5:00000000 00000000 MCH_FLT_7:00000000 00000000
+000438 MCH_FLT_8:00000000 00000000 MCH_FLT_9:00000000 00000000
+000448 MCH_FLT_10:00000000 00000000
+000450 MCH_FLT_11:00000000 00000000
+000458 MCH_FLT_12:00000000 00000000
+000460 MCH_FLT_13:00000000 00000000
+000468 MCH_FLT_14:00000000 00000000
+000470 MCH_FLT_15:00000000 00000000 MCH_FPC:00000000
+00047C MCH_APF_FLAGS:00

+0009E0 ABCC:00000000 HRC:00000000 RSM_SF_FMT:00
+0009E9 RSM_PH_CALLEE_FMT:00 SV1_FMT:00 RSM_PH_CALLEE:00000000
+0009F0 INT_FCN_EP:00000000 HDL_SF_FMT:00 HDL_PH_CALLEE_FMT:00
+0009F6 SV2_FMT:00 HDL_PH_CALLEE:00000000

CIB: 00023308
+000000 EYE:CIB BACK:00000000 FRWD:00000000 SIZ:010C
+00000E VER:0004 PLAT_ID:00015A38 COND_64:000300C6 59C3C5C5
+000020 MIB:00000000 MACHINE:00023414
+000028 OLD_COND_64:00030C89 59C3C5C5 OLD_MIB:00000001
+000034 FLG_1:00 FLG_2:00 FLG_3:00 FLG_4:04 HDL_SF:00022018
+00003C HDL_EPT:063A9C58 HDL_RST:00000000 RSM_SF:000221F8
+000048 RSM_POINT:063026D0 RSM_MACHINE:06319F18
+000050 COND_DEFAULT:00000003 PH_CALLEE_SF:FCFDFFFD HDL_SF_FMT:00
+000059 PH_CALLEE_SF_FMT:00 VSR:00000000 00000000 VSTOR:00000000
+00009C VRPSA:00000000 MCB:0631EE80 MRN:0000017A 068ED338
+0000AC MFLAG:00 FLG_5:48 FLG_6:23 FLG_7:00 FLG_8:00
+0000B4 ABCD:940C9000 ABRC:00000009 ABNAME:00000000 00000000
+0000C4 PL:06301B38 SV2:000221F8 SV1:000221F8
+0000D0 INT:063026CE Q_DATA_TOKEN:D3D34040 FDBK:00000000
+0000DC FUN:00000067 TOKE:00022018 MID:00000003
+0000E8 STATE:00000000 RTCC:FFFFFFFF PPAV:00000003
+0000F4 AB_TERM_EXIT:000232B0 00000000 SDWA_PTR:00000000
+000100 SIGNO:00000008 PPSD:0631A130

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 8 of 11)

```

[12]Message Processing Control Blocks
  CMXB: 00014B08
+000000 EYE:CMXB  SIZE:0110  FLAGS:C000  DHEAD1:00094000
+00000C DHEAD2:00014B28

MDST forward chain from CMXBDHEAD(1)

  MDST: 00094000
+000000 EYE:MDST  SIZE:00C8  CTL:40      CEEDUMPLOC:00
+000008 NEXT:00014B28  PREV:00000000  DDNAM:CEEDUMP

  MDST: 00014B28
+000000 EYE:MDST  SIZE:00C8  CTL:40      CEEDUMPLOC:00
+000008 NEXT:00000000  PREV:00094000  DDNAM:SYSOUT

MDST back chain from CMXBDHEAD(2)

  MDST: 00014B28
+000000 EYE:MDST  SIZE:00C8  CTL:40      CEEDUMPLOC:00
+000008 NEXT:00000000  PREV:00094000  DDNAM:SYSOUT

  MDST: 00094000
+000000 EYE:MDST  SIZE:00C8  CTL:40      CEEDUMPLOC:00
+000008 NEXT:00014B28  PREV:00000000  DDNAM:CEEDUMP

  TMXB: 0631A4E8
+000000 EYE:TMXB  MIB_CHAIN_PTR:068F5028

  MGF: 068F5028
+000000 EYE:CMIB  PREV:068F6550  NEXT:0631A520

  MGF: 0631A520
+000000 EYE:CMIB  PREV:068F5028  NEXT:068F6550

  MGF: 068F6550
+000000 EYE:CMIB  PREV:0631A520  NEXT:068F5028

[13]Information for enclave main

[14]Information for thread 0676992000000000

[15]Traceback:
  DSA Addr  Program Unit  PU Addr  PU Offset  Entry      E Addr  E  Offset  Statement  Load Mod  Service  Status
000253C0  CEEHSDMP  063D8838  +000E9600  CEEHSDMP  063D8838  +000E9600  CEEPLPKA  CEEPLPKA  Call
00022CA0  CEEHDSP  063CBCD0  +000027F2  CEEHDSP  063CBCD0  +000027F2  CEEPLPKA  CEEPLPKA  Call
000221F8  06301B90  +00000B3E  main      06301B90  +00000B3E  MYMOD  MYMOD  Exception
000220E0  066919A6  +00242CFA  EDCZMINV  066919A6  +00242CFA  CEEEV003  UQ31021  Call
00022018  CEEBBEXT  000082A8  +068CC3F6  CEEBBEXT  000082A8  +068CC3F6  CEEBINIT  CEEBINIT  Call

[16]Control Blocks Associated with the Thread:
  Thread Synchronization Queue Element (SQEL): 0631A618
+000000 0631A618  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  .....
+000020 0631A638  00016AC0  00000000  00000000  00000000  00000000  00000000  00000000  00000000  .....

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 9 of 11)

```

[17]Enclave Control Blocks:
Mutex and Condition Variable Blocks (MCVB+MHT+CHT): 068D1018
+000000 068D1018 00008F50 068D1044 000003F8 00001FC0 00000000 068D4130 068D1444 000000F8 |...&.....8.....|
+000020 068D1038 000007C0 00000000 068D4148 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 068D1058 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
:

Thread Synchronization Enclave Latch Table (EPALT): 068D1544
+000000 068D1544 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 068D1564 - +00055F 068D1AA3 same as above |.....|
+000560 068D1AA4 00000000 00000000 00000000 00000000 0646A478 00000000 00000000 00000000 |.....u.....|
+000580 068D1AC4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0005A0 068D1AE4 - +00061F 068D1B63 same as above |.....|
+000620 068D1B64 00000000 00000000 00000000 00000000 00000000 0646A478 00000000 00000000 |.....u.....|
+000640 068D1B84 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000660 068D1BA4 - +0009FF 068D1F43 same as above |.....|

HEAPCHK Option Control Block (HCOP): 068D3028
+000000 068D3028 C8C3D6D7 00000024 00000001 00000000 00000000 068F3028 068D304C 00000000 |HCOP.....<....|
+000020 068D3048 00000000 C8C3C6E3 00000200 00000000 00000000 00000000 00000000 00000000 |....HCFT.....|

HEAPCHK Element Table (HCEL) for Heapid 068F020C :
Header: 068F3028
+000000 068F3028 C8C3C5D3 068DD028 00000000 068F020C 000001F4 00000001 00000001 00000000 |HCEL.....4.....|
Address Seg Addr Length Address Seg Addr Length

Table: 068F3048
+000000 068F3048 068F2020 068F2000 000002A8 00000000 00000000 00000000 00000000 00000000 |.....y.....|

HEAPCHK Element Table (HCEL) for Heapid 00000000 :
Header: 068DD028
+000000 068DD028 C8C3C5D3 00000000 068F3028 00000000 000001F4 00000007 00000007 00000000 |HCEL.....4.....|
Address Seg Addr Length Address Seg Addr Length

Table: 068DD048
+000000 068DD048 068D5020 068D5000 00000110 00000000 068D5130 068D5000 00000828 00000000 |..&...&.....&.....|
+000020 068DD068 068D5958 068D5000 00000250 00000000 068D5BA8 068D5000 00000038 00000000 |.....&...&.....$y...|
+000040 068DD088 068D5BE0 068D5000 00000038 00000000 068D5C18 068D5000 00000010 00000000 |..$.&.....*...&.....|
+000060 068DD0A8 068D5C28 068D5000 00000010 00000000 00000000 00000000 00000000 00000000 |..*...&.....|
:

[18]Language Environment Trace Table:

Most recent trace entry is at displacement: 004900
Most recent trace entry is at displacement: 004480

Displacement Trace Entry in Hexadecimal Trace Entry in EBCDIC
-----
+000000 Time 20.55.18.050451 Date 2001.08.21 Thread ID... 0676992000000000
+000010 Member ID.... 03 Flags..... 000000 Entry Type..... 00000001
+000018 94818995 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |main
+000038 60606E4D F0F8F55D 40979989 95A3864D 5D404040 40404040 40404040 40404040 |-->(085) printf()
+000058 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000078 40404040 40404040

+000080 Time 20.55.18.068354 Date 2001.08.21 Thread ID... 0676992000000000
+000090 Member ID.... 03 Flags..... 000000 Entry Type..... 00000002
+000098 4C60604D F0F8F55D 40D9F1F5 7EF0F0F0 F0F0F0F0 C540C5D9 D9D5D67E F0F0F0F0 |<--(085) R15=0000000E ERRNO=0000
+0000B8 F0F0F0F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |0000.....
+0000D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0000F8 00000000 00000000

+000100 Time 20.55.18.068362 Date 2001.08.21 Thread ID... 0676992000000000
+000110 Member ID.... 03 Flags..... 000000 Entry Type..... 00000003
+000118 94818995 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |main
+000138 60606E4D F1F5F55D 4097A388 99858184 6D94A4A3 85A76D89 9589A34D 5D404040 |-->(155) pthread_mutex_init()
+000158 40404040 40404040 40404040 40404040 40404040 40404040 40000000 00000000 |.....
+000178 00000000 00000000

+000180 Time 20.55.18.068388 Date 2001.08.21 Thread ID... 0676992000000000
+000190 Member ID.... 03 Flags..... 000000 Entry Type..... 00000004
+000198 4C60604D F1F5F55D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(155) R15=00000000 ERRNO=0000
+0001B8 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0 F0F0F0F0 00000000 00000000 00000000 |0000 ERRNO2=00000000.....
+0001D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0001F8 00000000 00000000
:

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 10 of 11)

```

+000200 Time 20.55.18.068395 Date 2001.08.21 Thread ID... 0676992000000000
+000210 Member ID.... 03 Flags..... 000000 Entry Type..... 00000001
+000218 94818995 40404040 40404040 40404040 40404040 40404040 40404040 |main
+000238 60606E4D F0F8F55D 40979989 95A3864D 5D404040 40404040 40404040 |-->(085) printf()
+000258 40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000278 40404040 40404040

+004400 Time 20.55.23.736474 Date 2001.08.21 Thread ID... 0676992000000000
+004410 Member ID.... 03 Flags..... 000000 Entry Type..... 00000001
+004418 A3889985 81846D83 93858195 A4974040 40404040 40404040 40404040 |thread_cleanup
+004438 60606E4D F0F8F55D 40979989 95A3864D 5D404040 40404040 40404040 |-->(085) printf()
+004458 40404040 40404040 40404040 40404040 40404040 40404040 40404040
+004478 40404040 40404040

+004480 Time 20.55.23.736488 Date 2001.08.21 Thread ID... 0676992000000000
+004490 Member ID.... 03 Flags..... 000000 Entry Type..... 00000002
+004498 4C60604D F0F8F55D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(085) R15=00000000 ERRNO=0000
+0044B8 F0F0F0F0 00000000 00000000 00000000 00000000 00000000 00000000 |0000.....
+0044D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0044F8 00000000 00000000

[19]Process Control Blocks:
Thread Synchronization Process Latch Table (PPALT): 068D1F44
+000000 068D1F44 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+000020 068D1F64 - +0009FF 068D2943 same as above

Exiting Language Environment Data

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 11 of 11)

Sections of the Language Environment LEDATA Verbexit formatted output

The sections of the output listed here appear independently of the Language Environment-conforming languages used.

[1] - [8] Summary

These sections are included when the SUMMARY parameter is specified on the LEDATA invocation.

[1] Summary Header

The summary header section contains:

- Address of Thread control block (TCB)
- Release number
- Address Space ID (ASID)

[2] Active Members List

This list of active members is extracted from the enclave member list (MEML).

[3] CEECAA

This section formats the contents of the Language Environment common anchor area (CAA). Refer to “The Common Anchor Area” on page 62 for a description of the fields in the CAA.

[4] CEEPCB

This section formats the contents of the Language Environment process control block (PCB), and the process level member list.

[5] CEERCB

This section formats the contents of the Language Environment region control block (RCB).

[6] CEEEDB

This section formats the contents of the Language Environment enclave data block (EDB), and the enclave level member list.

[7] PMCB

This section formats the contents of the Language Environment program management control block (PMCB).

[8] Run-Time Options

This section lists the run-time options in effect at the time of the dump, and indicates where they were set.

[9] Heap Storage Control Blocks

This section is included when the HEAP or SM parameter is specified on the LEDATA invocation.

This section formats the Enclave-level storage management control block (ENSM) and for each different type of heap storage:

- Heap control block (HPCB)
- Chain of heap anchor blocks (HANC). A HANC immediately precedes each segment of heap storage.

This section includes a detailed heap segment report for each segment in the dump. For more information about the detailed heap segment report, see “Understanding the HEAP LEDATA output” on page 95.

[10] Stack Storage Control Blocks

This section is included when the STACK or SM parameter is specified on the LEDATA invocation.

This section formats:

- Storage management control block (SMCB)
- Chain of dynamic save areas (DSA)
Refer to “The upward-growing (non-XPLINK) stack frame section” on page 60 or “The downward-growing (XPLINK) stack frame section” on page 61 for a description of the fields in the DSA.
- Chain of stack segment headers (STKH)
An STKH immediately precedes each segment of stack storage.

[11] Condition Management Control Blocks

This section is included when the CM parameter is specified on the LEDATA invocation.

This section formats the chain of Condition Information Block Headers (CIBH) and Condition Information Blocks. The Machine State Information Block is contained with the CIBH starting with the field labeled MCH_EYE. Refer to “The condition information block” on page 69 for a description of fields in these control blocks.

[12] Message Processing Control Blocks

This section is included when the MH parameter is specified on the LEDATA invocation.

[13] - [19] CEEDUMP Formatted Control Blocks

These sections are included when the CEEDUMP parameter is specified on the LEDATA invocation.

[13] Enclave Identifier

This statement names the enclave for which information is provided.

[14] Information for thread

This section shows the system identifier for the thread. Each thread has a unique identifier.

[15] Traceback

There will be one or more Traceback sections, depending on the setting of the NTHREADS parameter in the VERBEXIT LEDATA invocation. For a description of NTHREADS, see “Report type parameters” on page 79.

For all active routines in a particular thread, the traceback section shows:

- Stack frame (DSA) address
- Program unit

The primary entry point of the external procedure. For COBOL programs, this is the PROGRAM-ID name. For C, Fortran, and PL/I routines, this is the compile unit name. For Language Environment-conforming assemblers, this is the EPNAME = value on the CEEPPA macro.

- Program unit address
- Program unit offset

The offset of the last instruction to run in the routine. If the offset is a negative number, zero, or a very large positive number, the routine associated with the offset probably did not allocate a save area, or the routine could have been called using SVC-assisted linkage. Adding the program unit address to the offset gives you the location of the current instruction in the routine. This offset is from the starting address of the routine.

- Entry

For COBOL, Fortran, and PL/I routines, this is the entry point name. For C/C++ routines, this is the function name. If a function name or entry point was not specified for a particular routine, then the string ‘** NoName **’ will appear.

- Entry point address
- Entry point offset
- Statement number

This field contains no Language Environment data.

- Load module
This field contains no Language Environment data.
- Service level
This field contains no Language Environment data.
- Status
Routine status can be call, exception, or running.

[16] Control Blocks Associated with the Thread

This section lists the contents of the thread synchronization queue element (SQEL).

[17] Enclave Control Blocks

If the POSIX run-time option was set to ON, this section lists the contents of the mutex and condition variable control blocks, the enclave level latch table, and the thread synchronization trace block and trace table. If the HEAPCHK run-time option is set to ON, this section lists the contents of the HEAPCHK options control block (HCOP) and the HEAPCHK element tables (HCEL). A HEAPCHK element table contains the location and length of all allocated storage elements for a heap in the order that they were allocated.

[18] Language Environment Trace Table

If the TRACE run-time option was set to ON, this section shows the contents of the Language Environment trace table.

[19] Process Control Blocks

If the POSIX run-time option was set to ON, this section lists the contents of the process level latch table.

Understanding the HEAP LEDATA output

The Language Environment IPCS Verbexit LEDATA generates a detailed heap segment report when the HEAP option is used with the DETAIL option, or when the SM,DETAIL option is specified. The detailed heap segment report is useful when trying to pinpoint damage because it provides very specific information. The report describes the nature of the damage, and specifies where the actual damage occurred. The report can also be used to diagnose storage leaks, and to identify heap fragmentation. Figure 15 on page 96 illustrates the output produced by specifying the HEAP option. “Heap report sections of the LEDATA output” on page 99 describes the information contained in the formatted output.

For easy reference, the sections of the dump are numbered to correspond with the description of each section that follows. Ellipses are used to summarize some sections of the dump.

Note: Language Environment does not provide support for alternative Vendor Heap Manager (VHM) data. LEDATA verb exit will state that an alternative VHM is in use.

```

IP VERBEXIT LEDATA 'HEAP'
*****
                        LANGUAGE ENVIRONMENT DATA
*****
Language Environment Productt 04 V01 R02.00

Heap Storage Control Blocks

    ENSM: 00014D30
+0000A8  ENSM_ADDL_HEAP:259B1120

User Heap Control Blocks

    HPCB: 00014D48
+000000  EYE_CATCHER:HPCB  FIRST:25995000  LAST:25995000

    HANC: 25995000
+000000  EYE_CATCHER:HANC  NEXT:00014D48  PREV:00014D48
+00000C  HEAPID:00000000  SEG_ADDR:25995000  ROOT_ADDR:259950B0
+000018  SEG_LEN:00008000  ROOT_LEN:00007F50

This is the last heap segment in the current heap.

[1]Free Storage Tree for Heap Segment 25995000

    Node      Node      Parent   Left      Right      Left      Right
Depth Address  Length  Node      Node      Node      Node
  0 259950B0  00007F50  00000000  00000000  00000000  00000000  00000000

[2]Map of Heap Segment 25995000

To display entire segment: IP LIST 25995000 LEN(X'00008000') ASID(X'0021')

25995020: Allocated storage element, length=00000038. To display: IP LIST 25995020 LEN(X'00000038') ASID(X'0021')
25995028: C3C4D3D3 00000000 40000000 00000000 24700F98 25993870 00000490 |CDLL....q.....r.....|

25995058: Allocated storage element, length=00000038. To display: IP LIST 25995058 LEN(X'00000038') ASID(X'0021')
25995060: C3C4D3D3 25995028 80000000 00000000 24700F0 24700770 2471CEB0 00000150 |CDLL.r&.....0.....&|

25995090: Allocated storage element, length=00000010. To display: IP LIST 25995090 LEN(X'00000010') ASID(X'0021')
25995098: 259ADBB8 00000000 |.....|

259950A0: Allocated storage element, length=00000010. To display: IP LIST 259950A0 LEN(X'00000010') ASID(X'0021')
259950A8: 259ADBE0 00000000 |.....|

259950B0: Free storage element, length=00007F50. To display: IP LIST 259950B0 LEN(X'00007F50') ASID(X'0021')

Summary of analysis for Heap Segment 25995000:

    Amounts of identified storage: Free:00007F50  Allocated:00000090  Total:00007FE0
    Number of identified areas   : Free:         1  Allocated:         4  Total:         5
    00000000 bytes of storage were not accounted for.
    No errors were found while processing this heap segment.
    This is the last heap segment in the current heap.

Anywhere Heap Control Blocks

    HPCB: 00014D78
+000000  EYE_CATCHER:HPCB  FIRST:24A91000  LAST:259C2000

    HANC: 24A91000
+000000  EYE_CATCHER:HANC  NEXT:25993000  PREV:00014D78
+00000C  HEAPID:00014D78  SEG_ADDR:24A91000  ROOT_ADDR:00000000
+000018  SEG_LEN:00F00028  ROOT_LEN:00000000

Free Storage Tree for Heap Segment 24A91000

    The free storage tree is empty.

```

Figure 15. Example formatted detailed heap segment report from LEDATA Verbexit (Part 1 of 4)

```

Map of Heap Segment 24A91000

To display entire segment: IP LIST 24A91000 LEN(X'00F00028') ASID(X'0021')

24A91020: Allocated storage element, length=00F00008. To display: IP LIST 24A91020 LEN(X'00F00008') ASID(X'0021')
24A91028: B035F6D8 B2C00081 24ABED80 00000000 03000000 00000001 94818995 40404040 |..6Q...a.....main |

Summary of analysis for Heap Segment 24A91000:
Amounts of identified storage: Free:00000000 Allocated:00F00008 Total:00F00008
Number of identified areas : Free: 0 Allocated: 1 Total: 1
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
:

HANC: 259AC000
+000000 EYE_CATCHER:HANC NEXT:259AF000 PREV:2599D000
+00000C HEAPID:00014D78 SEG_ADDR:259AC000 ROOT_ADDR:259AC020
+000018 SEG_LEN:00002000 ROOT_LEN:00000C30

Free Storage Tree for Heap Segment 259AC000

Depth Node Node Parent Left Right Left Right
Address Length Node Node Node Length Length
0 259AC020 00000C30 00000000 00000000 259ADC48 00000000 000003B8
1 259ADC48 000003B8 259AC020 00000000 00000000 00000000 00000000

Map of Heap Segment 259AC000

To display entire segment: IP LIST 259AC000 LEN(X'00002000') ASID(X'0021')

259AC020: Free storage element, length=00000C30. To display: IP LIST 259AC020 LEN(X'00000C30') ASID(X'0021')

259ACC50: Allocated storage element, length=00000728. To display: IP LIST 259ACC50 LEN(X'00000728') ASID(X'0021')
259ACC58: D3D3E340 071C0001 00000000 00000000 00000000 00000003 00000040 20010003 |LLT .....|

259AD378: Allocated storage element, length=00000080. To display: IP LIST 259AD378 LEN(X'00000080') ASID(X'0021')
259AD380: 00000000 00000000 247006F0 247006F0 000008A8 2471CEB0 00000000 00000001 |.....0...0...y.....|

259AD3F8: Allocated storage element, length=00000068. To display: IP LIST 259AD3F8 LEN(X'00000068') ASID(X'0021')
259AD400: C5E3C3E2 00000007 00000000 25993870 A4797478 247971E0 25993870 A4797478 |ETCS.....r..u.....r..u...|

259AD460: Allocated storage element, length=00000728. To display: IP LIST 259AD460 LEN(X'00000728') ASID(X'0021')
259AD468: C3D3D3E3 071C0001 00000000 00000000 00000000 00000001 00000040 60010005 |CLLT..... -...|

259ADB88: Allocated storage element, length=00000028. To display: IP LIST 259ADB88 LEN(X'00000028') ASID(X'0021')
259ADB90: 180F58FF 001007FF 24700AB8 2471CEB0 2479A6E8 FFFFFFFF 247006F0 259AD380 |.....wY.....0..L..|

259ADBB0: Allocated storage element, length=00000028. To display: IP LIST 259ADBB0 LEN(X'00000028') ASID(X'0021')
259ADBB8: 00000000 25995098 70004000 00000000 00000000 00000000 00000000 00000000 |.....r.&q.....|

259ADBDB8: Allocated storage element, length=00000028. To display: IP LIST 259ADBDB8 LEN(X'00000028') ASID(X'0021')
259ADBDE0: 00000000 259950A8 70004000 00000000 00000000 00000000 00000000 00000000 |.....r.&y.....|

259ADC00: Allocated storage element, length=00000048. To display: IP LIST 259ADC00 LEN(X'00000048') ASID(X'0021')
259ADC08: C1C4C8D7 F0F00000 259ADC14 C8D7C3C2 259AE000 259AE000 00001000 00001000 |ADHP00.....HPCB.....|

259ADC48: Free storage element, length=000003B8. To display: IP LIST 259ADC48 LEN(X'000003B8') ASID(X'0021')

Summary of analysis for Heap Segment 259AC000:
Amounts of identified storage: Free:00000FE8 Allocated:00000FF8 Total:00001FE0
Number of identified areas : Free: 2 Allocated: 8 Total: 10
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
:

```

Figure 15. Example formatted detailed heap segment report from LEDATA Verbexit (Part 2 of 4)

```

Below Heap Control Blocks

  HPCB: 00014DA8
+000000 EYE_CATCHER:HPCB FIRST:00044000 LAST:00044000

  HANC: 00044000
+000000 EYE_CATCHER:HANC NEXT:00014DA8 PREV:00014DA8
+00000C HEAPID:00014DA8 SEG_ADDR:80044000 ROOT_ADDR:00044388
+000018 SEG_LEN:00002000 ROOT_LEN:00001C78

This is the last heap segment in the current heap.

Free Storage Tree for Heap Segment 00044000

  Depth      Node      Node      Parent      Left      Right      Left      Right
            Address    Length    Node        Node        Node        Length    Length

    0 00044388 00001C78 00000000 00000000 00000000 00000000 00000000

Map of Heap Segment 00044000

To display entire segment: IP LIST 00044000 LEN(X'00002000') ASID(X'0021')

00044020: Allocated storage element, length=00000048. To display: IP LIST 00044020 LEN(X'00000048') ASID(X'0021')
00044028: C8C4D3E2 00000000 00044220 00000040 00010000 00000001 000241E0 24701038 |HDLS.....|

00044068: Allocated storage element, length=00000128. To display: IP LIST 00044068 LEN(X'00000128') ASID(X'0021')
00044070: 07000700 05E0900F E0A641DE 002258C0 E11258F0 E1160B0F E2C6E7D4 01200001 |.....w.....0....SFXM....|

00044190: Allocated storage element, length=00000088. To display: IP LIST 00044190 LEN(X'00000088') ASID(X'0021')
00044198: C3E2E3D2 00000000 00000000 00800001 00000001 00000068 04000000 00000000 |CSTK.....|

00044218: Allocated storage element, length=00000048. To display: IP LIST 00044218 LEN(X'00000048') ASID(X'0021')
00044220: C8C4D3E2 00044028 00000000 00000040 00010000 00000002 000241E0 259ADB90 |HDLS.. ..|

00044260: Allocated storage element, length=00000128. To display: IP LIST 00044260 LEN(X'00000128') ASID(X'0021')
00044268: 07000700 05E0900F E0A641DE 002258C0 E11258F0 E1160B0F E2C6E7D4 01200001 |.....w.....0....SFXM....|

00044388: Free storage element, length=00001C78. To display: IP LIST 00044388 LEN(X'00001C78') ASID(X'0021')

Summary of analysis for Heap Segment 00044000:
Amounts of identified storage: Free:00001C78 Allocated:00000368 Total:00001FE0
Number of identified areas : Free: 1 Allocated: 5 Total: 6
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
This is the last heap segment in the current heap.

Additional Heap Control Blocks

  ADHP: 259B1120
+000000 EYE_CATCHER:ADHP NEXT:259B24A8 HEAPID:259B112C

  HPCB: 259B112C
+000000 EYE_CATCHER:hpcb FIRST:259B112C LAST:259B112C

  ADHP: 259B24A8
+000000 EYE_CATCHER:ADHP NEXT:259ADC08 HEAPID:259B24B4

  HPCB: 259B24B4
+000000 EYE_CATCHER:hpcb FIRST:259B24B4 LAST:259B24B4

  ADHP: 259ADC08
+000000 EYE_CATCHER:ADHP NEXT:F0F00000 HEAPID:259ADC14

  HPCB: 259ADC14
+000000 EYE_CATCHER:HPCB FIRST:259AE000 LAST:259AE000

  HANC: 259AE000
+000000 EYE_CATCHER:HANC NEXT:259ADC14 PREV:259ADC14
+00000C HEAPID:259ADC14 SEG_ADDR:259AE000 ROOT_ADDR:259AE1E8
+000018 SEG_LEN:00001000 ROOT_LEN:00000E18

```

Figure 15. Example formatted detailed heap segment report from LEDATA Verbexit (Part 3 of 4)

```

This is the last heap segment in the current heap.

Free Storage Tree for Heap Segment 259AE000

Depth  Node      Node      Parent   Left      Right     Left      Right
      Address  Length   Node      Node      Node      Length    Length
0  259AE1E8  00000E18  00000000  00000000  00000000  00000000  00000000

Map of Heap Segment 259AE000

To display entire segment: IP LIST 259AE000 LEN(X'00001000') ASID(X'0021')

259AE020: Allocated storage element, length=000001C8. To display: IP LIST 259AE020 LEN(X'000001C8') ASID(X'0021')
259AE028: D7C3C9C2 00000000 00000000 000101BC 00000000 00000000 00000000 00000000 |PCIB.....|

259AE1E8: Free storage element, length=00000E18. To display: IP LIST 259AE1E8 LEN(X'00000E18') ASID(X'0021')

Summary of analysis for Heap Segment 259AE000:
Amounts of identified storage: Free:00000E18 Allocated:000001C8 Total:00000FE0
Number of identified areas   : Free:      1 Allocated:      1 Total:      2
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
This is the last heap segment in the current heap.

Exiting Language Environment Data

```

Figure 15. Example formatted detailed heap segment report from LEDATA Verbexit (Part 4 of 4)

Heap report sections of the LEDATA output

The Heap Report sections of the LEDATA output provide information for each heap segment in the dump. The detailed heap segment reports include information on the free storage tree in the heap segments, the allocated storage elements, and the cause of heap management data structure problems.

[1]Free Storage Tree Report

Within each heap segment, Language Environment keeps track of unallocated storage areas by chaining them together into a tree. Each free area represents a node in the tree. Each node contains a header, which points to its left and right child nodes. The header also contains the length of each child.

The LEDATA HEAP option formats the free storage tree within each heap, and validates all node addresses and lengths within each node. Each node address is validated to ensure that it:

- Falls on a doubleword boundary
- Falls within the current heap segment
- Does not point to itself
- Does not point to a node that was previously traversed

Each node length is validated to ensure that it:

- Is a multiple of 8
- Is not larger than the heap segment length
- Does not cause the end of the node to fall outside of the current heap segment
- Does not cause the node to overlap another node

If the formatter finds a problem, then it will place an error message describing the problem directly after the formatted line of the node that failed validation

[2]Heap Segment Map Report

The LEDATA HEAP option produces a report that lists all of the storage areas within each heap segment, and identifies the area as either allocated or freed. For each

allocated area the contents of the first X'20' bytes of the area are displayed in order to help identify the reason for the storage allocation.

Each allocated storage element has an 8 byte prefix used by Language Environment to manage the area. The first fullword contains a pointer to the start of the heap segment. The second fullword contains the length of the allocated storage element. The formatter validates this header to ensure that its heap segment pointer is valid. The length is also validated to ensure that it:

- Is a multiple of 8
- Is not zero
- Is not larger than the heap segment length
- Does not cause the end of the element to fall outside of the current heap segment
- Does not cause the element to overlap a free storage node

If the `heap_free_value` of the `STORAGE` run-time option was specified, then the formatter also checks that the free storage within each free storage element is set to the requested `heap_free_value`. If a problem is found, then an error message describing the problem is placed after the formatted line of the storage element that failed validation.

Diagnosing heap damage problems

Heap storage errors can occur when an application allocates a heap storage element that is too small for it to use, and therefore, accidentally overlays heap storage. If this situation occurs then some of the typical error messages generated are:

- The node address does not represent a valid node within the heap segment
- The length of the segment is not valid, or
- The heap segment pointer is not valid.

If one of the above error messages is generated by one of the reports, then examine the storage element that immediately precedes the damaged node to determine if this storage element is owned by the application program. Check the size of the storage element and ensure that it is sufficient for the program's use. If the size of the storage element is not sufficient then adjust the allocation size.

If an error occurs indicating that the node's pointers form a circular loop within the free storage tree, then check the Free Storage Tree Report to see if such a loop exists. If a loop exists, then contact the IBM support center for assistance because this may be a problem in the Language Environment heap management routines.

Additional diagnostic information regarding heap damage can be obtained by using the `HEAPCHK` run-time option. This option provides a more accurate time perspective on when the heap damage actually occurred, which could help to determine the program that caused the damage. For more information on `HEAPCHK`, see *z/OS Language Environment Programming Reference*.

Diagnosing storage leak problems

A storage leak occurs when a program does not return storage back to the heap after it has finished using it. To determine if this problem exists, do one of the following:

- The *call-level* suboption of the `HEAPCHK` run-time option causes a report to be produced in the `CEEDUMP`. Any still-allocated (that is, not freed) storage identified by `HEAPCHK` is listed in the report, along with the corresponding traceback. This shows any storage that wasn't freed, as well as all the calls that

were involved in allocating the storage. For more information about the HEAPCHK run-time option, see *z/OS Language Environment Programming Reference* .

- Examine the Heap Segment Map report to see if any data areas, within the allocated storage elements, appear more frequently than expected. If they do, then check to see if these data areas are still being used by the application program. If the data areas are not being used, then change the program to free the storage element after it is done with it.

Diagnosing heap fragmentation problems

Heap fragmentation occurs when allocated storage is interlaced with many free storage areas that are too small for the application to use. Heap fragmentation could indicate that the application is not making efficient use of its heap storage. Check the Heap Segment Map report for frequent free storage elements that are interspersed with the allocated storage elements.

Understanding the HEAPPOOLS trace LEDATA output

The Language Environment IPCS Verbexit LEDATA generates a detailed HEAPPOOLS trace report when the HPT option is used. The argument *value* is the id of the pool to be formatted in the report.

IP VERBEXIT LEDATA 'HPT(5)'

 LANGUAGE ENVIRONMENT DATA

Language Environment Product 04 V01 R05.00

[1] Heap Pool Trace Table

[2] POOLID: 00000005 ASID: 010B AVAILABLE ENTRIES: 6 OF 6

[3] Timestamp: 2003/09/03 21:01:32.828075

Type: FREE Cell Address: 20453850 CpuId: 01 Tcb: 008D6A68

[4] CALL NAME	CALL ADDRESS	CALL OFFSET
CEEVFQT	1FF144A0	00000000
foo8	1FF010B8	0000009E
foo7	1FF011D8	00000080
foo6	1FF012F8	00000080
foo5	1FF01418	00000080
foo4	1FF01538	00000080
foo3	1FF01658	00000080
foo2	1FF01778	00000080
foo1	1FF01898	00000080
main	1FF00AE8	00000000

Figure 16. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 1 of 2)

```

Timestamp: 2003/09/03 21:01:32.828074
Type: FREE Cell Address: 20453C58 Cpuid: 01 Tcb: 008D6A68
CALL NAME      CALL ADDRESS    CALL OFFSET
CEEVFQT        1FF14AA0      00000000
foo9           1FF00F98      0000009E
foo8           1FF010B8      00000080
foo7           1FF011D8      00000080
foo6           1FF012F8      00000080
foo5           1FF01418      00000080
foo4           1FF01538      00000080
foo3           1FF01658      00000080
foo2           1FF01778      00000080
foo1           1FF01898      00000000

Timestamp: 2003/09/03 21:01:32.819909
Type: GET Cell Address: 20453C58 Cpuid: 08 Tcb: 008D6A68
CALL NAME      CALL ADDRESS    CALL OFFSET
CEEVGQT        1FF14CF0      00000000
foo9           1FF00F98      00000068
foo8           1FF010B8      00000080
foo7           1FF011D8      00000080
foo6           1FF012F8      00000080
foo5           1FF01418      00000080
foo4           1FF01538      00000080
foo3           1FF01658      00000080
foo2           1FF01778      00000080
foo1           1FF01898      00000000

Timestamp: 2003/09/03 21:01:32.819907
Type: GET Cell Address: 20453850 Cpuid: 08 Tcb: 008D6A68
CALL NAME      CALL ADDRESS    CALL OFFSET
CEEVGQT        1FF14CF0      00000000
foo8           1FF010B8      00000068
foo7           1FF011D8      00000080
foo6           1FF012F8      00000080
foo5           1FF01418      00000080
foo4           1FF01538      00000080
foo3           1FF01658      00000080
foo2           1FF01778      00000080
foo1           1FF01898      00000080
main           1FF00AE8      00000000

Timestamp: 2003/09/03 21:01:32.819818
Type: GET Cell Address: 20453448 Cpuid: 08 Tcb: 008D6A68
CALL NAME      CALL ADDRESS    CALL OFFSET
CEEVGQT        1FF14CF0      00000000
setlocale      203AEB10      0000018C
tzset          20300CE8      0000059C
_cinit         201EF588      00002EDC
CEEZINV        200E27D0      00000000

Timestamp: 2003/09/03 21:01:32.819805
Type: GET Cell Address: 20453040 Cpuid: 08 Tcb: 008D6A68
CALL NAME      CALL ADDRESS    CALL OFFSET
CEEVGQT        1FF14CF0      00000000
setlocale      203AEB10      000000FC
tzset          20300CE8      0000059C
_cinit         201EF588      00002EDC
CEEZINV        200E27D0      00000000
Exiting Language Environment Data

```

Figure 16. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 2 of 2)

[1] Trace Header

HEAPPOOLS trace header information.

[2] Pool Information

Information includes the number of the pool (POOLID) which is currently being formatted, the ASID, and the number of entries formatted and the total number of entries taken.

Note: The trace wraps for each poolid after 1024 enties have been taken.

[3] Timestamp

The time this trace entry was taken.

Note: The trace entries are formatted in reverse order (most recent trace entry first).

[4] Trace Table Entry contents

The individual trace entry:

- The TYPE - GET or FREE.
- The Cell within the pool being acted upon.
- The CPU and TCB which requested or freed the cell.
- A traceback at the time of the request. The number of entries in this traceback is limited by the HEAPCHK run-time option.

Understanding the C/C++-specific LEDATA output

The Language Environment IPCS Verbexit LEDATA generates formatted output of C/C++-specific control blocks from a system dump when the ALL parameter is specified and C/C++ is active in the dump. Figure 17 on page 104 illustrates the C/C++-specific output produced. The system dump being formatted was obtained by specifying the TERMTHDACT(UADUMP) run-time option when running the program CELSAMP Figure 5 on page 44. “C/C++-specific sections of the LEDATA output” on page 108 describes the information contained in the formatted output. Ellipses are used to summarize some sections of the dump.

For easy reference, the sections of the dump are numbered to correspond with the description of each section that follows.

```

*****
***** CRTL ENVIRONMENT DATA *****
*****
[1]CGEN: 00015920
+00007C OS_SPCTYPE:00000000 CGENE:2471AD74 CRENT:25993870
+0001F8 CFLTINIT:4E000000 00000000 CPRMS:000149D0 TRACE:000000FF
+000208 CTHD:24719964 CURR_FECB:2471ABD4 CEDCXV:A489EB04
+000214 CGEN_CPCB:24719004 CGEN_CEDB:2471A5A4 CFLG3:00
+000220 CIO:247191AC FDSETFD:00000000 FCB_MUTEXOK:0000
+00022C T_C16:00000000 T_C17:00000000 CEDCOV:2489A69C
+000238 CTOFSV:00000000 TRTSPACE:24719D74

[2]CGENE: 2471AD74
+000000 CGENEYE:.... CGENESIZE:00000000 CGENEPTR:00000000
+0000D0 CERRNO:00000000 TEMPLONG:00000000 AMRC:00000000
+000104 STDINFILE:00000000 STDOUTFILE:00000000
+00010C STDERRFILE:00000000 CTYPE:00000000 LC_CTYPE:00010001
+000124 LC_CHARMAP:00000001
+000500 MIN_FLT:00F200F3 00F400F5 00F600F7 00F800F9
+000510 MAX_FLT:00F300AD 00E000E8 00E9001F 25993860
+000520 FLT_EPS:00000000 DBL_EPS:00000000 00000000
+000530 LDBL_EPS:00000000 00000000 C7C5D5C5 000006E0
+000544 IMSPCBLIST:000163BC ADDRBL:24719C7C
+0006D4 ABND_CODE:00000000 RSN_CODE:00000000

[3]CEDB: 2471A5A4
+000000 EYE:CEDB SIZE:000004D0 PTR:2471A5A4 CLLST:24704B40
+000010 CEELANG:0003 CASWITCH:0000 CLWA:2471B2DC
+000018 CALTLWA:2471B62C CCADDR:24702178 CFLGS:00000000
+000028 CANCHOR:00000000 RPLLEN:00000000 ACBLEN:00000000
+000034 LC:2471AA7C VALID_HIGH:2483D6E0 LOW:2483BD3C
+000040 HEAD_FECB:00000000 ATEXTIT_COUNT:00000000
+000048 _EMPTY_COUNT:00000000 MAINPRMS:25993D08
+000050 STDINFILE:2471A3A8 STDOUTFILE:24719FB8
+000058 STDERRFILE:2471A1B0 CTYPE:2484029A TZDFLT:00004650
+000064 CINFO:2471AB8C CMS_WRITE_DISK:4040 _DISK_SET:00000000
+000070 MIN_FLT:00100000 00000000 00000000 00000000
+000080 MAX_FLT:7FFFFFFF FFFFFFFF 71FFFFFF FFFFFFFF
+000090 FLT_EPS:3C100000 DBL_EPS:34100000 00000000
+0000A0 LDBL_EPS:26100000 00000000 I8000000 00000000 FLAGS1:02000000
+0000B4 MTF_MAINTASK_BLK:00000000 EMSG_SETTING:00 DEPTH:00000000
+0000C0 SCREEN_WIDTH:00000000 USERID:IBMUSER.
+0000CC HEAP24_ANCHOR:00000000 TCIC:00000000 TKCLI:00000000
+0000D8 ATEXTIT_FUNCS01:00000000 00000000 00000000 00000000
+0000EC ATEXTIT_FUNCS02:00000000 00000000 00000000 00000000

:

```

Figure 17. Example formatted C/C++ output from LEDATA Verbexit (Part 1 of 5)

```

+000330 ATEXT_FUNCS31:00000000 00000000 00000000 00000000 00000000
+000344 ATEXT_FUNCS32:00000000 00000000 00000000 00000000 00000000
+000358 HEAD_FOREIGN_FECB:00000000 SNAP_DUMP_COUNT:00000000
+000360 ENVIRON:00000000 GETENV_BUF:00000000
+000368 _BUF_LEN:00000000 INSPECT_GLOBALS:00000000
+000374 _JMP_BUFF:00025C44 _BACK_END:00000000 _FLAGS:00000000
+000380 _TAB:00000000 INTOFFLIST:00000000 CGEN_CRENT:25993870
+00038C _CPRMS:000149D0 _CEDCXV:A489EB04 _CEDCOV:2489A69C
+000398 _EPCBLIST:00000000 CAA_ADDR:00015920
+0003A4 USERIDLENGTH:00000007 MAXUNGETCOUNT:0004
+0003C4 IOGET_ANY:2493BFB0 _BELOW:2493B6D0 IOFREE_ANY:2493C470
+0003D0 _BELOW:2493BC10 MTFMAINTASKBLK:00000000
+0003E0 SIGTABLE:2471ADB4 INIT_STDIN:2471A3A8
+0003E8 _STDOUT:24719FB8 _STDERR:2471A1B0 TABNUM:00000008
+0003F8 FLAGS2:00000000 OPENMVS_FLAGS:00 MRPSTDR:2482D7F8
+000408 MWPSTDR:2482DA00 MRPSTDC:2482C928
+000410 MWPSTDC:2482CB30 OWRP1:24898BA4 OWRP3:2489EB04
+00041C STATIC_EDCOV:00000000 GETENV_BUF2:00000000
+000424 _BUF2_LEN:00000000 DLCB_MUTEX:25993DA8 _CONDV:25993DAC
+000430 EDCOV:2498B480 LCX:2471ACB4 _MUTEX_ATTR:25993D48
+000444 STOR_INIT:00003000 _INCR:00002000 DEMANGLE:00000000
+000454 TEMPR15:00000000 TERMINATE:00000000
+00045C CXX_INV:00000000 D4_JOIN_MUTEX_ATTR:25993D98
+000468 _MUTEX:25993D9C _CONDV_ATTR:25993DA0 _CONDV:25993DA4
+000474 DLLANCHOR:00000000 DLLLAST:00000000 MEM24P:000163C0
+000480 RTLMutex_ARRAYPTR:25993D4C MSGCATLIST:00000000
+000488 SRCHP:00000000 ETOAP:00000000 ATOEP:00000000
+000494 NDMGMTMP:00000000 POPENP:00000000 RND48P:00000000
+0004A0 BRK_HEAPID:00000000 _START:00000000 _CURRENT:00000000
+0004AC _END:00000000 RESTARTTABLE:2497BE48 SYSLOGP:00000000
+0004BC LOGIN_NAME:..... PREV_UMASK_VAL:00000000

```

```

[4]CTHD: 24719964
+000000 CTHDEYE:CTHD SIZE:00000310 CTHDPTR:24719964
+00000C STORPTR:00000000 TOKPTR:24837440
+000014 ASCTIME_RESULT:.....
+00002E SNAP_DUMP_FLAG:00 GMTIME_BKDN:24719D4C
+000034 TIMECALLED:00000000 DATECALLED:00000000
+00003C DTCALLED:00000000 LOC_CALLED:00000000
+000044 DOFMT0_DISCARDS:00000000 CERRNO:00000000 AMRC:24719854
+000050 AMRC2:2471993C GDATE:00000000 OPTARGV:00000000
+00005C OPTERRV:00000001 OPTINDV:00000001
+000064 OPTOPTV:00000000 OPTSIND:00000000 DLGHTV:00000000
+000070 TZONEV:00000000 GTDTERRV:00000000 OPTARGP:259938A8
+00007C OPTERRP:259938A4 OPTINDP:259938A0
+000084 OPTOPTP:2599389C DLGHTP:25993890 TZONEP:25993894
+000090 GTDTERRP:259938B0 RNDSTGP:00000000
+000098 LOCNAME:00000000 ENCRYPTP:00000000 CRYPTP:00000000
+0000A4 RND48P:00000000 L64AP:00000000 WCSTOKP:00000000
+0000B0 CUSERP:00000000 GPASSP:00000000 UTMXP:00000000
+0000BC NDMGMTMP:00000000 RECOMP:00000000 STACKPTR:00000000
+0000C8 STACKSIZE:00000000 STACKFLAGS:00 000000
+0000D0 MCVTP:00000000 H_ERRNO:00000000 SD:FFFFFFFF
+0000DC HOSTENT_DATA_P:00000000 HOSTENT_P:00000000
+0000E4 NETENT_DATA_P:00000000 NETENT_P:00000000
+0000EC PROTOENT_DATA_P:00000000 PROTOENT_P:00000000
+0000F4 SERVENT_DATA_P:00000000 SERVENT_P:00000000
+0000FC NTOA_BUF:..... _LOC1V:00000000

```

Figure 17. Example formatted C/C++ output from LEDATA Verbexit (Part 2 of 5)

```

+000118 HERRNOP:259938AC      _LOC1P:2599388C      REXECP:00000000
+000124 CXEXCEPTION:00000000 TEMPDCBE:24719644
+00012C T_ERRNOV:00000000 T_ERRNOP:25993870
+000148 THD_STORAGE:00000000 CONTEXT_LINK:00000000 FLAGS1:00000000
+000154 LABEL_VAR:24719E74 ABND_CODE:00000000
+00015C RSN_CODE:00000000 STRFTIME_ERADTCALLED:00000000
+000164 STRFTIME_ERADATECALLED:00000000
+000168 STRFTIME_ERATIMECALLED:00000000
+00016C STRFTIME_ERAYEARCALLED:00000000 MBRLN_STATE:0000
+000172 MBRTOWC_STATE:0000 WCRTOMB_STATE:0000
+000176 MBSRTOWCS_STATE:0000 WCSRTOMBS_STATE:0000 MBLEN_STATE:0000
+00017C MBTOWC_STATE:0000 CURR_HEAP_ID:00000000
+000184 CURR_CAA:00000000 CURR_MOD_HANDLE:00000000
+00018C CURR_BMR:00000000 CU_LIST:00000000 CURR_STATUS:00
+000198 RAND_NEXT:00000001 STRERRORBUF:247193BC
+0001A0 TMPAREA:00000000 IOWORKAREA:2471971C
+0001A8 TEMPDCB:00050088 TEMPJFCB:000500E8
+0001B0 TEMPDCB:2471967C NAMEBUF:259A0BC8
+0001B8 ERRNO_JR:00000000 RET_STRUCT:00000000
+0001C0 BKDN_IS_LOCALTIME:00000000 SWPRINTF_SIZE:00008000
+0001C8 SWPRINTF_BUF:00000000 S99P:24719624 MUTEXCTARRAY:24719EAC
+0001D4 STRFTIME_ERANAMECALLED:00000000 FCB_MUTEX:00000000
+000204 HSPABHWA:24719364 MUTEX_SAVE:24719EFC
+000210 INITIAL_CPU_TIME:4D000000 00053ADF FCB_MUTEX_OK:00000001
+00021C FCB_MUTEX_SAVE:00000000 ENTRY_ADDRTABLESIZE:00000000
+000224 ADDRESS:00000000 NUMBEROFNAMES:00000000
+00022C NAMES1:.....
+000245 NAMES2:.....
+00025E NAMES3:.....
+000277 NAMES4:.....
+000290 NAMES5:.....
+0002A9 NAMES6:.....
+0002C4 ENTRY_SITETABLESIZE:00000000 KIND:00
+0002CC NUM_ADDRS:00000000
+0002D0 ADDRESSES:00000000 00000000 00000000 00000000 00000000 00000000
+0002E8 NAME:00000000 00000000 00000000 00000000 00000000 00000000

```

```

[5]CPCB: 24719004
+000000 CPCB_EYE:CPCB CPCB_SIZE:00000038 CPCB_PTR:00000000
+00000C FLAGS1:40000000 TTKNHDR:00000000 TTKN:00000000
+000018 FOOTPRINT:2471A5A4 CODE370:00000000 CIO:247191AC
+000024 _Reuse:00000000 _RSAbove:24719004 _RSAboveLen:00003028
+000030 _RSBelow:000163B8 _RSBelowLen:00000328

```

```

[6]CIO: 247191AC
+000000 EYE:CIO SIZE:00000088 PTR:00000000 FLG1:08
+00000D FLG2:00 FLG3:00 FLG4:00 DUMMYF:24719234
+000014 EDCZ24:A49BF4E0 FCBSTART:259A0408 DUMMYFCB:2471924C
+000020 MFCBSTART:259A05F0 IOANYLIST:2599F000
+000028 IOBELOWLIST:00050000 FCBDDLST:24719FCC
+000030 PERRORBUF:24719074 TMPCOUNTER:00000000
+000038 TEMPMEM:00000000 PROMPTBUF:00000000 I024:000502D0
+000044 IOEXITS:00050F4C TERMINALCHAIN:00000000
+00004C VANCHOR:00000000 XTI:00000000 ENOWP24:249BFFD0
+000058 MAXNUMDESCRPS:00000000 DESCARRAY:00000000
+000060 PROC_RES_P:00000000 TEMPFILENUM:00000000 CSS:00000000
+00006C DUMMY_NAME:..... HOSTNAME_CACHE:00000000
+000078 HOSTADDR_CACHE:00000000

```

Figure 17. Example formatted C/C++ output from LEDATA Verbexit (Part 3 of 5)

```

[7]File name: memory.data
FCB: 259A0408
+000000 BUFPTR:259A07E5 COUNTIN:00000000 COUNTOUT:000003DB
+00000C READFUNC:259A04D8 WRITEFUNC:259A04F8 FLAGS1:0000
+000016 DEPTH:0000 NAME:259A05A4 _LENGTH:0000000B
+000020 _BUFSIZE:00000044 MEMBER:..... NEXT:2599F200
+000030 PREV:00000000 PARENT:259A0408 CHILD:00000000
+00003C DDNAME:..... FD:FFFFFFFF DEVTYPE:08 FCBTYPE:0055
+00004C FSCE:259A051C UNGETBUF:259A051C REPOS:24825EA0
+000058 GETPOS:24828418 CLOSE:24828678 FLUSH:24828AE0
+000064 UTILITY:2480D430 USERBUF:00000000 LRECL:00000400
+000070 BLKSIZE:00000400 REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000 BUFSIZE:00000400 BUF:259A07C0
+000084 CURSOR:259A07C0 ENDOFDATA:00000000 SAVEDBUF:00000000
+000090 REALCOUNTIN:00000000 REALCOUNTOUT:00000000
+000098 POSMAJOR:00000000 SAVEMAJOR:00000000
+0000A0 POSMINOR:00000000 SAVEMINOR:00000000 STATE:0000
+0000AA SAVESTATE:0000 EXITFTELL:00000000 EXITUNGETC:24815DB0
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:43020008 40001000
+0000C8 DBCSTATE:0000 FCB_CPCB:24719004
+0000D0 READGLUE:58FF0008 07FF0000 READ:248158B8
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 _RSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:248245D8
+0000FC WADDR_WSA:00000000 _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 _WSA:00000000

FCSE: 259A051C
+000000 GENERIC1:D4C5D4D6 259A05F0 259A0664
+00000C GENERIC2:00010000 00000000 248158B8
+000018 GENERIC3:248245D8 24825EA0 24828AE0
:
File name: DD:SYSPRINT

FCB: 24719FCC
+000000 BUFPTR:2599F0BD COUNTIN:00000000 COUNTOUT:00000084
+00000C READFUNC:2471A09C WRITEFUNC:2471A0BC FLAGS1:8000
+000016 DEPTH:0000 NAME:2471A168 _LENGTH:0000000B
+000020 _BUFSIZE:00000044 MEMBER:..... NEXT:2471A1C4
+000030 PREV:2471A3BC PARENT:24719FCC CHILD:00000000
+00003C DDNAME:SYSPRINT FD:FFFFFFFF DEVTYPE:02 FCBTYPE:0043
+00004C FSCE:2471A0E0 UNGETBUF:2471A0E0 REPOS:249C00D0
+000058 GETPOS:249C01F0 CLOSE:24A23150 FLUSH:24A23048
+000064 UTILITY:24A239A8 USERBUF:00000000 LRECL:00000089
+000070 BLKSIZE:00000372 REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000 BUFSIZE:0000008A BUF:2599F0B8
+000084 CURSOR:2599F0BC ENDOFDATA:00000000 SAVEDBUF:00000000
+000090 REALCOUNTIN:00000000 REALCOUNTOUT:00000000
+000098 POSMAJOR:00000000 SAVEMAJOR:00000000
+0000A0 POSMINOR:00000000 SAVEMINOR:00000000 STATE:0002
+0000AA SAVESTATE:0000 EXITFTELL:249C02A8 EXITUNGETC:249C0360
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:43128020 2A188000
+0000C8 DBCSTATE:0000 FCB_CPCB:24719004
+0000D0 READGLUE:58FF0008 07FF0000 READ:249BFE68
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000

```

Figure 17. Example formatted C/C++ output from LEDATA Verbexit (Part 4 of 5)

```

+0000E8 RCEESG003:00000000      RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:24A21A68
+0000FC WADDR_WSA:00000000      _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000      WWSA:00000000

```

This is the last heap segment in the current heap.

```

OSNS: 2471A0E0
+000000 OSNS_EYE:OSNS      READ:249BFE68      WRITE:24A21A68
+00000C REPOS:249C00D0      GETPOS:249C01F0      CLOSE:24A23150
+000018 FLUSH:24A23048      UTILITY:24A239A8      EXITFTELL:249C02A8
+000024 EXITUNGETC:249C0360      OSIOBLK:2599F020
+00002C NEWLINEPTR:2599F141      RECLENGTH:00000085      FLAGS:84800000

```

```

OSIO: 2599F020
+000000 OSIO_EYE:OSIO      DCBW:00050020      DCBRU:00000000
+00000C JFCB:00050F68      CURRMBUF:00051020      MBUFCOUNT:00000001
+000018 READMAX:00000001      CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF      BLKSPERTRK:00000000
+00002C FIRSTPOS:00000000      LASTPOS:00000000      NEWPOS:00000002
+000038 READFUNCNUM:00000005      WRITEFUNCNUM:24719FCC      FCB:2599F020
+000044 PARENT:80000000      FLAGS1:00000000      DCBERU:2599F078
+000050 DCBEW:80000040

```

```

DCB: 00050020
+000000 DCBRELAD:2599F078      DCBFDAD:00000000 00000019
+00000F DCBBUFNO:00      DCBSRG1:05 DCBEODAD:00005E DCBRECFM:A0
+000020 DCBEXLSA:860504      DCBDDNAM:;. . . . . DCBMACR1:9C
+00002E DCBMACR2:55      DCBSYNAD:000000 DCBBLKSI:0504 DCBNCP:00
+00004D DCBLRECL:9A2C

```

```

DCBE: 2599F078
+000000 DCBEID:DCBE      DCBELEN:0038      RESERVED0:0000
+000008 DCBEDCB:00050020      DCBERELA:00000000      DCBEFLG1:C0
+000011 DCBEFLG2:88      DCBENSTR:0000      DCBESIZE:00000000
+000028 DCBEOOA:00000000      DCBESYNA:00000000      MULTSDN:00

```

```

JFCB: 00050F68
+000000 JFCBDSNM:IBMUSER.PAHBAT.JOB00018.D0000101.?
+00002C JFCBELNM:      JFCBTSDM:20      JFCBDSDB:000000
+000046 JFCBVLSQ:0000      JFCBIND1:00      JFCBIND2:81
+000058 JFCBUFNO:00      JFCDSRG1:00      JFCDSRG2:00
+000064 JFCRECFM:00      JFCBLKSI:0000      JFCLRECL:0000      JFCNCP:00
+000075 JFCBNVOL:00      JFCFLGSI:00
:

```

Dummy FCB encountered at location 2471924C

Exiting CRTL Environment Data

Figure 17. Example formatted C/C++ output from LEDATA Verbexit (Part 5 of 5)

C/C++-specific sections of the LEDATA output

For the LEDATA output:

[1] CGEN

This section formats the C/C++-specific portion of the Language Environment common anchor area (CAA).

[2] CGENE

This section formats the extension to the C/C++-specific portion of the Language Environment common anchor area (CAA).

[3] CEDB

This section formats the C/C++-specific portion of the Language Environment enclave data block (EDB).

[4] CTHD

This section formats the C/C++ thread-level control block (CTHD).

[5] CPCB

This section formats the C/C++-specific portion of the Language Environment process control block (PCB).

[6] CIO

This section formats the C/C++ IO control block (CIO).

[7] File Control Blocks

This section formats the C/C++ file control block (FCB). The FCB and its related control blocks represent the information needed by each open stream.

Related Control Blocks

FSCE The file specific category extension control block. The FSCE represents the specific type of IO being performed. The following is a list of FSCEs that may be formatted.

OSNS — OS no seek

OSFS — OS fixed text

OSVF — OS variable text

OSUT — OS undefined format text

Other FSCEs will be displayed using a generic overlay.

OSIO The OS IO interface control block.

DCB The data control block. For more information about the DCB, refer to *z/OS DFSMS Macro Instructions for Data Sets*.

DCBE The data control block extension. For more information about the DCBE, refer to *z/OS DFSMS Macro Instructions for Data Sets*.

JFCB The job file control block (JFCB). For more information about the JFCB, refer to *z/OS MVS Data Areas, Vol 3 (IVT-RCWK)*.

Understanding the COBOL-specific LEDATA output

The Language Environment IPCS Verbexit LEDATA generates formatted output of COBOL-specific control blocks from a system dump when the ALL parameter is specified and COBOL is active in the dump. Figure 18 on page 110 illustrates the COBOL-specific output produced. The system dump being formatted was obtained by specifying the TERMTHDACT(UADUMP) run-time option. “COBOL-specific sections of the LEDATA Output” on page 111 describes the information contained in the formatted output.

For easy reference, the sections of the dump are numbered to correspond with the description of each section that follows.

```
*****
                                COBOL ENVIRONMENT DATA
*****

[1] RUNCOM: 00049038
+000000 IDENT:C3RUNCOM LENGTH:000002D8 FLAGS:00860000
+000010 RU_ID:000178B0 INVK_RSA:00005F80
+000024 MAIN_PGM_ADDR:00007DE8 MAIN_PGM_CLLE:00049328
+00002C ITBNAB:00000000 PARM_ADDR:000179D0 NEXT_RUNCOM:00000000
+000040 THDCOM:0001AA80 COBVEC:0001A1BC SUBCOM:00000000
+00004C COBVEC2:0001A7FC CAA:00018920 UPSI_SWITCHES:00000000
+00007C DUM_CLLE:0BF15BA8 1ST_FREE_CLLE:00000000
+000088 HAT:0BF157A8 1ST_CLLE:00049488
+000090 SORT_CONTROL_DCB:00000000 COBOL_ACTIVE:00000000
+0000A4 IO_FLAGS:00000000 UNSTR_WRK:00000000
+00011C INSP_WRK:00000000 INSP_WRK1:00000000
+00012C DDNAME_SORT_CONTROL:..... LEN_UNSTR_WRK:00000000
+000138 UNSTR_DELIMS:0000
+000154 CEEINT_PLIST:000491B0 00000000 00000006 000491B4 00000000 00000000
+00016C ----->:00000005 00000000 00000000 00000000 00000000
+0001C8 MAIN_ID:CALLSUBX
+000204 ----->:
+000240 ----->:

[2] THDCOM: 0001AA80
+000000 IDENT:C3THDCOM LENGTH:000001E8 FLAGS:81000000 00000100
+000018 COBCOM:0001A108 COBVEC:0001A1BC 1ST_RUNCOM:00049038
+000028 1ST_PROGRAM:CALLSUBX SUBCOM:00000000
+000034 CEEINT_PLIST:00000000 00000000 00000000 00000000 00000000 00000000
+00004C ----->:00000000 00000000 00000000 00000000 00000000
+000084 COBVEC2:0001A7FC ITBLK:00000000 STT_BST:00000000
+000098 CICS_EIB:00000000 SIBLING:00000000
+0000AC SORT_RETURN:00000000 INFO_MSG_LIMIT:0000
+0000C8 R12_SAVE:00000000 STP_DUM_TGT:00000000
+000180 LRR_COBCOM:00000000 CAA:00018920 DUM_THDCOM:00000000
+00019C ITBLK_TRAP_RSA:00000000 ITBLK_PLFPARMS:00000000
+0001A4 ITBLK_BS2PARMS:00000000 ITBLK_NAB:00000000
+0001AC DUM_MAIN_DSA:00000000 BDY_RSA:00000000
+0001D0 RRE_TAIL_RSA:00000000 ESTUB_TGT:00000000

[3] COBCOM: 0001A108
+000000 IDENT:C3COBCOM LENGTH:00000978 VERSION:010900
+000058 FLAGS:906000 ESM_ID:0 COBVEC:0001A1BC
+000060 COBVEC2:0001A7FC
+000064 LOADFG:00000100 00000000 80000000 00008000 00000000
+000078 THDCOM:0001AA80 INSH:00000000 LRR_THDCOM:00000000
+00009C LRR_ITBLK:00000000 LRR_SUBCOM:00000000
+0000A4 LRR_EPLF:00000000

[4] CLLE: 00049488
+000000 PGMNAME:PARM5 OPEN_NON_EXT_FILES:0000 TGT_FLAGS:00
+00000C LANG_LST:00050F98 INFO_FLAGS:8891 LOAD_ADDR:8004FF88
+000018 TGT_ADDR:00050248 LE_TOKEN:0BF150BC FLAGS2:00
```

Figure 18. Example formatted COBOL output from LEDATA Verbexit (Part 1 of 2)


```

[5]  TGT: 00050248
+000048 IDENT:3TGT LVL:05      FLAGS:40020220  RUNCOM:00049038
+00005C COBVEC:0001A7FC #FCBS:00000000  WS_LEN:00000000
+000070 SMG_WRK:00000000      CAA:00018920  LEN:00000154
+00008C EXT_FCBS:00000000      OUTDD:SYSOUT
+0000AC CALC_RSA:00000000 00000000 00000000 00000000 00000000 00000000
+0000C4 ----->:00000000 00000000 00000000 00000000 00000000 00000000
+0000DC ----->:00000000      ABINF:000500A5  TESTINF:00000000
+000100 PGMADDR:0004FF88      1STFCB:00000000  WS_ADDR:00000000
+000118 1STEXTFCB:00000000

      CLLE: 00049440
+000000 PGMNAME:PARM1 OPEN_NON_EXT_FILES:0000      TGT_FLAGS:00
+00000C LANG_LST:0004EF98      INFO_FLAGS:8891  LOAD_ADDR:8004DFE0
+000018 TGT_ADDR:0004E258      LE_TOKEN:0BF150A0  FLAGS2:00

      TGT: 0004E258
+000048 IDENT:3TGT LVL:05      FLAGS:40020220  RUNCOM:00049038
+00005C COBVEC:0001A7FC #FCBS:00000000  WS_LEN:00000000
+000070 SMG_WRK:00000000      CAA:00018920  LEN:00000144
+00008C EXT_FCBS:00000000      OUTDD:SYSOUT
+0000AC CALC_RSA:00000000 00000000 00000000 00000000 00000000 00000000
+0000C4 ----->:00000000 00000000 00000000 00000000 00000000 00000000
+0000DC ----->:00000000      ABINF:0004E0FD  TESTINF:00000000
+000100 PGMADDR:0004DFE0      1STFCB:00000000  WS_ADDR:00000000
+000118 1STEXTFCB:00000000

      CLLE: 00049370
+000000 PGMNAME:PARM0 OPEN_NON_EXT_FILES:0000      TGT_FLAGS:00
+00000C LANG_LST:0004CF98      INFO_FLAGS:8891  LOAD_ADDR:8004BFF8
+000018 TGT_ADDR:0004C260      LE_TOKEN:0BF15084  FLAGS2:00

      TGT: 0004C260
+000048 IDENT:3TGT LVL:05      FLAGS:40020220  RUNCOM:00049038
+00005C COBVEC:0001A7FC #FCBS:00000000  WS_LEN:00000000
+000070 SMG_WRK:00000000      CAA:00018920  LEN:00000140
+00008C EXT_FCBS:00000000      OUTDD:SYSOUT
+0000AC CALC_RSA:00000000 00000000 00000000 00000000 00000000 00000000
+0000C4 ----->:00000000 00000000 00000000 00000000 00000000 00000000
+0000DC ----->:00000000      ABINF:0004C115  TESTINF:00000000
+000100 PGMADDR:0004BFF8      1STFCB:00000000  WS_ADDR:00000000
+000118 1STEXTFCB:00000000

      CLLE: 00049328
+000000 PGMNAME:CALLSUBX OPEN_NON_EXT_FILES:0000      TGT_FLAGS:00
+00000C LANG_LST:00000000      INFO_FLAGS:9881  LOAD_ADDR:80007DE8
+000018 TGT_ADDR:00008220      LE_TOKEN:00000000  FLAGS2:00

      TGT: 00008220
+000048 IDENT:3TGT LVL:05      FLAGS:60020220  RUNCOM:00049038
+00005C COBVEC:0001A7FC #FCBS:00000000  WS_LEN:0000002C
+000070 SMG_WRK:00000000      CAA:00018920  LEN:00000150
+00008C EXT_FCBS:00000000      OUTDD:SYSOUT
+0000AC CALC_RSA:00000000 00000000 00000000 00000000 00000000 00000000
+0000C4 ----->:00000000 00000000 00000000 00000000 00000000 00000000
+0000DC ----->:00000000      ABINF:00007F34  TESTINF:00000000
+000100 PGMADDR:00007DE8      1STFCB:00000000  WS_ADDR:000083C0
+000118 1STEXTFCB:00000000

```

Exiting COBOL Environment Data

Figure 18. Example formatted COBOL output from LEDATA Verbexit (Part 2 of 2)

COBOL-specific sections of the LEDATA Output

For the LEDATA output:

[1] RUNCOM

This section formats the COBOL enclave-level control block (RUNCOM).

[2] THDCOM

This section formats the COBOL process-level control block (THDCOM).

[3] COBCOM

This section formats the COBOL region-level control block (COBCOM).

[4] CLLE

This section formats the COBOL loaded program control blocks (CLLE).

[5] TGT

This section formats the COBOL TGT control blocks.

Formatting individual control blocks

In addition to the full LEDATA output which contains many formatted control blocks, the IPCS Control block formatter can also format individual Language Environment control blocks.

The IPCS `cbf` command can be invoked from the "IPCS Subcommand Entry" screen, option 6 of the "IPCS PRIMARY OPTION MENU".

Syntax

►► `CBF—address—STRUCTure—(—cbname—)` ◀◀

address

The address of the control block in the dump. This is determined by browsing the dump or running the LEDATA verb exit.

cbname

The name of the control block to be formatted. The control blocks that can be individually formatted are listed in Table 8 on page 113. In general, the name of each control block is similar to that used by the LEDATA verb exit and is generally found in the control block's eyecatcher field. However, all control block names are prefixed with CEE in order to uniquely define the Language Environment control block names to IPCS.

For an example of the display which is the result of the command, see Figure 19 on page 113 :

```
CBF 15890 struct(CEECAA)
```

```

CEECAA: 00015890
+000000 FLAG:00 LANGP:08 BOS:00023000 EOS:00043000
+000044 TORC:00000000 TOVF:8000B5A0 ATTN:06412AF8
+00015C HLLEXIT:00000000 HOOK:50C0D064 05C058C0 C00605CC
+0001A4 DIMA:0000D176 ALLOC:0700C198 STATE:0700C198
+0001B0 ENTRY:0700C198 EXIT:0700C198 MEXIT:0700C198
+0001BC LABEL:0700C198 BCALL:0700C198 ACALL:0700C198
+0001C8 DO:0700C198 IFTRUE:0700C198 IFFALSE:0700C198
+0001D4 WHEN:0700C198 OTHER:0700C198 CGOTO:0700C198
+0001F4 CRENT:00000000 EDCV:864D9170 TCASRV_USERWORD:00000000
+00025C TCASRV_WORKAREA:06412448 TCASRV_GETMAIN:00000000
+000264 TCASRV_FREEMAIN:00000000 TCASRV_LOAD:8000E738
+00026C TCASRV_DELETE:8000E428 TCASRV_EXCEPTION:00000000
+000274 TCASRV_ATTENTION:00000000 TCASRV_MESSAGE:00000000
+000280 LWS:000174B0 SAVR:00000000 SYSTM:03 HRDWR:03
+0002AE SBSYS:02 FLAG2:00 LEVEL:08 PM:04 GETLS:00011CA0
+0002B8 CELV:00018038 GETS:00011BB0 LBOS:00021000
+0002C4 LEOS:00023000 LNAB:00022E98 DMC:00000000
+0002D0 ABCODE:00000000 RSNCODE:00000000 ERR:00021480
+0002DC GETSX:00011930 DDSA:00016128 SECTSIZE:00000000
+0002E8 PARTSUM:00000000 SSEXPN:00000000 EDB:000148B0
+0002F4 PCB:00014558 EYEPT:00015878 PTR:00015890
+000300 GETS1:00012730 SHAB:00000000 PRGCK:00000004 FLAG1:00
+000310 URC:00000000 ESS:00042F00 LESS:00022F00
+00031C OGETS:000120F8 OGETLS:00000000 PICICB:00000000
+000328 GETSX:00000000 GOSMR:0000 LEOV:00000000
+000334 SIGSCTR:00000000 SIGSFLG:00000000
+00033C THDID:80000000 00000000 DCRENT:00000000
+000348 DANCHOR:00000000 CTOC:00000000 RCB:00013918
+000354 CICSRSN:00000000 MEMBR:000161C8
+00035C SIGNAL_STATUS:00000000 FOR1:00000000 FOR2:00000000
+000378 THREADHEAPID:00000000 SIGNGPTR:00015C24 SIGNG:00000001
+000398 FORDBG:00000000 AB_STATUS:00 AB_GR0:00000000
+0003A4 AB_ICD1:00000000 AB_ABCC:00000000 AB_CRC:00000000
+0003D4 SMCB:00015F70 ERRCM:06412AB0 MIB_PTR:00000000
+000434 THDSTATUS:00000000 TICB_PTR:06413840
+00047C FWD_CHAIN:00015890 BKWD_CHAIN:00015890

```

Figure 19. The CAA formatted by the CBFORMAT IPCS command

For more information on using the IPCS CBF command refer to the "CBFORMAT subcommand" section in *z/OS MVS IPCS Commands*, SA22-7594.

Table 8. Language Environment Control blocks which can be individually formatted

Control Block	Description
CEEADHP	Additional Heap Control Block
CEECAA	Common Anchor Area
CEECIB	Condition Information Block
CEECIBH	Condition Information Block Header
CEECMXB	Message Services Block
CEEDSA	Dynamic Storage Area
CEEDSATR	XPLINK Transition Area
CEEDSAX	Dynamic Storage Area (XPLINK style)
CEEEDB	Enclave Data Block
CEEENSM	Enclave Level Storage Management
CEEHANC	Heap Anchor Node

Table 8. Language Environment Control blocks which can be individually formatted (continued)

Control Block	Description
CEEHCOM	CEL Exception Manager Communications Area
CEEHPCB	Thread Level Heap Control Block
CEEHPSB	Heap Statistics Block
CEEMDST	Message Destination
CEEMGF	Mapping of the Message Formatter (IBM1MGF)
CEEPCB	Process Control Block
CEEPMCB	Program Management Control Block
CEERCB	Region Control Block
CEESKSB	Stack Statistics Block
CEESMCB	Storage Management Control Block
CEESTKH	Stack Header Block
CEESTKHx	Stack Header Block (xplink style)
CEESTSB	Storage Report Statistics Block
CEETMXB	Thread Level Messages Extension Block

Requesting a Language Environment trace for debugging

Language Environment provides an in-storage, wrapping trace facility that can reconstruct the events leading to the point where a dump is taken. The trace facility can record two types of events: entry and exit library calls and, if the POSIX run-time option is set to ON, user mutex and condition variable activity such as init, lock/unlock, and wait. Language Environment produces a trace table in its dump report under the following conditions:

- The CEE3DMP callable service is invoked with the BLOCKS option and the TRACE run-time option is set to ON.
- The TRACE run-time option is set to NODUMP and the TERMTHDACT run-time option is set to DUMP, UADUMP, TRACE, or UATRACE.
- The TRACE run-time option is set to DUMP (the default).

For more information about the CEE3DMP callable service, the TERMTHDACT run-time option, or the TRACE run-time option, see *z/OS Language Environment Programming Reference*.

The TRACE run-time option activates Language Environment run-time library tracing and controls the size of the trace buffer, the type of trace events to record, and it determines whether a dump containing only the trace table should be unconditionally taken when the application (enclave) terminates. The trace table contents can be written out either upon demand or at the termination of an enclave.

The contents of the Language Environment dump depend on the values set in the TERMTHDACT run-time option. Under abnormal termination, the following dump contents are generated:

- TERMTHDACT(QUIET) generates a Language Environment dump containing the trace table only
- TERMTHDACT(MSG) generates a Language Environment dump containing the trace table only

- TERMTHDACT(TRACE) generates a Language Environment dump containing the trace table and the traceback
- TERMTHDACT(DUMP) generates a Language Environment dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block)
- TERMTHDACT(UAONLY) generates a system dump of the user address space
- TERMTHDACT(UATRACE) generates a Language Environment dump that contains traceback information, and a system dump of the user address space
- TERMTHDACT(UADUMP) generates a Language Environment dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block), and a user address space dump
- TERMTHDACT(UAIMM) generates a system dump of the user address space of the original abend or program interrupt that occurred prior to the Language Environment condition manager processing the condition.

Note: Under CICS, UAIMM yields UAONLY behavior. Under non-CICS, TRAP(ON,NOSPIE) must be in effect. When TRAP(ON,SPIE) is in effect, UAIMM yields UAONLY behavior. For software raised conditions or signals, UAIMM behaves the same as UAONLY.

Under normal termination, the following dump contents are generated:

- Independent of the TERMTHDACT setting, Language Environment generates a dump containing the trace table only based on the TRACE run-time option

Language Environment quiesces all threads that are currently running except for the thread that issued the call to CEE3DMP. When you call CEE3DMP in a multithread environment, only the current thread is dumped. Enclave- and process-related storage could have changed from the time the dump request was issued.

Locating the trace dump

If your application calls CEE3DMP, the Language Environment dump is written to the file specified in the FNAME parameter of CEE3DMP (the default is CEEDUMP).

If your application is running under TSO or batch, and a CEEDUMP DD is not specified, Language Environment writes the CEEDUMP to the batch log (SYSOUT=* by default). You can change the SYSOUT class by specifying a CEEDUMP DD, or by setting the environment variable, `_CEE_DMPTARG=SYSOUT(x)`, where x is the preferred SYSOUT class.

If your application is running under z/OS UNIX and is either running in an address space you issued a `fork()` to, or if it is invoked by one of the exec family of functions, the dump is written to the hierarchical file system (HFS). Language Environment writes the CEEDUMP to one of the following directories in the specified order:

1. The directory found in environment variable `_CEE_DMPTARG`, if found
2. The current working directory, if the directory is not the root directory (/), and the directory is writable
3. The directory found in environment variable `TMPDIR` (an environment variable that indicates the location of a temporary directory if it is not /tmp)
4. The `/tmp` directory

The name of this file changes with each dump and uses the following format:

`/path/Fname.Date.Time.Pid`

path	The path determined from the above algorithm.
Fname	The name specified in the FNAME parameter on the call to CEE3DMP (default is CEEDUMP).
Date	The date the dump is taken, appearing in the format YYYYMMDD (such as 20040918 for September 18, 2004).
Time	The time the dump is taken, appearing in the format HHMMSS (such as 175501 for 05:55:01 p.m.).
Pid	The process ID the application is running in when the dump is taken.

Using the Language Environment trace table format in a dump report

The Language Environment trace table is established unconditionally at enclave initialization time if the TRACE run-time option is set to ON. All threads in the enclave share the trace table; there is no thread-specific table, nor can the table be dynamically extended or enlarged.

Understanding the trace table entry (TTE)

Each trace table entry is a fixed-length record consisting of a fixed-format portion (containing such items as the timestamp, thread ID, and member ID) and a member-specific portion. The member-specific portion has a fixed length, of which some (or all) can be unused. For information about how participating products use the trace table entry, refer to the product-specific documentation. The format of the trace table entry is as follows:

Time of Day	Thread ID	Member ID and flags	Member entry type	Mbr-specific info up to a maximum of 104 bytes
Char (8)	Char (8)	Char (4)	Char (4)	Char (104)

Figure 20. Format of the trace table entry

Following is a definition of each field:

Time	The 64-bit value obtained from a store clock (STCK).
Thread ID	The 8-byte thread ID of the thread that is adding the trace table entry.
Member ID and Flags	Contains 2 fields:
Member ID	The 1-byte member ID of the member making the trace table entry, as follows:
ID	Name
01	CEL
03	C/C++
05	COBOL
07	Fortran
08	DCE
10	PL/I
12	Sockets

Flags 24 flags reserved for internal use.

Member Entry Type

A number that indicates the type of the member-specific trace information that follows the field.

To uniquely identify the information contained in a specific TTE, you must consider Member ID as well as Member Entry Type.

Member-Specific Information

Based on the member ID and the member entry type, this field contains the specific information for the entry, up to 104 bytes.

For C/C++, the entry type of 1 is a record that records an invocation of a base C run-time library function. The entry consists of the name of the invoking function and the name of the invoked function. Entry type 2 is a record that records the return from the base library function. It contains the returned value and the value of `errno`.

Member-specific information in the trace table entry

Global tracing is activated by using the `LE=n` suboption of the `TRACE` run-time option. This requests all Language Environment members to generate trace records in the trace table.

The settings for the global trace events are:

Level	Description
0	No global trace
1	Trace all run-time library (RTL) function entry and exits
2	Trace all RTL mutex init/destroy and lock/unlock
3	Trace all RTL function entry and exits, and all mutex init/destroy and lock/unlock
8	Trace all RTL storage allocation/deallocation

When `LE=1` is specified: The following C/C++ records may be generated.

Table 9. `LE=1` entry records

Member ID	Record Type	Description
03	00000001	Base C Library function Entry
03	00000002	Base C Library function Exit
03	00000003	Posix C Library function Entry
03	00000004	Posix C Library function Exit
03	00000005	XPLINK Base or Posix C Library function Entry
03	00000006	XPLINK Base or Posix C Library function Exit

For a detailed description of these records, see “C/C++ contents of the Language Environment trace tables” on page 167.

When `LE=2` is specified: The following Language Environment records may be generated.

Table 10. LE=2 entry records

Member ID	Record Type	Class	Event	Description
01	00000101	LT	A	Latch Acquire
01	00000102	LT	R	Latch Release
01	00000103	LT	W	Latch Wait
01	00000104	LT	AW	Latch Acquire after Wait
01	00000106	LT	I	Latch Increment (Recursive)
01	00000107	LT	D	Latch Decrement (Recursive)
01	000002FC	LE	EUO	Latch unowned (not released)
01	000002FD	LE	EO	Latch already owned (not acquired)
01	00000301	MX	A	Mutex acquire
01	00000302	MX	R	Mutex release
01	00000303	MX	W	Mutex wait
01	00000304	MX	AW	Mutex acquire after wait
01	00000305	MX	B	Mutex busy (Trylock failed)
01	00000306	MX	I	Mutex increment (recursive)
01	00000307	MX	D	Mutex decrement (recursive)
01	00000315	MX	IN	Mutex initialize
01	00000316	MX	DS	Mutex destroy
01	0000031D	MX	BI	Shared memory lock init
01	0000031E	MX	BD	Shared memory lock destroy
01	0000031F	MX	BO	shared memory lock obtain
01	00000320	MX	BC	Shared memory lock obtain on condition
01	00000321	MX	BR	Shared memory lock release
01	00000324	MX	CIN	Call to SMC_INIT
01	00000325	MX	CSD	Call to SMC_DESTROY
01	00000326	MX	CSO	Shared resource obtain
01	00000327	MX	CSR	Shared resource release
01	00000328	MX	CST	Call to SMC_SetupToWait
01	00000329	MX	CSP	Call to SMC_POST
01	000004CC	ME	FFR	Error - Forced release (shared mutex)
01	000004CD	ME	FFD	Error - Forced decrement (shared mutex)
01	000004CE	ME	FBD	Error - BPX_SMC(DESTROY) error return
01	000004CF	ME	FBU	Error - BPX_SMC(fail) returns EBUSY
01	000004D0	ME	FIV	Error - BPX_SMC(fail) returns EINVAL
01	000004D4	ME	FDU	Error - Destroy failed (uninitialized) (shared mutex/CV)
01	000004D5	ME	FP	Error - Program check (shared mutex/CV)
01	000004DB	ME	ESC	Error - BPX1SMC error return
01	000004DE	ME	EDL	Shared memory lock returns deadlock
01	000004DF	ME	EIV	Shared memory lock returns invalid

Table 10. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
01	000004E0	ME	EPM	Shared memory lock returns eperm
01	000004E1	ME	EAG	Shared memory lock returns eagain
01	000004E2	ME	EBU	Shared memory lock returns ebusy
01	000004E3	ME	ENM	Shared memory lock returns enomem
01	000004E4	ME	EBR	Shared memory lock release error
01	000004E5	ME	EBC	Shared memory lock obtain condition error
01	000004E6	ME	EBO	Shared memory lock obtain error
01	000004E7	ME	EBD	Shared memory lock destroy error
01	000004E8	ME	EBI	Shared memory lock initialize error
01	000004E9	ME	EFR	Mutex forced release
01	000004EA	ME	EFD	Mutex forced decrement
01	000004EB	ME	EDD	Mutex destroy failed (damage)
01	000004EC	ME	EDB	Mutex destroy failed (busy)
01	000004ED	ME	EIA	Mutex initialize failed (attribute)
01	000004EE	ME	EIS	Mutex initialize failed (storage)
01	000004EF	ME	EF	Mutex release (forced by quiesce)
01	000004F0	ME	EP	Mutex program check
01	000004FA	ME	EDU	Mutex destroy failed (uninitialized)
01	000004FB	ME	EUI	Mutex uninitialized
01	000004FC	ME	EUO	Mutex unowned (not released)
01	000004FD	E	EO	Mutex already owned (not acquired)
01	000004FE	ME	EIN	Mutex initialization failed (duplicate)
01	00000508	CV	MR	CV release mutex
01	00000509	CV	MA	CV reacquire mutex
01	0000050A	CV	MW	CV mutex wait
01	0000050B	CV	MAW	CV reacquire mutex after wait
01	0000050C	CV	CW	CV condition wait
01	0000050D	CV	CTW	CV condition timeout
01	0000050E	CV	CWP	CV wait posted
01	0000050F	CV	CWI	CV wait interrupted
01	00000510	CV	CTO	CV wait timeout
01	00000511	CV	CSS	CV condition signal success
01	00000512	CV	CSM	CV condition signal miss
01	00000513	CV	CBS	CV condition broadcast success
01	00000514	CV	CBM	CV condition broadcast miss
01	00000515	CV	IN	CV initialize
01	00000516	CV	DS	CV destroy
01	00000522	CV	CIN	Call to SMC_INIT
01	00000523	CV	CSD	Call to SMC_DESTROY

Table 10. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
01	00000529	CV	CSP	Call to SMC_POST
01	0000052A	CV	CSB	Call to SMC_POSTALL
01	0000052B	CV	CSW	Call to SMC_WAIT
01	0000052C	CV	DBM	Shared condition broadcast - miss
01	0000052D	CV	DBS	Shared condition broadcast - success
01	0000052E	CV	DDS	Destroy (shared mutex/CV)
01	0000052F	CV	DIN	Initialize (shared mutex/CV)
01	00000530	CV	DSM	Condition signal - miss (shared CV)
01	00000531	CV	DSS	Condition signal - success (shared CV)
01	00000532	CV	DWI	Wait interrupted (shared CV)
01	00000533	CV	DTO	Wait timeout (shared CV)
01	00000534	CV	DWP	Wait posted (shared CV)
01	000006CB	CE	FBT	Error - Invalid system TOD (shared)
01	000006D1	CE	FRM	Error - Recursive mutex (shared)
01	000006D2	CE	FUO	Error - Shared mutex unowned
01	000006D3	CE	FDB	Error - Destroy failed (busy) (shared mutex/CV)
01	000006D4	CE	FDU	Error - Destroy failed (uninitialized) (shared mutex/CV)
01	000006D5	CE	FP	Error - Program check (shared mutex/CV)
01	000006D6	CE	FUI	Error - Shared mutex or CV uninitialized
01	000006D7	CE	ENV	Error - BPX1SMC(fail) returns EINVAL
01	000006D8	CE	EPE	Error - BPX1SMC(fail) returns EPERM
01	000006D9	CE	EAN	Error - BPX1SMC(fail) returns EAGAIN
01	000006DA	CE	EIB	Error - BPX1SMC failed (EBUSY)
01	000006DB	CE	ESC	Error - BPX1SMC failed
01	000006EB	CE	EDD	CV destroy failed (damage)
01	000006EC	CE	EDB	CV destroy failed (busy)
01	000006ED	CE	EIA	CV initialization failed (attribute)
01	000006EE	CE	EIS	CV initialization failed (storage)
01	000006EF	CE	EF	CV forced by quiesce
01	000006F0	CE	EP	CV program check
01	000006F1	CE	EBT	CV invalid system TOD
01	000006F2	CE	EBN	CV invalid timespec (nanoseconds)
01	000006F3	CE	EBS	CV invalid timespec (seconds)
01	000006F4	CE	EPO	CV condition post callable service fail
01	000006F5	CE	ETW	CV condition timed wait callable service fail
01	000006F6	CE	EWA	CV condition wait callable service fail
01	000006F7	CE	ESE	CV condition setup callable service fail
01	000006F8	CE	ERM	CV recursive mutex

Table 10. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
01	000006F9	CE	EWM	CV wrong mutex
01	000006FA	CE	EDU	CV destroy failed (uninitialized)
01	000006FB	CE	EUI	CV mutex or CV uninitialized
01	000006FC	CE	EUO	CV mutex unowned
01	000006FE	CE	EIN	CV initialization failed (duplicate)
01	00000702	RW	R	Release
01	00000704	RW	AW	Acquire after wait
01	00000706	RW	I	Increment (recursive)
01	00000707	RW	D	Decrement (recursive)
01	00000715	RW	IN	Initialize
01	00000716	RW	DS	Destroy
01	00000717	RW	RA	Read acquire
01	00000718	RW	WA	Write acquire
01	00000719	RW	RB	Read busy (tryread failed)
01	0000071A	RW	WB	Write busy (trywrite failed)
01	0000071B	RW	RW	Read wait
01	0000071C	RW	WW	Write wait
01	0000071D	RW	BI	Call to SLK_INIT
01	0000071E	RW	BD	Call to SLK_DESTROY
01	0000071F	RW	BO	Call to SLK_OBTAIN
01	00000720	RW	BC	Call to SLK_OBTAIN_COND
01	00000721	RW	BR	Call to SLK_RELEASE
01	000008DC	RE	EOW	Error - Already owned for write (not acquired)
01	000008DD	RE	EOR	Error - Already owned for read (not acquired)
01	000008DE	RE	EDL	Error - BPX1SLK(fail) returns EDEADLK
01	000008DF	RE	EIV	Error - BPX1SLK(fail) returns EINVAL
01	000008E0	RE	EPM	Error - BPX1SLK(fail) returns EPERM
01	000008E1	RE	EAG	Error - BPX1SLK(fail) returns EAGAIN
01	000008E2	RE	EBS	Error - BPX1SLK(fail) returns EBUSY
01	000008E3	RE	ENM	Error - BPX1SLK(fail) returns ENOMEM
01	000008E4	RE	EBR	Error - BPX1SLK(RELEASE) error return
01	000008E5	RE	EBC	Error - BPX1SLK(OBTAIN_COND) error return
01	000008E6	RE	EBO	Error - BPX1SLK(OBTAIN) error return
01	000008E7	RE	EBD	Error - BPX1SLK(DESTROY) error return
01	000008E8	RE	EBI	Error - BPX1SLK(INIT) error return
01	000008E9	RE	EFR	Error - Forced release
01	000008EA	RE	EFD	Error - Forced decrement
01	000008ED	RE	EIA	Error - Initialization failed (attribute)
01	000008EE	RE	EIS	Error - Initialization failed (storage)

Table 10. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
01	000008EF	RE	EF	Error - Forced by quiesce
01	000008F0	RE	EP	Error - Program check
01	000008FB	RE	EUI	Error - Uninitialized
01	000008FC	RE	EUO	Error - Unowned (not released)
01	000008FD	RE	EO	Error - Already owned (not acquired)
01	000008FE	RE	EIN	Error - Initialization failed (duplicate)

The format for the Mutex – Condition Variable – Latch entries in the trace table is:

Table 11. Format of the mutex/CV/latch records

Class	Source		Event		Object Addr	Name1	Name2
unused							

Where each field represents:

Class Two character EBCDIC representation of the trace class.

LT	Latch
LE	Latch Exception
MX	Mutex
ME	Mutex Exception
CV	Condition Variable
CE	Condition Variable Exception

Source

One character EBCDIC representation of the event.

C	C/C++
D	DCE
S	Sockets

Blank Blank character

Event Two character EBCDIC representation of the event. See Table 10 on page 118.

Object address

Fullword address of the mutex object.

Name 1

Optional eight character field containing the name of the function or object to be recorded.

Name 2

Optional eight character field containing the name of the function or object to be recorded.

When LE=3 is specified: The trace table will include the records generated by both LE=1 and LE=2.

When LE=8 is specified: The trace table will contain only storage allocation records. Currently this is only supported by C/C++.

Table 12. LE=8 entry records

Member ID	Record Type	Description
03	00000001	Storage allocation entry
03	00000001	Storage allocation exit

For a detailed description of these records, see “C/C++ contents of the Language Environment trace tables” on page 167.

Sample dump for the trace table entry

The following is an example of a dump of the trace table when you specify the LE=1 suboption (the library call/return trace):

```

:
:
Enclave Control Blocks:
  EDB: 0001E920
      +000000 0001E920 C3C5C5C5 C4C24040 C5000001 00020E68 0001F040 00000000 00000000 00000000 |CEEEDB E.....0 .....|
:
:
  MEML: 00020E68
      +000000 00020E68 00000000 00000000 0007C5B8 00000000 00000000 00000000 0007C5B8 00000000 |.....E.....E.....|
:
:
Language Environment Trace Table:
Most recent trace entry is at displacement: 001B80

Displacement      Trace Entry in Hexadecimal      Trace Entry in EBCDIC
-----
+000000  Time 21.41.57.595359      Date 2001.08.26      Thread ID... 8000000000000000
+000010  Member ID.... 03      Flags..... 000000      Entry Type.... 00000001
+000018  6D6DA289 9589A3F6 F46D6D83 82836D83 938296A2 F2F46D89 96A2A399 8581946D |__sinit64__cbc_clbos24_iostream_
+000038  60606E4D F1F9F35D 406D6D87 85A38382 4D5D4040 40404040 40404040 40404040 |-->(193) __getc()
+000058  40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000078  40404040 40404040

+000080  Time 21.41.57.595367      Date 2001.08.26      Thread ID... 8000000000000000
+000090  Member ID.... 03      Flags..... 000000      Entry Type.... 00000002
+000098  4C60604D F1F9F35D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(193) R15=00000000 ERRNO=0000
+0000B8  F0F0F0F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |0000.....
+0000D8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0000F8  00000000 00000000

+000100  Time 21.41.57.595374      Date 2001.08.26      Thread ID... 8000000000000000
+000110  Member ID.... 03      Flags..... 000000      Entry Type.... 00000003
+000118  6D6DA289 9589A3F6 F46D6D83 82836D83 938296A2 F2F46D89 96A2A399 8581946D |__sinit64__cbc_clbos24_iostream_
+000138  60606E4D F1F9F35D 406D6D89 A2D796A2 89A7D695 4D5D4040 40404040 40404040 |-->(113) __isPosixOn()
+000158  40404040 40404040 40404040 40404040 40404040 40404040 40000000 00000000 |.....
+000178  00000000 00000000

+000180  Time 21.41.57.595380      Date 2001.08.26      Thread ID... 8000000000000000
+000190  Member ID.... 03      Flags..... 000000      Entry Type.... 00000004
+000198  4C60604D F1F9F35D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(113) R15=00000000 ERRNO=0000
+0001B8  F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0 F0F0F0F0 00000000 00000000 00000000 |0000 ERRNO2=00000000.....
+0001D8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0001F8  00000000 00000000

+000200  Time 21.41.57.595638      Date 2001.08.26      Thread ID... 8000000000000000
+000210  Member ID.... 03      Flags..... 000000      Entry Type.... 00000001
+000218  D3968392 A27A7AC9 95A2A381 9583854D 5D404040 40404040 40404040 40404040 |Locks::Instance()
+000238  60606E4D F1F2F45D 40948193 9396834D F1F6F0F0 5D404040 40404040 40404040 |-->(124) malloc(1600)
+000258  40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000278  40404040 40404040

+000280  Time 21.41.57.595690      Date 2001.08.26      Thread ID... 8000000000000000
+000290  Member ID.... 03      Flags..... 000000      Entry Type.... 00000002
+000298  4C60604D F1F2F45D 40D9F1F5 7EF2F4C2 F6C4F8C5 F840C5D9 D9D5D67E F0F0F0F0 |<--(124) R15=24B6D8E8 ERRNO=0000
+0002B8  F0F0F0F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |0000.....
+0002D8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0002F8  00000000 00000000

+000300  Time 21.41.57.595743      Date 2001.08.26      Thread ID... 8000000000000000
+000310  Member ID.... 03      Flags..... 000000      Entry Type.... 00000001
+000318  8785A394 9684856D 86999694 6D86844D 8995A35D 40404040 40404040 40404040 |getmode_from_fd(int)
+000338  60606E4D F1F9F35D 406D6D87 85A38382 4D5D4040 40404040 40404040 40404040 |-->(193) __getc()
+000358  40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000378  40404040 40404040

+000380  Time 21.41.57.595746      Date 2001.08.26      Thread ID... 8000000000000000
+000390  Member ID.... 03      Flags..... 000000      Entry Type.... 00000002
+000398  4C60604D F1F9F35D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(193) R15=00000000 ERRNO=0000
+0003B8  F0F0F0F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |0000.....
+0003D8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0003F8  00000000 00000000
:
:

```

Figure 21. Trace table in dump output

Part 2. Debugging language-specific routines

This part provides specific information for debugging applications written in C/C++, COBOL, Fortran, and PL/I. It also discusses techniques for debugging under CICS.

Chapter 4. Debugging C/C++ routines

This chapter provides specific information to help you debug applications that contain one or more C/C++ routines. It also provides information about debugging C/C++ applications compiled with XPLINK. It includes the following topics:

- Debugging C/C++ I/O routines
- Using C/C++ compiler listings
- Generating a Language Environment dump of a C/C++ routine
- Generating a Language Environment dump of a C/C++ routine with XPLINK
- Finding C/C++ information in a Language Environment dump
- Debugging example of C/C++ routines
- Debugging example of C/C++ routines with XPLINK

There are several debugging features that are unique to C/C++ routines. Before examining the C/C++ techniques to find errors, you might want to consider the following areas of potential problems:

- If you suspect that you are using uninitialized storage, you may want to use the STORAGE run-time option.
- If you are using the `fetch()` function, refer to *z/OS XL C/C++ Programming Guide* to ensure that you are creating the fetchable module correctly.
- If you are using DLLs, refer to *z/OS XL C/C++ Programming Guide* to ensure that you are using the DLL correctly.
- For non-System Programming C routines, ensure that the entry point of the load module is CEESTART.
- You should avoid:
 - Incorrect casting
 - Referencing an array element with a subscript outside the declared bounds
 - Copying a string to a target with a shorter length than the source string
 - Declaring but not initializing a pointer variable, or using a pointer to allocated storage that has already been freed

If a routine exception occurred and you need more information than the condition handler provided, run your routine with the following run-time options, TRAP(ON, NOSPIE) and TERMTHDACT(UAIMM). Setting these run-time options generates a system dump of the user address space of the original abend or program interrupt prior to the Language Environment condition manager processing the condition. After the system dump is taken by the operating system the Language Environment condition manager continues processing.

Debugging C/C++ input/output programs

You can use C/C++ conventions such as `__amrc` and `perror()` when you debug I/O operations.

Using the `__amrc` and `__amrc2` structures

`__amrc`, a structure defined in **stdio.h**, can help you determine the cause of errors resulting from an I/O operation, because it contains diagnostic information (for example, the return code from a failed VSAM operation).

There are two structures:

- `__amrc` (defined by type `__amrc_type`)
- `__amrc2` (defined by type `__amrc2_type`)

The `__amrc2_type` structure contains secondary information that C can provide.

Because any I/O function calls, such as `printf()`, can change the value of `__amrc` or `__amrc2`, make sure you save the contents into temporary structures of `__amrc_type` and `__amrc2_type` respectively, before dumping them.

Figure 22 shows the structure as it appears in **stdio.h**.

```
typedef struct __amrc_type {  
[1]     union {  
[2]         long int __error;  
           struct {  
               unsigned short __syscode,  
                               __rc;  
[3]         } __abend;  
           struct {  
               unsigned char __fdbk_fill,  
                               __rc,  
                               __ftnacd,  
                               __fdbk;  
[4]         } __feedback;  
           struct {  
               unsigned short __svc99_info,  
                               __svc99_error;  
[5]         } __alloc;  
[1]     } __code;  
[6]     unsigned long __RBA;  
[7]     unsigned int __last_op;  
           struct {  
               unsigned long __len_fill; /* __len + 4      */  
               unsigned long __len;  
               char __str[120];  
               unsigned long __parmr0;  
               unsigned long __parmr1;  
               unsigned long __fill2[2];  
               char __str2[64];  
[8]     } __msg;  
    } __amrc_type;  
}
```

Figure 22. `__amrc` structure

Figure 23 shows the `__amrc2` structure as it appears in **stdio.h**.

```
struct {  
[9]     long int __error2;  
           char __pad__error2[4];  
[10]    FILE *__fileptr;  
[11]    long int __reserved[6];  
}
```

Figure 23. `__amrc2` structure

- [1] **__code** The error or warning value from an I/O operation is in **__error**, **__abend**, **__feedback**, or **__alloc**. Look at **__last_op** to determine how to interpret the **__code** union.
- [2] **__error** A structure that contains error codes for certain macros or services your application uses. Look at **__last_op** to determine the error codes. **__syscode** is the system abend code.
- [3] **__abend** A structure that contains the abend code when **errno** is set to indicate a recoverable I/O abend. **__rc** is the return code. For more information on abend codes, see *z/OS MVS System Codes*.
- [4] **__feedback** A structure that is used for VSAM only. The **__rc** stores the VSAM register 15, **__fdbk** stores the VSAM error code or reason code, and **__RBA** stores the RBA after some operations.
- [5] **__alloc** A structure that contains errors during **fopen** or **freopen** calls when defining files to the system using SVC 99.
- [6] **__RBA** The RBA value returned by VSAM after an ESDS or KSDS record is written out. For an RRDS, it is the calculated value from the record number. It can be used in subsequent calls to **flocate**.
- [7] **__last_op** A field containing a value that indicates the last I/O operation being performed by C/C++ at the time the error occurred. These values are shown in Table 13.
- [8] **__msg** May contain the system error messages from read or write operations emitted from the DFSMS/MVS[®] SYNADAF macro instruction. Because the message can start with a hexadecimal address followed by a short integer, it is advisable to start printing at **MSG+6** or greater so the message can be printed as a string. Because the message is not null-terminated, a maximum of 114 characters should be printed. This can be accomplished by specifying a **printf** format specifier as **%.114s**.
- [9] **__error2** A secondary error code. For example, an unsuccessful rename or remove operation places its reason code here.
- [10] **__fileptr** A pointer to the file that caused a **SIGIOERR** to be raised. Use an **fldata()** call to get the actual name of the file.
- [11] **__reserved** Reserved for future use.

__last_op values

The **__last_op** field is the most important of the **__amrc** fields. It defines the last I/O operation C/C++ was performing at the time of the I/O error. You should note that the structure is neither cleared nor set by non-I/O operations, so querying this field outside of a **SIGIOERR** handler should only be done immediately after I/O operations. Table 13 lists **__last_op** values you could receive and where to look for further information.

Table 13. __last_op values and diagnosis information

Value	Further Information
__IO_INIT	Will never be seen by SIGIOERR exit value given at initialization.
__BSAM_OPEN	Sets __error with return code from OS OPEN macro.
__BSAM_CLOSE	Sets __error with return code from OS CLOSE macro.

Table 13. *__last_op* values and diagnosis information (continued)

Value	Further Information
<code>__BSAM_READ</code>	No return code (either <code>__abend</code> (errno == 92) or <code>__msg</code> (errno == 66) filled in).
<code>__BSAM_NOTE</code>	NOTE returned 0 unexpectedly, no return code.
<code>__BSAM_POINT</code>	This will not appear as an error lastop.
<code>__BSAM_WRITE</code>	No return code (either <code>__abend</code> (errno == 92) or <code>__msg</code> (errno == 65) filled in).
<code>__BSAM_CLOSE_T</code>	Sets <code>__error</code> with return code from OS CLOSE TYPE=T.
<code>__BSAM_BLDL</code>	Sets <code>__error</code> with return code from OS BLDL macro.
<code>__BSAM_STOW</code>	Sets <code>__error</code> with return code from OS STOW macro.
<code>__TGET_READ</code>	Sets <code>__error</code> with return code from TSO TGET macro.
<code>__TPUT_WRITE</code>	Sets <code>__error</code> with return code from TSO TPUT macro.
<code>__IO_DEVTYPE</code>	Sets <code>__error</code> with return code from I/O DEVTYPE macro.
<code>__IO_RDJFCB</code>	Sets <code>__error</code> with return code from I/O RDJFCB macro.
<code>__IO_TRKCALC</code>	Sets <code>__error</code> with return code from I/O TRKCALC macro.
<code>__IO_OBTAIN</code>	Sets <code>__error</code> with return code from I/O CAMLST OBTAIN.
<code>__IO_LOCATE</code>	Sets <code>__error</code> with return code from I/O CAMLST LOCATE.
<code>__IO_CATALOG</code>	Sets <code>__error</code> with return code from I/O CAMLST CAT. The associated macro is CATALOG.
<code>__IO_UNCATALOG</code>	Sets <code>__error</code> with return code from I/O CAMLST UNCAT. The associated macro is CATALOG.
<code>__IO_RENAME</code>	Sets <code>__error</code> with return code from I/O CAMLST RENAME.
<code>__SVC99_ALLOC</code>	Sets <code>__alloc</code> structure with info and error codes from SVC 99 allocation.
<code>__SVC99_ALLOC_NEW</code>	Sets <code>__alloc</code> structure with info and error codes from SVC 99 allocation of NEW file.
<code>__SVC99_UNALLOC</code>	Sets <code>__unalloc</code> structure with info and error codes from SVC 99 unallocation.
<code>__C_TRUNCATE</code>	Set when C or C++ truncates output data. Usually this is data written to a text file with no newline such that the record fills up to capacity and subsequent characters cannot be written. For a record I/O file this refers to an <code>fwrite()</code> writing more data than the record can hold. Truncation is always rightmost data. There is no return code.
<code>__C_FCBCHECK</code>	Set when C or C++ FCB is corrupted. This is due to a pointer corruption somewhere. File cannot be used after this.
<code>__C_DBCS_TRUNCATE</code>	This occurs when writing DBCS data to a text file and there is no room left in a physical record for anymore double byte characters. A new-line is not acceptable at this point. Truncation will continue to occur until an SI is written or the file position is moved. Cannot happen if <code>MB_CUR_MAX</code> is 1.
<code>__C_DBCS_SO_TRUNCATE</code>	This occurs when there is not enough room in a record to start any DBCS string or else when a redundant SO is written to the file before an SI. Cannot happen if <code>MB_CUR_MAX</code> is 1.
<code>__C_DBCS_SI_TRUNCATE</code>	This occurs only when there was not enough room to start a DBCS string and data was written anyways, with an SI to end it. Cannot happen if <code>MB_CUR_MAX</code> is 1.

Table 13. *__last_op* values and diagnosis information (continued)

Value	Further Information
<code>__C_DBCS_UNEVEN</code>	This occurs when an SI is written before the last double byte character is completed, thereby forcing C or C++ to fill in the last byte of the DBCS string with a padding byte X'FE'. Cannot happen if MB_CUR_MAX is 1.
<code>__C_CANNOT_EXTEND</code>	This occurs when an attempt is made to extend a file that allows writing, but cannot be extended. Typically this is a member of a partitioned data set being opened for update.
<code>__VSAM_OPEN_FAIL</code>	Set when a low level VSAM OPEN fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_OPEN_ESDS</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_OPEN_RRDS</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_OPEN_KSDS</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_OPEN_ESDS_PATH</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_OPEN_KSDS_PATH</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_MODCB</code>	Set when a low level VSAM MODCB macro fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_TESTCB</code>	Set when a low level VSAM TESTCB macro fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_SHOWCB</code>	Set when a low level VSAM SHOWCB macro fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_GENCB</code>	Set when a low level VSAM GENCB macro fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_GET</code>	Set when the last op was a low level VSAM GET; if the GET fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_PUT</code>	Set when the last op was a low level VSAM PUT; if the PUT fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_POINT</code>	Set when the last op was a low level VSAM POINT; if the POINT fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_ERASE</code>	Set when the last op was a low level VSAM ERASE; if the ERASE fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_ENDREQ</code>	Set when the last op was a low level VSAM ENDREQ; if the ENDREQ fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_CLOSE</code>	Set when the last op was a low level VSAM CLOSE; if the CLOSE fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__QSAM_GET</code>	<code>__error</code> is not set (if <code>abend (errno == 92)</code> , <code>__abend</code> is set, otherwise if read error (<code>errno == 66</code>), look at <code>__msg</code> .
<code>__QSAM_PUT</code>	<code>__error</code> is not set (if <code>abend (errno == 92)</code> , <code>__abend</code> is set, otherwise if write error (<code>errno == 65</code>), look at <code>__msg</code> .
<code>__QSAM_TRUNC</code>	This is an intermediate operation. You will only see this if an I/O abend occurred.
<code>__QSAM_FREEPOL</code>	This is an intermediate operation. You will only see this if an I/O abend occurred.
<code>__QSAM_CLOSE</code>	Sets <code>__error</code> to result of OS CLOSE macro.

Table 13. *__last_op* values and diagnosis information (continued)

Value	Further Information
<code>__QSAM_OPEN</code>	Sets <code>__error</code> to result of OS OPEN macro.
<code>__CMS_OPEN</code>	Sets <code>__error</code> to result of FSOPEN.
<code>__CMS_CLOSE</code>	Sets <code>__error</code> to result of FSCLOSE.
<code>__CMS_READ</code>	Sets <code>__error</code> to result of FSREAD.
<code>__CMS_WRITE</code>	Sets <code>__error</code> to result of FSWRITE.
<code>__CMS_STATE</code>	Sets <code>__error</code> to result of FSSTATE.
<code>__CMS_ERASE</code>	Sets <code>__error</code> to result of FSERASE.
<code>__CMS_RENAME</code>	Sets <code>__error</code> to result of CMS RENAME command.
<code>__CMS_EXTRACT</code>	Sets <code>__error</code> to result of DMS EXTRACT call.
<code>__CMS_LINERD</code>	Sets <code>__error</code> to result of LINERD macro.
<code>__CMS_LINEWRT</code>	Sets <code>__error</code> to result of LINEWRT macro.
<code>__CMS_QUERY</code>	<code>__error</code> is not set.
<code>__HSP_CREATE</code>	Indicates last op was a DSPSERV CREATE to create a hiperspace for a hiperspace memory file. If CREATE fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__HSP_DELETE</code>	Indicates last op was a DSPSERV DELETE to delete a hiperspace for a hiperspace memory file during termination. If DELETE fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__HSP_READ</code>	Indicates last op was a HSPSERV READ from a hiperspace. If READ fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__HSP_WRITE</code>	Indicates last op was a HSPSERV WRITE to a hiperspace. If WRITE fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__HSP_EXTEND</code>	Indicates last op was a HSPSERV EXTEND during a write to a hiperspace. If EXTEND fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__CICS_WRITEQ_TD</code>	Sets <code>__error</code> with error code from EXEC CICS WRITEQ TD.
<code>__LFS_OPEN</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
<code>__LFS_CLOSE</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
<code>__LFS_READ</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .

Table 13. `__last_op` values and diagnosis information (continued)

Value	Further Information
<code>__LFS_WRITE</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
<code>__LFS_LSEEK</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
<code>__LFS_FSTAT</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .

Displaying an error message with the `perror()` function

To find a failing routine, check the return code of all function calls. After you have found the failing routine, use the `perror()` function after the routine to display the error message. `perror()` displays the string that you pass to it and an error message corresponding to the value of `errno`. `perror()` writes to the standard error stream (`stderr`).

If you need additional diagnostic information set the environment variable, `_EDC_ADD_ERRNO2` to 1, and that will append the current `errno2` value to the end of the `perror()` string.

Figure 24 is an example of a routine using `perror()`.

```
#include <stdio.h>
int main(void){
    FILE *fp;

    fp = fopen("myfile.dat", "w");
    if (fp == NULL)
        perror("fopen error");
}
```

Figure 24. Example of a routine using `perror()`

Using `__errno2()` to diagnose application problems

Use `__errno2()` when diagnosing problems in a z/OS UNIX or an OpenExtensions application. This function enables C/C++ application programs to access diagnostic information returned to the C/C++ run-time library from an underlying kernel callable service. `__errno2()` returns the reason code of the last failing kernel callable service called by the C/C++ run-time library. The returned value is intended for diagnostic display purposes only. The function call is always successful.

Note: Since the `__errno2()` function returns the reason code of the kernel callable service that last failed, and not all function calls invoke the kernel, the value returned by `__errno2()` may be misleading.

Figure 25 is an example of a routine using `__errno2()`.

```
#include <stdio.h>
#include <errno.h>
FILE *myfopen(const char *fn, const char *mode) {
    FILE *f;
    f = fopen(fn,mode);
    if (f==NULL) {
        perror("fopen() failed");
        printf("__errno2 = %08x\n", __errno2());
    }
    return(f);
}
```

Figure 25. Example of a routine using `__errno2()`

Figure 26 is an example of a routine using the environment variable `_EDC_ADD_ERRNO2`, and Figure 27 shows the sample output from that routine.

```
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int main(void) {
    FILE *fp;

    /* add errno2 to perror message */
    setenv("_EDC_ADD_ERRNO2","1",1);

    fp = fopen("testfile.dat", "r");
    if (fp == NULL)
        perror("fopen error");

}
```

Figure 26. Example of a routine using `_EDC_ADD_ERRNO2`

```
fopen error: EDC5129I  No such file or directory.
(errno2=0x05620062)
```

Figure 27. Sample output of a routine using `_EDC_ADD_ERRNO2`

Using C/C++ listings

The following sections discuss C/C++ listings generated when the executable program is created. They also explain how to use these listings to locate information, such as variable values and the timestamp, in the dump.

Generating C/C++ listings and maps

The two techniques for creating an executable program are:

- When the executable program is to be stored in a PDSE or HFS, use the binder to combine the output from the XL C/C++ compiler.
- When the executable program is to be stored in a PDS, use Language Environment Prelinker Utility to combine the output from the XL C/C++ compiler and pass the prelinker output to the binder.

Note: Executable programs using XPLINK can only be created by using the binder.

The listings and maps created by the compile, prelink (optional), and link-edit steps provide many pieces of information necessary for performing problem analysis tasks. When creating an executable program without using the prelink step, the map of the Writable Static Area (WSA) is provided by the binder in the output listing in the C_WSA section.

In addition, the @STATIC is replaced by the binder with \$PRIVnnnnnn and to find the source listing use the cross reference to associate \$PRIVnnnnnn with the defining section name and use the section name to find the source in the module map. So, the output listing provided by the binder should be used when locating variables in executable programs created without using the prelink step.

When you are debugging, you can use various options depending upon which compiler you are using. The following section provides an overview of each listing and specifies the compiler option to use. For a detailed description of available listings, see *z/OS XL C/C++ User's Guide*.

Table 14. Contents of listing and associated compiler options

Name	Compiler Option	Function
Pseudo-assembler listing	LIST	Generates a pseudo-assembler listing, which shows the source listing for the current routine.
Storage Offset Listing	XREF	Produces a storage offset listing, which includes in the source listing a cross reference table of names used in the routine and the line numbers on which they were declared or referenced, and a static map.
Structure Map	AGGREGATE (C only)	Causes a structure map to be included in the source listing. The structure map shows the layout of variables for the type struct or union.
Inline Report	INLRPT and INLINE(,REPORT,,)	Generates an inline report that summarizes all functions inlined and provides a detailed call structure of all the functions.
Prelinker Map	MAP (prelink option)	Creates the prelinker map when invoking the Prelinker. It is the default. You can use prelinker maps to determine the location of static and external C variables compiled with the RENT option and all C++ variables.
Link-edit Output Listing	MAP, LIST, XREF (linker option)	These options control the listing output from the link-edit process.
Source Listing	SOURCE	Generates the source listing, which contains the original source input statements and any compiler diagnostic messages issued for the source.
Cross-Reference Listing	XREF	Cross-reference table containing a list of the identifiers from the source program and the line numbers in which they appear.
External Symbol Cross Reference Listing	ATTR (C only) or XREF	Shows the original name and corresponding mangled name for each symbol.

Table 14. Contents of listing and associated compiler options (continued)

Name	Compiler Option	Function
Object File Map	IPA(MAP)	Displays the names of the object files that were used as input to the IPA Link step.
Source File Map	IPA(MAP)	Identifies the source files included in the object files.
Compiler Options Map	IPA(MAP)	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.
Global Symbols Map	IPA(MAP)	Shows how global symbols are mapped into members of global data structures by the global variable coalescing optimization process. This section is only generated if the IPA Link phase coalesces global variables.
Inline Report for IPA Inliner	INLRPT and INLINE(,REPORT,,)	Describes the actions performed by the IPA Inliner. INLRPT and/or INLINE(,REPORT,,) must be passed to the IPA link phase as a suboption of -WI,I. For example, -WI,I,INLRPT.

C, C++, and C/C++ IPA listings

The options for each listing vary depending upon which XL C compiler is used. The -V option will cause all listing sections to be output. The following section illustrates which options are available for each listing.

C compiler listings: The following table specifies which listings are available for the XL C compiler, and which option(s) must be specified to obtain it.

Table 15. XL C compiler listings

Name	Compiler Option
Source Program	SOURCE
Cross-Reference Listing	XREF
Structure and Union Maps	AGGREGATE
Inline Report	OPTIMIZE and INLINE(,REPORT,,) or INLRPT
Pseudo Assembly Listing	LIST
Storage Offset Listing	XREF

XL C++ compiler listings: The following table specifies which listings are available for the XL C++ compiler, and which option(s) must be specified to obtain it.

Table 16. XL C++ compiler listings

Name	Compiler Option
Source Program	SOURCE
Cross-Reference Listing	ATTR and XREF
Inline Report	INLINE(,REPORT,,) or INLRPT
Pseudo Assembly Listing	LIST
External Symbol Cross Reference Listing	ATTR or XREF

XL C/C++ IPA link step listings: The following table specifies which listings are available for the XL C/C++ IPA Link Step, and which option(s) must be specified to obtain it.

Table 17. XL C/C++ IPA link step listings

Name	Compiler Option
Object File Map	IPA (MAP)
Source File Map	IPA (MAP)
Compiler Options Map	IPA (MAP)
Global Symbols Map	IPA (MAP)
Inline Report for IPA Inliner	Inline Report INLINE(,REPORT,,) or INLRPT)
Partition Map	IPA (MAP)

Finding variables

You can determine the value of a variable in the routine at the point of interrupt by using the compiled code listing as a guide to its address, then finding this address in the Language Environment dump. The method you use depends on the storage class of variable.

This method is generally used when no symbolic variables have been dumped (by using the TEST compiler option).

It is possible for the routine to be interrupted before the value of the variable is placed in the location provided for it. This can explain unexpected values in the dump.

Steps for finding automatic variables

Perform the following steps to find automatic variables in the Language Environment dump:

1. Identify the start of the stack frame. If a dump has been taken, each stack frame is dumped. The stack frames can be cross-referenced to the function name in the traceback.
2. Determine the value of the base register (in this example, GPR13) in the Saved Registers section for the function you are interested in.
3. Find the offset of the variable (which is given in decimal) in the storage offset listing.

Example:

```
aa1    85-0:85    Class = automatic,  Offset = 164(r13),    Length = 40
```

4. Add this base address to the offset of the variable.

When you are done, the contents of the variable can be read in the DSA Frame section corresponding to the function the variable is contained in.

Locating the Writable Static Area (WSA)

The Writable Static Area (WSA) address is the base address of the writable static area which is available for all C and C++ programs except C programs compiled with the NORENT compiler option. If you have C code compiled with the RENT option or C++ code (hereafter called RENT code) you must determine the base address of the WSA if you want to calculate the address of a static or external variable. Use the following table to determine where to find the WSA base address:

Table 18. Finding the WSA base address

If you want the WSA base address for:	Locate the WSA base address in:
application code	the WSA address field in the Enclave Control Blocks section
a fetched module	the WSA address field of the Fetch() Information section for the fetch() function pointer for which you are interested
a DLL	the corresponding WSA address in the DLL Information section

Use the WSA base address to locate the WSA in the Enclave Storage section.

Steps for finding the static storage area

If you have C code compiled with the NORENT option (hereafter called NORENT code) you must determine the base address of the static storage area if you want to calculate the address of a static or external variable.

Perform the following steps to find the static storage area:

1. Name the static storage area CSECT by using the pragma csect directive. Once this is done, a CSECT is generated for the static storage area for each source file.
2. Determine the origin and length of the CSECT from the linker map.
3. Locate the external variables corresponding to the CSECT with the same name.
4. Determine the origin and length of the external variable CSECT from the linker map.

Notes:

1. Address calculation for static and external variables uses the static storage area as a base address with 1 or more offsets added to this address.
2. The storage associated with these CSECTs is not dumped when an exception occurs. It is dumped when cdump or CEE3DMP is called, but it is written to a separate ddname called CEESNAP. For information about cdump, CEE3DMP, and enabling the CEESNAP ddname, see “Generating a Language Environment dump of a C/C++ routine” on page 145.

Steps for finding RENT static variables

Before you begin: You need to know the WSA. To find this information, see “Locating the Writable Static Area (WSA)” on page 137. For this procedure’s example, the address of writable static is X'02D66E40'.

Perform the following steps to find RENT static variables:

1. Find the offset of @STATIC (associated with the file where the static variable is located) in the Writable Static Map section of the prelinker map.

Example:

Writable Static Map			
OFFSET	LENGTH	FILE ID	INPUT NAME
0	1	00001	DFHC0011
4	1	00001	DFHC0010
8	2	00001	DFHDUMMY
C	2	00001	DFHB0025
10	2	00001	DFHB0024
14	2	00001	DFHB0023
18	2	00001	DFHB0022
1C	2	00001	DFHB0021
20	2	00001	DFHB0020
24	2	00001	DFHEIB0
28	4	00001	DFHEIPTR
2C	4	00001	DFHCP011
30	4	00001	DFHCP010
34	4	00001	DFHBP025
38	4	00001	DFHBP024
3C	4	00001	DFHBP023
40	4	00001	DFHBP022
44	4	00001	DFHBP021
48	4	00001	DFHBP020
4C	4	00001	DFHEICB
50	4	00001	DFHEID0
54	4	00001	DFHLDVER
58	278	00001	@STATIC
720	30	00002	@STATIC

Figure 28. Writable static map produced by prelinker

In this Writable Static Map section of a prelinker map the offset is X'58'.

2. Add the offset to the WSA to get the base address of static variables.

Example: X'02D66E40' + X'58' = X'2D66E98'

3. Find the offset of the static variable in the partial storage offset compiler listing.

Example:

sa0 66-0:66 Class = static, **Location = WSA + @STATIC + 96**, Length = 4

The offset is 96 (X'60').

4. Add the offset of the static variable in the partial storage offset compiler listing (found in step 3) to the base address of static variables (calculated in step 2).

Example: X'2D66E98' + X'60' = X'2D66EF8'

When you are done, you have the address of the value of the static variable in the Language Environment dump.

Figure 29 on page 140 shows the path to locate RENT C++ and C static variables by adding the address of writable static, the offset of @STATIC, and the variable offset.

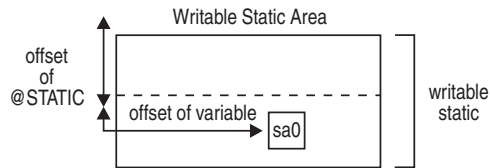


Figure 29. Location of RENT static variable in storage

Steps for finding external RENT variables

Before you begin: You need to know the WSA. To find this information see “Locating the Writable Static Area (WSA)” on page 137. For this procedure’s example, the address of writable static is X'02D66E40'.

Perform the following steps to find external RENT variables:

1. Find the offset of the external variable in the Prelinker Writable Static Map.

Example:

In this example, the offset for DFHEIPTR is X'28'.

Writable Static Map			
OFFSET	LENGTH	FILE ID	INPUT NAME
0	1	00001	DFHC0011
4	1	00001	DFHC0010
8	2	00001	DFHDUMMY
C	2	00001	DFHB0025
10	2	00001	DFHB0024
14	2	00001	DFHB0023
18	2	00001	DFHB0022
1C	2	00001	DFHB0021
20	2	00001	DFHB0020
24	2	00001	DFHEIB0
28	4	00001	DFHEIPTR
2C	4	00001	DFHCP011
30	4	00001	DFHCP010
34	4	00001	DFHBP025
38	4	00001	DFHBP024
3C	4	00001	DFHBP023
40	4	00001	DFHBP022
44	4	00001	DFHBP021
48	4	00001	DFHBP020
4C	4	00001	DFHEICB
50	4	00001	DFHEID0
54	4	00001	DFHLDVER
58	420	00001	@STATIC

Figure 30. Writable static map produced by prelinker

2. Add the offset of the external variable to the address of writable static.

Example: X'02D66E40' + X'28' = X'2D66E68'

When you are done, you have the address of the value of the external variable in the Language Environment dump.

Steps for finding NORENT static variables

Before you begin: You need to know the name and address of the static storage area. To find this information see “Steps for finding the static storage area” on page 138. For this procedure’s example, the static storage area is called **STATSTOR** and has an address of X'02D66E40'.

Perform the following steps to find external RENT variables:

1. Find the offset of the static variable in the partial storage offset compiler listing.

Example:

```
sa0    66-0:66    Class = static,    Location = STATSTOR +96,    Length = 4
```

The offset is 96 (X'60').

2. Add the offset to the base address of static variables.

Example: X'2D66E40' + X'60' = X'2D66EA0'

When you are done, you have the address of the value of the static variable in the Language Environment dump.

Figure 31 shows how to locate NORENT C static variables by adding the Static Storage Area CSECT address to the variable offset.

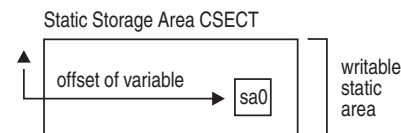


Figure 31. Location of NORENT static variable in storage

Steps for finding external NORENT variables

Before you begin: You need to find the address of the external variable CSECT. To find this information, see “Steps for finding the static storage area” on page 138. For this procedure’s example, the address of the external variable CSECT is X'02D66E40'.

The address of the external variable CSECT is the address of the value of the external variable in the Language Environment dump.

Steps for finding the C/370 parameter list

Perform the following steps to locate a parameter in the Language Environment dump:

1. Identify the address of the start of the parameter list. A pointer to the parameter list is passed to the called function in register 1. This is the address of the start of the parameter list. Figure 32 on page 142 shows an example code for the parameter variable.

Example:

```

func0() {
    :
    func1(a1,a2);
    :
}

func1(int ppx, int pp0) {
    :
}

```

Figure 32. Example code for parameter variable

Parameters *ppx* and *pp0* correspond to copies of *a1* and *a2* in the stack frame belonging to *func0*.

-
2. Use the address of the start of the parameter list to find the register and offset in the partial storage offset listing.

Example:

```
pp0      62-0:62      Class = parameter,      Location = 4(r1),      Length = 4
```

The offset is 4 (X'4') from register 1.

3. Determine the value of GPR1 in the Saved Registers section for the function that called the function you are interested in.
4. Add this base address to the offset of the parameter.

When you are done, the contents of the variable can then be read in the DSA frame section corresponding to the function the parameter was passed from.

Steps for finding the C++ parameter list

Before you begin: To locate C++ functions with extern C attributes, see “Steps for finding the C/370 parameter list” on page 141.

Perform the following steps to find the C++ parameter list:

1. Identify the address of the start of the parameter list. A pointer to the parameter list is passed to the called function in register 1. This is the address of the start of the parameter list. Figure 33 shows an example code for the parameter variable.

Example:

```

func0() {
    :
    func1(a1,a2);
    :
}

func1(int ppx, int pp0) {
    :
}

```

Figure 33. Example code for parameter variable

Parameters *ppx* and *pp0* correspond to copies of *a1* and *a2* in the stack frame belonging to *func1*.

2. Locate the value of the base register in the Saved Registers section of the function you are interested in.
3. Find the offset of the static variable in the partial storage offset compiler listing.

Example:

ppx	62-0:62	Class = parameter,	Location = 188(r13),	Length = 4
pp0	62-0:62	Class = parameter,	Location = 192(r13),	Length = 4

Figure 34. Partial storage offset listing

4. Add the value of the base register to the offset.
5. Locate the parameter.

Restriction: When OPTIMIZE is on, the parameter value might never be stored, since the first few parameters might be passed in registers and there might be no need to save them.

Steps for finding members of aggregates

You can define aggregates in any of the storage classes or pass them as parameters to a called function. The first step is to find the start of the aggregate. You can compute the start of the aggregate as described in previous sections, depending on the type of aggregate used.

The aggregate map provided for each declaration in a routine can further assist in finding the offset of a specific variable within an aggregate. Structure maps are generated using the AGGREGATE compiler option. Figure 35 shows an example of a static aggregate.

```
static struct {
    short int ss01;
    char      ss02[56];
    int       sz0[6];
    int       ss03;
} ss0;
```

Figure 35. Example code for structure variable

Figure 36 on page 144 shows an example aggregate map.

Aggregate map for: ss0		
Offset Bytes(Bits)	Length Bytes(Bits)	Member Name
0	2	ss01
2	56	ss02[56]
58	2	***PADDING***
60	24	sz0[6]
84	4	ss03

Figure 36. Example of aggregate map

Assume the structure has been compiled as RENT. To find the value of variable `sz0[0]`:

1. Find the address of the writable static. For this example the address of writable static is X'02D66E40'.
2. Find the offset of @STATIC in the Writable Static Map. In this example, the offset is X'58'. Add this offset to the address of writable static. The result is X'2D66E98' (X'02D66E40' + X'58'). Figure 37 shows the Writable Static Map produced by the prelinker.

Writable Static Map			
OFFSET	LENGTH	FILE ID	INPUT NAME
0	1	00001	DFHC0011
4	1	00001	DFHC0010
8	2	00001	DFHDUMMY
C	2	00001	DFHB0025
10	2	00001	DFHB0024
14	2	00001	DFHB0023
18	2	00001	DFHB0022
1C	2	00001	DFHB0021
20	2	00001	DFHB0020
24	2	00001	DFHEIB0
28	4	00001	DFHEIPTR
2C	4	00001	DFHCP011
30	4	00001	DFHCP010
34	4	00001	DFHBP025
38	4	00001	DFHBP024
3C	4	00001	DFHBP023
40	4	00001	DFHBP022
44	4	00001	DFHBP021
48	4	00001	DFHBP020
4C	4	00001	DFHEICB
50	4	00001	DFHEID0
54	4	00001	DFHLDVER
58	320	00001	@STATIC

Figure 37. Writable static map produced by prelinker

3. Find the offset of the static variable in the storage offset listing. The offset is 96 (X'60'). Following is an example of a partial storage offset listing.

```
ss0      66-0:66      Class = static,      Location = GPR13(96),      Length = 4
```

 Add this offset to the result from step 2. The result is X'2D66EF8' (X'2D66E98' + X'60'). This is the address of the value of the static variable in the dump.
4. Find the offset of sz0 in the Aggregate Map, shown in Figure 36 on page 144. The offset is 60.

Add the offset from the Aggregate Map to the address of the ss0 struct. The result is X'60' (X'3C' + X'60'). This is the address of the values of sz0 in the dump.

Finding the timestamp

The timestamp is in the compile unit block. The address for the compile unit block is located at eight bytes past the function entry point. The compile unit block is the same for all functions in the same compilation. The fourth word of the compile unit block points to the timestamp. The timestamp is 16 bytes long and has the following format:

```
YYYYMMDDHHMMSSSS
```

Generating a Language Environment dump of a C/C++ routine

You can use either the CEE3DMP callable service or the `cdump()`, `csnap()`, and `ctrace()` C/C++ functions to generate a Language Environment dump of C/C++ routines. These C/C++ functions call CEE3DMP with specific options.

`cdump()`

If your routine is running under z/OS or CICS, you can generate useful diagnostic information by using the `cdump()` function. `cdump()` produces a main storage dump with the activation stack. This is equivalent to calling CEE3DMP with the option string: TRACEBACK BLOCKS VARIABLES FILES STORAGE STACKFRAME(ALL) CONDITION ENTRY.

When `cdump()` is invoked from a user routine, the C/C++ library issues an OS SNAP macro to obtain a dump of virtual storage. The first invocation of `cdump()` results in a SNAP identifier of 0. For each successive invocation, the ID is increased by one to a maximum of 256, after which the ID is reset to 0.

The output of the dump is directed to the CEESNAP data set. The DD definition for CEESNAP is as follows:

```
//CEESNAP DD SYSOUT= *
```

If the data set is not defined, or is not usable for any reason, `cdump()` returns a failure code of 1. This occurs even if the call to CEE3DMP is successful.

If the SNAP is not successful, the CEE3DMP DUMP file displays the following message:

```
Snap was
unsuccessful
```

If the SNAP is successful, CEE3DMP displays this message:

```
Snap was
successful; snap ID = nnn
```

Where *nnn* corresponds to the SNAP identifier described above. An unsuccessful SNAP does not result in an incrementation of the identifier.

Because `cdump()` returns a code of 0 only if the SNAP was successful or 1 if it was unsuccessful, you cannot distinguish whether a failure of `cdump()` occurred in the call to CEE3DMP or SNAP. A return code of 0 is issued only if both SNAP and CEE3DMP are successful.

Support for SNAP dumps using the `_cdump` function is provided only under z/OS and z/VM. SNAP dumps are not supported under CICS; no SNAP is produced in this environment. A successful SNAP results in a large quantity of output. A routine calling `cdump()` under CICS receives a return code of 0 if the ensuing call to CEE3DMP is successful. In addition to a SNAP dump, a Language Environment formatted dump is also taken.

csnap()

The `csnap()` function produces a condensed storage dump. `csnap()` is equivalent to calling CEE3DMP with the option string: TRACEBACK FILES BLOCKS VARIABLES NOSTORAGE STACKFRAME(ALL) CONDITION ENTRY.

To use these functions, you must add `#include <ctest.h>` to your C/C++ code. The dump is directed to output *dumprname*, which is specified in a `//CEEDUMP DD` statement in MVS/JCL or a `FILEDEF CEEDUMP` command in z/VM.

`cdump()`, `csnap()`, and `ctrace()` all return a 1 code in the SPC environment because they are not supported in SPC.

Refer to the *z/OS XL C/C++ Run-Time Library Reference* for more details about the syntax of these functions.

ctrace()

The `ctrace()` function produces a traceback and includes the offset addresses from which the calls were made. `ctrace()` is equivalent to calling CEE3DMP with the option string: TRACEBACK NOFILES NOBLOCKS NOVARIABLES NOSTORAGE STACKFRAME(ALL) NOCONDITION NOENTRY.

Sample C routine that calls `cdump()`

Figure 38 on page 147 shows a sample C routine that uses the `cdump` function to generate a dump.

Figure 43 on page 151 shows the dump output.

```

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

void hsigfpe(int);
void hsigterm(int);
void atfl(void);

typedef int (*FuncPtr_T)(void);

int st1    = 99;
int st2    = 255;
int xcount = 0;

int main(void) {
    /*
     * 1) Open multiple files
     * 2) Register 2 signals
     * 3) Register 1 atexit function
     * 4) Fetch and execute a module
     */

    FuncPtr_T fetchPtr;
    FILE*     fp1;
    FILE*     fp2;
    int        rc;
    fp1 = fopen("myfile.data", "w");
    if (!fp1) {
        perror("Could not open myfile.data for write");
        exit(101);
    }

    fprintf(fp1, "record 1\n");
    fprintf(fp1, "record 2\n");
    fprintf(fp1, "record 3\n");

    fp2 = fopen("memory.data", "wb,type=memory");
    if (!fp2) {
        perror("Could not open memory.data for write");
        exit(102);
    }
    fprintf(fp2, "some data");
    fprintf(fp2, "some more data");
    fprintf(fp2, "even more data");

    signal(SIGFPE, hsigfpe);
    signal(SIGTERM, hsigterm);

    rc = atexit(atfl);
    if (rc) {
        fprintf(stderr, "Failed on registration of atexit function atfl\n");
        exit(103);
    }
}

```

Figure 38. Example C routine using `cdump()` to generate a dump (Part 1 of 2)

```

    fetchPtr = (FuncPtr_T) fetch("MODULE1");
    if (!fetchPtr) {
        fprintf(stderr, "Failed to fetch MODULE1\n");
        exit(104);
    }
    fetchPtr();
    return(0);
}

void hsigfpe(int sig) {
    ++st1;
    return;
}

void hsigterm(int sig) {
    ++st2;
    return;
}

void atf1() {
    ++xcount;
}

```

Figure 38. Example C routine using cdump() to generate a dump (Part 2 of 2)

Figure 39 shows a fetched C module:

```

#include <ctest.h>

#pragma linkage(func1, fetchable)
int func1(void) {
    cdump("This is a sample dump");
    return(0);
}

```

Figure 39. Fetched module for C routine

Sample C++ routine that generates a Language Environment dump

Figure 40 on page 149 shows a sample C++ routine that uses a protection exception to generate a dump.

```

#include <iostream.h>
#include <ctest.h>
#include "stack.h"

int main() {
    cout << "Program starting:\n";
    cerr << "Error report:\n";

    Stack<int> x;
    x.push(1);
    cout << "Top value on stack : " << x.pop() << '\n';
    cout << "Next value on stack: " << x.pop() << '\n';
    return(0);
}

```

Figure 40. Example C++ routine with protection exception generating a dump

Figure 41 shows the template file `stack.c`

```

#ifndef __STACK__
#include "stack.h"
#endif

template <class T> T Stack<T>::pop() {
    T value = head->value;
    head = head->next;

    return(value);
}

template <class T> void Stack<T>::push(T value) {
    Node* newNode = new Node;
    newNode->value = value;
    newNode->next = head;
    head = newNode;
}

```

Figure 41. Template file STACK.C

Figure 42 on page 150 shows the header file **`stack.h`**.

```

#ifndef __STACK
#define __STACK__
template <class T> class Stack {
public:
    Stack() {
        char* badPtr = 0; badPtr -= (0x01010101);
        head = (Node*) badPtr; /* head initialized to 0xFEFEFEFF */
    }
    T pop();
    void push(T);
private:
    struct Node {
        T value;
        struct Node* next;
    }* head;
};
#endif

```

Figure 42. Header file STACK.H

Sample Language Environment dump with C/C++-specific information

This sample dump was produced by compiling the routine in Figure 38 on page 147 with the TEST(SYM) compiler option, then running it. Notice the sequence of calls in the traceback section - EDCZMINV is the C-C++ management module that invokes main and @@FECBMODULE1 fetches the user-defined function func1, which in turn calls the library routine __cdump.

If source code is compiled with the GONUMBER or TEST compile option, statement numbers are shown in the traceback. If source code is compiled with the TEST(SYM) compile option, variables and their associated type and value are dumped out. For more information about C/C++-specific information contained in a dump, see “Finding C/C++ information in a Language Environment dump” on page 156.

```

CEE3DMP V1 R3.0: This is a sample dump

CEE3DMP called by program unit (entry point __cdump) at offset +00000184.

Snap was unsuccessful

Registers on Entry to CEE3DMP:

PM..... 0100
GPR0..... 00000000 GPR1..... 000305D0 GPR2..... 00030564 GPR3..... 8DD5CB06
GPR4..... 00000001 GPR5..... 00000015 GPR6..... 0DEB54D8 GPR7..... 00000001
GPR8..... 00000000 GPR9..... 0DEB5038 GPR10..... 8DB50310 GPR11..... 8DB50310
GPR12..... 00023890 GPR13..... 000304E0 GPR14..... 800260DE GPR15..... 8DB6C670
FPR0..... 4D000000 00060388 FPR2..... 00000000 00000000

FPR4..... 00000000 00000000 FPR6..... 00000000 00000000

:
:
Information for enclave main

Information for thread 8000000000000000

Registers on Entry to CEE3DMP:
PM..... 0100
GPR0..... 00000000 GPR1..... 000305D0 GPR2..... 00030564 GPR3..... 8DD5CB
GPR4..... 00000001 GPR5..... 00000015 GPR6..... 0DEB54D8 GPR7..... 000000
GPR8..... 00000000 GPR9..... 0DEB5038 GPR10..... 8DB50310 GPR11..... 8DB503
GPR12..... 00023890 GPR13..... 000304E0 GPR14..... 800260DE GPR15..... 8DB6C6
FPR0..... 4D000000 00060388 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000

:
:
Traceback:
DSA Addr Program Unit PU Addr PU Offset Entry E Addr E Offset Statement Load Mod Service Status
000304E0 00030440 POSIX.CRTL.C(MODULE1) 0DD5CAB8 +00000184 __cdump 0DD5CAB8 +00000184 CEEEV003 Call
00030350 0DB50310 +0000006E func1 0DB50310 +0000006E 5 MODULE1 Call
00030298 0DEB54F8 -0DEB54F3 @@FECBMODULE1 0DEB54F8 -0DEB54F3 Call
000301E0 0DCB4AE8 +0000001A @@GETFN 0DCB4A40 +000000C2 CEEEV003 Call
000300C8 0DB51078 +000000392 main 0DB51078 +000000392 64 CSAMPLE Call
00030018 0DC626EE +000000B4 EDCZMINV 0DC626EE +000000B4 CEEEV003 Call
0001B898 0001B898 +0000013C CEEBBEXT 0001B898 +0000013C CEEBINIT Call
Parameters, Registers, and Variables for Active Routines:

:
:
main (DSA address 000301E0):

Saved Registers:

GPR0..... 0DEB5330 GPR1..... 8DC9EE8A GPR2..... 8DC627A2 GPR3..... 8DB510C6
GPR4..... 8001B97C GPR5..... 0DEB5098 GPR6..... 0DEB5330 GPR7..... 0DB523DC
GPR8..... 00000001 GPR9..... 80000000 GPR10..... 8DC626E2 GPR11..... 8001B898
GPR12..... 00023890 GPR13..... 000301E0 GPR14..... 8DB5140C GPR15..... 0DCB4A40

:
:
Local Variables:

fetchPtr signed int (*) (void)
0xDEB5330

fp2 struct __ffile * 0xDEBEA1C
fp1 struct __ffile * 0xDEBD024
rc signed int 0

```

Figure 43. Example dump from sample C routine (Part 1 of 6)

[1] Storage for Active Routines:

Control Blocks for Active Routines:

```

:
:
DSA for func1: 00030440
+000000  FLAGS.... 1002      member... 47D0      BKC..... 00030350  FWC..... F7F20000  R14..... 8DB50380
+000010  R15..... 0DD5CAB8  R0..... 000304E0  R1..... 000304D8  R2..... 8DC5A732  R3..... 8DB5035E
+000024  R4..... 8001B97C  R5..... 0DEB54D8  R6..... 0DEB5330  R7..... 0DB523DC  R8..... 00000001
+000038  R9..... 0DB6E66E  R10..... 0DB6D66F  R11..... 8DB6C670  R12..... 00023890  reserved. 000247D0
+00004C  NAB..... 000304E0  PNAB..... 000304B0  reserved. 8DBFE208  0DB50B08  000304EC  00000000
+000064  reserved. 000303D4  reserved. 00030350  MODE..... 00030538  reserved. 00000000  00000000
+000078  reserved. 00000000  reserved. D3C5F140

:
:
DSA frame: 00030440
+000000 00030440 100247D0 00030350 F7F20000 8DB50380 0DD5CAB8 000304E0 000304D8 8DC5A732 | .....&72.....N.....Q.Ex.
+000010 00030460 8DB5035E 8001B97C 0DEB54D8 0DEB5330 0DB523DC 00000001 0DB6E66E 0DB6D66F | ...;...@...Q.....w>..0?
+000020 00030480 8DB6C670 00023890 000247D0 000304E0 000304B0 8DBFE208 0DB50B08 000304EC | ..F.....S.....
+000030 000304A0 00000000 000303D4 00030350 00030538 00000000 00000000 00000000 D3C5F140 | .....M...&.....LE1
+000040 000304C0 40404040 40404040 0DB68414 40404040 40404040 40404040 0DEB54D8 40404040 | ..d. ...Q

:
:
[2] Control Blocks Associated with the Thread:
CAA: 00023890
+000000 00023890 00000800 00000000 00030000 00050000 00000000 00000000 00000000 00000000 | .....
+000010 000238B0 00000000 00000000 00024A10 00000000 00000000 00000000 00000000 00000000 | .....
+000020 000238D0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000030 000238F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000040 00023910 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....

:
:
[2A] C/370 CAA information :
C-C++ Specific CTHD..... 0DB6740C
C-C++ Specific CEDB..... 0DB67DE4

C-C++ Specific Thread block: 0DB6740C
+000000 0DB6740C C3E3C8C4 00000308 0DB6740C 00000000 0DC3865E 00000000 00000000 00000000 | CTHD.....Cf;.....
+000010 0DB6742C 00000000 00000000 00000000 00000000 0DB677EC 00000000 00000000 00000000 | .....
+000020 0DB6744C 00000000 00000000 00000000 00024640 00024728 00000000 00000000 00000001 | .....
+000030 0DB6746C 00000001 00000000 00000000 00000000 00000000 00000000 0DEB54B8 0DEB54B4 | .....
+000040 0DB6748C 0DEB54B0 0DEB54AC 0DEB54A0 0DEB54A4 0DEB54C0 00000000 00000000 00000000 | .....u.....

:
:
C-C++ Specific EDB block: 0DB67DE4
+000000 0DB67DE4 C3C5C4C2 000004D0 0DB67DE4 0DB52B40 00030000 0DB691C4 0DB69514 0DB51078 | CEDB.....'U... ..jd..n....
+000010 0DB67E04 00000080 0DB50FD8 00000000 00000000 00000000 0DB682BC 0DC3DFB0 0DC3DE20 | .....Q.....b..C..C..
+000020 0DB67E24 0DEB54F8 00000001 00000000 0DEB5388 0DB67CB0 0DB67A58 0DB67B84 0DC95196 | ..8.....h..@.....#d.I.o
+000030 0DB67E44 00004650 0DB683CC 40400000 00000000 00100000 00000000 00000000 00000000 | ...&..c. ....

```

Figure 43. Example dump from sample C routine (Part 2 of 6)

```

[2B] errno value..... 0
memory file block chain.... 0DEBEBA0
open FCB chain..... 0DEBEA30
GTAB table..... 0DB6771C

[3] signal information :
SIGFPE :
function pointer... 0DB51D20   WSA address... 8DEB5038   function name... hsigfpe

SIGTERM :
function pointer... 0DB51E90   WSA address... 8DEB5038   function name... hsigterm

SIGOBJECT :
function pointer... 0DB67A58   WSA address... 0DB67B84   function name... (unknown)

Enclave variables:
*..C(CSAMPLE):>hsigterm
void ()                0xDB51E90
*..C(CSAMPLE):>hsigfpe
void ()                0xDB51D20
*..C(CSAMPLE):>xcount
signed int              0
*..C(CSAMPLE):>main
signed int (void)
0xDB51078
*..C(CSAMPLE):>atf1
void (void)            0xDB51FF8
*..C(CSAMPLE):>st2
signed int              255
*..C(CSAMPLE):>st1
signed int              99
*..C(MODULE1):>func1
signed int (void)
0xDB50310

Enclave Control Blocks:
EDB: 000228B0
+000000 000228B0 C3C5C5C5 C4C24040 C0000001 00023750 00022EF8 00000000 00000000 00000000 CEEEDB .....&...8.....
+000020 000228D0 00022D78 00022DA8 00025038 00022558 00000000 80021808 000229D0 00008000 .....y..&.....
+000040 000228F0 00000000 00000000 0000CFB0 00000000 00000000 00000000 0DB646F0 0DEB54C8 .....0...H
+000060 00022910 8001C8D8 00000000 0DB69864 00000000 00024AE0 00000000 0DC2A620 00023890 ..HQ.....q.....Bw.....
+000080 00022930 00000000 00000000 00000000 00000000 00000001 00000000 00008A08 008DA738 .....x
+0000A0 00022950 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000001 .....
MEML: 00023750
+000000 00023750 00000000 00000000 0DBBB0B8 00000000 00000000 00000000 0DBBB0B8 00000000 .....
+000020 00023770 00000000 00000000 0DBBB0B8 00000000 0DB67DE4 00000000 0DC2B6B8 00000000 .....U...B.....
+000040 00023790 00000000 00000000 0DBBB0B8 00000000 00000000 00000000 0DBBB0B8 00000000 .....
+000060 000237B0 - +00011F 0002386F same as above

[4] WSA address.....0DEB5038

[5] atexit information :
function pointer... 8DB51FF8   WSA address... 0DEB5038   function name... atf1

[6] fetch information :
fetch pointer : 0DEB54F8
function pointer... 8DB50310   WSA address... 0DEB5480

Enclave Storage:
Initial (User) Heap : 0DEB5000
+000000 0DEB5000 C8C1D5C3 00022D48 00022D48 00000000 0DEB5000 0DEB55D0 00008000 00007A30 HANC.....&.....
+000020 0DEB5020 0DEB5000 00000190 00000000 00000000 00000000 00000000 00000000 0DB67B98 ..&.....#q
+000040 0DEB5040 0DB67A6C 0DB67CC4 00000000 00000000 00000000 00000000 00000000 00000000 ..:%.@D.....
:
LE/370 Anywhere Heap : 0DEB1000
+000000 0DEB1000 C8C1D5C3 00022D78 00022D78 00022D78 0DEB1000 0DEB3C18 00004000 000013E8 HANC.....Y
+000020 0DEB1020 0DEB1000 00001008 00000000 00000000 00000000 00000000 00000000 00000000 .....
+000040 0DEB1040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 .....
:

```

Figure 43. Example dump from sample C routine (Part 3 of 6)

```

LE/370 Below Heap                                : 00050000
+000000 00050000 C8C1D5C3 00022DA8 00022DA8 00022DA8 80050000 000500A8 00002000 00001F58 |HANC...y...y...y.....y.....|
+000020 00050020 00050000 00000088 C3E2E3D2 00000000 00000000 00000000 00000001 00000002 00000068 |.....hCSTK.....|
+000040 00050040 04000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 00050060 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 00050080 - +001FFF 00051FFF                same as above
Additional Heap, heapid = 0DEB3BE4                : 0DB14000
+000000 0DB14000 C8C1D5C3 0DEB3BE4 0DEB3BE4 0DEB3BE4 0DB14000 0DB143D0 000003E8 00000018 |HANC...U...U...U.. ....Y....|
+000020 0DB14020 0DB14000 00000050 00000000 0DB503AC 0DB50310 0DB14098 0DEB54F8 46F11000 |.. ...&..... q...8.1..|
+000040 0DB14040 00000003 0DB50580 00020000 0DB14078 00000000 0DB50310 0DB50310 00000000 |.....|
+000060 0DB14060 0DB50568 0DB503C0 00000000 00000000 0DB14000 00000020 0015D7D6 E2C9E74B |.....POSIX..|
+000080 0DB14080 C3D9E3D3 4BC34DD4 D6C4E4D3 C5F15D00 0DB14000 00000088 0DB14028 00010000 |CRTL.C(MODULE1)... ..h.. ..|

:
File Status and Attributes:

[7] File Control Block: 0DEBEA30
+000000 0DEBEA30 0DEBED65 00000000 000003DB 0DEBEB00 0DEBEB20 00000000 0DEBEB50 00000011 |.....&....|
+000020 0DEBEA50 00000014 00000000 00000000 0DEBD038 00000000 0DEBEA30 00000000 00000000 |.....|
+000040 0DEBEA70 00000000 FFFFFFFF 00080055 0DEBEB70 0DEBEB44 0DC81810 0DC83A08 0DC83C88 |.....H...H...H.h|
+000060 0DEBEA90 0DC840F8 0DC71578 00000000 00000400 00000400 00000000 00000000 00000400 |.H 8.G.....|
+000080 0DEBEAB0 0DEBED40 0DEBED40 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000A0 0DEBEAD0 00000000 00000000 00000000 00000000 0DC68140 00000000 00000000 00000000 |.....Fa ..|
+0000C0 0DEBEAF0 43020008 40001000 00000000 0DB66DDC 58FF0008 07FF0000 0DC67BF8 00000000 |....._.....F#8....|
+0000E0 0DEBEB10 00000000 00000000 00000000 00000000 58FF0008 07FF0000 0DC80088 00000000 |......H.h....|
+000100 0DEBEB30 00000000 00000000 00000000 00000000 00000000 00000000 80000020 0DEBD000 |.....|

fldata FOR FILE: HEALY.MEMORY.DATA
__recfmF:1..... 1
__recfmV:1..... 0
__recfmU:1..... 0
__recfmS:1..... 0
__recfmBlk:1..... 0
__recfmASA:1..... 0
__recfmM:1..... 0
__recfmPO:1..... 0
__dsorgPDSmem:1... 0
__dsorgPDSdir:1... 0
__dsorgPS:1..... 0
__dsorgConcat:1... 0
__dsorgMem:1..... 1
__dsorgHiper:1... 0
__dsorgTemp:1.... 0
__dsorgVSAM:1.... 0
__dsorgHFS:1..... 0
__openmode:2.... 1
__modeflag:4..... 2
__dsorgPDSE:1.... 0
__reserve2:8..... 0
__device..... 8
__blksize..... 1024
__maxreclen..... 1024
__dsname..... HEALY.MEMORY.DATA
__reserve4..... 0

FILE pointer..... 0DEBEA1C

Buffer at current file position: 0DEBED40
+000000 0DEBED40 A2969485 408481A3 81A29694 85409496 99854084 81A38185 A5859540 94969985 |some datasome more dataeven more|
+000020 0DEBED60 408481A3 81000000 00000000 00000000 00000000 00000000 00000000 00000000 |data.....|
+000040 0DEBED80 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 0DEBEDA0 - +0003FF 0DEBF13F                same as above
Saved Buffer..... NULL

File Control Block: 0DEBD038
+000000 0DEBD038 0DEBD234 00000000 00000400 0DEBD108 0DEBD128 80000000 0DEBD158 00000011 |..K.....J...J.....J....|
+000020 0DEBD058 00000014 00000000 00000000 0DB6701C 0DEBEA30 0DEBD038 00000000 E2E8E2F0 |.....SYS0|
+000040 0DEBD078 F0F2F7F2 FFFFFFFF 0000003C 0DEBD178 0DEBD14C 0DE789B8 0DE768F0 0DE81178 |0272.....J...J<.Xi..X.0.Y..|
+000060 0DEBD098 0DE7F988 0DE85530 00000000 00000404 00001800 0DEBEA0A 00000000 00001801 |.X9h.Y.....|
+000080 0DEBD0B8 0DEBD208 0DEBD234 0DEBD234 00000000 00000000 00000000 00000000 00000000 |..K...K.....|
+0000A0 0DEBD0D8 00000018 00000000 00000000 00000000 0DE01740 00000000 00000000 00000000 |.....|
+0000C0 0DEBD0F8 43120020 28440000 00000000 0DB66DDC 58FF0008 07FF0000 0DE011F8 00000000 |....._.8....|
+0000E0 0DEBD118 00000000 00000000 00000000 00000000 58FF0008 07FF0000 0DE6F3E8 00000000 |.....W3Y....|
+000100 0DEBD138 00000000 00000000 00000000 00000000 00000000 00000000 80000020 0DEBD000 |.....|

```

Figure 43. Example dump from sample C routine (Part 4 of 6)

```

fldata FOR FILE: 'HEALY.MYFILE.DATA'
__recfmF:1..... 0
__recfmV:1..... 1
__recfmU:1..... 0
__recfmS:1..... 0
__recfmBlk:1..... 1
__recfmASA:1..... 0
__recfmM:1..... 0
__recfmPO:1..... 0
__dsorgPDsmem:1... 0
__dsorgPDSdir:1... 0
__dsorgPS:1..... 1
__dsorgConcat:1... 0
__dsorgMem:1..... 0
__dsorgHiper:1.... 0
__dsorgTemp:1.... 0
__dsorgVSAM:1.... 0
__dsorgHFS:1..... 0
__openmode:2..... 0
__modeflag:4..... 2
__dsorgPDSE:1.... 0
__reserve2:8..... 0
__device..... 0
__blksize..... 6144
__maxreclen..... 1024
__dsname..... HEALY.MYFILE.DATA
__reserve4..... 0

FILE pointer..... 0DEBD024
ddname..... SYS00272

Buffer at current file position: 0DEBD208
+000000 0DEBD208 00280000 000C0000 99858396 998440F1 000C0000 99858396 998440F2 000C0000 |.....record 1....record 2....|
+000020 0DEBD228 99858396 998440F3 00040000 00000000 00000000 00000000 00000000 00000000 |record 3.....|
+000040 0DEBD248 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 0DEBD268 - +0003FF 0DEBD607 same as above
Saved Buffer..... NULL

Write Data Control Block: 00052020
+000000 00052020 00052E20 00000000 00000008 00EFD08C 002FE5A2 00000001 00004000 0000CE38 |.....Vs.....|
+000020 00052040 8605223A 50052DAD 01582424 0089A044 12BEE1B0 00C129D8 0A0521B8 00001800 |f...&.....i.....A.Q.....|
+000040 00052060 30013030 0000CEA8 01D448F8 00D448F8 00000404 00D45470 47F0F026 01C3C5C5 |.....y.M.8.M.8.....M...00..CEE|
read/update DCB..... NULL
Write Data Control Block Extension: 00052E20
+000000 00052E20 C4C3C2C5 00380000 00052020 00000000 C0C80000 00000000 00000000 00000000 |DCBE.....H.....|
+000020 00052E40 00000000 00000000 00000000 00000000 00000000 00000100 800000B8 00052000 |.....|
read/update DCBE..... NULL

Job File Control Block: 00052E60
+000000 00052E60 C8C5C1D3 E84BD4E8 C6C9D3C5 4BC4C1E3 C1404040 40404040 40404040 40404040 |HEALY.MYFILE.DATA|
+000020 00052E80 40404040 40404040 40404040 40404040 40404040 80001F1D 00000000 00000000 |.....|
+000040 00052EA0 00000200 00000000 00000000 00000000 61004900 00000040 00000000 00000000 |...../.....|
+000060 00052EC0 00000000 00000000 00000000 00000000 00000000 0001E2D4 E2F0F0F6 40404040 |.....SMS006|
+000080 00052EE0 40404040 40404040 40404040 40404040 40404040 0089DDA0 00000050 00001800 |.....i.....&...|
+0000A0 00052F00 00000000 00000000 00000000 20000100 80000038 00052000 00052F18 0DEBD208 |.....K.....|

[8] __amrc_type structure: 00031B18
+000000 00031B18 00000000 00000000 00000007 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 00031B38 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 00031B58 - +0000BF 00031BD7 same as above
+0000C0 00031BD8 00000000 00000000 00000000 00000000 00000000 00000000 000000DC |.....|

```

Figure 43. Example dump from sample C routine (Part 5 of 6)

```

amrc __code union fields
__error..... 0(0)
__abend.__syscode..... 0(0)
__abend.__rc..... 0(0)
__feedback.rc..... 0(0)
__feedback.__ftncd..... 0(0)
__feedback.__fdbk..... 0(0)
__alloc.__svc99_info.... 0(0)
__alloc.__svc99_error... 0(0)
__RBA..... 0(0)
__last_op..... 7(7)
__msg.__str..... NULL
__msg.__parmr0..... 0(0)
__msg.__parmr1..... 0(0)
__msg.__str2..... NULL

__amrc2 type structure: 00031A1C
+000000 00031A1C 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
__error2..... 0(0)
__fileptr..... NULL

Process Control Blocks:

PCB: 00022558
+000000 00022558 C3C5C5D7 C3C24040 03030398 00000000 00000000 00000000 00022788 0DC294C8 |CEEPCB ...q.....h.BmH|
+000020 00022578 0DC27A50 0DC2A338 0DC2A810 0DB58928 00021918 00000000 00000000 000228B0 |.B:&.Bt..By...i.....|
+000040 00022598 0DC2A470 7C000000 00000000 00000000 00000000 00000000 00000000 00000000 |.Bu.@.....|

MEML: 00022788
+000000 00022788 00000000 00000000 0DBBB0B8 00000000 00000000 00000000 0DBBB0B8 00000000 |.....|
+000020 000227A8 00000000 00000000 0DBBB0B8 00000000 0DB66DDC 00000000 0DC2B6B8 00000000 |....._.....B.....|
+000040 000227C8 00000000 00000000 0DBBB0B8 00000000 00000000 00000000 0DBBB0B8 00000000 |.....|
+000060 000227E8 - +00011F 000228A7 same as above

Additional Language Specific Information:
[9] errno information :
Thread Id .... 8000000000000000 Errno ..... 00000000 Errnojr .... 00000000

```

Figure 43. Example dump from sample C routine (Part 6 of 6)

Finding C/C++ information in a Language Environment dump

When a Language Environment traceback or dump is generated for a C/C++ routine, information is provided that is unique to C/C++ routines. C/C++-specific information includes:

- Control block information for active routines
- Condition information for active routines
- Enclave level data

Each of the unique C/C++ sections of the Language Environment dump are described.

[1] Storage for Active Routines

The Storage for Active Routines section of the dump shows the DSAs for the active C and C++ routines. To relate a DSA frame to a particular function name, use the address associated with the frame to find the corresponding DSA. In this example, the function func1 DSA address is X'00030440'.

[2] Control Blocks Associated with the Active Thread

In the Control Blocks Associated with the Thread section of the dump, the following information appears:

- Fields from the CAA
- Fields specific from the CTHD and CEDB
- Signal information

[2A] C/C++ CAA Fields

The CAA contains several fields that the C/C++ programmer can use to find information about the run-time environment. For each C/C++ program, there is a C-C++ Specific Thread area and a C-C++ Specific Enclave area.

[2B] C-C++ Specific CAA

The C-C++ specific CAA fields that are of interest to users are described below.

errno value	A variable used to display error information. Its value can be set to a positive number that corresponds to an error message. The functions <code>perror()</code> and <code>strerror()</code> print the error message that corresponds to the value of <code>errno</code> .
Memory file control block	You can use the memory file control block (MFCB) to locate additional information about memory files. This control block resides at the C/C++ thread level. For more information about the MFCB, see 158.
Open FCB chain	A pointer to the start of a linked list of open file control blocks (FCBs). For more information about FCBs, see 158.

[3] Signal Information

When the `POSIX(OFF)` run-time option is specified, signal information is provided in the dump to aid you in debugging. For each signal that is disabled with `SIG_IGN`, an entry value of 00000001 is made in the first field of the Signal Information field for the specified signal name.

For each signal that has a handler registered, the signal name and the handler name are listed. If the handler is a fetched C function, the value `@@FECB` is entered as the function name and the address of the fetched pointer is in the first field.

If you compile a C routine as `NORENT`, the WSA address is not available (N/A). For more information about the signal function, see *z/OS XL C/C++ Programming Guide*.

[4] WSA Address

The WSA Address is the base address of the writable static area which is available for all C and C++ programs except C programs compiled with the `NORENT` compile option.

[5] atexit() Information

The `atexit()` information lists the functions registered with the `atexit()` function that would be run at normal termination. The functions are listed in chronological order of registration.

If you compile a C routine as `NORENT`, the WSA address is not available (N/A). For more information about the `atexit()` function, see *z/OS XL C/C++ Run-Time Library Reference*.

[6] fetch() Information

The `fetch()` information shows information about modules that you have dynamically loaded using `fetch()`. For each module that was fetched, the `fetch()` pointer and the function pointer are included.

```
ptr1 = fetch("MOD");
```

If you compile a C routine as `NORENT`, the WSA address is not available (N/A). For more information about the `fetch()` function, see *z/OS XL C/C++ Programming Guide*.

[7] File Control Block Information

This section of the dump includes the file control block (FCB) information for each C/C++ file. The FCB contains file status and attributes for files open during C/C++ active routines. You can use this information to find the data set or file name.

The FCB is a handle that points to the following file information, which is displayed when applicable, for the file:

- Access method control block (ACB) address
- Data control block (DCB) address
- Data control block extension (DCBE) address
- Job file control block (JFCB) address
- RPL address
- Current[®] buffer address
- Saved buffer address
- ddname

Not all FCB fields are always filled in. For example, RPLs are used only for VSAM data sets. The ddname field contains blanks if it is not used.

The save block buffer represents auxiliary buffers that are used to save the contents of the main buffers. Such saving occurs only when a reposition is performed and there is new data; for example, an incomplete text record or an incomplete fixed-block standard (FBS) block in the buffers that cannot be flushed out of the system.

Because the main buffers represent the current position in the file, while the save buffers merely indicate a save has occurred, check the save buffers only if data appears to be missing from the external device and is not found in the main buffers. Also, do not infer that the presence of save buffers means that data present there belongs at the end of the file. (The buffers remain, even when the data is eventually written.)

For information about the job file control block, refer to *z/OS MVS Data Areas, Vol 3 (IVT-RCWK)*.

Memory File Control Block

This section of the dump holds the memory file control block information for each memory file the routine uses. A sample memory file control block is shown in Figure 44 on page 159.

```

Memory File Control Block: 046F5CD0
+000000 046F5CD0 046F5D40 00000001 00000000 00000000 046F5BA8 00010000 046F5E48 00000013 |.?) .....?$.y.....?;.....|
+000020 046F5CF0 00000014 00000000 00000000 00000000 00000000 046F5BA8 00000000 00000000 |.....?$.y.....|
+000040 046F5D10 046F5CD0 00000000 00000000 00000000 00000000 00000000 00000000 046F5D40 |.?*.....?)|
+000060 046F5D30 00010000 00000000 00000000 00000000 00000000 00000000 046F5E68 00000001 |.....?;.....|
memory file name..... TSQID.MEMORY.DATA
First memory data space: 046F5E68
+000000 046F5E68 93899585 40F19389 958540F2 93899585 40F30000 00000000 00000000 00000000 |line 1line 2line 3.....|

```

Figure 44. Memory file control block

Memory file name	The name assigned to this memory file.
First memory data space	A dump of the first 1K maximum of actual user data associated with this memory file.

[8] Information for `__amrc`

`__amrc` is a structure defined in the **stdio.h** header file to assist in determining errors resulting from I/O operations. The contents of `__amrc` can be checked for system information, such as the return code for VSAM. Certain fields of the `__amrc` structure can provide useful information about what occurred previously in your routine.

For more information about `__amrc`, refer to “Debugging C/C++ input/output programs” on page 127 and to *z/OS XL C/C++ Programming Guide*.

[9] Errno Information

The Errno information shows the thread id of the thread that generated the dump and the settings of the `errno` and `errnojr` variables for that thread.

Both the `errno` and the `errnojr` variables contain the return code of the last failing z/OS UNIX system service call. These variables provide z/OS UNIX application programs access to diagnostic information returned from an underlying z/OS UNIX callable service. For more information on these return and reason codes, refer to *z/OS UNIX System Services Messages and Codes*.

Additional Floating-Point registers

The Language Environment dump formats Additional Floating Point (AFP™) registers and Floating Point Control (FPC) registers when the AFP suboption of the FLOAT XL C/C++ compiler option is specified and the registers are needed. These floating-point registers are displayed in three sections of the CEE3DUMP; Registers on Entry to CEE3DMP; Parameters, Registers, and Variables; and Condition Information for Active Routines. Samples of each section are given. For information on the FLOAT XL C/C++ compiler option, see *z/OS XL C/C++ User's Guide*.

Registers on entry to CEE3DMP: This section of the Language Environment dump displays the twelve floating-point registers. A sample output is shown.

```
CEE3DMP V1 R3.0: Sample dump produced by calling CEE3DMP                                08/30/01 5:19:52 PM
CEE3DMP called by program unit ./celdll.c (entry point dump_n_perc) at statement 34 (offset +0000017A).
```

Registers on Entry to CEE3DMP:

```
PM..... 0100
GPR0..... 183F8BE8  GPR1..... 00023D38  GPR2..... 00023E98  GPR3..... 1840E792
GPR4..... 00023D98  GPR5..... 183F8CD0  GPR6..... 00023D48  GPR7..... 0002297F
GPR8..... 17F4553D  GPR9..... 183F6870  GPR10..... 17F4353F  GPR11..... 17FA0550
GPR12..... 00015920  GPR13..... 00023CA0  GPR14..... 800180E2  GPR15..... 97F57FE8
FPC..... 40084000
FPR0..... 40260000  00000000      FPR1..... 41086A00  00000000
FPR2..... 00000000  00000000      FPR3..... 3F8CAC08  3126E979
FPR4..... 3FF33333  33333333      FPR5..... 40C19400  00000000
FPR6..... 3F661E4F  765FD8AE      FPR7..... 3FF06666  66666666
FPR8..... 3FF33333  33333333      FPR9..... 00000000  00000000
FPR10..... 3FF33333  33333333      FPR11..... 00000000  00000000
FPR12..... 40260000  00000000      FPR13..... 00000000  00000000
FPR14..... 40220000  00000000      FPR15..... 00000000  00000000
:
```

Figure 45. Registers on entry to CEE3DMP

Parameters, registers, and variables for active routines: This section of the Language Environment dump displays the non-volatile floating-point registers that are saved in the stack frame. The registers are only displayed if the program owning the stack frame saved them. Dashes are displayed in the registers when the register values are not saved. A sample output is shown.

```
Parameters, Registers, and Variables for Active Routines:
:
goo (DSA address 000213B0):
  Saved Registers:
    GPR0..... 183F6CC0  GPR1..... 00021278  GPR2..... 183F6870  GPR3..... 17F01DC2
    GPR4..... 000000F8  GPR5..... 183F6968  GPR6..... 17F02408  GPR7..... 000212EC
    GPR8..... 000212F0  GPR9..... 80000000  GPR10..... 98125022  GPR11..... 80007F98
    GPR12..... 00015920  GPR13..... 000213B0  GPR14..... 97F01E1E  GPR15..... 0000002F
    FPR8..... 3FF33333  33333333      FPR9..... -----
    FPR10..... 3FF33333  33333333      FPR11..... -----
    FPR12..... 40260000  00000000      FPR13..... -----
    FPR14..... 40220000  00000000      FPR15..... -----
  GPREG STORAGE:
    Storage around GPR0 (183F6CC0)
  :
```

Figure 46. Parameters, registers, and variables for active routines

Condition information for active routines: This section of the Language Environment dump displays the floating-point registers when they are saved in the machine state. A sample output is shown.

```

:
Condition Information for Active Routines
Condition Information for ./celsamp.c (DSA address 000213B0)
CIB Address: 00021F90
Current Condition:
  CEE3224S The system detected an IEEE division-by-zero exception.
Location:
  Program Unit: ./celsamp.c
  Program Unit:Entry:      goo Statement: 78 Offset: +000000BA
Machine State:
  ILC..... 0004      Interruption Code..... 0007
  PSW..... 078D0400 97F01E46
  GPR0..... 183F6CC0  GPR1..... 00021278  GPR2..... 183F6870  GPR3..... 17F01DC2
  GPR4..... 000000F8  GPR5..... 183F6968  GPR6..... 17F02408  GPR7..... 000212EC
  GPR8..... 000212F0  GPR9..... 80000000  GPR10..... 98125022  GPR11..... 80007F98
  GPR12..... 00015920  GPR13..... 000213B0  GPR14..... 97F01E1E  GPR15..... 0000002F
  FPC..... 40084000
  FPR0..... 40260000 00000000      FPR1..... 41086A00 00000000
  FPR2..... 00000000 00000000      FPR3..... 3F8CAC08 3126E979
  FPR4..... 3FF33333 33333333      FPR5..... 40C19400 00000000
  FPR6..... 3F661E4F 765FD8AE      FPR7..... 3FF06666 66666666
  FPR8..... 3FF33333 33333333      FPR9..... 00000000 00000000
  FPR10..... 3FF33333 33333333      FPR11..... 00000000 00000000
  FPR12..... 40260000 00000000      FPR13..... 00000000 00000000
  FPR14..... 40220000 00000000      FPR15..... 00000000 00000000
Storage dump near condition, beginning at location: 17F01E32
+000000 17F01E32 68201008 5810D0F0 68401010 B31B0024 B31D0002 B3050000 5820D0F4 584031C2 |.....0. ....4. .B|
:

```

Figure 47. Condition information for active routines

Sample Language Environment dump with XPLINK-specific information

The programs `tranmain` shown in Figure 48 on page 162 and `trandll` shown in Figure 49 on page 163 were used to produce a Language Environment dump. The dump shows XPLINK-compiled routines calling NOXPLINK-compiled routines, and NOXPLINK-compiled routines calling XPLINK-compiled routines. The program `tranmain` was compiled XPLINK and `trandll` was compiled NOXPLINK. Each was link-edited as a separate program object with the sidedeck from the other. The Language Environment dump produced by running these program is shown in Figure 50 on page 164. Explanations for some of the sections are in “Finding XPLINK information in a Language Environment dump” on page 166.

```

#pragma runopts (TRACE(ON,1M,NODUMP,LE=1),XPLINK(ON),TERMTHDACT(UADUMP))
#include <stdio.h>
#pragma export(tran2)

int tran1(int, int, int, long double, int);
int tran3(int, int, int, long double, int);

void main(void) {

    int      parm1 = 0x11111111;
    int      parm2 = 0x22222222;
    int      parm3 = 0x33333333;
    long double parm4 = 1234.56789;
    int      parm5 = 0x55555555;
    int      retval;

    printf("Main: Call Tran1\n");
    retval = tran1(parm1,parm2,parm3,parm4,parm5);
    printf("Main: Return value from Tran1 = %d\n",retval);

}
int tran2(int parm1,int parm2,int parm3,long double parm4,int parm5) {

    int      retval;

    printf("Tran2: Call Tran3\n");
    retval = tran3(parm1,parm2,parm3,parm4,parm5);
    printf("Tran2: Return value from Tran3 = %d\n",retval);
    return retval;

}

```

Figure 48. Sample XPLINK-compiled program (tranmain) which calls a NOXPLINK-compiled program

```

#include <stdio.h>
#include <ctest.h>
#include <leawi.h>
#pragma export(tran1)
#pragma export(tran3)

int tran2(int, int, int, long double, int);

int tran1(int parm1,int parm2,int parm3,long double parm4,int parm5) {
    int        retval;

    printf("Tran1: Call Tran2\n");
    retval = tran2(parm1,parm2,parm3,parm4,parm5);
    printf("Tran1: Return value from Tran2 = %d\n",retval);
    return retval;
}

int tran3(int parm1,int parm2,int parm3,long double parm4,int parm5) {
    _INT4 code, timing;

    code = 1001;    /* Abend code to issue */
    timing = 1;
    printf("Tran3: About to ABEND\n");
    CEE3ABD(&code,&timing);

    return parm1 + parm2 + parm3;
}

```

Figure 49. Sample NOXPLINK-compiled program (trandll) which calls an XPLINK-compiled program

Information for enclave main

Information for thread 8000000000000000

[1] Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
23ED4C18	CEEHDSR	23BA6538	+000038DA	CEEHDSR	23BA6538	+000038DA			CEEPLPKA	Call
23ED4998	CEEHABD	23AD34A0	+0000012A	CEEHABD	23AD34A0	+0000012A			CEEPLPKA	Exception
23ED48E0	./trandll.c	240848C0	+000000D6	tran3	240848C0	+000000D6	26	XNTDLL		Call
23ED4730	CEEVRONU	23BB0470	+00000706	CEEVRONU	23BB0470	+00000706			CEEPLPKA	Call
24077530	./tranmain.c	23A000E8	+00000070	tran2	23A000E8	+00000070	27	XNTRAN		Call
240775B0		23BAEE38	+000009A4	CEEVROND	23BAEE90	+0000094C			CEEPLPKA	Call
23ED44E8	./trandll.c	240844A0	+000000F2	tran1	240844A0	+000000F2	14	XNTDLL		Call
23ED4338	CEEVRONU	23BB0470	+00000706	CEEVRONU	23BB0470	+00000706			CEEPLPKA	Call
24077680	./tranmain.c	23A00218	+0000008C	main	23A00218	+0000008C	18	XNTRAN		Call
24077720		23BAEE38	+000009A4	CEEVROND	23BAEE90	+0000094C			CEEPLPKA	Call
23ED40E0	EDCZHINV	23E8EC28	+0000009A	EDCZHINV	23E8EC28	+0000009A			CELHV003	Call
23ED4018	CEEBBEXT	000082A8	+000001A6	CEEBBEXT	000082A8	+000001A6			CEEBINIT	Call

Condition Information for Active Routines

Condition Information for CEEHABD (DSA address 23ED4998)

CIB Address: 23ED5438

Current Condition:

CEE0198S The termination of a thread was signaled due to an unhandled condition.

Original Condition:

CEE3250C The system or user abend U1001 R=00000000 was issued.

Location:

Program Unit: CEEHABD Entry: CEEHABD Statement: Offset: +0000012A

Machine State:

ILC..... 0002 Interruption Code..... 000D

PSW..... 078D1400 A3AD35CA

GPR0..... 84000000 GPR1..... 840003E9 GPR2..... 23ED4984 GPR3..... 240848FA

GPR4..... 23ED4980 GPR5..... 000159D0 GPR6..... 00000000 GPR7..... 00000000

GPR8..... A3A000F2 GPR9..... 23A1BE80 GPR10..... 23ED4980 GPR11..... A3AD34A0

GPR12..... 00016AC0 GPR13..... 23ED4998 GPR14..... A4084998 GPR15..... 00000000

ABEND code: 000003E9 Reason code: 00000000

Storage dump near condition, beginning at location: 23AD35BA

+000000 23AD35BA 88100008 41000084 89000018 16100A0D 47F0B1B6 5840C2F0 9506400B 4770B1A0 |h.....di.....0... B0n.|

[2] Parameters, Registers, and Variables for Active Routines:

:

tran3 (DSA address 23ED48E0):

UPSTACK DSA

Parameters:

parm5	signed int	1431655765
parm4	long double	1.23456788999999977135303197E+03
parm3	signed int	858993459
parm2	signed int	572662306
parm1	signed int	286331153

Saved Registers:

GPR0.....	23A1BD70	GPR1.....	23ED4978	GPR2.....	23ED4984	GPR3.....	240848FA
GPR4.....	23ED4980	GPR5.....	23A1BE10	GPR6.....	00000000	GPR7.....	00000000
GPR8.....	A3A000F2	GPR9.....	23A1BE80	GPR10.....	240848C0	GPR11.....	A3BB04FA
GPR12.....	00016AC0	GPR13.....	23ED48E0	GPR14.....	A4084998	GPR15.....	A3AD34A0

:

Local Variables:

timing	signed long int	1
code	signed long int	1001

Figure 50. Example dump of calling between XPLINK and non-XPLINK programs (Part 1 of 3)

```

CEEVRONU (DSA address 23ED4730):
TRANSITION DSA
Saved Registers:
  GPR0..... 23A1BD70  GPR1..... 24077D70  GPR2..... 23ED4818  GPR3..... 00000010
  GPR4..... 24077530  GPR5..... 23FF58D0  GPR6..... 00000000  GPR7..... 00000000
  GPR8..... A3A000F2  GPR9..... 23A1BE80  GPR10..... 240848C0  GPR11..... A3BB04FA
  GPR12..... 00016AC0  GPR13..... 23ED4730  GPR14..... A3BB0B78  GPR15..... 240848C0
:
:

tran2 (DSA address 24077530):
DOWNSTACK DSA
Parameters:
  parm5          signed int          1431655765
  parm4          long double         1.23456788999999977135303197E+03
  parm3          signed int          858993459
  parm2          signed int          572662306
  parm1          signed int          286331153
Saved Registers:
  GPR0..... 55555555  GPR1..... 11111111  GPR2..... 22222222  GPR3..... 33333333
  GPR4..... 24077530  GPR5..... 23FF58D0  GPR6..... 23BB04D8  GPR7..... A3A0015A
  GPR8..... A3A000F2  GPR9..... 23A1BE80  GPR10..... 23A000B8  GPR11..... 23BAEE38
  GPR12..... 00016AC0  GPR13..... 23ED45B0  GPR14..... 23A000B8  GPR15..... 0000000C
:
:

Local Variables:
  retval          signed int          -455613482
CEEVROND (DSA address 240775B0):
TRANSITION DSA
Saved Registers:
  GPR0..... *****  GPR1..... *****  GPR2..... *****  GPR3..... *****
  GPR4..... 240775B0  GPR5..... 23A1BE80  GPR6..... 23A000E8  GPR7..... A3BAF7DE
  GPR8..... 23A000B8  GPR9..... 23BAFE37  GPR10..... *****  GPR11..... *****
  GPR12..... *****  GPR13..... *****  GPR14..... *****  GPR15..... *****
:
:

tran1 (DSA address 23ED44E8):
UPSTACK DSA
Saved Registers:
  GPR0..... 23FF5098  GPR1..... 23ED4580  GPR2..... 55555555  GPR3..... 24084A7A
  GPR4..... 33333333  GPR5..... 23A1BE10  GPR6..... 22222222  GPR7..... 11111111
  GPR8..... A3A00222  GPR9..... 23A00320  GPR10..... 24084A40  GPR11..... A3BB04FA
  GPR12..... 00016AC0  GPR13..... 23ED44E8  GPR14..... A4084B34  GPR15..... 23BAEE38
:
:

```

Figure 50. Example dump of calling between XPLINK and non-XPLINK programs (Part 2 of 3)

[3] Control Blocks for Active Routines:

```
:
:
DSA for tran3: 23ED48E0
+000000  FLAGS.... 10A0      member... 1D8C      BKC..... 23ED4730  FWC..... 23ED4998  R14..... A4084998
+000010  R15..... A3AD34A0    R0..... 23A1BD70  R1..... 23ED4978  R2..... 23ED4984  R3..... 240848FA
+000024  R4..... 23ED4980    R5..... 23A1BE10  R6..... 00000000  R7..... 00000000  R8..... A3A000F2
+000038  R9..... 23A1BE80    R10..... 240848C0 R11..... A3BB04FA R12..... 00016AC0 reserved. 00017630
+00004C  NAB..... 23ED4998    PNAB..... 00000000 reserved. 00000000 00000000 00000000
+000064  reserved. 00000000    reserved. 23A00000 reserved. 80000000 00000000
+000078  reserved. 23A001D0    reserved. 23A00480
DSA for CEEVRONU: 23ED4730
+000000  FLAGS.... 0000      member... 0000      BKC..... FFFFFFFF  FWC..... E3D9C1D5  R14..... A3BB0B78
+000010  R15..... 240848C0    R0..... 23A1BD70  R1..... 24077D70  R2..... 23ED4818  R3..... 00000010
+000024  R4..... 24077530    R5..... 23FF58D0  R6..... 00000000  R7..... 00000000  R8..... A3A000F2
+000038  R9..... 23A1BE80    R10..... 240848C0 R11..... A3BB04FA R12..... 00016AC0 reserved. 00017630
+00004C  NAB..... 23ED48E0    PNAB..... 23ED48E0 reserved. 00000000 00000000 00000000
+000064  reserved. 23ED47B0    reserved. 00000000 MODE..... 00000000 reserved. 00000000 00000000
+000078  reserved. 00000000    reserved. 00000000
DSA for CEEVRONU: 23ED47B0
+000000  EYE..... DOWNTOUN  TRTYPE... 00000003  BOS..... 00000000  STACKFLR. 00000000  SSTOPD... 24077680
+000018  SSDSAU... 23ED4730  TRANEP... 23BB0470  TR_R0.... 55555555  TR_R1.... 11111111  TR_R2.... 22222222
+00002C  TR_R3.... 33333333  TR_R4.... 24077530  TR_R5.... 23FF58D0  TR_R6.... 23BB04D8  TR_R7.... A3A0015A
+000040  TR_R8.... A3A000F2  TR_R9.... 23A1BE80  TR_R10... 23A000B8  TR_R11... 23BAEE38  TR_R12... 00016AC0
+000054  TR_R13... 23ED45B0  TR_R14... 23A000B8  TR_R15... 0000000C  CRENT.... 00000000  ROND_DSA. 23ED45B0
+000068  INTF MAP. 018F1000
DSA for tran2: 24077D30
+000000  R4..... 240775B0    R5..... 23A1BE80  R6..... 23A000E8  R7..... A3BAF7DE  R8..... 23A000B8
+000014  R9..... 23BAFE37    R10..... 23A000B8 R11..... 23BAEE38 R12..... 00016AC0 R13..... 23ED45B0
+000028  R14..... 23A000B8  R15..... 0000000C reserved. 00000A68 reserved. 23BF87B8 HPTRAN... 00000000
+00003C  reserved. 55555555 reserved. 11111111
DSA for CEEVROND: 24077DB0
+000000  R4..... E3D9C1D5    R5..... 00000000  R6..... 23BAEE90  R7..... 00000000  R8..... 23ED4860
+000014  R9..... 00000000    R10..... 00000000 R11..... A3B6931C R12..... 00000000 R13..... 00000000
+000028  R14..... 24077E10  R15..... 00000000 reserved. 23A1BE1C reserved. 23ED45A8 HPTRAN... 24077E10
+00003C  reserved. FFFFFFFF reserved. 11111111
DSA for CEEVROND: 24077E10
+000000  EYE..... UPTODOWN  TRTYPE... 00000002  BOS..... 00000000  STACKFLR. 00000000  SSTOPD... 24077680
+000018  SSDSAU... 23ED4338  TRANEP... 23BAEE90  TR_R0.... 00000000  TR_R1.... 00000000  TR_R2.... 00000000
+00002C  TR_R3.... 00000000  TR_R4.... 23ED44E8  TR_R5.... 00000000  TR_R6.... 00000000  TR_R7.... A4084B34
+000040  TR_R8.... 00000000  TR_R9.... 00000000  TR_R10... 00000000  TR_R11... 00000000  TR_R12... 00000000
+000054  TR_R13... 00000000  TR_R14... 00000000  TR_R15... 00000000  CRENT.... 23BAEE38  ROND_DSA. 00000000
+000068  INTF MAP. 00000000
DSA for tran1: 23ED44E8
+000000  FLAGS.... 1000      member... 59D0      BKC..... 23ED4338  FWC..... 23ED4860  R14..... A4084B34
+000010  R15..... 23BAEE38    R0..... 23FF5098  R1..... 23ED4580  R2..... 55555555  R3..... 24084A7A
+000024  R4..... 33333333    R5..... 23A1BE10  R6..... 22222222  R7..... 11111111  R8..... A3A00222
+000038  R9..... 23A00320    R10..... 24084A40 R11..... A3BB04FA R12..... 00016AC0 reserved. 00017630
+00004C  NAB..... 23ED45B0    PNAB..... A3B9104C reserved. 23ED4408 00000001 23A01CF0 0000000C
+000064  reserved. 00000000    reserved. 23A01D38 MODE..... 00100028 reserved. 00000000 00000000
+000078  reserved. 23A01BB0    reserved. A3B90CF8
:
:
```

Figure 50. Example dump of calling between XPLINK and non-XPLINK programs (Part 3 of 3)

Finding XPLINK information in a Language Environment dump

[1] Traceback

When an XPLINK-compiled routine calls a NOXPLINK-compiled routine, a glue routine gets control to convert the linkage conventions of the XPLINK caller to those of the NOXPLINK callee. In the sample dump, this routine is CEEVRONU and it appears between `main()` and `tran1()` and again between `tran2()` and `tran3()`.

When a NOXPLINK-compiled routine calls an XPLINK-compiled routine, a glue routine gets control to convert the linkage conventions of the NOXPLINK caller to those of the XPLINK callee. In the sample dump, this routine is CEEVROND and it appears between `EDCZHINV` and `main()` and again between `tran1()` and `tran2()`.

[2] Parameters, Registers, and Variables for Active Routines

In this section, each DSA is identified as one of the following:

UPSTACK DSA	The DSA format is that for a NOXPLINK-compiled program that uses an upward growing stack.
DOWNSTACK DSA	The DSA format is that for an XPLINK-compiled program that uses a downward growing stack.
TRANSITION DSA	The DSA format is that of its callee. A transition DSA can occur between an UPSTACK DSA and a DOWNSTACK DSA where it represents a transition from one linkage convention to another. A transition DSA can also occur between two DOWNSTACK DSAs where it represents a transition from one stack segment to another (a stack overflow).

[3] Control Blocks for Active Routines

In this section, DSAs are formatted. Those previously identified as UPSTACK DSAs will have one format and those identified as DOWNSTACK DSAs will have a different format. Those identified as TRANSITION DSAs will have two parts — the first will be either the downstack or upstack format, the second is unique to transition DSAs and contains information about the transition.

It is important to understand that the registers saved in an upstack DSA are those saved by a routine that the DSA-owning routine called. Typically register 15 is the entry point of the routine that was called, and register 14 is the return address into the DSA-owning routine. In contrast, the registers saved in a downstack DSA are those saved by the DSA-owning routine on entry. Register 7 is the return address back to the caller of the DSA-owning routine. Register 6 may be the entry point of the DSA-owning routine. (This is not true when the Branch Relative and Save instruction is used to implement the call.)

C/C++ contents of the Language Environment trace tables

Language Environment provides four C/C++ trace table entry types that contain character data:

- Trace entry 1 occurs when a base C library function is called.
- Trace entry 2 occurs when a base C library function returns.
- Trace entry 3 occurs when a POSIX C library function is called.
- Trace entry 4 occurs when a POSIX C library function returns.
- Trace entry 5 occurs when an XPLINK base C or POSIX C library function is called.
- Trace entry 6 occurs when an XPLINK base C or POSIX C library function returns.

The format for trace table entry 1 is:

```
NameOfCallingFunction  
—>(xxx) NameOfCalledFunction
```

or, for called functions calloc, free, malloc, and realloc:

```
NameOfCallingFunction  
—>(xxx) NameOfCalledFunction<(input_parameters)>
```

In addition, when the call is due to one of these C++ operators:

```
-new,  
-new[],  
-delete,  
-delete[]
```

then the C++ operator will appear and the format becomes:

```
NameOfCallingFunction  
—>(xxx) NameOfCalledFunction<(input_parameters)>  
  
NameOfC++Operator
```

The format for trace table entry 2 is:

```
<—(xxx) R15=value ERRN0=value
```

The format for trace table entry 3 is:

```
NameOfCallingFunction  
—>(xxx) NameOfCalledFunction
```

The format for trace table entry 4 is:

```
<—(xxx) R15=value ERRN0=value ERRN02=value
```

The format for trace table entry 5 is:

```
NameOfCallingFunction  
-->(xxxx) NameOfCalledFunction<(input_parameters)>
```

Trace table entry 5 is just like trace table entry 1. The `input_parameters` and `NameOfC++Operator` only appear for the appropriate functions. The angle brackets (<>) indicate that this information does not always appear.

The format for trace table entry 6 is:

```
<--(xxxx) R1=xxxxxxx R2=xxxxxxx R3=xxxxxxx ERRN0=xxxxxxx ERRN02=xxxxxxx
```

In all six entry types, (xxx) and (xxxx) are numbers associated with the called library function and are used to associate a specific entry record with its corresponding return record.

For entry types 5 and 6, the number will be the same as the number of the function as seen in the C run-time library definition side-deck, SCEELIB dataset member CELHS003, on the IMPORT statement for that function.

Figure 51 on page 169 shows a non-XPLINK trace which has examples of C/C++ trace table entry types 1 thru 4.

Figure 52 on page 171 shows an XPLINK trace which has examples of the trace entries 5 and 6.

:

Language Environment Trace Table:

Most recent trace entry is at displacement: 02D500

Displacement	Trace Entry in Hexadecimal								Trace Entry in EBCDIC	
+000000	Time 20.52.46.666280	Date 2001.08.26	Thread ID...	8000000000000000						
+000010	Member ID.... 03	Flags..... 000000	Entry Type....	00000001						
+000018	94818995 40404040	40404040 40404040	40404040 40404040	40404040	40404040	40404040	40404040	main		
+000038	60606E4D F1F3F95D	40A2A399 8397A84D	5D404040	40404040	40404040	40404040	40404040	-->(139) strcpy()		
+000058	40404040 40404040	40404040 40404040	40404040	40404040	40404040	40404040	40404040			
+000078	40404040 40404040									
+000080	Time 20.52.46.666286	Date 2001.08.26	Thread ID...	8000000000000000						
+000090	Member ID.... 03	Flags..... 000000	Entry Type....	00000002						
+000098	4C60604D F1F3F95D	40D9F1F5 7EF2F4C2	F7F3F1C4 F840C5D9	D9D5D67E F0F0F0F0				<--(139) R15=24B731D8	ERRNO=0000	
+0000B8	F0F0F0F0 00000000	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000			0000.....		
+0000D8	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000					
+0000F8	00000000 00000000									
+000100	Time 20.52.46.666289	Date 2001.08.26	Thread ID...	8000000000000000						
+000110	Member ID.... 03	Flags..... 000000	Entry Type....	00000001						
+000118	94818995 40404040	40404040 40404040	40404040 40404040	40404040	40404040	40404040	40404040	main		
+000138	60606E4D F1F3F95D	40A2A399 8397A84D	5D404040	40404040	40404040	40404040	40404040	-->(139) strcpy()		
+000158	40404040 40404040	40404040 40404040	40404040	40404040	40404040	40404040	40404040			
+000178	40404040 40404040									
+000180	Time 20.52.46.666293	Date 2001.08.26	Thread ID...	8000000000000000						
+000190	Member ID.... 03	Flags..... 000000	Entry Type....	00000002						
+000198	4C60604D F1F3F95D	40D9F1F5 7EF2F4C2	F7F3F2F2 F840C5D9	D9D5D67E F0F0F0F0				<--(139) R15=24B73228	ERRNO=0000	
+0001B8	F0F0F0F0 00000000	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000			0000.....		
+0001D8	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000					
+0001F8	00000000 00000000									
+000200	Time 20.52.46.666303	Date 2001.08.26	Thread ID...	8000000000000000						
+000210	Member ID.... 03	Flags..... 000000	Entry Type....	00000003						
+000218	C98785A3 97819994	A2404040 40404040	40404040 40404040	40404040	40404040	40404040	40404040	Igetparms		
+000238	60606E4D F0F5F25D	4089A2B1 A3A3A84D	5D404040	40404040	40404040	40404040	40404040	-->(052) isatty()		
+000258	40404040 40404040	40404040 40404040	40404040	40404040	40404040	40000000	00000000			
+000278	00000000 00000000									
+000280	Time 20.52.46.673289	Date 2001.08.26	Thread ID...	8000000000000000						
+000290	Member ID.... 03	Flags..... 000000	Entry Type....	00000004						
+000298	4C60604D F0F5F25D	40D9F1F5 7EF0F0F0	F0F0F0F0 F040C5D9	D9D5D67E F0F0F0F0				<--(052) R15=00000000	ERRNO=0000	
+0002B8	F0F0F7F1 40C5D9D9	D5D6F27E F0F5C6C3	F0F1F1C3 00000000	00000000 00000000				0071 ERRNO2=05FC011C.....		
+0002D8	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000					
+0002F8	00000000 00000000									
+000300	Time 20.52.46.673296	Date 2001.08.26	Thread ID...	8000000000000000						
+000310	Member ID.... 03	Flags..... 000000	Entry Type....	00000003						
+000318	C98785A3 97819994	A2404040 40404040	40404040 40404040	40404040	40404040	40404040	40404040	Igetparms		
+000338	60606E4D F0F5F25D	4089A2B1 A3A3A84D	5D404040	40404040	40404040	40404040	40404040	-->(052) isatty()		
+000358	40404040 40404040	40404040 40404040	40404040	40404040	40400000	00000000				
+000378	00000000 00000000									
+000380	Time 20.52.46.673334	Date 2001.08.26	Thread ID...	8000000000000000						
+000390	Member ID.... 03	Flags..... 000000	Entry Type....	00000004						
+000398	4C60604D F0F5F25D	40D9F1F5 7EF0F0F0	F0F0F0F0 F040C5D9	D9D5D67E F0F0F0F0				<--(052) R15=00000000	ERRNO=0000	
+0003B8	F0F0F7F1 40C5D9D9	D5D6F27E F0F5C6C3	F0F1F1C3 00000000	00000000 00000000				0071 ERRNO2=05FC011C.....		
+0003D8	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000					
+0003F8	00000000 00000000									
+000400	Time 20.52.46.673338	Date 2001.08.26	Thread ID...	8000000000000000						
+000410	Member ID.... 03	Flags..... 000000	Entry Type....	00000003						
+000418	C98785A3 97819994	A2404040 40404040	40404040 40404040	40404040	40404040	40404040	40404040	Igetparms		
+000438	60606E4D F0F5F25D	4089A2B1 A3A3A84D	5D404040	40404040	40404040	40404040	40404040	-->(052) isatty()		
+000458	40404040 40404040	40404040 40404040	40404040	40404040	40400000	00000000				
+000478	00000000 00000000									

Figure 51. Trace table with C/C++ trace table entry types 1 thru 4 (Part 1 of 2)

```

+000480 Time 20.52.46.673373 Date 2001.08.26 Thread ID... 8000000000000000
+000490 Member ID.... 03 Flags..... 000000 Entry Type..... 00000004
+000498 4C60604D F0F5F25D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(052) R15=00000000 ERRNO=0000
+0004B8 F0F0F7F1 40C5D9D9 D5D6F27E F0F5C6C3 F0F1F1C3 00000000 00000000 00000000 |0071 ERRNO2=05FC011C.....
+0004D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0004F8 00000000 00000000

+000500 Time 20.52.46.673379 Date 2001.08.26 Thread ID... 8000000000000000
+000510 Member ID.... 03 Flags..... 000000 Entry Type..... 00000001
+000518 C98785A3 97819994 A2404040 40404040 40404040 40404040 40404040 40404040 |Igetparms
+000538 60606E4D F1F2F95D 408785A3 8595A54D 5D404040 40404040 40404040 40404040 |-->(129) getenv()
+000558 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000578 40404040 40404040

+000580 Time 20.52.46.673392 Date 2001.08.26 Thread ID... 8000000000000000
+000590 Member ID.... 03 Flags..... 000000 Entry Type..... 00000002
+000598 4C60604D F1F2F95D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(129) R15=00000000 ERRNO=0000
+0005B8 F0F0F7F1 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |0071.....
+0005D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0005F8 00000000 00000000

+000600 Time 20.52.46.673401 Date 2001.08.26 Thread ID... 8000000000000000
+000610 Member ID.... 03 Flags..... 000000 Entry Type..... 00000001
+000618 C9A285A3 A4974040 40404040 40404040 40404040 40404040 40404040 40404040 |Isetup
+000638 60606E4D F1F9F15D 408685A3 83884D5D 40404040 40404040 40404040 40404040 |-->(191) fetch()
+000658 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000678 40404040 40404040

+000680 Time 20.52.47.553343 Date 2001.08.26 Thread ID... 8000000000000000
+000690 Member ID.... 03 Flags..... 000000 Entry Type..... 00000002
+000698 4C60604D F1F9F15D 40D9F1F5 7EF2F4C2 F7F6F0F6 F040C5D9 D9D5D67E F0F0F0F0 |<--(191) R15=24B76060 ERRNO=0000
+0006B8 F0F0F7F1 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |0071.....
+0006D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0006F8 00000000 00000000

+000700 Time 20.52.47.553355 Date 2001.08.26 Thread ID... 8000000000000000
+000710 Member ID.... 03 Flags..... 000000 Entry Type..... 00000001
+000718 C9A285A3 A4974040 40404040 40404040 40404040 40404040 40404040 40404040 |Isetup
+000738 60606E4D F1F2F45D 40948193 9396834D F2F0F6F8 5D404040 40404040 40404040 |-->(124) malloc(2068)
+000758 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000778 40404040 40404040

+000780 Time 20.52.47.553366 Date 2001.08.26 Thread ID... 8000000000000000
+000790 Member ID.... 03 Flags..... 000000 Entry Type..... 00000002
+000798 4C60604D F1F2F45D 40D9F1F5 7EF2F4C2 F7F6F2F3 F040C5D9 D9D5D67E F0F0F0F0 |<--(124) R15=24B76230 ERRNO=0000
+0007B8 F0F0F7F1 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |0071.....
+0007D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0007F8 00000000 00000000

```

⋮

Figure 51. Trace table with C/C++ trace table entry types 1 thru 4 (Part 2 of 2)

Figure 52 on page 171 shows an XPLINK trace which has examples of the trace entries 5 and 6.

⋮
Language Environment Trace Table:

Most recent trace entry is at displacement: 000D80

Displacement	Trace Entry in Hexadecimal	Trace Entry in EBCDIC
+000000	Time 22.41.35.433944 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000010	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000018	4C60604D F0F0F5F9 5D40D9F1 7EF2F3C6 C6C3C1C2 F040D9F2 7EF2F3C3 F5F8F9C4	<--(0059) R1=23FFCAB0 R2=23C589D 0 R3=23FFD000 ERRNO=00000074 ERR NO2=00000000.....
+000038	F040D9F3 7EF2F3C6 C6C4F0F0 F040C5D9 D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9	
+000058	D5D6F27E F0F0F0F0 F0F0F0F0 00000000 00000000 00000000 00000000	
+000078	00000000 00000000	
+000080	Time 22.41.35.433948 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000090	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000098	C9D9E3D3 D985A296 A4998385 7A7AA1C9 D9E3D3D9 85A296A4 9983854D 5D404040	IRTLResource::.IRTLResource() -->(0204) pthread_mutex_destroy()
+0000B8	60606E4D F0F2F0F4 5D4097A3 88998581 846D94A4 A385A76D 8485A2A3 9996A84D	
+0000D8	5D404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+0000F8	40404040 40404040	
+000100	Time 22.41.35.433952 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000110	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000118	4C60604D F0F2F0F4 5D40D9F1 7EF2F3C6 C6C3C1F3 C340D9F2 7EF2F3C3 F5F8F9C4	<--(0204) R1=23FFCA3C R2=23C589D 0 R3=00000000 ERRNO=00000074 ERR NO2=00000000.....
+000138	F040D9F3 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9	
+000158	D5D6F27E F0F0F0F0 F0F0F0F0 00000000 00000000 00000000 00000000	
+000178	00000000 00000000	
+000180	Time 22.41.35.433957 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000190	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000198	C9D9E3D3 D985A296 A4998385 7A7AA1C9 D9E3D3D9 85A296A4 9983854D 5D404040	IRTLResource::.IRTLResource() -->(0059) free(0x24004C20) delete
+0001B8	60606E4D F0F0F5F9 5D408699 85854DF0 A7F2F4F0 F0F4C3F2 F05D4040 40404040	
+0001D8	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+0001F8	84859385 A3854040	
+000200	Time 22.41.35.433959 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000210	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000218	4C60604D F0F0F5F9 5D40D9F1 7EF2F3C6 C6C3C1C2 F040D9F2 7EF2F3C3 F5F8F9C4	<--(0059) R1=23FFCAB0 R2=23C589D 0 R3=23FFD000 ERRNO=00000074 ERR NO2=00000000.....
+000238	F040D9F3 7EF2F3C6 C6C4F0F0 F040C5D9 D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9	
+000258	D5D6F27E F0F0F0F0 F0F0F0F0 00000000 00000000 00000000 00000000	
+000278	00000000 00000000	
+000280	Time 22.41.35.433963 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000290	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000298	C9D9E3D3 D985A296 A4998385 7A7AA1C9 D9E3D3D9 85A296A4 9983854D 5D404040	IRTLResource::.IRTLResource() -->(0204) pthread_mutex_destroy()
+0002B8	60606E4D F0F2F0F4 5D4097A3 88998581 846D94A4 A385A76D 8485A2A3 9996A84D	
+0002D8	5D404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+0002F8	40404040 40404040	
+000300	Time 22.41.35.433967 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000310	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000318	4C60604D F0F2F0F4 5D40D9F1 7EF2F3C6 C6C3C1F3 C340D9F2 7EF2F3C3 F5F8F9C4	<--(0204) R1=23FFCA3C R2=23C589D 0 R3=00000000 ERRNO=00000074 ERR NO2=00000000.....
+000338	F040D9F3 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9	
+000358	D5D6F27E F0F0F0F0 F0F0F0F0 00000000 00000000 00000000 00000000	
+000378	00000000 00000000	
+000380	Time 22.41.35.433972 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000390	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000398	C9D9E3D3 D985A296 A4998385 7A7AA1C9 D9E3D3D9 85A296A4 9983854D 5D404040	IRTLResource::.IRTLResource() -->(0059) free(0x24004C38) delete
+0003B8	60606E4D F0F0F5F9 5D408699 85854DF0 A7F2F4F0 F0F4C3F3 F85D4040 40404040	
+0003D8	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+0003F8	84859385 A3854040	
+000400	Time 22.41.35.433974 Date 2001.08.30 Thread ID... 26C70D0000000000	
+000410	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000418	4C60604D F0F0F5F9 5D40D9F1 7EF2F3C6 C6C3C1C2 F040D9F2 7EF2F3C3 F5F8F9C4	<--(0059) R1=23FFCAB0 R2=23C589D 0 R3=23FFD000 ERRNO=00000074 ERR NO2=00000000.....
+000438	F040D9F3 7EF2F3C6 C6C4F0F0 F040C5D9 D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9	
+000458	D5D6F27E F0F0F0F0 F0F0F0F0 00000000 00000000 00000000 00000000	
+000478	00000000 00000000	
⋮		

Figure 52. Trace table with XPLINK trace table entries 5 and 6.

For more information about the Language Environment trace table format, see
“Understanding the trace table entry (TTE)” on page 116.

Debugging examples of C/C++ routines

This section contains examples that demonstrate the debugging process for C/C++ routines. Important areas of the output are highlighted. Data unnecessary to the debugging examples has been replaced by ellipses.

Divide-by-zero error

Figure 53 illustrates a C program that contains a divide-by-zero error. The code was compiled with RENT so static and external variables need to be calculated from the WSA field. The code was compiled with XREF, LIST and OFFSET to generate a listing, which is used to calculate addresses of functions and data. The code was processed by the binder with MAP to generate a binder map, which is used to calculate the addresses of static and external variables.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
int statint = 73;
int fa;
void funcb(int *pp);

int main(void) {
    int aa, bb=1;
    aa = bb;
    funcb(&aa);
    return(99);
}

void funcb(int *pp) {
    int result;
    fa = *pp;
    result = fa/(statint-73);
    return;
}
```

Figure 53. C routine with a divide-by-zero error

To debug this routine, use the following steps:

1. Locate the Current Condition message in the Condition Information for Active Routines section of the dump. In this example, the message is CEE3209S. The system detected a fixed—point divide exception. This message indicates the error was caused by an attempt to divide by zero. For additional information about CEE3209S, see *z/OS Language Environment Run-Time Messages*.

The traceback section of the dump indicates that the exception occurred at offset X'7E' within function funcb. This information is used along with the compiler-generated Pseudo Assembly Listing to determine where the problem occurred.

If the TEST compiler option is specified, variable information is in the dump. If the GONUMBER compiler option is specified, statement number information is in the dump. Figure 54 on page 173 shows the generated traceback from the dump.

Information for enclave main

Information for thread 0B672E6800000000

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
00015018	CEEHDSP	088AFB00	+000025D2	CEEHDSP	088AFB00	+000025D2		CEEPLPKA	UQ13157	Call
00017288		0B309C18	+0000007E	funcb	0B309C18	+0000007E		*PATHNAM		Exception
000171E0		0B309B28	+0000006E	main	0B309B28	+0000006E		*PATHNAM		Call
000170C8		0876ED36	-08765998	EDCZMINV	0876ED36	-08765998		CEEV003		Call
00017018	CEEBEXT	00CA0508	+0000013C	CEEBEXT	00CA0508	+0000013C		CEEBINIT	UQ09246	Call

Condition Information for Active Routines

Condition Information for (DSA address 00017288)

CIB Address: 00015498

Current Condition:

CEE0198S The termination of a thread was signaled due to an unhandled condition.

Original Condition:

CEE3209S The system detected a fixed-point divide exception.

Location:

Program Unit: Entry: funcb Statement: Offset: +0000007E

Machine State:

ILC..... 0002 Interruption Code..... 0009

PSW..... 078D2400 8B309C98

GPR0..... 00017330 GPR1..... 00017280 GPR2..... 8876EDEA GPR3..... 8B309C62

GPR4..... 80CA05EC GPR5..... 0B3076C8 GPR6..... 00000000 GPR7..... 00000001

GPR8..... 00000000 GPR9..... 80000000 GPR10..... 8876ED2A GPR11..... 80CA0508

GPR12..... 00008910 GPR13..... 00017288 GPR14..... 00017288 GPR15..... 0B309C18

Storage dump near condition, beginning at location: 0B309C86

+000000 0B309C86 4B803052 5860304A 58656000 8E600020 1D685070 D0A058D0 D00458E0 D00C9838 |.....-.....-.....&.....q.|
:
:

Figure 54. Sections of the dump from example C/C++ routine

2. Locate the instruction with the divide-by-zero error in the Pseudo Assembly Listing in Figure 55 on page 174.

The offset (within funcb) of the exception from the traceback (X'7E') reveals the divide instruction: DR r6,r8 at that location. Instructions X'66' through X'80' refer to the result = fa/(statint-73); line of the C/C++ routine.

OFFSET	OBJECT	CODE	LINE#	FILE#	P S E U D O	A S S E M B L Y	L I S T I N G
000000			00015		* funcb	DS	0D
000000	47F0	F026	00015			B	38(,r15)
000004	01C3	C5C5					CEE eyecatcher
000008	0000	00A8					DSA size
00000C	****	****					=A(PPA1-funcb)
000010	47F0	F001	00015			B	1(,r15)
000014	183F		00015			LR	r3,r15
000016	58F0	C31C	00015			L	r15,796(,r12)
00001A	184E		00015			LR	r4,r14
00001C	05EF		00015			BALR	r14,r15
00001E	0000	0000					=F'0'
000022	47F0	303A	00015			B	58(,r3)
000026	90E8	D00C	00015			STM	r14,r8,12(r13)
00002A	58E0	D04C	00015			L	r14,76(,r13)
00002E	4100	E0A8	00015			LA	r0,168(,r14)
000032	5500	C314	00015			CL	r0,788(,r12)
000036	4720	F014	00015			BH	20(,r15)
00003A	5000	E04C	00015			ST	r0,76(,r14)
00003E	9210	E000	00015			MVI	0(r14),16
000042	50D0	E004	00015			ST	r13,4(,r14)
000046	18DE		00015			LR	r13,r14
000048	0530		00015			BALR	r3,r0
00004A					End of Prolog		
00004A	5010	D098	00015		* void funcb(int *pp) {	ST	r1,152(,r13)
00004E	5850	C1F4	00015			L	r5,500(,r12)
			00017		* int result;		
			00017		* fa = *pp;		
000052	5860	D098	00017			L	r6,152(,r13)
000056	5860	6000	00017			L	r6,0(,r6)
00005A	5870	****	00017			L	r7,=Q(fa)
00005E	5860	6000	00017			L	r6,0(,r6)
000062	5065	7000	00017			ST	r6,0(r5,r7)
			00018		* result = fa/(statint-73);		
000066	5880	****	00018			L	r8,=Q(statint)
00006A	5885	8000	00018			L	r8,0(r5,r8)
00006E	4B80	****	00018			SH	r8,=H'73'
000072	5860	****	00018			L	r6,=Q(fa)
000076	5865	6000	00018			L	r6,0(r5,r6)
00007A	8E60	0020	00018			SRDA	r6,32
00007E	1D68		00018			DR	r6,r8
000080	5070	D0A0	00018			ST	r7,160(,r13)
			00019		* return;		
			00020		* }		
000084					Start of Epilog		
000084	58D0	D004	00020			L	r13,4(,r13)
000088	58E0	D00C	00020			L	r14,12(,r13)
00008C	9838	D020	00020			LM	r3,r8,32(r13)
000090	051E		00020			BALR	r1,r14
000092	0707		00020			NOPR	r7
000094					Start of Literals		
000094	0000	0000					=Q(fa)
000098	0000	0000					=Q(statint)
00009C	0049						=H'73'
00009E					End of Literals		
					PPA1: Entry Point Constants		
00009E	10CE	A106					=F'281977094'
0000A2	FFFF	FF9C					=A(PPA2-funcb)
0000A6	0000	0000					=F'0'
0000AA	0000	0000					=F'0'
0000AE	FFE0	0000					=F'-2097152'
0000B2	0000	0000					=F'0'
0000B6	90						AL1(144)
0000B7	0000	00					AL3(0)
0000BA	0240						=H'576'
0000BC	0014						=H'20'
0000BE	0005	****					AL2(5),C'funcb'
0000C6	5000	0049					=F'1342177353'
0000CA	FFFF	FF62					=F'-158'
0000CE	3825	0000					=F'941948928'
0000D2	4007	0042					=F'1074200642'
0000D6	0000						=H'0'
					PPA1 End		
							Flags
							No PPA3
							No EPD
							Register save mask
							Member flags
							Flags
							Callee's DSA use/8
							Flags
							Offset/2 to CDL
							CDL function length/2
							CDL function EP offset
							CDL prolog
							CDL epilog
							CDL end

Figure 55. Pseudo assembly listing

- Verify the value of the divisor `statint`. The procedure specified below is to be used for determining the value of static variables only. If the divisor is an automatic variable, there is a different procedure for finding the value of the variable. For more information about finding automatic variables in a dump, see “Steps for finding automatic variables” on page 137.
Because this routine was compiled with the RENT option, find the WSA address in the Enclave Control Blocks section of the dump. In this example, this address is X'0B3076C8'. Figure 56 shows the WSA address.



Figure 56. C/C++ CAA information in dump

- Routines compiled with the RENT option must also be processed by the binder. The binder produces the Writable Static Map. Find the offset of `statint` in the Writable Static Map in Figure 57. In this example, the offset is X'4'.

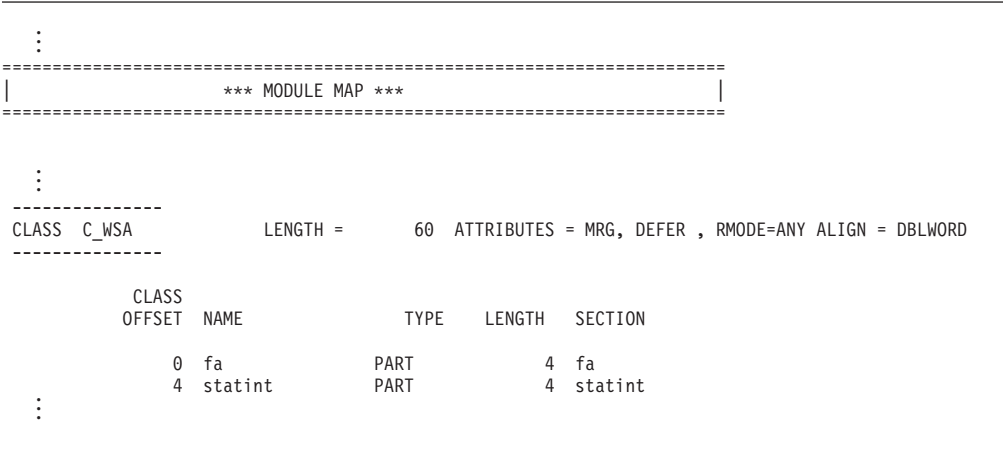


Figure 57. Writable static map

- Add the WSA address of X'0B3076C8' to the offset of `statint`. The result is X'0B3076CC'. This is the address of the variable `statint`, which is in the writable static area. The writable static area is storage allocated by the C/C++ run-time for the C/C++ user, so it is in the user heap. The heap content is displayed in the Enclave Storage section of the dump, shown in Figure 58 on page 176.
- To find the variable `statint` in the heap, locate the closest address listed that is before the address of `statint`. In this case, that address is X'0B3076CB'. Count across X'04' to location X'0B3076CC'. The value at that location is X'49' (that is, `statint` is 73), and hence the fixed point divide exception.

```

:
:
Enclave Storage:
Initial (User) Heap                : 0B325000
:
:
WSA for Program Object(s)
WSA: 0B3076C8
+000000 0B3076C8 00000001 00000049 0B31B6F0 00000000 00000000 00000000 00000000 00000000 | .....0.....
+000020 0B3076E8 00000001 00000001 00000000 00000000 00004650 00000001 00000000 0B31EEEC | .....&.....
+000040 0B307708 00000000 00000000 0B31F018 0B31EDC0 00000000 00000000 0B31F85C 0B31F866 | .....0.....8*..8.
:
:

```

Figure 58. Enclave storage section of dump

Calling a nonexistent non-XPLINK function

Figure 59 demonstrates the error of calling a nonexistent function. This routine was compiled with the compiler options LIST and RENT and was run with the option TERMTHDACT(DUMP). The code was processed by the binder with MAP to generate a binder map, which is used to calculate the addresses of static and external variables.

This routine was not compiled with the TEST(ALL) compiler option. As a result, arguments and variables do not appear in the dump.

The only prelinker option used was MAP.

```

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
void funca(int* aa);
int (*func_ptr)(void)=0;
int main(void) {
    int aa;
    funca(&aa);
    printf("result of funca = %d\n",aa);
    return;
}
void funca(int* aa) {
    *aa = func_ptr();
    return;
}

```

Figure 59. C/C++ example of calling a nonexistent subroutine

To debug this routine, use the following steps:

1. Locate the Current Condition message in the Condition Information for Active Routines section of the dump, shown in Figure 60 on page 177. In this example, the message is CEE3201S The system detected an operation exception (System Completion Code=0C1). This message suggests that the error was caused by an attempt to branch to an unknown address. For additional information about CEE3201S, see *z/OS Language Environment Run-Time Messages*.

The traceback section of the dump indicates that the exception occurred at offset X'-04500616' within function funca. The negative offset indicates that the

offset cannot be used to locate the instruction that caused the error. Another indication of bad data is the value of X'80000004' in the instruction address of the PSW. This address indicates that an instruction in the routine branched outside the bounds of the routine.

```

Information for enclave main

Information for thread 8000000000000000

Registers on Entry to CEE3DMP:

PM..... 0100
GPR0..... 00000000 GPR1..... 00077448 GPR2..... 053AD9AF GPR3..... 853AD514
GPR4..... 00000001 GPR5..... 053AD314 GPR6..... 80077454 GPR7..... 00000000
GPR8..... 00000001 GPR9..... 80000000 GPR10..... 00077470 GPR11..... 000F7490
GPR12..... 0006A520 GPR13..... 000773C8 GPR14..... 80060712 GPR15..... 853F7918
FPR0..... 4D000000 00043C31 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000

:
Traceback:
DSA Addr Program Unit PU Addr PU Offset Entry E Addr E Offset Statement Load Mod Service Status
0006B018 CEEHDSP 0001A5B8 +00001A18 CEEHDSP 0001A5B8 +00001A18 CEEPLPKA Call
00075278 04500618 -04500616 funca 04500618 -04500616 Exception
000751D8 04500558 +00000068 main 04500558 +00000068 Call
000750C8 046ABBAE +000000B0 @@MNINV 046ABBAE +000000B0 CEEEV003 Call
00075018 CEEBBEXT 00007768 +00000138 CEEBBEXT 00007768 +00000138 CEEBINIT Call

Condition Information for Active Routines
Condition Information for (DSA address 00075278)
CIB Address: 0006B3C8
Current Condition:
CEE0198S The termination of a thread was signalled.
Original Condition:
CEE3201S The system detected an operation exception (System Completion Code=0C1).
Location:
Program Unit: Entry: funca Statement: Offset: -04500616
Machine State:
ILC..... 0002 Interruption Code..... 0001
PSW..... 078D0000 80000004
GPR0..... 00075308 GPR1..... 00075270 GPR2..... 00075278 GPR3..... 84500666
GPR4..... 046E5618 GPR5..... 046E5620 GPR6..... 00075268 GPR7..... 00000000
GPR8..... 04500698 GPR9..... 80000000 GPR10..... 846ABBA2 GPR11..... 80007768
GPR12..... 00068520 GPR13..... 00075278 GPR14..... 84500680 GPR15..... 00000000

:
Parameters, Registers, and Variables for Active Routines:
CEEHDSP (DSA address 0006B018):
Saved Registers:
GPR0..... 0001BE62 GPR1..... 0006B32C GPR2..... 00000003 GPR3..... 0006BEC8
GPR4..... 00000000 GPR5..... 0005D8C0 GPR6..... 0001C8BB GPR7..... 00000003
GPR8..... 00000001 GPR9..... 0001C5B6 GPR10..... 0001B5B7 GPR11..... 0001A5B8
GPR12..... 00068520 GPR13..... 0006B018 GPR14..... 8005E712 GPR15..... 846FC918

:
funca (DSA address 00075278):
Saved Registers:
GPR0..... 00075308 GPR1..... 00075270 GPR2..... 00075278 GPR3..... 84500666
GPR4..... 046E5618 GPR5..... 046E5620 GPR6..... 00075268 GPR7..... 00000000
GPR8..... 04500698 GPR9..... 80000000 GPR10..... 846ABBA2 GPR11..... 80007768
GPR12..... 00068520 GPR13..... 00075278 GPR14..... 84500680 GPR15..... 00000000

:

```

Figure 60. Sections of the dump from example C routine

- Find the branch instructions for funca in the listing in Figure 61 on page 178. Notice the BALR r14,r15 instruction at offset X'126'. This branch is part of the instruction.

0000C0		92	funca	DS	0F
:					
00010C	0530	107		BALR	r3,r0
00010E			End of Prolog		
00010E	5840 C1F4	109		L	r4,500(,r12)
000112	5010 D088	110		ST	r1,136(,r13)
			*	return;	
			*	}	
			*		
	00016		*	void funca(int *aa) {	
	00017		*	*aa = func_ptr();	
000116	5870 D088	116		L	r7,136(,r13)
00011A	5860 7000	117		L	r6,0(,r7)
00011E	5870 ****	118		L	r7,=Q(func_ptr)
000122	58F4 7000	119		L	r15,0(r4,r7)
000126	05EF	120		BALR	r14,r15
000128	50F0 6000	125		ST	r15,0(,r6)
:					

Figure 61. Pseudo assembly listing

- Find the offset of FUNC@PTR in the Writable Static Map, shown in Figure 62, as produced by the prelinker.

:				
=====				
			Writable Static Map	
=====				
OFFSET	LENGTH	FILE ID	INPUT NAME	
0	4	00001	FUNC@PTR	
8	18	00001	@STATIC	
:				

Figure 62. Writable static map

- Add the offset of FUNC@PTR (X'0') to the address of WSA (X'46E5618'). The result (X'46E5618') is the address of the function pointer func_ptr in the writable static storage area within the heap. This value is 0, indicating the variable is uninitialized.

Figure 63 shows the sections of the dump.

Enclave Control Blocks:									
:									
:									
:	WSA address.....							046E5618	
:									
Enclave Storage:									
:	Initial (User) Heap:							046E4000	
:									
:	+001600	046E5600	C05FA15A	7B4F5B7C	79000000	00000000	046E4000	00000028	00000000
:	+001620	046E5620	9985A2A4	93A34096	864086A4	95838140	7E406C84	15000000	00000000
:	+001640	046E5640	00000000	00000000	046E4000	00000010	00000000	80054152	046E4000
:	+001660	046E5660	00000000	00000000	00000000	00000000	046E5638	00000000	00000010
:	+001680	046E5680	00000000	00000000	00000000	00000000	00000000	00000000	00000000
:									

Figure 63. Enclave control blocks and storage sections in dump

Calling a nonexistent XPLINK function

Figure 64 demonstrates the error of calling a nonexistent function. This routine was compiled with the compiler options XPLINK, LIST and RENT and was run with the option TERMTHDACT(DUMP).

This routine was not compiled with the TEST(ALL) compile option. As a result, arguments and variables do not appear in the dump.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
void funca(int* aa);
int (*func_ptr)(void)=0;
int main(void) {
    int aa;
    funca(&aa);
    printf("result of funca = %d\n",aa);
    return;
}
void funca(int* aa) {
    *aa = func_ptr();
    return;
}
```

Figure 64. C/C++ example of calling a nonexistent subroutine

To debug this routine, use the following steps:

1. Locate the Current Condition message in the Condition Information for Active Routines section of the dump, shown in Figure 65 on page 180. In this example, the message is CEE3201S The system detected an operation exception (System Completion Code=0C1). This message suggests that the error was caused by an attempt to branch to an unknown address. For additional information about CEE3201S, see *z/OS Language Environment Run-Time Messages*.

The traceback section of the dump indicates that the exception occurred at offset X'-23B553DE7' within function funca. The negative offset indicates that the offset cannot be used to locate the instruction that caused the error. Another indication of bad data is the value of X'80000004' in the instruction address of the PSW. This address indicates that an instruction in the routine branched outside the bounds of the routine.

Information for enclave main

Information for thread 8000000000000000

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
241694E8	CEEHDSR	23D7D3F8	+000038DA	CEEHDSR	23D7D3F8	+000038DA		CEEPLPKA		Call
24169338	CEEHRNUH	23E78028	+00000082	CEEHRNUH	23E78028	+00000082		CEEPLPKA		Call
24209620		23B553E0	-23B553DE	funca	23B553E0	-23B553DE		XEXIST		Exception
242096A0		23B55358	+00000012	main	23B55358	+00000012		XEXIST		Call
24209720		23E7AD10	+000009A4	CEEVROND	23E7AD68	+0000094C		CEEPLPKA		Call
241690E0	EDCZHINV	2413BFC0	+0000009A	EDCZHINV	2413BFC0	+0000009A		CELVH003		Call
24169018	CEEBEXT	00073380	+000001A6	CEEBEXT	00073380	+000001A6		CEEBINIT		Call

Condition Information for Active Routines

Condition Information for (DSA address 24209620)

CIB Address: 24169D08

Current Condition:

CEE0198S The termination of a thread was signaled due to an unhandled condition.

Original Condition:

CEE3201S The system detected an operation exception (System Completion Code=0C1).

Location:

Program Unit: Entry: funca Statement: Offset: -23B553DE

Machine State:

ILC..... 0002 Interruption Code..... 0001

PSW..... 078D2300 80000004

GPR0..... 23C7BFC0 GPR1..... 24209F00 GPR2..... 24169128 GPR3..... 2416912C

GPR4..... 24209620 GPR5..... 00FCD178 GPR6..... 00000000 GPR7..... A3B553FE

GPR8..... A3B55362 GPR9..... 23E7BD0F GPR10..... 00000000 GPR11..... A3E7AD10

GPR12..... 0007FAC0 GPR13..... 241691B8 GPR14..... 23E7AD68 GPR15..... 00000000

Storage dump near condition, beginning at location: 00000000

+000000 00000000 Inaccessible storage.

Information for enclave main

Information for thread 8000000000000000

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
241694E8	CEEHDSR	23D7D3F8	+000038DA	CEEHDSR	23D7D3F8	+000038DA		CEEPLPKA		Call
24169338	CEEHRNUH	23E78028	+00000082	CEEHRNUH	23E78028	+00000082		CEEPLPKA		Call
24209620		23B553E0	-23B553DE	funca	23B553E0	-23B553DE		XEXIST		Exception
242096A0		23B55358	+00000012	main	23B55358	+00000012		XEXIST		Call
24209720		23E7AD10	+000009A4	CEEVROND	23E7AD68	+0000094C		CEEPLPKA		Call
241690E0	EDCZHINV	2413BFC0	+0000009A	EDCZHINV	2413BFC0	+0000009A		CELVH003		Call
24169018	CEEBEXT	00073380	+000001A6	CEEBEXT	00073380	+000001A6		CEEBINIT		Call

Condition Information for Active Routines

Condition Information for (DSA address 24209620)

CIB Address: 24169D08

Current Condition:

CEE0198S The termination of a thread was signaled due to an unhandled condition.

Original Condition:

CEE3201S The system detected an operation exception (System Completion Code=0C1).

Location:

Program Unit: Entry: funca Statement: Offset: -23B553DE

Machine State:

ILC..... 0002 Interruption Code..... 0001

PSW..... 078D2300 80000004

GPR0..... 23C7BFC0 GPR1..... 24209F00 GPR2..... 24169128 GPR3..... 2416912C

GPR4..... 24209620 GPR5..... 00FCD178 GPR6..... 00000000 GPR7..... A3B553FE

GPR8..... A3B55362 GPR9..... 23E7BD0F GPR10..... 00000000 GPR11..... A3E7AD10

GPR12..... 0007FAC0 GPR13..... 241691B8 GPR14..... 23E7AD68 GPR15..... 00000000

Storage dump near condition, beginning at location: 00000000

+000000 00000000 Inaccessible storage.

Figure 65. Sections of the dump from example C routine (Part 1 of 2)

```

Parameters, Registers, and Variables for Active Routines:
CEEHDSR (DSA address 241694E8):
UPSTACK DSA
Saved Registers:
GPR0..... 23C7BFC0 GPR1..... 24169904 GPR2..... 2416AF40 GPR3..... 00000003
GPR4..... 23CA7D20 GPR5..... 00000002 GPR6..... 23CB29D0 GPR7..... 00000000
GPR8..... A3D80A78 GPR9..... 2416A4E7 GPR10.... 23D817FC GPR11.... 23D7D3F8
GPR12.... 0007FAC0 GPR13.... 241694E8 GPR14.... A3D80CD4 GPR15.... A3D615D8
FPR8..... *****
FPR10.... *****
FPR12.... *****
FPR14.... *****
FPR9..... *****
FPR11.... *****
FPR13.... *****
FPR15.... *****

.
.
.

CEEHRNUH (DSA address 24169338):
TRANSITION DSA
Saved Registers:
GPR0..... 23C7BFC0 GPR1..... 00000000 GPR2..... 23CB1CA0 GPR3..... 23CB1D38
GPR4..... 24209620 GPR5..... 24169338 GPR6..... 23CB29D0 GPR7..... 24209620
GPR8..... 940C1000 GPR9..... 00000000 GPR10.... 23CB1D20 GPR11.... 23E78028
GPR12.... 0007FAC0 GPR13.... 24169338 GPR14.... A3E780AC GPR15.... 23D7D3F8
FPR8..... *****
FPR10.... *****
FPR12.... *****
FPR14.... *****
FPR9..... *****
FPR11.... *****
FPR13.... *****
FPR15.... *****

.
.
.

funca (DSA address 24209620):
DOWNSTACK DSA
Saved Registers:
GPR0..... 23C7BFC0 GPR1..... 24209F00 GPR2..... 24169128 GPR3..... 2416912C
GPR4..... 24209620 GPR5..... 00FCD178 GPR6..... 00000000 GPR7..... A3B553FE
GPR8..... A3B55362 GPR9..... 23E7BD0F GPR10.... 00000000 GPR11.... A3E7AD10
GPR12.... 0007FAC0 GPR13.... 241691B8 GPR14.... 23E7AD68 GPR15.... 00000000
FPR8..... *****
FPR10.... *****
FPR12.... *****
FPR14.... *****
FPR9..... *****
FPR11.... *****
FPR13.... *****
FPR15.... *****

```

Figure 65. Sections of the dump from example C routine (Part 2 of 2)

2. Find the branch instructions for funca in the listing in Figure 66 on page 182. Notice the BASR r7,r6 instruction at offset X'001C'. This branch is part of the instruction.

```

      :
      :
000020      :      00015 |      * void funca(int* aa) {
      :
000020      :      @2L0      DS      0D
000020      :      :      =F'12779717'      XPLink entrypoint marker
000024      :      :      =F'12910833'
000028      :      :      =F'-32'
00002C      :      :      =F'128'
000000      :      00015 |      funca      DS      0D
000000      :      :      STM      r5,r7,1924(r4)
000004      :      :      AHI      r4,H'-128'
000008      :      :      End of Prolog

000008      :      5010      48C0      00015 |      ST      r1,aa(,r4,2240)
000008      :      :      *      *aa = func_ptr();
00000C      :      :      L      r6,#Save_ADA_Ptr_2(,r4,2052)
000010      :      :      L      r6,=A(func_ptr)(,r6,24)
000014      :      :      L      r6,func_ptr(,r6,0)
000018      :      :      LM      r5,r6,&ADA_EPA(r6,16)
00001C      :      :      BASR   r7,r6
00001E      :      :      NOP      4
000022      :      :      LR      r0,r3
000024      :      :      L      r6,aa(,r4,2240)
000028      :      :      ST      r0,(*)int(,r6,0)
      :      :      *      return;
      :      :      *      }
00002C      :      :      @2L3      DS      0H

00002C      :      :      Start of Epilog
00002C      :      :      5870      480C      00018 |      L      r7,2060(,r4)
000030      :      :      4140      4080      00018 |      LA     r4,128(,r4)
000034      :      :      07F7      00018 |      BR     r7
      :
      :

```

Figure 66. Pseudo assembly listing

- Find the offset of func_ptr in the Writable Static Map, shown in Figure 67.

```

      :
      :
-----
CLASS   C_WSA              LENGTH =      3C  ATTRIBUTES = MRG, DEFER , RMODE=ANY
                        OFFSET =      0  IN SEGMENT 002      ALIGN = DBLWORD
-----

      CLASS
      OFFSET  NAME              TYPE      LENGTH  SECTION
      0      $PRIV000011      PART      10
      10     exist            PART      28      EXIST
      38     func_ptr          PART      4      func_ptr
      :

```

Figure 67. Writable static map

- Add the offset of func_ptr (X'38') to the address of WSA (X'23C7BFC0'). The result (X'23C7BFF8') is the address of the function pointer func_ptr in the writable static storage area within the heap. This value is 0, indicating the variable is uninitialized.

Figure 68 on page 183 shows the sections of the dump.

```

Enclave Control Blocks:
:
:
DLL Information:
WSA Addr  Module Addr  Thread ID      Use Count  Name
23C7BFC0                                00000001  main

WSA address.....23C7BFC0

:

Enclave Storage:

:

WSA for Program Object(s)
WSA: 23C7BFC0
+000000 23C7BFC0  C36DE6E2 C1404040 40404040 40404040 9985A2A4 93A34096 864086A4 95838140 | C_WSA      result of funca |
+000020 23C7BFE0  7E406C84 15000000 23C7BFF8 00000000 00000060 23EA78A0 00000000 00000000 | =%d.....G.8.....>..... |

:

```

Figure 68. Enclave control blocks and storage sections in dump

Handling dumps written to the z/OS UNIX file system

When a z/OS UNIX C/C++ application program is running in an address space created as a result of a call to `spawnp()`, `vfork()`, or one of the `exec` family of functions, the `SYSDUMP` DD allocation information is not inherited. Even though the `SYSDUMP` allocation is not inherited, a `SYSDUMP` allocation must exist in the parent in order to obtain a HFS storage dump. If the program terminates abnormally while running in this new address space, the kernel causes an unformatted storage dump to be written to an HFS file in the user's working directory. The file is placed in the current working directory or into `/tmp` if the current working directory is not defined. The file name has the following format:

```
/directory/coredump.pid
```

where `directory` is the current working directory or `tmp`, and `pid` is the hexadecimal process ID (PID) for the process that terminated. For details on how to generate the system dump, see "Steps for generating a system dump in a z/OS UNIX shell" on page 77.

To debug the dump, use the MVS Interactive Problem Control System (IPCS). If the dump was written to an HFS file, you must allocate a data set that is large enough and has the correct attributes for receiving a copy of the HFS file. For example, from the `ISPF DATA SET™` UTILITY panel you can specify a volume serial and data set name to allocate. Doing so brings up the `DATA SET INFORMATION` panel for specifying characteristics of the data set to be allocated. The following filled-in panel shows the characteristics defined for the `URCOMP.JRUSL.COREDUMP` dump data set:

```

----- DATA SET INFORMATION -----
Command ==>

Data Set Name . . . : URCOMP.JRUSL.COREDUMP

General Data                      Current Allocation
Management class . . : STANDARD   Allocated cylinders : 30
Storage class . . . : OS390       Allocated extents . : 1
Volume serial . . . : DPXDUI
Device type . . . . : 3380
Data class . . . . .
Organization . . . : PS           Current Utilization
Record format . . . : FB          Used cylinders . . . : 0
Record length . . . : 4160        Used extents . . . : 0
Block size . . . . : 4160
1st extent cylinders: 30
Secondary cylinders : 10
Data set name type  :

Creation date . . . : 2001/08/30
Expiration date . . : ***None***

F1=Help    F2=Split    F3=End    F4=Return    F5=Rfind    F6=Rchange
F7=Up      F8=Down    F9=Swap   F10=Left    F11=Right   F12=Cancel

```

Figure 69. IPCS panel for entering data set information

Fill in the information for your data set as shown, and estimate the number of cylinders required for the dump file you are going to copy.

Use the TSO/E OGET or OCOPY command with the BINARY keyword to copy the file into the data set. For example, to copy the HFS storage dump file coredump.00060007 into the data set URCOMP.JRUSL.COREDUMP just allocated, a user with the user ID URCOMP enters the following command:

```
OGET '/u/urcomp/coredump.00060007' 'urcomp.jrusl.coredump' BINARY
```

For more information on using the copy commands, see *z/OS UNIX System Services User's Guide*.

After you have copied the storage dump file to the data set, you can use IPCS to analyze the dump. Refer to “Formatting and analyzing system dumps” on page 78 for information about formatting Language Environment control blocks.

Multithreading consideration

Certain control blocks are locked while a dump is in progress. For example, a `csnap()` of the file control block would prevent another thread from using or dumping the same information. An attempt to do so causes the second thread to wait until the first one completes before it can continue.

Understanding C/C++ heap information in storage reports

Storage reports that contain specific C/C++ heap information can be generated in two ways:

- By setting the Language Environment RPTSTG(ON) run-time option for Language Environment created heaps
- By issuing a stand-alone call to the C function `__uheapreport()` for user-created heaps.

Details on how to request and interpret the reports are provided in the following sections.

Language Environment storage report with HeapPools statistics

To request a Language Environment storage report set RPTSTG(ON). If the C/C++ application specified the HEAPPOOLS(ON) run-time option, then the storage report displays HeapPools statistics. For a sample storage report showing HeapPools statistics for a multithreaded C/C++ application, see Figure 3 on page 15.

The following describes the C/C++ specific heap pool information.

HeapPools storage statistics

The HEAPPOOLS run-time option controls usage of the heap pools storage algorithm at the enclave level. The heap pools algorithm allows for the definition of one to twelve heap pools, each consisting of a number of storage cells of a specified length.

Usage Note: The use of an alternative Vendor Heap Manager (VHM) overrides the use of the HEAPPOOLS run-time option.

HeapPools statistics:

- Pool *p* size: *ssss*
 - *p* — the number of the pool
 - *ssss* — the cell size specified for the pool.
- Successful Get Heap requests: *xxxx-yyyy n*
 - *xxxx* — the low side of the 8 byte range
 - *yyyy* — the high side of the 8 byte range
 - *n* — the number of requests in the 8 byte range.
- Requests greater than the largest cell size — the number of storage requests that are not satisfied by heap pools.

Note: Values displayed in the HeapPools Statistics report are not serialized when collected, therefore the values are not necessarily exact.

HeapPools summary: The HeapPools Summary displays a report of the HeapPool Statistics and provides suggested percentages for current cell sizes as well as suggested cell sizes.

- Cell Size — the size of the cell specified in the HEAPPOOLS run-time option
- Extent Percent — the cell pool percent specified by the HEAPPOOLS run-time option
- Cells Per Extent — the number of cells per extent. This number is calculated using the following formula:

$$\text{Initial Heap Size} * (\text{Extent Percent}/100) / (8 + \text{Cell Size})$$

with a minimum of four cells.

Note: Having a small number of cells per extent is not recommended since the pool could allocate many extents, which would cause the HeapPool algorithm to perform inefficiently.

- Extents Allocated — the number of times that each pool allocated an extent.
In order to optimize storage usage, the extents allocated should be either one or two. If the number of extents allocated is too high, then increase the percentage for the pool.
- Maximum Cells Used — the maximum number of cells used for each pool.
- Cells In Use — the number of cells that were never freed.

Note: A large number in this field could indicate a storage leak.

- Suggested Percentages for current Cell Sizes — percentages calculated to find the optimal size of the cell pool extent. The calculation is based on the following formula:

$$(\text{Maximum Cells Used} * (\text{Cell Size} + 8) * 100) / \text{Initial Heap Size}$$

With a minimum of 1% and a maximum of 90%

Make sure that your cell pool extents are neither too large nor too small. If your percentages are too large then additional, unreferenced virtual storage will be allocated, thereby causing the program to exhaust the region size. If the percentages are too small then the HeapPools algorithm will run inefficiently.

- Suggested Cell Sizes — sizes that are calculated to optimally use storage (assuming that the application will malloc/free with the same frequency).

Note: The suggested cell sizes are given with no percentages because the usage of each new cell pool size is not known. If there are less than 12 cell sizes calculated and the last calculated cell size is smaller than the largest cell size currently in effect, the largest cell size currently in effect will be used for the last suggested cell size.

For more information about stack and heap storage, see *z/OS Language Environment Programming Guide*.

C function `__uheapreport()` storage report

To generate a user-created heap storage report use the C function, `__uheapreport()`. Use the information in the report to assist with tuning your application's use of the user-created heap. For a description of the information contained in the report, see "HeapPools storage statistics" on page 185.

For more information on the `__uheapreport()` function, see *z/OS XL C/C++ Run-Time Library Reference*. For tuning tips, see *z/OS Language Environment Programming Guide*.

A sample storage report generated by `__uheapreport()` is shown in Figure 70 on page 187.

```
Storage Report for Enclave 08/30/01 11:42:23 AM
Language Environment V01 R03.00
```

```
HeapPools Statistics:
Pool 1 size: 32
  Successful Get Heap requests: 1- 32 11250
Pool 2 size: 128
  Successful Get Heap requests: 97- 128 3306
Pool 3 size: 512
  Successful Get Heap requests: 481- 512 864
Pool 4 size: 2048
  Successful Get Heap requests: 2017- 2048 216
Pool 5 size: 8192
  Successful Get Heap requests: 8161- 8192 54
Pool 6 size: 16384
  Successful Get Heap requests: 16353-16384 27
Requests greater than the largest cell size: 0
HeapPools Summary:
Cell Extent Cells Per Extents Maximum Cells In
Size Percent Extent Allocated Cells Used Use
-----
32 15 3750 1 3750 0
128 15 1102 1 1102 0
512 15 288 1 288 0
2048 15 72 1 72 0
8192 15 18 1 18 0
16384 15 9 1 9 0
-----
Suggested Percentages for current Cell Sizes:
32,15,128,15,512,15,2048,15,8192,15,16384,15
Suggested Cell Sizes:
32,,128,,512,,2048,,8192,,16384,
End of Storage Report
```

Figure 70. Storage report generated by `__uheapreport()`

MEMCHECK VHM memory leak analysis tool

The MEMCHECK VHM memory leak analysis tool is an alternative vendor heap manager used to diagnose memory problems. MEMCHECK VHM performs the following functions:

- check for heap storage leaks, double free, and overlays
- trace user heap storage allocation and deallocation requests

The results are displayed in two reports.

Restrictions

- MEMCHECK VHM works with C/C++ and Enterprise PL/I applications, but is not enabled for COBOL or FORTRAN.
- MEMCHECK VHM and HEAPPOOLS are mutually exclusive. HEAPPOOLS will be ignored when MEMCHECK VHM is active.
- MEMCHECK VHM should not be used in PIPI, PICI, CICS, and SPC environments.

Invoking MEMCHECK VHM

As with any alternate vendor heap manager, you must specify the `dllname` with the environment variable `_CEE_HEAP_MANAGER` to indicate that MEMCHECK VHM will be used to manage the user heap. Since `CEE_HEAP_MANAGER` must be set

before any user code gains control, use the ENVAR run-time option to set the variable or set it inside the file specified by the environment variable, `_CEE_ENVFILE`. The format is

`_CEE_HEAP_MANAGER=dllname`

There are two DLLs associated with MEMCHECK VHM:

CEL4MCHK

31-bit base and xplink.

CELQMCHK

64-bit.

They use the following events:

_VHM_INIT

replaces C-RTL malloc(), calloc(), realloc(), and free() with the corresponding MEMCHECK VHM functions. This event is only at Language Environment Initialization and only called by Language Environment.

_VHM_TERM

terminates Vendor Heap Manager to free the memcheck storage functions. This event is called only by Language Environment at Language Environment Termination.

_VHM_REPORT

generates the Heap Leak Report and the optional Trace Report. This new event will be called by Language Environment at Language Environment Termination and will write the Heap Leak Report (and the optional Trace Report if the `_CEE_MEMCHECK_TRACE` environment variable is active) in the output file name specified in `_CEE_MEMCHECK_OUTFILENAME`. This event can also be called dynamically by the `__vhm_event()` API.

MEMCHECK VHM environment variables

The MEMCHECK VHM environment variables control the tool, the call levels of the Heap Leak Report and Trace Report, the Overlay Analysis, the pad length added in the user heap allocation for overlay analysis, and the output file name for the reports.

They should be activated through the ENVAR run-time option, the file specified by the `_CEE_ENV_FILE` environment variable, or using the export command from the USS shell before any user code gets control (prior to the HLL user exit, static constructors, or main getting control). Setting these environment variables once the user code has begun execution will not activate them and the default values will be used.

_CEE_MEMCHECK_DEPTH

Description: Controls the number of call-levels to be generated on the Heap Leak Report.

Valid settings: integer value : the minimum is 1 and the maximum is MAX_CALL_LEVELS (100). If an invalid value is specified, the default value will be used.

Default: 10.

_CEE_MEMCHECK_OVERLAY

Description: Activates the storage overlays analysis beyond the end of the malloc'd storage.

Valid settings: ON to activate the analysis, OFF to deactivate. If an invalid value is specified, the default value will be used.

Default: OFF

_CEE_MEMCHECK_OVERLAYLEN

Description: Sets the pad length added in the user heap allocation for overlay analysis. This environment variable will be used only if `_CEE_MEMCHECK_OVERLAY` is active.

Valid settings: integer value, multiple of 4 : the minimum is 4 and the maximum is `MAX_OVERLAY_LENGTH` (80). Non-multiples of 4 will be rounded up to the next multiple.

Default: 4

_CEE_MEMCHECK_TRACE

Description: Enables tracing of all heap storage allocation and deallocation and a Trace Report will be generated at Language Environment Termination.

Valid settings: ON to activate the analysis, OFF to deactivate. If an invalid value is specified, the default value will be used.

Default: OFF

_CEE_MEMTRACE_DEPTH

Description: Controls the number of call-levels to be generated in the Trace Report, on each call to a library function that deals with heap. This environment variable will be used only if `_CEE_MEMCHECK_TRACE` is active.

Valid settings: integer value: the minimum is 1 and the maximum is `MAX_CALL_LEVELS` (100). If an invalid value is specified, the default value will be used.

Default: 10

_CEE_MEMCHECK_OUTFILENAME

Description: Sets the name of the fully qualified path name of the file in which the Heap Leak Report and Trace Report should be directed. The report name could be any valid name used in C-RTL `fopen()` function, then it could also generates the reports in a Data Set.

Valid settings: string value. If an invalid value is specified, the default value will be used.

Default: standard error output

MEMCHECK VHM report sample scenario

In this example, the MEMCHECK VHM tool is used by specifying the environment variables from the USS shell. The user specifies a depth of 8 call levels in the Heap Leak Report and 8 call levels in the Trace Report for 31-bit.

1. `Export _CEE_MEMCHECK_DEPTH=8`
specifies the depth to trace on storage requests (written to the Heap Leak Report)
2. `Export _CEE_MEMCHECK_TRACE=ON`
activates the Trace Report option
3. `Export _CEE_MEMTRACE_DEPTH=8`
specifies the depth to trace on storage requests (written to the Trace Report)
4. `Export _CEE_MEMCHECK_OVERLAY=ON`
activates the Overlay analysis option

5. Export `_CEE_HEAP_MANAGER=CEL4MCHK`
activates the tool with the 31-bit DLL (automatically generating the Heap Leak Report)

MEMCHECK VHM report examples

Both reports are written at Language Environment termination (`_VHM_TERM` event). They are written in the output file name specified in `_CEE_MEMCHECK_OUTFILENAME` and are consistent with the format of other Language Environment reports.

The Trace Report will be generated at Language Environment termination (`_VHM_TERM` event) if the `_CEE_MEMCHECK_TRACE` environment variable is active. The report generates the traceback information of all heap storage allocations and deallocations.

```

MEMCHECK
Language Environment V1 R7
TRACE REPORT for enclave main, termination report

DEALLOCATE of storage at 0x25a2ea30
- sequence 12
  Called from: 25a43c78 +00000120 MemFree
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 257f6888 +000002b0 _cterm
  Called from: 05d46788 +0000040c (unknown)
DEALLOCATE of storage at 0x25a2e0c8
- sequence 11
  Called from: 25a43c78 +00000120 MemFree
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 257f6888 +000001bc _cterm
  Called from: 05d46788 +0000040c (unknown)
DEALLOCATE of storage at 0x25a2ecf8
- sequence 10
  Called from: 25a43c78 +00000120 MemFree
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601ae8 +000000b2 function3
  Called from: 25601bb8 +0000008c function2
  Called from: 25601c68 +000000ca function1
  Called from: 25601a60 +00000062 main
ALLOCATE of storage at 0x25a2ecf8 for 5 bytes
- sequence 9
  Called from: 25a44330 +000000fc MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601ae8 +00000084 function3
  Called from: 25601bb8 +0000008c function2
  Called from: 25601c68 +000000ca function1
  Called from: 25601a60 +00000062 main
ALLOCATE of storage at 0x25a2ecd8 for 8 bytes
- sequence 8
  Called from: 25a44330 +000000fc MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601bb8 +0000007e function2
  Called from: 25601c68 +000000ca function1
  Called from: 25601a60 +00000062 main
DEALLOCATE of storage at 0x25a2ecd8
- sequence 7
  Called from: 25a43c78 +00000120 MemFree
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601c68 +000000bc function1
  Called from: 25601a60 +00000062 main
DEALLOCATE of storage at 0x25a2ecd8
- sequence 6
  Called from: 25a43c78 +00000120 MemFree
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601c68 +0000009e function1
  Called from: 25601a60 +00000062 main
ALLOCATE of storage at 0x25a2ecd8 for 4 bytes
- sequence 5
  Called from: 25a44330 +000000fc MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601c68 +0000007e function1
  Called from: 25601a60 +00000062 main
ALLOCATE of storage at 0x25a2ec90 for 48 bytes
- sequence 4
  Called from: 25a44330 +000000fc MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25725c08 +000000a0 dllinit
  Called from: 05d49c88 +000007dc (unknown)

```

Figure 71. Trace report generated by MEMCHECK VHM (Part 1 of 2)

```

ALLOCATE of storage at 0x25a2ea30 for 584 bytes
- sequence 3
  Called from: 25a44330 +000000fc MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 258c6d70 +00000186 setlocale
  Called from: 25862540 +00000059e tzset
  Called from: 257f8d30 +00002df2 _cinit
  Called from: 05d4abb0 +00000cb4 (unknown)
ALLOCATE of storage at 0x25a2e1f8 for 2074 bytes
- sequence 2
  Called from: 25a44330 +000000fc MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 258c6958 +00000070 realloc_name_buffer
  Called from: 258c6d70 +00000132 setlocale
  Called from: 25862540 +00000059e tzset
  Called from: 257f8d30 +00002df2 _cinit
  Called from: 05d4abb0 +00000cb4 (unknown)
ALLOCATE of storage at 0x25a2e0c8 for 280 bytes
- sequence 1
  Called from: 25a44330 +000000fc MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 258c6d70 +000000f6 setlocale
  Called from: 25862540 +00000059e tzset
  Called from: 257f8d30 +00002df2 _cinit
  Called from: 05d4abb0 +00000cb4 (unknown)

```

Figure 71. Trace report generated by MEMCHECK VHM (Part 2 of 2)

The Heap Leak Report will be generated with any remaining entries in the memory leak control block. The allocated entries will be reported as storage leaks, while the deallocated entries will be reported as duplicated deallocations and the overlay entries as overlay damage.

```

MEMCHECK
Language Environment V1 R7
HEAP LEAK REPORT for enclave main, termination report

Total number of ALLOCATE calls = 7
Total number of DEALLOCATE calls = 5

Current number of bytes allocated = 288928
Maximum number of bytes allocated = 289824

Total number of unmatched ALLOCATE calls = 3
Unmatched ALLOCATE of 8 bytes at address 0x25a2ecd8
- sequence 8
  Called from: 25a44330 +000000d2 MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601bb8 +0000007e function2
  Called from: 25601c68 +000000ca function1
  Called from: 25601a60 +00000062 main
Unmatched ALLOCATE of 48 bytes at address 0x25a2ec90
- sequence 4
  Called from: 25a44330 +000000d2 MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25725c08 +000000a0 dllinit
  Called from: 05d49c88 +000007dc (unknown)
Unmatched ALLOCATE of 2074 bytes at address 0x25a2e1f8
- sequence 2
  Called from: 25a44330 +000000d2 MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 258c6958 +00000070 realloc_name_buffer
  Called from: 258c6d70 +00000132 setlocale
  Called from: 25862540 +0000059e tzset
  Called from: 257f8d30 +00002df2 _cinit
  Called from: 05d4abb0 +00000cb4 (unknown)

Total number of unmatched DEALLOCATE calls = 1
Unmatched DEALLOCATE at address 0x25a2ecd8
- sequence 7
  Called from: 25a43c78 +000000f2 MemFree
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601c68 +000000bc function1
  Called from: 25601a60 +00000062 main

Total number of OVERLAY calls = 1
OVERLAY damage using more than 5 bytes requested at address 0x25a2ecf8
  Called from: 25a44330 +000000d2 MemAlloc
  Called from: 05cd9918 +0000005c CEEPGTFN
  Called from: 25601ae8 +00000084 function3
  Called from: 25601bb8 +0000008c function2
  Called from: 25601c68 +000000ca function1
  Called from: 25601a60 +00000062 main

```

Figure 72. Heap Leak Report generated by MEMCHECK VHM

Note: The following names are used within MEMCHECK to denote special cases and may be displayed in any of the reports:

(unknown)

Name of the routine is not known.

(noname)

Routine does not have a name in the PPA section. (For example, module compiled with **compress** option).

(nospace)

Internal memory space reserved by MEMCHECK is full, so name was not saved for the traceback information. No action is needed from the user.

Chapter 5. Debugging COBOL programs

This chapter provides information for debugging applications that contain one or more COBOL programs. It includes information about:

- Determining the source of error
- Generating COBOL listings and the Language Environment dump
- Finding COBOL information in a dump
- Debugging example COBOL programs

Determining the source of error

The following sections describe how you can determine the source of error in your COBOL program. They explain how to simplify the process of debugging COBOL programs by using features such as the DISPLAY statement, declaratives, and file status keys. The following methods for determining errors are covered:

- Tracing program logic
- Finding and handling input/output errors
- Validating data
- Assessing switch problems
- Generating information about procedures

After you have located and fixed any problems in your program, you should delete all debugging aids and recompile it before running it in production. Doing so helps the program run more efficiently and use less storage.

Tracing program logic

You can add DISPLAY statements to help you trace through the logic of the program in a non-CICS environment. If, for example, you determine that the problem appears in an EVALUATE statement or in a set of nested IF statements, DISPLAY statements in each path tell you how the logic flows. You can also use DISPLAY statements to show you the value of interim results.

For example, to check logic flow, you might insert:

```
DISPLAY "ENTER CHECK PROCEDURE".  
      .  
      . (checking procedure routine)  
      .  
DISPLAY "FINISHED CHECK PROCEDURE".
```

to determine whether you started and finished a particular procedure. After you are sure that the program works correctly, comment out the DISPLAY statement lines by putting asterisks in position 7 of the appropriate lines. For a detailed description of the DISPLAY statement, see *Enterprise COBOL for z/OS Language Reference*.

Scope terminators can also help you trace the logic of your program because they clearly indicate the end of a statement. For a detailed description of scope terminators, see *Enterprise COBOL for z/OS Programming Guide* or *COBOL for OS/390 & VM Programming Guide*.

Finding input/output errors

VSAM file status keys can help you determine whether routine errors are due to the logic of your routine or are I/O errors occurring on the storage media.

To use file status keys as a debugging aid, include a test after each I/O statement to check for a value other than 0 in the file status key. If the value is other than 0, you can expect to receive an error message. You can use a nonzero value to indicate how the I/O procedures in the routine were coded. You can also include procedures to correct the error based on the file status key value.

The file status key values and their associated meanings are described in *Enterprise COBOL for z/OS Language Reference* and in *COBOL for OS/390 & VM Language Reference*.

Handling input/output errors

If you have determined that the problem lies in one of the I/O procedures in your program, you can include the `USE EXCEPTION/ERROR` declarative to help debug the problem. If the file does not open, the appropriate `USE EXCEPTION/ERROR` declarative is activated. You can specify the appropriate declarative for the file or for the different open attributes—`INPUT`, `OUTPUT`, `I/O`, or `EXTEND`.

Code each `USE AFTER STANDARD ERROR` statement in a separate section immediately after the Declarative Section keyword of the Procedure Division. See the rules for coding such usage statements in *Enterprise COBOL for z/OS Language Reference* or in *COBOL for OS/390 & VM Language Reference*.

Validating data (class test)

If you suspect that your program is trying to perform arithmetic on nonnumeric data or is somehow receiving the wrong type of data on an input record, you can use the class test to validate the type of data. For a detailed discussion of how to use the class test to check for incompatible data, see *Enterprise COBOL for z/OS Programming Guide* or *COBOL for OS/390 & VM Programming Guide*.

Assessing switch problems

Using `INITIALIZE` or `SET` statements to initialize a table or data item is useful when you suspect that a problem is caused by residual data left in those fields. If your problem occurs intermittently and not always with the same data, the problem could be that a switch is not initialized, but is generally set to the right value (0 or 1). By including a `SET` statement to ensure that the switch is initialized, you can determine whether or not the uninitialized switch is the cause of the problem. For a detailed discussion of how to use the `INITIALIZE` and `SET` statements, see *Enterprise COBOL for z/OS Programming Guide* or *COBOL for OS/390 & VM Programming Guide*.

Generating information about procedures

You can use the `USE FOR DEBUGGING` declarative to include COBOL statements in a COBOL program and specify when they should run. Use these statements to generate information about your program and how it is running.

For example, to check how many times a procedure is run, include a special procedure for debugging (in the `USE FOR DEBUGGING` declarative) that adds 1 to a counter each time control passes to that procedure. The adding-to-a-counter technique can be used as a check for:

- How many times a `PERFORM` ran. This shows you whether the control flow you are using is correct.
- How many times a loop routine actually runs. This tells you whether the loop is running and whether the number you have used for the loop is accurate.

Code each USE FOR DEBUGGING declarative in a separate section in the DECLARATIVES SECTION of the PROCEDURE DIVISION. See the rules for coding them in *Enterprise COBOL for z/OS Language Reference* or in *COBOL for OS/390 & VM Language Reference*.

You can use debugging lines, debugging statements, or both in your program. Debugging lines are placed in your program, and are identified by a D in position 7. Debugging statements are coded in the DECLARATIVES SECTION of the PROCEDURE DIVISION.

- The USE FOR DEBUGGING declaratives must:
 - Be only in the DECLARATIVES SECTION
 - Follow a DECLARATIVES header USE FOR DEBUGGING
- With USE FOR DEBUGGING, the TEST compiler option must have the NONE hook-location suboption specified or the NOTEST compiler option must be specified. The TEST compiler option and the DEBUG run-time option are mutually exclusive, with DEBUG taking precedence.
- Debugging lines must have a D in position 7 to identify them.

To use debugging lines and statements in your program, you must include both:

- WITH DEBUGGING MODE in the SOURCE-COMPUTER paragraph in the ENVIRONMENT DIVISION
- The DEBUG run-time option

Figure 73 on page 198 shows how to use the DISPLAY statement and the USE FOR DEBUGGING declarative to debug a program.

```

Environment Division
Source Computer . . . With Debugging Mode.

:
Data Division.

:
File Section.

Working-Storage Section.

*(among other entries you would need:)

01 Trace-Msg    PIC X(30)
                Value " Trace for Procedure-Name : ".
01 Total        PIC 99 Value Zeros.

*(balance of Working-Storage Section)

Procedure Division.
Declaratives.
Debug-Declar Section.
    Use For Debugging On 501-Some-Routine.
Debug-Declar-Paragraph.
    Display Trace-Msg, Debug-Name, Total.
Debug-Declar-End.
Exit.

End Declaratives.

Begin-Program Section.
:
    Perform 501-Some-Routine.

*(within the module where you want to test, place:)

Add 1 To Total

* (whether you put a period at the end depends on
* where you put this statement.)

```

Figure 73. Example of using the WITH DEBUGGING MODE clause

In the example in Figure 73, portions of a program are shown to illustrate the kind of statements needed to use the USE FOR DEBUGGING declarative. The DISPLAY statement specified in the DECLARATIVES SECTION issues the:

```
Trace For Procedure-Name : 501-Some-Routine nn
```

message every time the PERFORM 501-SOME-ROUTINE runs. The total shown, *nn*, is the value accumulated in the data item named TOTAL.

Another use for the DISPLAY statement technique shown above is to show the flow through your program. You do this by changing the USE FOR DEBUGGING declarative in the DECLARATIVES SECTION to:

```
USE FOR DEBUGGING ON ALL PROCEDURES.
```

and dropping the word TOTAL from the DISPLAY statement.

Using COBOL listings

When you are debugging, you can use one or more of the following listings:

- Sorted Cross-Reference listing
- Data Map listing
- Verb Cross-Reference listing
- Procedure Division Listings

This section gives an overview of each of these listings and specifies the compiler option you use to obtain each listing. For a detailed description of available listings, sample listings, and a complete description of COBOL compiler options, see *Enterprise COBOL for z/OS Programming Guide* or *COBOL for OS/390 & VM Programming Guide*.

Table 19. Compiler-generated COBOL listings and their contents

Name	Contents	Compiler Option
Sorted Cross-Reference Listings	Provides sorted cross-reference listings of DATA DIVISION, PROCEDURE DIVISION, and program names. The listings provide the location of all references to this information.	XREF
Data Map listing	Provides information about the locations of all DATA DIVISION items and all implicitly declared variables. This option also supplies a nested program map, which indicates where the programs are defined and provides program attribute information.	MAP
Verb Cross-Reference listing	Produces an alphabetic listing of all the verbs in your program and indicates where each is referenced.	VBREF
Procedure Division listings	Tells the COBOL compiler to generate a listing of the PROCEDURE DIVISION along with the assembler coding produced by the compiler. The list output includes the assembler source code, a map of the task global table (TGT), information about the location and size of WORKING-STORAGE and control blocks, and information about the location of literals and code for dynamic storage usage.	LIST
Procedure Division listings	Instead of the full PROCEDURE DIVISION listing with assembler expansion information, you can use the OFFSET compiler option to get a condensed listing that provides information about the program verb usage, global tables, WORKING-STORAGE, and literals. The OFFSET option takes precedence over the LIST option. That is, OFFSET and LIST are mutually exclusive; if you specify both, only OFFSET takes effect.	OFFSET

Generating a Language Environment dump of a COBOL program

The two sample programs shown in Figure 74 on page 200 and Figure 75 on page 201 generate Language Environment dumps with COBOL-specific information.

COBOL program that calls another COBOL program

In this example, program COBDUMP1 calls COBDUMP2, which in turn calls the Language Environment dump service CEE3DMP.

```
CBL  TEST(STMT,SYM),RENT
      IDENTIFICATION DIVISION.
        PROGRAM-ID.   COBDUMP1.
        AUTHOR. USER NAME

      ENVIRONMENT DIVISION.

      DATA DIVISION.

      WORKING-STORAGE SECTION.
        01 SOME-WORKINGSTG.
          05 SUB-LEVEL PIC X(80).

        01 SALARY-RECORDA.
          02 NAMEA  PIC X(10).
          02 DEPTA  PIC 9(4).
          02 SALARYA PIC 9(6).

      PROCEDURE DIVISION.
        START-SEC.
          DISPLAY "STARTING TEST COBDUMP1".
          MOVE "THIS IS IN WORKING STORAGE" TO SUB-LEVEL.
          CALL "COBDUMP2" USING SALARY-RECORDA.
          DISPLAY "END OF TEST COBDUMP1"
          STOP RUN.
        END PROGRAM COBDUMP1.
```

Figure 74. COBOL program COBDUMP1 calling COBDUMP2

COBOL program that calls the Language Environment CEE3DMP callable service

In the example in Figure 75 on page 201, program COBDUMP2 calls the Language Environment dump service CEE3DMP.

```

CBL  TEST(STMT,SYM),RENT
      IDENTIFICATION DIVISION.
        PROGRAM-ID.    COBDUMP2.
        AUTHOR. USER NAME

      ENVIRONMENT DIVISION.
      INPUT-OUTPUT SECTION.
      FILE-CONTROL.
        SELECT OPTIONAL IOFSS1 ASSIGN AS-ESDS1DD
          ORGANIZATION SEQUENTIAL ACCESS SEQUENTIAL.

      DATA DIVISION.
      FILE SECTION.
      FD  IOFSS1 GLOBAL.
        1 IOFSS1R PIC X(40).

      WORKING-STORAGE SECTION.
        01 TEMP4.
          05 A-1 OCCURS 2 TIMES.
            10 A-2 OCCURS 2 TIMES.
              15 A-3V PIC X(3).
              15 A-6 PIC X(3).
        77 DMPTITLE PIC X(80).
        77 OPTIONS PIC X(255).
        77 FC PIC X(12).

      LINKAGE SECTION.
        01 SALARY-RECORD.
          02 NAME PIC X(10).
          02 DEPT PIC 9(4).
          02 SALARY PIC 9(6).

      PROCEDURE DIVISION USING SALARY-RECORD.
      START-SEC.
        DISPLAY "STARTING TEST COBDUMP2"
        MOVE "COBOL DUMP" TO DMPTITLE.
        MOVE "XXX" TO A-6(1, 1).
        MOVE "YYY" TO A-6(1, 2).
        MOVE "ZZZ" TO A-6(2, 1).
        MOVE " BLOCKS STORAGE PAGE(55) FILES" TO OPTIONS.
        CALL "CEE3DMP" USING DMPTITLE, OPTIONS, FC.
        DISPLAY "END OF TEST COBDUMP2"
        GOBACK.
      END PROGRAM COBDUMP2.

```

Figure 75. COBOL program COBDUMP2 calling the Language Environment dump service CEE3DMP

Sample Language Environment dump with COBOL-specific information

The call in program COBDUMP2 to CEE3DMP generates a Language Environment dump, shown in Figure 76 on page 202. The dump includes a traceback section, which shows the names of both programs; a section on register usage at the time the dump was generated; and a variables section, which shows the storage and data items for each program. Character fields in the dump are indicated by single quotes. For an explanation of these sections of the dump, see “Finding COBOL information in a dump” on page 203.

```

:
:
CEE3DMP V1 R3.0: COBOL DUMP                                08/30/01 11:49:15 AM                Page:    1

CEE3DMP called by program unit COBDUMP2 at statement 40 (offset +00000430).

Registers on Entry to CEE3DMP:

PM..... 0000
GPR0..... 0D41F838  GPR1..... 00027158  GPR2..... 0D4232C8  GPR3..... 0D302302
GPR4..... 0D301FA0  GPR5..... 00047038  GPR6..... 00000000  GPR7..... 00FCAB00
GPR8..... 0D423160  GPR9..... 0D41F700  GPR10..... 0D302070  GPR11..... 0D302234
GPR12..... 00016A48  GPR13..... 000270C0  GPR14..... 8001E0E2  GPR15..... 8D353858
FPR0..... 00000000  00000000  FPR2..... 00000000  00000000
FPR4..... 00000000  00000000  FPR6..... 00000000  00000000
GPREG STORAGE:
Storage around GPR0 (0D41F838)
-0020 0D41F818  00000000  00000000  00000000  0D423110  0D423160  00000000  0D4230D8  0004C038  .....~.....Q...
+0000 0D41F838  00000000  00000000  0D302450  07FE07FE  00000000  00000000  00001FFF  07FE0000  .....&.....
+0020 0D41F858  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  .....

:
:
Information for enclave COBDUMP1

Information for thread 8000000000000000

Registers on Entry to CEE3DMP:

PM..... 0000
GPR0..... 0D41F838  GPR1..... 00027158  GPR2..... 0D4232C8  GPR3..... 0D302302
GPR4..... 0D301FA0  GPR5..... 00047038  GPR6..... 00000000  GPR7..... 00FCAB00
GPR8..... 0D423160  GPR9..... 0D41F700  GPR10..... 0D302070  GPR11..... 0D302234
GPR12..... 00016A48  GPR13..... 000270C0  GPR14..... 8001E0E2  GPR15..... 8D353858
FPR0..... 00000000  00000000  FPR2..... 00000000  00000000
FPR4..... 00000000  00000000  FPR6..... 00000000  00000000
GPREG STORAGE:
Storage around GPR0 (0D41F838)
-0020 0D41F818  00000000  00000000  00000000  0D423110  0D423160  00000000  0D4230D8  0004C038  .....~.....Q...
+0000 0D41F838  00000000  00000000  0D302450  07FE07FE  00000000  00000000  00001FFF  07FE0000  .....&.....
+0020 0D41F858  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  .....

:
:
Traceback:
DSA Addr  Program Unit  PU Addr  PU Offset  Entry      E Addr  E Offset  Statement  Load Mod  Service  Status
000270C0  COBDUMP2      0D301F68  +00000430  COBDUMP2   0D301F68  +00000430      40  GO              Call
00027018  COBDUMP1      0D300100  +0000033E  COBDUMP1   0D300100  +0000033E      23  GO              Call

Parameters, Registers, and Variables for Active Routines:  COBDUMP2 (DSA address 000270C0):
Saved Registers:
GPR0..... 0D41F838  GPR1..... 00027158  GPR2..... 0D4232C8  GPR3..... 0D302302
GPR4..... 0D301FA0  GPR5..... 00047038  GPR6..... 00000000  GPR7..... 00FCAB00
GPR8..... 0D423160  GPR9..... 0D41F700  GPR10..... 0D302070  GPR11..... 0D302234
GPR12..... 00016A48  GPR13..... 000270C0  GPR14..... 8001E0E2  GPR15..... 8D353858
GPREG STORAGE:
Storage around GPR0 (0D41F838)
-0020 0D41F818  00000000  00000000  00000000  0D423110  0D423160  00000000  0D4230D8  0004C038  .....~.....Q...
+0000 0D41F838  00000000  00000000  0D302450  07FE07FE  00000000  00000000  00001FFF  07FE0000  .....&.....
+0020 0D41F858  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  .....

:
:

```

Figure 76. Sections of the Language Environment dump called from COBDUMP2 (Part 1 of 2)

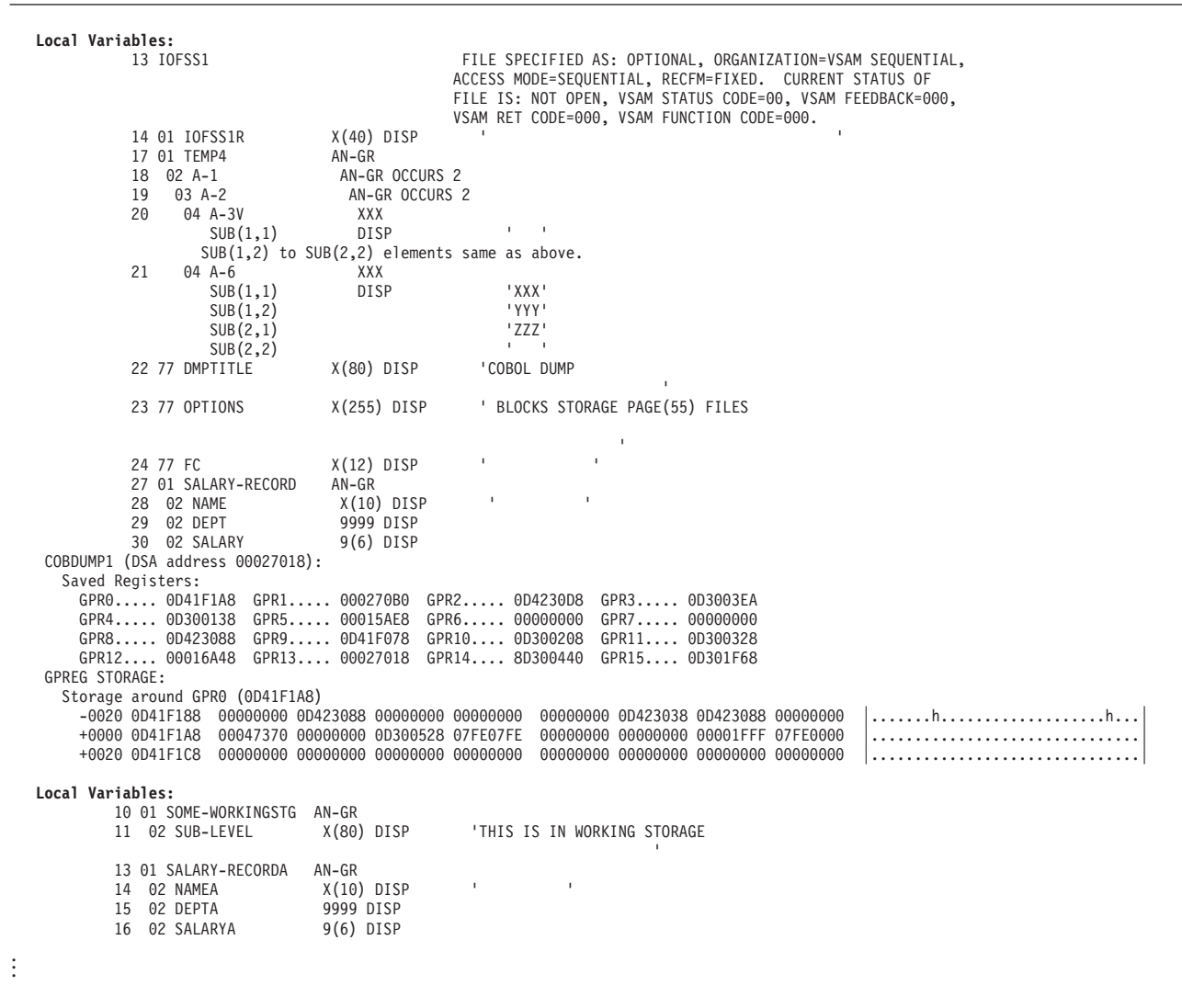


Figure 76. Sections of the Language Environment dump called from COBDUMP2 (Part 2 of 2)

Finding COBOL information in a dump

Like the standard Language Environment dump format, dumps generated from COBOL programs contain:

- Control block information for active programs
- Storage for each active program
- Enclave-level data
- Process-level data

Control block information for active routines

The Control Blocks for Active Routines section of the dump, shown in Figure 77 on page 204, displays the following information for each active COBOL program:

- DSA
- Program name and date/time of compile
- COBOL compiler Version, Release, Modification, and User Level
- COBOL control blocks TGT and CLLE. The layout of the TGT can be found by looking at the compiler listing of the COBOL program. The CLLE is a COBOL

control block that is allocated by the COBOL runtime for each program. The CLLE is dumped for IBM service personnel use.

```

:
:
Control Blocks for Active Routines:
DSA for COBDUMP2: 000270C0
+000000  FLAGS.... 0010      member... 4001      BKC..... 00027018  FWC..... 00027168  R14..... 8001E0E2
+000010  R15..... 8D353858  R0..... 0D41F838  R1..... 00027158  R2..... 0D4232C8  R3..... 0D302302
+000024  R4..... 0D301FA0  R5..... 00047038  R6..... 00000000  R7..... 00FCAB00  R8..... 0D423160
+000038  R9..... 0D41F700  R10..... 0D302070  R11..... 0D302234  R12..... 00016A48  reserved. 00000000
+00004C  NAB..... 00027168  PNAB..... 00000000  reserved. 00000000 000270C0 0D41F700 00101001
+000064  reserved. 00027018  reserved. 000270E4  MODE..... 8D30239A  reserved. 000270EC 000270F4
+000078  reserved. 000270F0  reserved. 000270F8

:
:
Program COBDUMP2 was compiled 08/04/01 11:49:09 AM
COBOL Version = 02 Release = 01 Modification = 01      User Level = ' '
TGT for COBDUMP2: 0D41F700
+000000 0D41F700 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 0D41F720 - +00003F 0D41F73F same as above |.....|
+000040 0D41F740 00000000 00000000 F3E3C7E3 00000000 05000000 42030220 00047038 000187FC |.....3TGT.....g|
+000060 0D41F760 0D41F920 00000001 00000174 00000000 00000000 0D423100 00000000 00000000 |..9.....|
+000080 0D41F780 00016A48 0000021C 00000000 00000000 00000000 00000001 E2E8E2D6 E4E34040 |.....SYSOUT|
+0000A0 0D41F7A0 C9C7E9E2 D9E3C3C4 00000000 00000000 00000000 00000000 00000000 00000000 |IGZSRSTD.....|
+0000C0 0D41F7C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000E0 0D41F7E0 00000000 00000000 0D302064 00000001 0D41F908 00047370 0D3020F3 0D41F83C |.....9.....3..8|
+000100 0D41F800 0D301F68 0D302078 0D41F904 0D30206C 0D41F904 0D423160 00000000 00000000 |.....9...%.9...-.....|
+000120 0D41F820 00000000 0D423110 0D423160 00000000 0D4230D8 0004C038 00000000 00000000 |.....-.....Q.....|
+000140 0D41F840 0D302450 07FE07FE 00000000 00000000 00001FFF 07FE0000 00000000 00000000 |...&.....|
+000160 0D41F860 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000180 0D41F880 - +0001FF 0D41F8FF same as above |.....|
+000200 0D41F900 00000000 0D41F950 40000000 00000000 00000000 0D41FA50 00000001 00000000 |.....9& .....&.....|

:
:
CLLE for COBDUMP2: 00047370
+000000 00047370 C3D6C2C4 E4D4D7F2 00000100 00000000 84810000 0D301F68 0D41F700 00000000 |COBDUMP2.....da.....7.....|
+000020 00047390 00000000 0D41F5AC 0D41F6B8 00047328 00047000 000000C8 000000C0 00000000 |.....5...6.....H.....|

:

```

Figure 77. Control block information for active COBOL routines

Storage for each active routine

The Storage for Active Routines section of the dump, shown in Figure 78 on page 205, displays the following information for each COBOL program:

- Program name
- Contents of the base locators for files, WORKING-STORAGE, LINKAGE SECTION, LOCAL-STORAGE SECTION, variably-located areas, and EXTERNAL data.
- File record contents.
- WORKING-STORAGE, including the base locator for WORKING-STORAGE (BLW) and program class storage.

```

:
:
Storage for Active Routines:
COBDUMP2:
  Contents of base locators for files are:
    0-0004C038

  Contents of base locators for working storage are:
    0-0D423160

  Contents of base locators for the linkage section are:
    0-00000000      1-0D4230D8

  No variably located areas were used in this program.

  No EXTERNAL data was used in this program.

  No object instance data were used in this program.

  No local storage was used in this program.

  No DSA indexes were used in this program.

  No indexes were used in this program.
File record contents for COBDUMP2
ESDS1DD (BLF-0): 0004C038
+000000 0004C038 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000020 0004C058 - +00003F 0004C077 same as above

Working storage for COBDUMP2
BLW-0: 0D423160
+000000 0D423160 000000E7 E7E70000 00E8E8E8 000000E9 E9E90000 00000000 C3D6C2D6 D340C4E4 | ...XXX...YYY...ZZZ.....COBOL DU
+000020 0D423180 D4D74040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 | MP
+000040 0D4231A0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+000060 0D4231C0 40404040 40404040 40C2D3D6 C3D2E240 E2E3D6D9 C1C7C540 D7C1C7C5 4DF5F55D | BLOCKS STORAGE PAGE(55)
+000080 0D4231E0 40C6C9D3 C5E24040 40404040 40404040 40404040 40404040 40404040 40404040 | FILES
+0000A0 0D423200 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+0000C0 0D423220 - +00015F 0D4232BF same as above
+000160 0D4232C0 40404040 40404000 00000000 00000000 00000000 00000000 00000000 00000000 | .....

Program class storage: 0D423100
+000000 0D423100 000001DB 00000000 00000000 0D41F940 00000000 00000000 00000000 00000000 | .....9 .....
+000020 0D423120 C9C7E9E2 D9E3C3C4 00000000 00000000 00000000 00000000 00000000 00000000 | IGZSRITCD.....
+000040 0D423140 E2E8E2D6 E4E34040 00000000 00000000 0E000000 00000000 0F000000 00000000 | SYSOUT .....
+000060 0D423160 000000E7 E7E70000 00E8E8E8 000000E9 E9E90000 00000000 C3D6C2D6 D340C4E4 | ...XXX...YYY...ZZZ.....COBOL DU
+000080 0D423180 D4D74040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 | MP
+0000A0 0D4231A0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+0000C0 0D4231C0 40404040 40404040 40C2D3D6 C3D2E240 E2E3D6D9 C1C7C540 D7C1C7C5 4DF5F55D | BLOCKS STORAGE PAGE(55)
+0000E0 0D4231E0 40C6C9D3 C5E24040 40404040 40404040 40404040 40404040 40404040 40404040 | FILES
+000100 0D423200 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+000120 0D423220 - +0001BF 0D4232BF same as above
+0001C0 0D4232C0 40404040 40404000 00000000 00000000 00000000 00000000 00000000 00000000 | .....

Program class storage: 0D41F940
+000000 0D41F940 000001BE 00000000 0D423100 0004C028 C6C3C200 01020000 FFFFFFFF FFFFFFFF | .....FCB.....
+000020 0D41F960 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 00000000 00000000 | .....
+000040 0D41F980 00000000 00000000 00000000 8001B9E0 8001B9E0 8001B9E0 8D414938 8001B9E0 | .....
+000060 0D41F9A0 8001B9E0 8001B9E0 8D414938 00000000 00000000 00000000 00000000 00000000 | .....
+000080 0D41F9C0 00000000 00000000 00000000 00000000 00000000 00000000 C3D6C2C4 E4D4D7F2 | .....COBDUMP2
+0000A0 0D41F9E0 C5E2C4E2 F1C4C440 00000000 00000000 00000000 0D302194 00000000 00000000 | ESDS1DD .....m.....
+0000C0 0D41FA00 00000000 00000000 00000000 00000000 00000000 00000000 00008800 00000000 | .....h.....
+0000E0 0D41FA20 00000000 00000000 00000028 00000000 00000000 00000000 00000000 00000000 | .....
+000100 0D41FA40 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000120 0D41FA60 - +0001BF 0D41FAFF same as above
Program class storage: 0004C028
+000000 0004C028 0000003F 00000000 0D41F940 00000000 00000000 00000000 00000000 00000000 | .....9 .....
+000020 0004C048 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
:
:

```

Figure 78. Storage for active COBOL programs

Enclave-level data

The Enclave Control Blocks section of the dump, shown in Figure 79 on page 206, displays the following information:

- RUNCOM control block. The RUNCOM is a control block that is allocated by the COBOL runtime to anchor enclave level resources. The RUNCOM is dumped for IBM service personnel use.

- Storage for all run units
- COBOL control blocks FCB, FIB, and GMAREA. The FCB, FIB, and GMAREA are control blocks used for COBOL file processing. These control blocks are dumped for IBM service personnel use.

```

Enclave Control Blocks:
:
RUNCOM: 00047038
+000000 00047038 C3F3D9E4 D5C3D6D4 000002D8 04860000 000159C8 00000001 00005F78 00000000 |C3RUNCOM...Q.f....H.....^.....|
+000020 00047058 00000000 00300100 00047328 00000000 00015AE8 00000000 00000000 00000000 |.....1Y.....|
+000040 00047078 00018A80 0001818C 00000000 000187FC 00000000 00000000 00016A48 00000000 |...a.....g.....|
+000060 00047098 00000000 000473A8 00000000 00000000 00000000 F0F0F0F0 F0F0F0F0 0041F6B8 |...y.....00000000..6.

Enclave Storage:
:
Rununit class storage: 000473A8
+000000 000473A8 000000C0 00000000 00000000 0041F6F0 00000000 00000000 00000000 00000000 |.....60.....|
+000020 000473C8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 000473E8 - +0000BF 00047467 same as above
Rununit class storage: 0041F6F0
+000000 0041F6F0 00000248 00000000 000473A8 00047360 00000000 00000000 00000000 00000000 |.....y...-.....|
+000020 0041F710 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 0041F730 00000000 00000000 00000000 00000000 00000000 F3E3C7E3 00000000 00000000 |.....3TGT...|
+000060 0041F750 05000000 42030220 00047038 000187FC 0041F920 00000001 00000174 00000000 |.....g...9.....|

Rununit class storage: 00047360
+000000 00047360 00000040 00000000 0041F6F0 0041F2A0 C3D6C2C4 E4D4D7F2 00000100 00000000 |... ..60..2.COBDUMP2.....|
+000020 00047380 84810000 00301F68 0041F700 00000000 00000000 0041F5AC 0041F6B8 00047328 |da.....7.....5...6.....|
Rununit class storage: 0041F2A0
+000000 0041F2A0 00000448 00000000 00047360 0041F068 C8C1E340 00000100 00000000 00000000 |.....0.HAT .....|
+000020 0041F2C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 0041F2E0 - +00031F 0041F5BF same as above

Rununit class storage: 0041F068
+000000 0041F068 0000022C 00000000 0041F2A0 00047318 00000000 00000000 00000000 00000000 |.....2.....|
+000020 0041F088 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 0041F0A8 00000000 00000000 00000000 00000000 00000000 F3E3C7E3 00000000 00000000 |.....3TGT...|
+000060 0041F0C8 05000000 60030220 00047038 000187FC 0041F288 00000000 00000064 00000000 |.....g...2h.....|

Rununit class storage: 00047318
+000000 00047318 00000040 00000000 0041F068 00000000 C3D6C2C4 E4D4D7F1 00000100 00000000 |... ..0....COBDUMP1.....|
+000020 00047338 94810000 8D300100 0041F078 00000000 00000000 0041F5A8 0041F6B8 00000000 |ma.....0.....5y..6.....|

File Control Blocks:
FCB for file ESDS1DD in program COBDUMP2: 0041F950
+000000 0041F950 C6C3C200 01020000 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF |FCB.....|
+000020 0041F970 FFFFFFFF 00000000 00000000 00000000 00000000 00000000 00000000 800189E0 |.....|
+000040 0041F990 800189E0 800189E0 8D414938 800189E0 800189E0 800189E0 8D414938 00000000 |.....|
+000060 0041F9B0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....COBDUMP2ESDS1DD .....|
+000080 0041F9D0 00000000 00000000 C3D6C2C4 E4D4D7F2 C5E2C4E2 F1C4C440 00000000 00000000 |.....|
+0000A0 0041F9F0 00000000 00302194 00000000 00000000 00000000 00000000 00000000 00000000 |.....m.....|
+0000C0 0041FA10 00000000 00000000 00000000 00000000 00000000 00000000 00000028 00000000 |.....h.....|
+0000E0 0041FA30 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
FIB for file ESDS1DD in program COBDUMP2: 00302194
+000000 00302194 C6C9C200 0103C5E2 C4E2F1C4 C4400088 8000A000 00008000 00000000 00000028 |FIB...ESDS1DD .h.....|
+000020 003021B4 00010000 00000000 00000000 00000000 00000000 00000000 00302180 00000000 |.....|
+000040 003021D4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 003021F4 - +00007F 00302213 same as above
+000080 00302214 0000C9D6 C6E2E2F1 40404040 40404040 40404040 40404040 40404040 |..10FSS1 |
GMAREA for file ESDS1DD in program COBDUMP2: 00000000
+000000 00000000 Inaccessible storage.

```

Figure 79. Enclave-level data for COBOL programs

Process-level data

The Process Control Block section of the dump, shown in Figure 80 on page 207, displays COBOL process-level control blocks THDCOM, COBCOM, COBVEC, and ITBLK.

In a non-CICS environment, the ITBLK control block only appears when a VS COBOL II program is active. In a CICS environment, the ITBLK control block always appears.

COBOL control blocks THDCOM, COBCOM, COBVEC and ITBLK are dumped for IBM service personnel use.

```

Process Control Blocks:
:
THDCOM: 00018A80
+000000 00018A80 C3F3E3C8 C4C3D6D4 000001E8 81000000 00000100 00000000 00018108 0001818C |C3THDCOM...Va.....a...a.|
+000020 00018AA0 00047038 00000000 C3D6C2C4 E4D4D7F1 00000000 00000000 00000000 00000000 |.....COBOLUMP1.....|
+000040 00018AC0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 00018AE0 - +00007F 00018AFF same as above
:
COBCOM: 00018108
+000000 00018108 C3F3C3D6 C2C3D6D4 00000978 F0F2F0F9 F0F00000 00000000 00000000 00000000 |C3COBCOM...020900.....|
+000020 00018128 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 00018148 00000000 00000000 00000000 00000000 00000000 00000000 906000D6 0001818C |.....0...a.|
+000060 00018168 000187FC 00000100 00820000 00000000 00000000 00000000 00018A80 00000000 |..g.....b.....|
:
COBVEC: 000181BC
+000000 000181BC 0001843C 00018442 00018448 0001844E 00018454 0001845A 00018460 00018466 |..d...d...d...d+.d...d!..d-..d.|
+000020 000181DC 0001846C 00018472 00018478 0001847E 00018484 0001848A 00018490 00018496 |..d%.d...d...d=.dd..d...d..do|
+000040 000181FC 0001849C 000184A2 000184A8 000184AE 000184B4 000184BA 000184C0 000184C6 |..d...ds...dy..d...d...d...d..dF|
+000060 0001821C 000184CC 000184D2 000184D8 000184DE 000184E4 000184EA 000184F0 000184F6 |..d...dK...dQ...d...dU...d...d0...d6|
:

```

Figure 80. Process-level control blocks for COBOL programs

Debugging example COBOL programs

The following examples help demonstrate techniques for debugging COBOL programs. Important areas of the dump output are highlighted. Data unnecessary to debugging has been replaced by vertical ellipses.

Subscript range error

Figure 81 illustrates the error of using a subscript value outside the range of an array. This program was compiled with LIST, TEST(STMT,SYM), and SSRANGE. The SSRANGE compiler option causes the compiler to generate code that checks (during run time) for data that has been stored or referenced outside of its defined area because of incorrect indexing and subscripting. The SSRANGE option takes effect during run time, unless you specify CHECK(OFF) as a run-time option.

The program was run with TERMTHDACT(TRACE) to generate the traceback information shown in Figure 82 on page 208.

```

CBL LIST,SSRANGE,TEST(STMT,SYM)
ID DIVISION.
PROGRAM-ID. COBOLX.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 J PIC 9(4) USAGE COMP.
01 TABLE-X.
02 SLOT PIC 9(4) USAGE COMP OCCURS 8 TIMES.
PROCEDURE DIVISION.
MOVE 9 TO J.
MOVE 1 TO SLOT (J).
GOBACK.

```

Figure 81. COBOL example of moving a value outside an array range

To understand the traceback information and debug this program, use the following steps:

1. Locate the current error message in the Condition Information for Active Routines section of the Language Environment traceback, shown in Figure 82 on page 208. The message is IGZ0006S The reference to table SLOT by verb

number 01 on line 000011 addressed an area outside the region of the table. The message indicates that line 11 was the current COBOL statement when the error occurred. For more information about this message, see *z/OS Language Environment Run-Time Messages*.

2. Statement 11 in the traceback section of the dump occurred in program COBOLX.

CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.

08/30/01 11:48:58 AM

Page: 1

Information for enclave COBOLX

Information for thread 8000000000000000

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
0002A768	CEEHDSP	0D3386F0	+00003032	CEEHDSP	0D3386F0	+00003032		CEEPLPKA		Call
0002A5D0	CEEHSGLT	0D344250	+0000005C	CEEHSGLT	0D344250	+0000005C		CEEPLPKA		Exception
0002A0B8	IGZCMSG	0D400BF8	+0000038C	IGZCMSG	0D400BF8	+0000038C		IGZCPAC		Call
0002A018	COBOLX	00007808	+00000286	COBOLX	00007808	+00000286	11	G0		Call

Condition Information for Active Routines

Condition Information for CEEHSGLT (DSA address 0002A5D0)

CIB Address: 0002ADE0

Current Condition:

IGZ0006S The reference to table SL0T by verb number 01 on line 000011 addressed an area outside the region of the table.

Location:

Program Unit: CEEHSGLT Entry: CEEHSGLT Statement: Offset: +0000005C

Storage dump near condition, beginning at location: 0D34429C

+000000 0D34429C F010D20B D0801000 58A0C2B8 58F0A01C 05EFD20B D098B108 41A0D098 50A0D08C |0.K.....B..0....K..q....q&...|

Figure 82. Sections of Language Environment dump for COBOLX (Part 1 of 2)

```

Parameters, Registers, and Variables for Active Routines:
CEEHDSP (DSA address 0002A768):
  Saved Registers:
    GPR0..... 0D33BB6C  GPR1..... 0002AB80  GPR2..... 00000001  GPR3..... 00000003
    GPR4..... 00000008  GPR5..... 0002ADE0  GPR6..... 00020038  GPR7..... 0002B767
    GPR8..... 0D33B6ED  GPR9..... 0D33A6EE  GPR10.... 0D3396EF  GPR11.... 8D3386F0
    GPR12.... 00019A48  GPR13.... 0002A768  GPR14.... 800210E2  GPR15.... 8D34E858
  GPREG STORAGE:
    Storage around GPR0 (0D33BB6C)
      -0020 0D33BB4C  0D33BB78  0D33BBBC  0D33BB80  0D33BBC0  0D33BBB0  00000000  00000001  00000002  .....
      +0000 0D33BB6C  00000003  00000004  00000006  00000007  00000008  00000009  0000000A  0000000B  .....
      +0020 0D33BB8C  0000000D  0000000E  00000010  00000015  00000017  00000064  00000069  00000080  .....
:
CEEHSGLT (DSA address 0002A5D0):
  Saved Registers:
    GPR0..... 000079B8  GPR1..... 0002A290  GPR2..... 0002A290  GPR3..... 00000000
    GPR4..... 00020038  GPR5..... 00020038  GPR6..... 0002A3D4  GPR7..... 00000005
    GPR8..... 0001BA80  GPR9..... 00009140  GPR10.... 00020038  GPR11.... 8D344250
    GPR12.... 00019A48  GPR13.... 0002A5D0  GPR14.... 800210CE  GPR15.... 8D343AA8
  GPREG STORAGE:
    Storage around GPR0 (000079B8)
      -0020 00007998  08000004  00224000  00000006  C0000140  00040800  00040028  02400002  08000004  .....
      +0000 000079B8  001F03C0  00060800  0004001C  40000000  0040C000  01400006  08000004  002202C0  .....
      +0020 000079D8  00060800  00040022  183F4100  10A05500  C00C05F0  47D0F00C  58F0C300  05EF181F  .....0..0..0C.....
:
IGZCMSG (DSA address 0002A0B8):
  Saved Registers:
    GPR0..... 000079B8  GPR1..... 0002A290  GPR2..... 0002A290  GPR3..... 00000000
    GPR4..... 00020038  GPR5..... 00020038  GPR6..... 0002A3D4  GPR7..... 00000005
    GPR8..... 0001BA80  GPR9..... 00009140  GPR10.... 0004A038  GPR11.... 8D400BF8
    GPR12.... 00019A48  GPR13.... 0002A0B8  GPR14.... 8002463E  GPR15.... 8D344250
  GPREG STORAGE:
    Storage around GPR0 (000079B8)
      -0020 00007998  08000004  00224000  00000006  C0000140  00040800  00040028  02400002  08000004  .....
      +0000 000079B8  001F03C0  00060800  0004001C  40000000  0040C000  01400006  08000004  002202C0  .....
      +0020 000079D8  00060800  00040022  183F4100  10A05500  C00C05F0  47D0F00C  58F0C300  05EF181F  .....0..0..0C.....
:
COBOLX (DSA address 0002A018):
  Saved Registers:
    GPR0..... 0002A0B8  GPR1..... 0000799E  GPR2..... 00000010  GPR3..... 0001B7FC
    GPR4..... 00007840  GPR5..... 00018AE8  GPR6..... 00000000  GPR7..... 00000000
    GPR8..... 000093A0  GPR9..... 00009140  GPR10.... 00007910  GPR11.... 00007A3E
    GPR12.... 00007904  GPR13.... 0002A018  GPR14.... 80007A90  GPR15.... 8D400BF8
  GPREG STORAGE:
    Storage around GPR0 (0002A0B8)
      -0020 0002A098  0D30021C  0D300218  0D300220  000179A4  00000000  00000000  00000000  00000000  .....u.....
      +0000 0002A0B8  00101001  0002A018  0002A5D0  8002463E  8D344250  000079B8  0002A290  0002A290  .....v.....&.....s...s.
      +0020 0002A0D8  00000000  00020038  00020038  0002A3D4  00000005  0001BA80  00009140  0004A038  .....tM.....j ....
:
Local Variables:
      6 77 J          9999 COMP          00009
      7 01 TABLE-X  AN-GR
      8 02 SLOT      9999 OCCURS 8
      SUB(1)        COMP          00000

```

Figure 82. Sections of Language Environment dump for COBOLX (Part 2 of 2)

- Find the statement on line 11 in the listing for program COBOLX, shown in Figure 83 on page 210. This statement moves the 1 value to the array SLOT (J).

```

PP 5648-A25 IBM COBOL for OS/390 & VM 2.1.1          COBOLX   Date 11/04/1999   Time 11:48:54   Page    3
LineID  PL SL  ----*A-1-B-+----2-+----3-+----4-+----5-+----6-+----7-|-+----8   Map and Cross Reference
/* COBOLX
000001      ID DIVISION.
000002      PROGRAM-ID. COBOLX.
000003      ENVIRONMENT DIVISION.
000004      DATA DIVISION.
000005      WORKING-STORAGE SECTION.
000006      77 J    PIC 9(4) USAGE COMP.
000007      01 TABLE-X.
000008          02 SLOT PIC 9(4) USAGE COMP OCCURS 8 TIMES.
000009      PROCEDURE DIVISION.
000010          MOVE 9 TO J.
000011          MOVE 1 TO SLOT (J).
000012          GOBACK.
*/ COBOLX
:
```

Figure 83. COBOL listing for COBOLX

- Find the values of the local variables in the Parameters, Registers, and Variables for Active Routines section of the traceback, shown in Figure 82 on page 208. J, which is of type PIC 9(4) with usage COMP, has a 9 value. J is the index to the array SLOT.

The array SLOT contains eight positions. When the program tries to move a value into the J or 9th element of the 8-element array named SLOT, the error of moving a value outside the area of the array occurs.

Calling a nonexistent subroutine

Figure 84 demonstrates the error of calling a nonexistent subroutine in a COBOL program. In this example, the program COBOLY was compiled with the compiler options LIST, MAP and XREF. The TEST option was also specified with the suboptions NONE and SYM. Figure 84 shows the program.

```

CBL  LIST,MAP,XREF,TEST(NONE,SYM)
      ID DIVISION.
      PROGRAM-ID. COBOLY.
      ENVIRONMENT DIVISION.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      77 SUBNAME PIC X(8) USAGE DISPLAY VALUE 'UNKNOWN'.
      PROCEDURE DIVISION.
          CALL SUBNAME.
          GOBACK.
```

Figure 84. COBOL example of calling a nonexistent subroutine

To understand the traceback information and debug this program, use the following steps:

- Locate the error message for the original condition under the Condition Information for Active Routines section of the dump, shown in Figure 85 on page 211. The message is CEE3501S The module UNKNOWN was not found. For more information about this message, see *z/OS Language Environment Run-Time Messages*.
- Note the sequence of calls in the Traceback section of the dump. COBOLY called IGZCFCC; IGZCFCC (a COBOL library subroutine used for dynamic calls) called IGZCLDL; then IGZCLDL (a COBOL library subroutine used to load library routines) called CEESGLT, a Language Environment condition handling routine.

This sequence indicates that the exception occurred in IGZCLDL when COBOLY was attempting to make a dynamic call. The call statement in COBOLY is located at offset +00000338.

```
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.      08/30/01 5:17:40 PM      Page:      1

Information for enclave COBOLY

Information for thread 8000000000000000

Traceback:
DSA Addr  Program Unit  PU Addr  PU Offset  Entry      E Addr  E  Offset  Statement  Load Mod  Service  Status
0002A5A8  CEEHDSP      0D3386F0 +000028DE CEEHDSP    0D3386F0 +000028DE CEEPLPKA    Call
0002A410  CEEHSGLT     0D344250 +0000005C CEEHSGLT   0D344250 +0000005C CEEPLPKA    Exception
0002A2A8  IGZCLDL      0D3FF098 +0000011A IGZCLDL    0D3FF098 +0000011A IGZCPAC     Call
0002A0C0  IGZCFCC      0001E128 +00000398 IGZCFCC    0001E128 +00000398 IGZCFCC     Call
0002A018  COBOLY       000077E0 +00000338 COBOLY     000077E0 +00000338      8  G0      Call

Condition Information for Active Routines
Condition Information for CEEHSGLT (DSA address 0002A410)
CIB Address: 0002AC20
Current Condition:
CEE0198S The termination of a thread was signaled due to an unhandled condition.
Original Condition:
CEE3501S The module UNKNOWN was not found.
Location:
Program Unit: CEEHSGLT Entry: CEEHSGLT Statement: Offset: +0000005C
Storage dump near condition, beginning at location: 0D34429C
+000000 0D34429C F010D20B D0801000 58A0C2B8 58F0A01C 05EFD20B D098B108 41A0D098 50A0D08C |0.K.....B..0....K..q....q&...|
:
```

Figure 85. Sections of Language Environment dump for COBOLY

- Use the offset of X'338' from the COBOL listing, shown in Figure 86 on page 212, to locate the statement that caused the exception in the COBOLY program. At offset X'338' is an instruction for statement 8. Statement 8 is a call with the identifier SUBNAME specified.

```

:
PP 5648-A25 IBM COBOL for OS/390 & VM 2.1.1 COBOLY Date 11/08/1999 Time 17:17:35 Page 11
0002A8 START EQU *
0002A8 183F LR 3,15
0002AA 4100 10A8 LA 0,168(0,1)
0002AE 5500 C00C CL 0,12(0,12)
0002B2 05F0 BALR 15,0
0002B4 47D0 F00C BC 13,12(0,15)
0002B8 58F0 C300 L 15,768(0,12)
0002BC 05EF BALR 14,15
0002BE 181F LR 1,15
0002C0 50D0 1004 ST 13,4(0,1)
0002C4 5000 104C ST 0,76(0,1)
0002C8 D203 1000 3058 MVC 0(4,1),88(3)
0002CE 18D1 LR 13,1
0002D0 58C0 90E8 L 12,232(0,9) TGTFIXD+232
0002D4 1812 LR 1,2
0002D6 50D0 D058 ST 13,88(0,13)
0002DA 5090 D05C ST 9,92(0,13)
0002DE 58A0 C004 L 10,4(0,12) CBL=1
0002E2 5880 9128 L 8,296(0,9) BLW=0
0002E6 D203 90EC A010 MVC 236(4,9),16(10) TGTFIXD+236 PGMLIT AT +8
0002EC BF2F 9208 ICM 2,15,520(9) IPCB=1+16
0002F0 58B0 C008 L 11,8(0,12) PBL=1
0002F4 4780 B000 BC 8,0(0,11) GN=7(000306)
0002F8 5830 905C L 3,92(0,9) TGTFIXD+92
0002FC 58F0 30F4 L 15,244(0,3) V(IGZCMMSG )
000300 4110 A180 LA 1,384(0,10) PGMLIT AT +376
000304 05EF BALR 14,15
000306 EQU *
000306 5A20 C000 A 2,0(0,12) SYSLIT AT +0
00030A 5020 9208 ST 2,520(0,9) IPCB=1+16
00030E 9640 91F8 OI 504(9),X'40' IPCB=1
000008 *
000008 CALL
000312 D207 D098 8000 MVC 152(8,13),0(8) TS2=0 SUBNAME
000318 DC07 D098 A01A TR 152(8,13),26(10) TS2=0 PGMLIT AT +18
00031E D203 D0A0 A15A MVC 160(4,13),346(10) TS2=8 PGMLIT AT +338
000324 4120 D098 LA 2,152(0,13) TS2=0
000328 5020 D0A4 ST 2,164(0,13) TS2=12
00032C 4110 D0A0 LA 1,160(0,13) TS2=8
000330 5820 905C L 2,92(0,9) TGTFIXD+92
000334 58F0 2100 L 15,256(0,2) V(IGZCFCC )
000338 05EF BALR 14,15
00033A 5830 9124 L 3,292(0,9) BL=1
00033E 40F0 3000 STH 15,0(0,3) RETURN-CODE
000009 GOBACK
000342 47F0 B052 BC 15,82(0,11) GN=2(000358)
000346 9120 9054 TM 84(9),X'20' TGTFIXD+84
00034A 47E0 B052 BC 14,82(0,11) GN=2(000358)
00034E 58F0 20F4 L 15,244(0,2) V(IGZCMMSG )
000352 4110 A16E LA 1,366(0,10) PGMLIT AT +358
000356 05EF BALR 14,15
000358 EQU *
000358 5840 9208 L 4,520(0,9) IPCB=1+16
00035C 5840 C000 S 4,0(0,12) SYSLIT AT +0
000360 5040 9208 ST 4,520(0,9) IPCB=1+16
000364 9140 9055 TM 85(9),X'40' TGTFIXD+85
000368 47E0 B070 BC 14,112(0,11) GN=8(000376)
00036C 4110 0008 LA 1,8(0,0)
000370 58F0 2020 L 15,32(0,2) V(IGZCCTL )
000374 05EF BALR 14,15
000376 EQU *
000376 9128 9054 TM 84(9),X'28' TGTFIXD+84
00037A 4770 B08A BC 7,138(0,11) GN=9(000390)
00037E 48F0 3000 LH 15,0(0,3) RETURN-CODE
000382 58D0 D004 L 13,4(0,13)
000386 58E0 D00C L 14,12(0,13)
00038A 980C D014 LM 0,12,20(13)
00038E 07FE BCR 15,14

```

Figure 86. COBOL Listing for COBOLY (Part 1 of 2)

000390		GN=9	EQU	*		
000390	D20B D098 A14E		MVC	152(12,13),334(10)	TS2=0	PGMLIT AT +326
000396	4840 3000		LH	4,0(0,3)	RETURN-CODE	
00039A	5040 D0A4		ST	4,164(0,13)	TS2=12	
00039E	4110 D098		LA	1,152(0,13)	TS2=0	
0003A2	58F0 2224		L	15,548(0,2)	V(IGZETRM)	
0003A6	05EF		BALR	14,15		
:						
:						

Figure 86. COBOL Listing for COBOLY (Part 2 of 2)

- Find the value of the local variables in the Parameters, Registers, and Variables for Active Routines section of the dump, shown in Figure 87. Notice that the value of SUBNAME with usage DISP, has a value of 'UNKNOWN'.

Correct the problem by either changing the subroutine name to one that is defined, or by ensuring that the subroutine is available at compile time.

:	
:	Parameters, Registers, and Variables for Active Routines:
:	:
:	COBOLY (DSA address 0002A018):
	Saved Registers:
	GPR0..... 0002A0C0 GPR1..... 0002A0B8 GPR2..... 0001B7FC GPR3..... 000077E0
	GPR4..... 00007818 GPR5..... 00018AE8 GPR6..... 00000000 GPR7..... 00000000
	GPR8..... 000093B0 GPR9..... 00009150 GPR10..... 000078E8 GPR11..... 00007AE6
	GPR12..... 000078DC GPR13..... 0002A018 GPR14..... 80007B1A GPR15..... 8001E128
	GPREG STORAGE:
	Storage around GPR0 (0002A0C0)
	-0020 0002A0A0 0D300220 000179A4 00000000 00000000 E4D5D2D5 D6E6D540 A2080000 0002A0B0 u.....UNKNOWN s.....
	+0000 0002A0C0 00102001 0002A018 00000000 8001E4C2 8D3FF098 0002A2A8 0002A260 0001B7FC UB..0q..sy..s-....
	+0020 0002A0E0 000077E0 0002A0B8 0001B7FC 00000000 0002A250 0001BA80 00009150 0004A038 s&.....j&....
:	
:	Local Variables:
	6 77 SUBNAME X(8) DISP 'UNKNOWN '

Figure 87. Parameters, registers, and variables for active routines section of dump for COBOLY

Divide-by-zero error

The following example demonstrates the error of calling an assembler routine that tries to divide by zero. Both programs were compiled with TEST(STMT,SYM) and run with the TERMTHDACT(TRACE) run-time option. Figure 88 on page 214 shows the main COBOL program (COBOLZ1), the COBOL subroutine (COBOLZ2), and the assembler routine.

```

[Main Program]

CBL  TEST(STMT,SYM),DYN,XREF(FULL),MAP
      ID DIVISION.
      PROGRAM-ID. COBOLZ1.
      ENVIRONMENT DIVISION.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      77 D-VAL PIC 9(4) USAGE COMP VALUE 0.
      PROCEDURE DIVISION.
          CALL "COBOLZ2" USING D-VAL.
          GOBACK.

[Subroutine]

CBL  TEST(STMT,SYM),DYN,XREF(FULL),MAP
      ID DIVISION.
      PROGRAM-ID. COBOLZ2.
      ENVIRONMENT DIVISION.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      77 DV-VAL PIC 9(4) USAGE COMP.
      LINKAGE SECTION.
      77 D-VAL PIC 9(4) USAGE COMP.
      PROCEDURE DIVISION USING D-VAL.
          MOVE D-VAL TO DV-VAL.
          CALL "ASSEMZ3" USING DV-VAL.
          GOBACK.

[Assembler Routine]

      PRINT NOGEN
ASSEMZ3 CEEENTRY MAIN=NO,PPA=MAINPPA
      LA    5,2348          Low order part of quotient
      SR    4,4             Hi order part of quotient
      L     6,0(1)          Get pointer to divisor
      LA    6,0(6)          Clear hi bit
      D     4,0(6)          Do division
      CEETERM RC=0          Terminate with return code zero
*
MAINPPA CEEPPA              Constants describing the code block
      CEEDSA              Mapping of the Dynamic Save Area
      CEECAA              Mapping of the Common Anchor Area
      END  ASSEMZ3

```

Figure 88. Main COBOL program, COBOL subroutine, and assembler routine

To debug this application, use the following steps:

1. Locate the error message for the current condition in the Condition Information section of the dump, shown in Figure 89 on page 215. The message is CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

For additional information about this message, see *z/OS Language Environment Run-Time Messages*.

2. Note the sequence of calls in the call chain. COBOLZ1 called IGZCFCC, which is a COBOL library subroutine used for dynamic calls; IGZCFCC called COBOLZ2; COBOLZ2 then called IGZCFCC; and IGZCFCC called ASSEMZ3. The exception occurred at this point, resulting in a call to CEEHDSP, a Language Environment condition handling routine.

The call to ASSEMZ3 occurred at statement 11 of COBOLZ2. The exception occurred at offset +64 in ASSEMZ3.

Information for enclave COBOLZ1

Information for thread 8000000000000000

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
0002A5E0	CEEHDS	0D3386F0	+00003032	CEEHDS	0D3386F0	+00003032		CEEPLPKA		Call
0002A560	ASSEMZ3	0D315048	+00000064	ASSEMZ3	0D315048	+00000064		ASSEMZ3		Exception
0002A378	IGZCFCC	0001E128	+00000270	IGZCFCC	0001E128	+00000270		IGZCFCC		Call
0002A2C0	COBOLZ2	0004C758	+0000026E	COBOLZ2	0004C758	+0000026E	11	COBOLZ2		Call
0002A0D8	IGZCFCC	0001E128	+00000270	IGZCFCC	0001E128	+00000270		IGZCFCC		Call
0002A018	COBOLZ1	000077C0	+00000258	COBOLZ1	000077C0	+00000258	8	60		Call

Condition Information for Active Routines

Condition Information for ASSEMZ3 (DSA address 0002A560)

CIB Address: 0002AC58

Current Condition:

CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

Location:

Program Unit: ASSEMZ3 Entry: ASSEMZ3 Statement: Offset: +00000064

Machine State:

ILC..... 0004 Interruption Code..... 0009

PSW..... 078D0000 8D3150B0

GPR0..... 0002A5E0 GPR1..... 0002A360 GPR2..... 0D41AA84 GPR3..... 0004E27C

GPR4..... 00000000 GPR5..... 0000092C GPR6..... 0004E3B0 GPR7..... 0002A360

GPR8..... 00000000 GPR9..... 0004E148 GPR10..... 0004A038 GPR11..... 8D315048

GPR12..... 00019A48 GPR13..... 0002A560 GPR14..... 8001E39A GPR15..... 8D315048

Storage dump near condition, beginning at location: 0D31509C

+000000 0D31509C 10184150 092C1B44 58610000 41660000 5D460000 58F0B0F0 5800B0F0 58DD0004 |...&...../.....)....0.0...0....|

Parameters, Registers, and Variables for Active Routines:

:

COBOLZ2 (DSA address 0002A2C0):

Saved Registers:

GPR0..... 0002A378 GPR1..... 0002A368 GPR2..... 0001B7FC GPR3..... 0004E27C

GPR4..... 0002A360 GPR5..... 0001B1BC GPR6..... 0004A370 GPR7..... 00FCAB00

GPR8..... 0004E3B0 GPR9..... 0004E148 GPR10..... 0004C860 GPR11..... 0004C96E

GPR12..... 0004C854 GPR13..... 0002A2C0 GPR14..... 8004C9C8 GPR15..... 8001E128

GPREG STORAGE:

Storage around GPR0 (0002A378)

-0020 0002A358 0002A378 0D41ABA8 8004E3B0 00000000 96080000 0004C870 0004E27C 0002A360 |..t....y..T.....o.....H...S@..t-

+0000 0002A378 00102401 0002A2C0 0002A560 8001E39A 8D315048 0002A560 0002A360 0D41AA84 |.....s...v-.T...&...v-.t-.d

+0020 0002A398 0004E27C 0002A368 0004E27C 0004A3B8 0002A360 00000000 0004E148 0004A038 |..S@..t...S@..t...t-.....

:

Local Variables:

6 77 DV-VAL 9999 COMP 00000

8 77 D-VAL 9999 COMP 00000

:

COBOLZ1 (DSA address 0002A018):

Saved Registers:

GPR0..... 0002A0D8 GPR1..... 0002A0C8 GPR2..... 0001B7FC GPR3..... 00009280

GPR4..... 0002A0C0 GPR5..... 00018AE8 GPR6..... 00000000 GPR7..... 00000000

GPR8..... 000093B0 GPR9..... 00009150 GPR10..... 000078C8 GPR11..... 000079CE

GPR12..... 000078BC GPR13..... 0002A018 GPR14..... 80007A1A GPR15..... 8001E128

GPREG STORAGE:

Storage around GPR0 (0002A0D8)

-0020 0002A0B8 00000000 00000000 800093B0 00000000 96080000 000078DC 00009280 0002A0C0 |.....l.....o.....k.....

+0000 0002A0D8 00102401 0002A018 0002A2C0 8001E39A 0004C758 0002A2C0 0002A0C0 0001B7FC |.....s...T...G...s.....

+0020 0002A0F8 00009280 0002A0C8 00009280 0004A370 0002A0C0 00000000 00009150 0004A038 |.k....H..k...t.....j&....

:

Local Variables:

6 77 D-VAL 9999 COMP 00000

:

Figure 89. Sections of Language Environment dump for program COBOLZ1

3. Locate statement 11 in the COBOL listing for the COBOLZ2 program, shown in Figure 90 on page 216. This is a call to the assembler routine ASSEMZ3.

```

PP 5648-A25 IBM COBOL for OS/390 & VM 2.1.1          COBOLZ2  Date 11/04/1999  Time 12:13:28  Page    3
LineID  PL  SL  -----A-1-B-----2-----3-----4-----5-----6-----7-----8  Map and Cross Reference
/* COBOLZ2
000001          ID DIVISION.
000002          PROGRAM-ID, COBOLZ2.
000003          ENVIRONMENT DIVISION.
000004          DATA DIVISION.
000005          WORKING-STORAGE SECTION.
000006          77 DV-VAL PIC 9(4) USAGE COMP.          BLW=0000+000      2C
000007          LINKAGE SECTION.
000008          77 D-VAL PIC 9(4) USAGE COMP.          BLL=0001+000      2C
000009          PROCEDURE DIVISION USING D-VAL.
000010          MOVE D-VAL TO DV-VAL.
000011          CALL "ASSEMZ3" USING DV-VAL.          EXT 6
000012          GOBACK.
*/ COBOLZ2
:
:

```

Figure 90. COBOL listing for COBOLZ2

4. Check offset +64 in the listing for the assembler routine ASSEMZ3, shown in Figure 91.

This shows an instruction to divide the contents of register 4 by the variable pointed to by register 6. You can see the two instructions preceding the divide instruction load register 6 from the first word pointed to by register 1 and prepare register 6 for the divide. Because of linkage conventions, you can infer that register 1 contains a pointer to a parameter list that passed to ASSEMZ3. Register 6 points to a 0 value because that was the value passed to ASSEMZ3 when it was called by a higher level routine.

Note: To translate assembler instructions, see *z/Architecture™ Principles of Operation, SA22-7832*.

		IBMC		HLASM Option Summary		IBMC		(PTF R3PLUS9)		Page	1
								HLASM R3.0		1999/11/04 12.13	
										Page	2
								HLASM R3.0		1999/11/04 12.13	

		IBMC External Symbols		IBMC							
		LD ID		Flags Alias-of							
				07							

Symbol	Type	Id	Address	Length							
ASSEMZ3	SD	00000001	00000000	000000F4							
CEESTART	ER	00000002									
CEEBETBL	ER	00000003									

										Page	3

Active Usings: None											
Loc	Object	Code	Addr1	Addr2	Stmt	Source	Stmt	IBMC			
HLASM R3.0 1999/11/04 12.13											
000000	47F0	F014		00014	1	PRINT NOGEN					
000056	4150	092C		0092C	2	ASSEMZ3	CEEENTRY	MAIN=NO,PPA=MAINPPA			
00005A	1B44				37	LA	5,2348	Low order part of quotient			
00005C	5861	0000		00000	38	SR	4,4	Hi order part of quotient			
000060	4166	0000		00000	39	L	6,0(1)	Get pointer to divisor			
000064	5D46	0000		00000	40	LA	6,0(6)	Clear hi bit			
000068	58F0	B0F0		000F0	41	D	4,0(6)	Do division			
					42	CEETERM	RC=0	Terminate with return code zero			
					49 *						
000080	10				50	MAINPPA	CEEPPA	Constants describing the code block			
					116**	Time Stamp =	1999/11/04 12:13:00		01-CEEPP		
					117**	Version 1 Release 1 Modification	0		01-CEEPP		
					128	CEEDSA		Mapping of the Dynamic Save Area			
					173	CEECAA		Mapping of the Common Anchor Area			
000000					376	END	ASSEMZ3				
0000F0	00000000				377		=A(0)				

Figure 91. Listing for ASSEMZ3

5. Check local variables for COBOLZ2 in the Local Variables section of the dump shown in Figure 92 on page 217. From the dump and listings, you know that COBOLZ2 called ASSEMZ3 and passed a parameter in the variable DV-VAL. The two variables DV-VAL and D-VAL have 0 values.

```

:
:      Local Variables:
:          6 77 DV-VAL      9999 COMP      00000
:          8 77 D-VAL      9999 COMP      00000
:

```

Figure 92. Variables section of Language Environment dump for COBOLZ2

- In the COBOLZ2 subroutine, the variable D-VAL is moved to DV-VAL, the parameter passed to the assembler routine. D-VAL appears in the Linkage section of the COBOLZ2 listing, shown in Figure 93, indicating that the value did pass from COBOLZ1 to COBOLZ2.

```

PP 5648-A25 IBM COBOL for OS/390 & VM 2.1.1      COBOLZ2   Date 11/04/1999 Time 12:13:28 Page    3
LineID  PL SL  ----+--A-1-B-+-----2-----3-----4-----+-----5-----+-----6-----+-----7-|-+-----8  Map and Cross Reference
/* COBOLZ2
000001      ID DIVISION.
000002      PROGRAM-ID. COBOLZ2.
000003      ENVIRONMENT DIVISION.
000004      DATA DIVISION.
000005      WORKING-STORAGE SECTION.
000006      77 DV-VAL PIC 9(4) USAGE COMP.          BLW=0000+000      2C
000007      LINKAGE SECTION.
000008      77 D-VAL PIC 9(4) USAGE COMP.          BLL=0001+000      2C
000009      PROCEDURE DIVISION USING D-VAL.
000010      MOVE D-VAL TO DV-VAL.                  8 6
000011      CALL "ASSEMZ3" USING DV-VAL.          EXT 6
000012      GOBACK.
*/ COBOLZ2

```

Figure 93. Listing for COBOLZ2

- In the Local Variables section of the dump for program COBOLZ1, shown in Figure 94, D-VAL has a 0 value. This indicates that the error causing a fixed-point divide exception in ASSEMZ3 was actually caused by the value of D-VAL in COBOLZ1.

```

:
:      Local Variables:
:          6 77 D-VAL      9999 COMP      00000
:

```

Figure 94. Variables section of Language Environment dump for COBOLZ1

Chapter 6. Debugging FORTRAN routines

This chapter provides information to help you debug applications that contain one or more FORTRAN routines. It includes the following topics:

- Determining the source of errors in FORTRAN routines
- Using FORTRAN compiler listings
- Generating a Language Environment dump of a FORTRAN routine
- Finding FORTRAN information in a dump
- Examples of debugging FORTRAN routines

Determining the source of errors in FORTRAN routines

Most errors in FORTRAN routines can be identified by the information provided in FORTRAN run-time messages, which begin with the prefix FOR.

The FORTRAN compiler cannot identify all possible errors. The following list identifies several errors not detected by the compiler that could potentially result in problems:

- Failing to assign values to variables and arrays before using them in your program.
- Specifying subscript values that are not within the bounds of an array. If you assign data outside the array bounds, you can inadvertently destroy data and instructions.
- Moving data into an item that is too small for it, resulting in truncation.
- Making invalid data references to EQUIVALENCE items of differing types (for example, integer or real).
- Transferring control into the range of a DO loop from outside the range of the loop. The compiler issues a warning message for all such branches if you specify OPT(2), OPT(3), or VECTOR.
- Using arithmetic variables and constants that are too small to give the precision you need in the result. For example, to obtain more than 6 decimal digits in floating-point results, you must use double precision.
- Concatenating character strings in such a way that overlap can occur.
- Trying to access services that are not available in the operating system or hardware.
- Failing to resolve name conflicts between FORTRAN and C library routines using the procedures described in *z/OS Language Environment Programming Guide*.

Identifying run-time errors

FORTRAN has several features that help you find run-time errors. FORTRAN run-time messages are discussed in *z/OS Language Environment Run-Time Messages*. Other debugging aids include the optional traceback map, program interruption messages, abnormal termination dumps, and operator messages.

- The optional traceback map helps you identify where errors occurred while running your application. The TERMTDACT(TRACE) run-time option, which is set by default under Language Environment, generates a dump containing the traceback map.

You can also get a traceback map at any point in your routine by invoking the ERRTRA subroutine.

- Program interruption messages are generated whenever the program is interrupted during execution. Program interruption messages are written to the Language Environment message file.

The program interruption message indicates the exception that caused the termination; the completion code from the system indicates the specification or operation exception resulting in termination.

- Program interruptions causing an abnormal termination produce a dump, which displays the completion code and the contents of registers and system control fields.

To display the contents of main storage as well, you must request an abnormal termination (ABEND) dump by including a SYSUDUMP DD statement in the appropriate job step. The following example shows how the statement can be specified for IBM-supplied cataloged procedures:

```
//GO.SYSUDUMP DD SYSOUT=A
```

- You can request various dumps by invoking any of several dump service routines while your program runs. These dump service routines are discussed in “Generating a Language Environment dump of a FORTRAN routine” on page 221.
- Operator messages are displayed when your program issues a PAUSE or STOP *n* statement. These messages help you understand how far execution has progressed before reaching the PAUSE or STOP statement.

The operator message can take the following forms:

<i>n</i>	String of 1–5 decimal digits you specified in the PAUSE or STOP statement. For the STOP statement, this number is placed in R15.
<i>'message'</i>	Character constant you specified in the PAUSE or STOP statement.
0	Printed when a PAUSE statement containing no characters is executed (not printed for a STOP statement).

A PAUSE message causes the program to stop running pending an operator response. The format of the operator's response to the message depends on the system being used.

- Under Language Environment, error messages produced by Language Environment and FORTRAN are written to a common message file. Its ddname is specified in the MSGFILE run-time option. The default ddname is SYSOUT.

FORTRAN information directed to the message file includes:

- Error messages resulting from unhandled conditions
- Printed output from any of the dump services (SDUMP, DUMP/PDUMP, CDUMP/CPDUMP)
- Output produced by a WRITE statement with a unit identifier having the same value as the FORTRAN error message unit
- Output produced by a WRITE statement with * given as the unit identifier (assuming the FORTRAN error message unit and standard print unit are the same unit)
- Output produced by the PRINT statement (assuming the FORTRAN error message unit and the standard print unit are the same unit)

For more information about handling message output using the Language Environment MSGFILE run-time option, see *z/OS Language Environment Programming Guide*.

Using FORTRAN compiler listings

FORTRAN listings provide you with:

- The date of compilation including information about the compiler
- A listing of your source program
- Diagnostic messages telling you of errors in the source program
- Informative messages telling you the status of the compilation

The following table contains a list of the contents of the various compiler-generated listings that you might find helpful when you use information in dumps to debug FORTRAN programs.

Table 20. Compiler-generated FORTRAN listings and their contents

Name	Contents	Compiler Option
Diagnostic message listing	Error messages detected during compilation.	FLAG
Source program	Source program statements.	SOURCE
Source program	Source program statements and error messages.	SRCFLG
Storage map and cross reference	Variable use, statement function, subprogram, or intrinsic function within a program.	MAP and XREF
Cross reference	Cross reference of names with attributes.	XREF
Source program map	Offsets of automatic and static internal variables (from their defining base).	MAP
Object code	Contents of the program control section in hexadecimal notation and translated into a pseudo-assembler format. To limit the size of the object code listing, specify the statement or range of statements to be listed; for example, LIST(20) or LIST(10,30).	LIST
Variable map, object code, static storage	Same as MAP and LIST options above, plus contents of static internal and static external control sections in hexadecimal notation with comments.	MAP and LIST
Symbolic dump	Internal statement numbers, sequence numbers, and symbol (variable) information.	SDUMP

Generating a Language Environment dump of a FORTRAN routine

To generate a dump containing FORTRAN information, call either DUMP/PDUMP, CDUMP/CPDUMP, or SDUMP.

DUMP/PDUMP and CDUMP/CPDUMP produce output that is unchanged from the output generated under FORTRAN. Under Language Environment, however, the output is directed to the message file.

When SDUMP is invoked, the output is also directed to the Language Environment message file. The dump format differs from other FORTRAN dumps, however, reflecting a common format shared by the various HLLs under Language Environment.

You cannot make a direct call to CEE3DMP from a FORTRAN program. It is possible to call CEE3DMP through an assembler routine called by your FORTRAN program. FORTRAN programs are currently restricted from directly invoking Language Environment callable services.

DUMP/PDUMP	Provides a dump of a specified area of storage.
CDUMP/CPDUMP	Provides a dump of a specified area of storage in character format.
SDUMP	Provides a dump of all variables in a program unit.

DUMP/PDUMP subroutines

The DUMP/PDUMP subroutine dynamically dumps a specified area of storage to the system output data set. When you use DUMP, the processing stops after the dump; when you use PDUMP, the processing continues after the dump.

Syntax

```
CALL {DUMP | PDUMP} (a1, b1,k1, a2,b2, k2,...)
```

a and *b*

Variables in the program unit. Each indicates an area of storage to be dumped. Either *a* or *b* can represent the upper or lower limit of the storage area.

k The dump format to be used. The values that can be specified for *k*, and the resulting dump formats, are:

Value	Format Requested
0	Hexadecimal
1	LOGICAL*1
2	LOGICAL*4
3	INTEGER*2
4	INTEGER*4
5	REAL*4
6	REAL*8
7	COMPLEX*8
8	COMPLEX*16
9	CHARACTER
10	REAL*16
11	COMPLEX*32
12	UNSIGNED*1
13	INTEGER*1
14	LOGICAL*2
15	INTEGER*8
16	LOGICAL*8

Usage considerations for DUMP/PDUMP

A load module or phase can occupy a different area of storage each time it is executed. To ensure that the appropriate areas of storage are dumped, the following conventions should be observed.

If an array and a variable are to be dumped at the same time, a separate set of arguments should be used for the array and for the variable. The specification of limits for the array should be from the first element in the array to the last element. For example, assume that A is a variable in common, B is a real number, and

TABLE is an array of 20 elements. The following call to the storage dump routine could be used to dump TABLE and B in hexadecimal format, and stop the program after the dump is taken:

```
CALL DUMP(TABLE(1),TABLE(20),0,B,B,0)
```

If an area of storage in common is to be dumped at the same time as an area of storage not in common, the arguments for the area in common should be given separately. For example, the following call to the storage dump routine could be used to dump the variables A and B in REAL*8 format without stopping the program:

```
CALL PDUMP(A,A,6,B,B,6)
```

If variables not in common are to be dumped, each variable must be listed separately in the argument list. For example, if R, P, and Q are defined implicitly in the program, the statement

```
CALL PDUMP(R,R,5,P,P,5,Q,Q,5)
```

should be used to dump the three variables in REAL*4 format. If the statement

```
CALL PDUMP(R,Q,5)
```

is used, all main storage between R and Q is dumped, which might or might not include P, and could include other variables.

CDUMP/CPDUMP subroutines

The CDUMP/CPDUMP subroutine dynamically dumps a specified area of storage containing character data. When you use CDUMP, the processing stops after the dump; when you use CPDUMP, the processing continues after the dump.

Syntax

```
CALL {CDUMP | CPDUMP} (a1, b1, a2, b2,...)
```

a and *b*

Variables in the program unit. Each indicates an area of storage to be dumped. Either *a* or *b* can represent the upper or lower limit of each storage area.

The dump is always produced in character format. A dump format type (unlike for DUMP/PDUMP) must not be specified.

Usage considerations for CDUMP/CPDUMP

A load module can occupy a different area of storage each time it is executed. To ensure that the appropriate areas of storage are dumped, the following conventions should be observed.

If an array and a variable are to be dumped at the same time, a separate set of arguments should be used for the array and for the variable. The specification of limits for the array should be from the first element in the array to the last element. For example, assume that B is a character variable and TABLE is a character array of 20 elements. The following call to the storage dump routine could be used to dump TABLE and B in character format, and stop the program after the dump is taken:

```
CALL CDUMP(TABLE(1), TABLE(20), B, B)
```

SDUMP subroutine

The SDUMP subroutine provides a symbolic dump that is displayed in a format dictated by variable type as coded or defaulted in your source. Data is dumped to the error message unit. The symbolic dump is created by program request, on a program unit basis, using CALL SDUMP. Variables can be dumped automatically after abnormal termination using the compiler option SDUMP. For more information on the SDUMP compiler option, see *VS FORTRAN Version 2 Programming Guide for CMS and MVS*.

Items displayed are:

- All referenced, local, named, and saved variables in their FORTRAN-defined data representation
- All variables contained in a static common area (blank or named) in their FORTRAN-defined data representation
- All variables contained in a dynamic common area in their FORTRAN-defined data representation
- Nonzero or nonblank character array elements only
- Array elements with their correct indexes

The amount of output produced can be very large, especially if your program has large arrays, or large arrays in common blocks. For such programs, you might want to avoid calling SDUMP.

Syntax

```
CALL SDUMP [(rtn1,rtn2,...)]
```

rtn₁,rtn₂,...

Names of other program units from which data will be dumped. These names must be listed in an EXTERNAL statement.

Usage considerations for SDUMP

- To obtain symbolic dump information and location of error information, compilation must be done either with the SDUMP option or with the TEST option.
- Calling SDUMP and specifying program units that have not been entered gives unpredictable results.
- Calling SDUMP with no parameters produces the symbolic dump for the current program unit.
- An EXTERNAL statement must be used to identify the names being passed to SDUMP as external routine names.
- At higher levels of optimization (1, 2, or 3), the symbolic dump could show incorrect values for some variables because of compiler optimization techniques.
- Values for uninitialized variables are unpredictable. Arguments in uncalled subprograms or in subprograms with argument lists shorter than the maximum can cause the SDUMP subroutine to fail.
- The display of data can also be invoked automatically. If the run-time option TERMTHDACT(DUMP) is in effect and your program abends in a program unit compiled with the SDUMP option or with the TEST option, all data in that program unit is automatically dumped. All data in any program unit in the save area traceback chain compiled with the SDUMP option or with the TEST option is also dumped. Data occurring in a common block is dumped at each occurrence, because the data definition in each program unit could be different.

Examples of calling SDUMP from the main program and from a subprogram follow. Figure 95 on page 226 shows a sample program calling SDUMP and Figure 96 on page 227 shows the resulting output that is generated. In the main program, the statement

```
EXTERNAL PGM1,PGM2,PGM3
```

makes the address of subprograms PGM1, PGM2, and PGM3 available for a call to SDUMP as follows:

```
CALL SDUMP (PGM1,PGM2,PGM3)
```

This causes variables in PGM1, PGM2, and PGM3 to be printed.

In the subprogram PGM1, the statement

```
EXTERNAL PGM2,PGM3
```

makes PGM2 and PGM3 available. (PGM1 is missing because the call is in PGM1.) The statements

```
CALL SDUMP  
CALL SDUMP (PGM2,PGM3)
```

dump variables PGM1, PGM2, and PGM3.

```

OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
                    NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NOBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                    NREORDER NOPC
                    OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
IF DO  ISN  *....*1.....2.....3.....4.....5.....6.....7.*.....8
      1      PROGRAM FORTMAIN
      2      EXTERNAL PGM1,PGM2,PGM3
      3      INTEGER*4 ANY_INT
      4      INTEGER*4 INT_ARR(3)
      5      CHARACTER*20 CHAR_VAR
      6      ANY_INT = 555
      7      INT_ARR(1) = 1111
      8      INT_ARR(2) = 2222
      9      INT_ARR(3) = 2222
     10      CHAR_VAR = 'SAMPLE CONSTANT '
     11      CALL PGM1(ANY_INT,CHAR_VAR)
     12      CALL SDUMP(PGM1,PGM2,PGM3)
     13      STOP
     14      END

OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
                    NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NOBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                    NREORDER NOPC
                    OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
IF DO  ISN  *....*1.....2.....3.....4.....5.....6.....7.*.....8
      1      SUBROUTINE PGM1(ARG1,ARG2)
      2      EXTERNAL PGM2,PGM3
      3      INTEGER*4 ARG1
      4      CHARACTER*20 ARG2
      5      ARG1 = 1
      6      ARG2 = 'ARGUMENT'
      7      CALL PGM2
      8      CALL SDUMP
      9      CALL SDUMP(PGM2,PGM3)
     10      RETURN
     11      END

OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
                    NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NOBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                    NREORDER NOPC
                    OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
IF DO  ISN  *....*1.....2.....3.....4.....5.....6.....7.*.....8
      1      SUBROUTINE PGM2
      2      INTEGER*4 PGM2VAR
      3      PGM2VAR = 555
      4      CALL PGM3
      5      RETURN
      6      END

OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
                    NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NOBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                    NREORDER NOPC
                    OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
IF DO  ISN  *....*1.....2.....3.....4.....5.....6.....7.*.....8
      1      SUBROUTINE PGM3
      2      CHARACTER*20 PGM3VAR
      3      PGM3VAR = 'PGM3 VAR'
      4      RETURN
      5      END

```

Figure 95. Example program that calls SDUMP

Figure 96 on page 227 shows the resulting output generated by the example in Figure 95.

```

Parameters, Registers, and Variables for Active Routines:
PGM1      (DSA address 06D004C8):
Parameters:
  ARG2      CHARACTER*20      ARGUMENT

  ARG1      INTEGER*4          1
Local Variables:
Parameters, Registers, and Variables for Active Routines:
PGM2      (DSA address 000930FC):
Parameters:
Local Variables:
  PGM2VAR   INTEGER*4          555
Parameters, Registers, and Variables for Active Routines:
PGM3      (DSA address 000930FC):
Parameters:
Local Variables:
  PGM3VAR   CHARACTER*20      PGM3 VAR

Parameters, Registers, and Variables for Active Routines:
PGM1      (DSA address 000930FC):
Parameters:
  ARG2      CHARACTER*20      ARGUMENT

  ARG1      INTEGER*4          1
Local Variables:
Parameters, Registers, and Variables for Active Routines:
PGM2      (DSA address 000930FC):
Parameters:
Local Variables:
  PGM2VAR   INTEGER*4          555
Parameters, Registers, and Variables for Active Routines:
PGM3      (DSA address 000930FC):
Parameters:
Local Variables:
  PGM3VAR   CHARACTER*20      PGM3 VAR

```

Figure 96. Language Environment dump generated using SDUMP

Finding FORTRAN information in a Language Environment dump

To locate FORTRAN-specific information in a Language Environment dump, you must understand how to use the traceback section and the section in the symbol table dump showing parameters and variables.

Information for enclave SAMPLE

Information for thread 8000000000000000

[1]

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
0002D018	CEEHDSP	05936760	+0000277C	CEEHDSP	05936760	+0000277C		CEEPLPKA		Call
0002F018	AFHCSGLE	059DF718	+000001A8	AFHCSGLE	059DF718	+000001A8		AFHPRNAG		Exception
05A44060	AFH00PNR	05A11638	+00001EDE	AFH00PNR	05A11638	+00001EDE		AFHPRNAG		Call
05900A90	SAMPLE	059009A8	+0000021C	SAMPLE	059009A8	+0000021C	6_ISN	GO		Call

[2]

Condition Information for Active Routines

Condition Information for AFHCSGLE (DSA address 0002F018)

CIB Address: 0002D468

Current Condition:

FOR1916S The OPEN statement for unit 999 failed. The unit number was either less than 0 or greater than 99, the highest unit number allowed at your installation.

Location:

Program Unit: AFHCSGLE Entry: AFHCSGLE Statement: Offset: +000001A8

Storage dump near condition, beginning at location: 059DF8B0

+000000 059DF8B0 5060D198 5880C2B8 58F0801C 4110D190 05EFD502 D3019751 4770A1F0 4820D2FE |&-Jq..B..0....J...N.L.p....0..K.|

Parameters, Registers, and Variables for Active Routines:

CEEHDSP (DSA address 0002D018):

Saved Registers:

GPR0..... 000003E7	GPR1..... 0002D3B4	GPR2..... 0002DFD7	GPR3..... 0002E027
GPR4..... 0002DF94	GPR5..... 00000000	GPR6..... 00000004	GPR7..... 00000000
GPR8..... 0002E017	GPR9..... 0593875E	GPR10..... 0593775F	GPR11..... 85936760
GPR12..... 00014770	GPR13..... 0002D018	GPR14..... 800250DE	GPR15..... 85949C70

GPREG STORAGE:

Storage around GPR0 (000003E7)

-000020 000003C7 Inaccessible storage.

+000000 000003E7 Inaccessible storage.

+000020 00000407 Inaccessible storage.

Storage around GPR1 (0002D3B4)

-000020 0002D394 00000006 00000000 0002E017 0593875E 0593775F 85936760 00014770 00000000

+000000 0002D3B4 0002DFD7 0002E027 0002DF94 0002DF94 0002DDF4 0002DEC4 0002E158 0002D018

+000020 0002D3D4 0002D468 00000000 00000000 00000007 859D67E0 00000000 00000000 05914848

.....lg;.l.-el.-.....
...P.....m...m...4...D.....
..M.....e.....j..

:

[3]

Local Variables:

ABC	CHARACTER*3	123
J	INTEGER*4	444

[4]

File Status and Attributes:

The total number of units defined is 100.

The default unit for the PUNCH statement is 7.

The default unit for the Fortran error messages is 6.

The default unit for formatted sequential output is 6.

The default unit for formatted sequential input is 5.

Figure 97. Sections of the Language Environment dump

Understanding the Language Environment traceback table

Examine the traceback section of the dump, labeled with **[1]** in Figure 97, for condition information about your routine and information about the statement number and address where the exception occurred. The traceback section helps you locate where an error occurred in your program. The information in this section begins with the most recent program unit and ends with the first program unit.

Identifying condition information

The section labeled **[2]** in Figure 97 shows the condition information for the active routines, indicating the program message, program unit name, the statement number, and the offset within the program unit where the error occurred.

Identifying variable information

The local variable section of the dump, shown in the section labeled [3] in Figure 97 on page 228, contains information on all variables and arrays in each program unit in the save area chain, including the program causing the dump to be invoked. The output shows variable items (one line only) and array (more than one line) items.

Use the local variable section of the dump to identify the variable name, type, and value at the time the dump was called. Variable and array items can contain either character or noncharacter data, but not both.

Identifying file status information

The section labeled [4] in Figure 97 on page 228 shows the file status and attribute section of the dump. This section displays the total number of units defined, the default units for error messages, and the default unit numbers for formatted input or formatted output.

Examples of debugging FORTRAN routines

This section contains examples of FORTRAN routines and instructions for using information in the Language Environment dump to debug them.

Calling a nonexistent routine

Figure 98 illustrates an error caused by calling a nonexistent routine. The options in effect at compile time appear at the top of the listing.

```
OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
                   NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                   NOREORDER NOPC
                   OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
1      PROGRAM CALLNON
2      INTEGER*4 ARRAY_END
      C
3      CALL SUBNAM
4      STOP
5      END
```

Figure 98. Example of calling a nonexistent routine

Figure 99 on page 230 shows sections of the dump generated by a call to SDUMP.

Information for enclave CALLNON

Information for thread 8000000000000000

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
0002D018	CEEHDSP	05936760	+0000277C	CEEHDSP	05936760	+0000277C		CEEPLPKA		Call
05900C10	CALLNON	05900B28	-05900B26	CALLNON	05900B28	-05900B26	3_ISN	GO		Exception

Condition Information for Active Routines

Condition Information for CALLNON (DSA address 05900C10)

CIB Address: 0002D468

Current Condition:

CEE3201S The system detected an operation exception.

Location:

Program Unit: CALLNON

Entry: CALLNON

Statement: 3_ISN Offset: -05900B26

Machine State:

ILC..... 0002 Interruption Code..... 0001

PSW..... 078D3D00 80000004

GPR0..... FD000008 GPR1..... 00000000 GPR2..... 05900D04 GPR3..... 05900C10

GPR4..... 007F6930 GPR5..... 007FD238 GPR6..... 007BFFF8 GPR7..... FD000000

GPR8..... 007FD968 GPR9..... 807FD4F8 GPR10..... 00000000 GPR11..... 007FD238

GPR12..... 00E21ED2 GPR13..... 05900C10 GPR14..... 85900CE8 GPR15..... 00000000

Storage dump near condition, beginning at location: 00000000

+000000 00000000 Inaccessible storage.

Parameters, Registers, and Variables for Active Routines:

CEEHDSP (DSA address 0002D018):

Saved Registers:

GPR0..... 00000000 GPR1..... 0002D3B4 GPR2..... 0002DFD7 GPR3..... 0002E027

GPR4..... 0002DF94 GPR5..... 00000000 GPR6..... 00000004 GPR7..... 00000000

GPR8..... 0002E017 GPR9..... 0593875E GPR10..... 0593775F GPR11..... 05936760

GPR12..... 00014770 GPR13..... 0002D018 GPR14..... 800250DE GPR15..... 85949C70

GPREG STORAGE:

Storage around GPR0 (00000000)

+000000 00000000 Inaccessible storage.

+000020 00000020 Inaccessible storage.

+000040 00000040 Inaccessible storage.

Storage around GPR1 (0002D3B4)

-000020 0002D394 00000006 00000000 0002E017 0593875E 0593775F 05936760 00014770 00000000 |.....lg;l..l..-.....|

+000000 0002D3B4 0002DFD7 0002E027 0002DF94 0002DF94 0002DDF4 0002DEC4 0002E158 00000000 |...P.....m...m...4...D.....|

+000020 0002D3D4 0002D468 00000000 00000000 00000007 859D67E0 00000000 00000000 05914848 |..M.....e.....j..|

:

File Status and Attributes:

The total number of units defined is 100.

The default unit for the PUNCH statement is 7.

The default unit for the Fortran error messages is 6.

The default unit for formatted sequential output is 6.

The default unit for formatted sequential input is 5.

Figure 99. Sections of the Language Environment dump resulting from a call to a nonexistent routine

To understand the traceback section, and debug this example routine, do the following:

1. Find the Current Condition information in the Condition Information for Active Routines section of the dump. The message is CEE3201S. The system detected an operation exception at statement 3. For more information about this message, see *z/OS Language Environment Run-Time Messages*. This section of the dump also provides such information as the name of the active routine and the current statement number at the time of the dump.
2. Locate statement 3 in the routine shown in Figure 98 on page 229. This statement calls subroutine SUBNAM. The message CEE3201S in the Condition Information section of the dump indicates that the operation exception was generated because of an unresolved external reference.
3. Check the linkage editor output for error messages.

Divide-by-zero error

Figure 100 demonstrates a divide-by-zero error. In this example, the main FORTRAN program passed 0 to subroutine DIVZEROSUB, and the error occurred when DIVZEROSUB attempted to use this data as a divisor.

```
OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
NOREORDER NOPC
OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
1  PROGRAM DIVZERO
2  INTEGER*4 ANY_NUMBER
3  INTEGER*4 ANY_ARRAY(3)
4  PRINT *, 'EXAMPLE STARTING'
5  ANY_NUMBER = 0
6  DO I = 1,3
1  7  ANY_ARRAY(I) = I
1  8  END DO
9  CALL DIVZEROSUB(ANY_NUMBER, ANY_ARRAY)
10 PRINT *, 'EXAMPLE ENDING'
11 STOP
12 END
OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
NOREORDER NOPC
OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
1  SUBROUTINE DIVZEROSUB(DIVISOR, DIVIDEND)
2  INTEGER*4 DIVISOR
3  INTEGER*4 DIVIDEND(3)
4  PRINT *, 'IN SUBROUTINE DIVZEROSUB'
5  DIVIDEND(1) = DIVIDEND(3) / DIVISOR
6  PRINT *, 'END OF SUBROUTINE DIVZEROSUB'
7  RETURN
8  END
```

Figure 100. FORTRAN routine with a divide-by-zero error

Figure 101 on page 232 shows the Language Environment dump for routine DIVZERO.

Information for enclave DIVZERO

Information for thread 8000000000000000

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Statement	Load Mod	Service	Status
0002D018	CEEHDSP	05936760	+0000277C	CEEHDSP	05936760	+0000277C		CEEPLPKA		Call
05900640	DIVZSUB	05900558	+00000258	DIVZSUB	05900558	+00000258	5_ISN	GO		Exception
0002F018	AFHLCLNR	0001B150	+00000000	AFHLCLNR	0001B150	+00000000		AFHPRNBG		Call
059002E8	DIVZERO	05900200	+00000298	DIVZERO	05900200	+00000298	9_ISN	GO		Call

Condition Information for Active Routines

Condition Information for DIVZSUB (DSA address 05900640)

CIB Address: 0002D468

Current Condition:

CEE3209S The system detected a fixed-point divide exception.

Location:

Program Unit: DIVZSUB

Entry: DIVZSUB

Statement: 5_ISN Offset: +00000258

Machine State:

ILC..... 0004 Interruption Code..... 0009

PSW..... 078D2A00 859007B4

GPR0..... 00000000 GPR1..... 00000003 GPR2..... 059003FC GPR3..... 05900400

GPR4..... 007F6930 GPR5..... 05900468 GPR6..... 0000000C GPR7..... 059003E0

GPR8..... 85900400 GPR9..... 807FD4F8 GPR10..... 00000000 GPR11..... 007FD238

GPR12..... 00E21ED2 GPR13..... 05900640 GPR14..... 8590079C GPR15..... 05900BA0

Storage dump near condition, beginning at location: 059007A0

+000000 059007A0 5870D118 5800700C 5870D120 8E000020 5D007000 5870D118 50107004 58F0D128 |..J.....J.....).....J.&....0J.|

Parameters, Registers, and Variables for Active Routines:

CEEHDSP (DSA address 0002D018):

Saved Registers:

GPR0..... 00000000 GPR1..... 0002D3B4 GPR2..... 0002DFD7 GPR3..... 0002E027

GPR4..... 0002DF94 GPR5..... 00000000 GPR6..... 00000004 GPR7..... 00000000

GPR8..... 0002E017 GPR9..... 0593875E GPR10..... 0593775F GPR11..... 05936760

GPR12..... 00014770 GPR13..... 0002D018 GPR14..... 800250DE GPR15..... 85949C70

GPREG STORAGE:

Storage around GPR0 (00000000)

+000000 00000000 Inaccessible storage.

+000020 00000020 Inaccessible storage.

+000040 00000040 Inaccessible storage.

Storage around GPR1 (0002D3B4)

-000020 0002D394 00000006 00000000 0002E017 0593875E 0593775F 05936760 00014770 00000000

+000000 0002D3B4 0002DFD7 0002E027 0002DF94 0002DF94 0002DDF4 0002DEC4 0002E158 00000000

+000020 0002D3D4 0002D468 00000000 00000000 00000007 859D67E0 00000000 00000000 05914848

|.....lg;.l..l.-.....
|...P.....m...m...4...D.....
|..M.....e.....j..

:

Local Variables:

I INTEGER*4 4

ANY_ARRAY(3) INTEGER*4

ANY_ARRAY(1) 1 2 3

ANY_NUMBER INTEGER*4 0

File Status and Attributes:

The total number of units defined is 100.

The default unit for the PUNCH statement is 7.

The default unit for the Fortran error messages is 6.

The default unit for formatted sequential output is 6.

The default unit for formatted sequential input is 5.

Figure 101. Language Environment dump from divide-by-zero FORTRAN example

To debug this application, do the following:

1. Locate the error message for the current condition in the Condition Information section of the dump, shown in Figure 101. The message is CEE3209S. The system detected a fixed-point divide exception. See *z/OS Language Environment Run-Time Messages* for additional information about this message.
2. Note the sequence of the calls in the call chain:
 - a. DIVZERO called AFHLCLNR, which is a FORTRAN library subroutine.
 - b. AFHLCLNR called DIVZSUB.

Note: When a program-unit name is longer than 7 characters, the name as it appears in the dump consists of the first 4 and last 3 characters concatenated together.

- c. DIVZEROSUB attempted a divide-by-zero operation at statement 5.
 - d. This resulted in a call to CEEHDSP, a Language Environment condition handling routine.
3. Locate statement 5 in the FORTRAN listing for the DIVZEROSUB subroutine in Figure 101 on page 232. This is an instruction to divide the contents of DIVIDEND(3) by DIVISOR.
 4. Since DIVISOR is a parameter of subroutine DIVZEROSUB, go to the Parameters section of the dump shown in Figure 101 on page 232. The parameter DIVISOR shows a value of 0.
 5. Since DIVISOR contains the value passed to DIVZEROSUB, check its value. ANY_NUMBER is the actual argument passed to DIVZEROSUB, and the dump and listing of DIVZERO indicate that ANY_NUMBER had value 0 when passed to DIVZEROSUB, leading to the divide-by-zero exception.

Chapter 7. Debugging PL/I routines

This chapter contains information that can help you debug applications that contain one or more PL/I routines. Following a discussion about potential errors in PL/I routines, the first part of this chapter discusses how to use compiler-generated listings to obtain information about PL/I routines, and how to use PLIDUMP to generate a Language Environment dump of a PL/I routine. The last part of the chapter provides examples of PL/I routines and explains how to debug them using information contained in the traceback information provided in the dump. The topics covered are listed below.

- Determining the source of errors in PL/I routines
- Using PL/I compiler listings
- Generating a Language Environment dump of a PL/I routine
- Finding PL/I information in a dump
- Debugging example of PL/I routines

Determining the source of errors in PL/I routines

Most errors in PL/I routines can be identified by the information provided in PL/I run-time messages, which begin with the prefix IBM. For a list of these messages, see *z/OS Language Environment Run-Time Messages*.

A malfunction in running a PL/I routine can be caused by:

- Logic errors in the source routine
- Invalid use of PL/I
- Unforeseen errors
- Invalid input data
- Compiler or run-time routine malfunction
- System malfunction
- Unidentified routine malfunction
- Overlaid storage

Logic errors in the source routine

Errors of this type are often difficult to detect because they often appear as compiler or library malfunctions.

Some common errors in source routines are:

- Incorrect conversion from arithmetic data
- Incorrect arithmetic and string manipulation operations
- Unmatched data lists and format lists

Invalid use of PL/I

A misunderstanding of the language or a failure to provide the correct environment for using PL/I can result in an apparent malfunction of a PL/I routine.

Any of the following, for example, might cause a malfunction:

- Using uninitialized variables
- Using controlled variables that have not been allocated
- Reading records into incorrect structures
- Misusing array subscripts
- Misusing pointer variables
- Incorrect conversion
- Incorrect arithmetic operations
- Incorrect string manipulation operations

Unforeseen errors

If an error is detected during run time and no ON-unit is provided in the routine to terminate the run or attempt recovery, the job terminates abnormally. However, the status of a routine at the point where the error occurred can be recorded by using an ERROR ON-unit that contains the statements:

```
ON ERROR
BEGIN;
  ON ERROR SYSTEM;
  CALL PLIDUMP;           /*generates a dump*/
  PUT DATA;             /*displays variables*/
END;
```

The statement ON ERROR SYSTEM ensures that further errors do not result in a permanent loop.

Invalid input data

A routine should contain checks to ensure that any incorrect input data is detected before it can cause the routine to malfunction.

Use the COPY option of the GET statement to check values obtained by stream-oriented input. The values are listed on the file named in the COPY option. If no file name is given, SYSPRINT is assumed.

Compiler or run-time routine malfunction

If you are certain that the malfunction is caused by a compiler or run-time routine error, you can either open a PMR or submit an APAR for the error. See either *PL/I for MVS & VM Diagnosis Guide* or *VisualAge PL/I for OS/390 Diagnosis Guide* for more information about handling compiler and run-time routine malfunctions. Meanwhile, you can try an alternative way to perform the operation that is causing the trouble. A bypass is often feasible, since the PL/I language frequently provides an alternative method of performing operations.

System malfunction

System malfunctions include machine malfunctions and operating system errors. System messages identify these malfunctions and errors to the operator.

Unidentified routine malfunction

In most circumstances, an unidentified routine malfunction does not occur when using the compiler. If your routine terminates abnormally without an accompanying Language Environment run-time diagnostic message, the error causing the termination might also be inhibiting the production of a message. Check for the following:

- Your job control statements might be in error, particularly in defining data sets.
- Your routine might overwrite main storage areas containing executable instructions. This can happen if you have accidentally:
 - Assigned a value to a nonexistent array element. For example:

```
DCL ARRAY(10);

:
DO I = 1 TO 100;
  ARRAY(I) = VALUE;
```

To detect this type of error in a compiled module, set the SUBSCRIPTRANGE condition so that each attempt to access an element outside the declared

range of subscript values raises the SUBSCRIPTRANGE condition. If there is no ON-unit for this condition, a diagnostic message is printed and the ERROR condition is raised. This facility, though expensive in run time and storage space, is a valuable routine-testing aid.

- Used an incorrect locator value for a locator (pointer or offset) variable. This type of error can occur if a locator value is obtained by means of record-oriented transmission. Ensure that locator values created in one routine, transmitted to a data set, and subsequently retrieved for use in another routine, are valid for use in the second routine.

- Attempted to free a nonbased variable. This can happen when you free a based variable after its qualifying pointer value has been changed. For example:

```
DCL A STATIC,B BASED (P);  
ALLOCATE B;  
P = ADDR(A);  
FREE B;
```

- Used the SUBSTR pseudovalue to assign a string to a location beyond the end of the target string. For example:

```
DCL X CHAR(3);  
I=3  
SUBSTR(X,2,I) = 'ABC';
```

To detect this type of error, enable the STRINGRANGE condition during compilation.

Storage overlay problems

If you suspect an error in your PL/I application is a storage overlay problem, check for the following:

- The use of a subscript outside the declared bounds (check the SUBSCRIPTRANGE condition)
- An attempt to assign a string to a target with an insufficient maximum length (check the STRINGSIZE condition)
- The failure of the arguments to a SUBSTR reference to comply with the rules described for the SUBSTR built-in function (check the STRINGRANGE condition)
- The loss of significant last high-order (left-most) binary or decimal digits during assignment to an intermediate result or variable or during an input/output operation (check the SIZE condition)
- The reading of a variable-length file into a variable
- The misuse of a pointer variable
- The invocation of a Language Environment callable service with fewer arguments than are required

The first four situations are associated with the listed PL/I conditions, all of which are disabled by default. If you suspect one of these problems exists in your routine, use the appropriate condition prefix on the suspected statement or on the BEGIN or PROCEDURE statement of the containing block.

The fifth situation occurs when you read a data record into a variable that is too small. This type of problem only happens with variable-length files. You can often isolate the problem by examining the data in the file information and buffer.

The sixth situation occurs when you misuse a pointer variable. This type of storage overlay is particularly difficult to isolate. There are a number of ways pointer variables can be misused:

- When a READ statement runs with the SET option, a value is placed in a pointer. If you then run a WRITE statement or another READ SET option with another pointer, you overlay your storage if you try to use the original pointer.
- When you try to use a pointer to allocate storage that has already been freed, you can also cause a storage overlay.
- When you attempt to use a pointer set with the ADDR built-in function as a base for data with different attributes, you can cause a storage overlay.

The seventh situation occurs when a Language Environment callable service is passed fewer arguments than its interface requires. The following example might cause a storage overlay because Language Environment assumes that the fourth item in the argument list is the address of a feedback code, when in reality it could be residue data pointing anywhere in storage.

Invalid calls:

```
DCL CEEDATE ENTRY OPTIONS(ASM);
CALL CEEDATE(x,y,z);      /* invalid */
```

Valid calls:

```
DCL CEEDATE ENTRY(*,*,*,* OPTIONAL) OPTIONS(ASM);
CALL CEEDATE(x,y,z,*);    /* valid */
CALL CEEDATE(x,y,z,fc);   /* valid */
```

Using PL/I compiler listings

The following sections explain how to generate listings that contain information about your routine. PL/I listings show machine instructions, constants, and external or internal addresses that the linkage editor resolves. This information can help you find other information, such as variable values, in a dump of a PL/I routine.

Note: VisualAge PL/I shares a common compiler back-end with C/C++. The VisualAge PL/I assembler listing will, consequently, have a similar form to those from the XL C/C++ compiler.

The PL/I compiler listings included below are from the PL/I for MVS & VM product.

Generating PL/I listings and maps

The following table shows compiler-generated listings that you might find helpful when you use information in dumps to debug PL/I routines. For more information about supported compiler options that generate listings, reference either the *PL/I for MVS & VM Programming Guide* or the *VisualAge PL/I for OS/390 Programming Guide*.

Table 21. Compiler-generated PL/I listings and their contents

Name	Contents	Compiler Option
Source program	Source program statements	SOURCE
Cross reference	Cross reference of names with attributes	XREF and ATTRIBUTES
Aggregate table	Names and layouts of structures and arrays	AGGREGATE
Variable map	Offsets of automatic and static internal variables (from their defining base)	MAP

Table 21. Compiler-generated PL/I listings and their contents (continued)

Name	Contents	Compiler Option
Object code	Contents of the program control section in hexadecimal notation and translated into a pseudo-assembler format. To limit the size of the object code listing, specify a certain statement or range of statements to be listed; for example, LIST(20) or LIST(10,30).	LIST
Variable map, object code, static storage	Same as MAP and LIST options above, plus contents of static internal and static external control sections in hexadecimal notation with comments	MAP and LIST

Finding information in PL/I listings

Figure 102 shows an example PL/I routine that was compiled with LIST and MAP.

```

*PROCESS SOURCE, LIST, MAP;

          SOURCE LISTING

          STMT
1  | EXAMPLE: PROC OPTIONS(MAIN);
2  |     DCL EXTR ENTRY EXTERNAL;
3  |     DCL A FIXED BIN(31);
4  |     DCL B(2,2)  FIXED BIN(31) STATIC EXTERNAL INIT((4)0);
5  |     DCL C CHAR(20) STATIC INIT('SAMPLE CONSTANT');
6  |     DCL D FIXED BIN(31) STATIC;
7  |     DCL E FIXED BIN(31);
8  |     FETCH EXTR;
9  |     CALL EXTR(A,B,C,D,E);
10 |     DISPLAY(C);
11 |     END;

```

Figure 102. PL/I routine compiled with LIST and MAP

Figure 103 on page 240 shows the output generated from this routine, including the static storage map, variable storage map, and the object code listing. The sections following this example describe the contents of each type of listing.

```

                STATIC INTERNAL STORAGE MAP

000000 E00000E8      PROGRAM ADCON
000004 00000008      PROGRAM ADCON
000008 00000096      PROGRAM ADCON
00000C 00000096      PROGRAM ADCON
000010 00000096      PROGRAM ADCON
000014 00000000      A..IBMSJDSA
000018 00000000      A..IBMSPFRA
00001C 00000000      A..STATIC
000020 000000000000044 LOCATOR..B
000028 0000008800140000 LOCATOR..C
000030 91E091E0      CONSTANT
000034 0A000000C5E7E3D9 FECB..EXTR
        40404040
000040 80000034      A..FECB..EXTR
000044 0000000C00000008 DESCRIPTOR
        0000000200000001
        0000000400000002
        00000001
000060 80000034      A..FECB..EXTR
000064 00000000      A..B
000068 00000000      A..A
00006C 00000020      A..LOCATOR
000070 00000028      A..LOCATOR
000074 000000A0      A..D
000078 80000000      A..E
00007C 00000000      A..ENTRY EXTR
000080 80000028      A..LOCATOR
000084
000088 E2C1D4D7D3C540C3 INITIAL VALUE..C
        D6D5E2E3C1D5E340
        40404040

                STATIC EXTERNAL CSECTS

000000 0000000000000000 CSECT FOR EXTERNAL VARIABLE
0000000000000000

:
:
                VARIABLE STORAGE MAP

IDENTIFIER          LEVEL      OFFSET      (HEX)  CLASS   BLOCK

E                    1          184         B8     AUTO   EXAMPLE
D                    1          160         A0     STATIC  EXAMPLE
C                    1          136         88     STATIC  EXAMPLE
A                    1          188         BC     AUTO   EXAMPLE

:
:
OBJECT LISTING

* STATEMENT NUMBER 1
000000          DC  C'EXAMPLE'
000007          DC  AL1(7)

* PROCEDURE          EXAMPLE
0000AE          05 EF
0000B0          58 FB 0 000
0000B4          0000B4

CL.5
EQU  *

* REAL ENTRY
000008 90 EC D 00C      STM  14,12,12(13)
00000C 47 F0 F 04C      B    **72
000010 00000000      DC  A(STMT. NO. TABLE) * STATEMENT NUMBER 9
000014 000000D8      DC  F'216' 0000B4 D2 13 D 0C0 3 068 MVC 192(20,13),104(3)
000018 00000000      DC  A(STATIC CSECT) 0000BA 41 70 D 0BC LA 7,A
00001C 00000000      DC  A(SYMTAB VECTOR) 0000BE 50 70 D 0C0 ST 7,192(0,13)
000020 00000000      DC  A(COMPILEATION INFO) 0000C2 41 70 D 0B8 LA 7,E
000024 A8000000      DC  X'A8000000' 0000C6 50 70 D 0D0 ST 7,208(0,13)
000028 00010100      DC  X'00010100' 0000CA 96 80 D 0D0 OI 208(13),X'80'
00002C 00000000      DC  X'00000000' 0000CE 58 FB 0 000 L 15,PR..EXTR
000030 00000000      DC  X'00000000' 0000D2 59 F0 C 064 C 15,100(0,12)
000034 00000000      DC  A(ENTRY LIST VECTOR)0000D6 47 70 2 052 BNE CL.6

```

Figure 103. Compiler-generated listings from example PL/I routine (Part 1 of 2)

```

000038 00000000      DC  X'00000000'      0000DA 41 10 3 060      LA  1,96(0,3)
00003C 01008000      DC  X'01008000'      0000DE 58 F0 3 018      L   15,A..IBMSPFRA
000040 00000000      DC  A(REGION TABLE)  0000E2 05 EF          BALR 14,15
000044 00000002      DC  X'00000002'      0000E4 58 FB 0 000      L   15,PR..EXTR
000048 00000000      DC  A(PRIMARY ENTRY)  0000E8          CL.6    EQU  *
00004C 00000000      DC  X'00000000'      0000E8 1B 55          SR   5,5
000050 00000000      DC  X'00000000'      0000EA 41 10 D 0C0     LA  1,192(0,13)
000054 58 30 F 010    L   3,16(0,15)      0000EE 05 EF          BALR 14,15
000058 58 10 D 04C    L   1,76(0,13)
00005C 58 00 F 00C    L   0,12(0,15)
000060 1E 01          ALR  0,1
000062 55 00 C 00C    CL   0,12(0,12)      * STATEMENT NUMBER 10
000066 47 D0 F 068    BNH  **+10          0000F0 41 10 3 080     LA  1,128(0,3)
00006A 58 F0 C 074    L   15,116(0,12)     0000F4 58 F0 3 014     L   15,A..IBMSJDASA
00006E 05 EF          BALR 14,15          0000F8 05 EF          BALR 14,15
000070 58 E0 D 048    L   14,72(0,13)
000074 18 F0          LR   15,0
000076 90 E0 1 048    STM  14,0,72(1)      * STATEMENT NUMBER 11
00007A 50 D0 1 004    ST   13,4(0,1)      0000FA 18 0D          LR   0,13
00007E 92 80 1 000    MVI  0(1),X'80'      0000FC 58 D0 D 004     L   13,4(0,13)
000082 92 25 1 001    MVI  1(1),X'25'      000100 58 E0 D 00C     L   14,12(0,13)
000086 92 02 1 076    MVI  118(1),X'02'    000104 98 2C D 01C     LM   2,12,28(13)
00008A 41 D1 0 000    LA   13,0(1,0)      000108 05 1E          BALR 1,14
00008E D2 03 D 054 3 030 MVC  84(4,13),48(3)
000094 05 20          BALR 2,0
                                * END PROCEDURE
                                00010A 07 07
                                NOPR 7
* PROCEDURE BASE
                                * END PROGRAM

```

Figure 103. Compiler-generated listings from example PL/I routine (Part 2 of 2)

Static internal storage map

To get a complete variable storage map and static storage map, but not a complete LIST, specify a single statement for LIST to minimize the size of the listing; for example, LIST(1).

Each line of the static storage map contains the following information:

1. Six-digit hexadecimal offset.
2. Hexadecimal text, in 8-byte sections where possible.
3. Comment, indicating the type of item to which the text refers. The comment appears on the first line of the text for an item.

Some typical comments you might find in a static storage listing:

Table 22. Typical comments in a PL/I static storage listing

Comment	Explanation
A...xxx	Address constant for xxx
COMPILER LABEL CL.n	Compiler-generated label n
CONDITION CSECT	Control section for programmer-named condition
CONSTANT	Constant
CSECT FOR EXTERNAL VARIABLE	Control section for external variable
D...xxx	Descriptor for xxx
DED...xxx	Data element descriptor for xxx
DESCRIPTOR	Data descriptor
ENVB	Environment control block
FECB...xxx	Fetch control block for xxx
DCLCB	Declare control block
FED...xxx	Format element descriptor for xxx

Table 22. Typical comments in a PL/I static storage listing (continued)

Comment	Explanation
KD..xxx	Key descriptor for xxx
LOCATOR..xxx	Locator for xxx
ONCB	ON statement control block
PICTURED DED..xxx	Pictured data element descriptor for xxx
PROGRAM ADCON	Program address constant
RD..xxx	Record descriptor for xxx
SYMBOL TABLE ELEMENT	Symbol table address
SYMBOL TABLE..xxx	Symbol table for xxx
SYMTAB DED..xxx	Symbol table DED for xxx
USER LABEL..xxx	Source program label for xxx
xxx	Variable with name xxx. If the variable is not initialized, no text appears against the comment. There is also no static offset if the variable is an array (the static offset can be calculated from the array descriptor, if required).

Variable storage map

For automatic and static internal variables, the variable storage map contains the following information:

- PL/I identifier name
- Level
- Storage class
- Name of the PL/I block in which it is declared
- Offset from the start of the storage area, in both decimal and hexadecimal form

If the LIST option is also specified, a map of the static internal and external control sections, called the static storage map, is also produced.

Object code listing

The object code listing consists of the machine instructions and a translation of these instructions into a form that resembles assembler and includes comments, such as source program statement numbers.

The machine instructions are formatted into blocks of code, headed by the statement or line number in the PL/I source program listing. Generally, only executable statements appear in the listing. DECLARE statements are not normally included. The names of PL/I variables, rather than the addresses that appear in the machine code, are listed. Special mnemonics are used to refer to some items, including test hooks, descriptors, and address constants.

Statements in the object code listing are ordered by block, as they are sequentially encountered in the source program. Statements in the external procedure are given first, followed by the statements in each inner block. As a result, the order of statements frequently differs from that of the source program.

Every object code listing begins with the name of the external procedure. The actual entry point of the external procedure immediately follows the heading comment REAL ENTRY. The subsequent machine code is the prolog for the block,

which performs block activation. The comment PROCEDURE BASE marks the end of the prolog. Following this is a translation of the first executable statement in the PL/I source program.

Following are the comments used in the listing:

Table 23. Comments in a PL/I object code listing

Comment	Function
BEGIN BLOCK <i>xxx</i>	Indicates the start of the begin block with label <i>xxx</i>
BEGIN BLOCK NUMBER <i>n</i>	Indicates the start of the begin block with number <i>n</i>
CALCULATION OF COMMONED EXPRESSION FOLLOWS	Indicates that an expression used more than once in the routine is calculated at this point
CODE MOVED FROM STATEMENT NUMBER <i>n</i>	Indicates object code moved by the optimization process to a different part of the routine and gives the number of the statement from which it originated
COMPILER GENERATED SUBROUTINE <i>xxx</i>	Indicates the start of compiler-generated subroutine <i>xxx</i>
CONTINUATION OF PREVIOUS REGION	Identifies the point at which addressing from the previous routine base recommences
END BLOCK	Indicates the end of a begin block
END INTERLANGUAGE PROCEDURE <i>xxx</i>	Identifies the end of an ILC procedure <i>xxx</i>
END OF COMMON CODE	Identifies the end of code used in running more than one statement
END OF COMPILER GENERATED SUBROUTINE	Indicates the end of the compiler-generated subroutine
END PROCEDURE	Identifies the end of a procedure
END PROGRAM	Indicates the end of the external procedure
INITIALIZATION CODE FOR <i>xxx</i>	Indicates the start of initialization code for variable <i>xxx</i>
INITIALIZATION CODE FOR OPTIMIZED LOOP FOLLOWS	Indicates that some of the code that follows was moved from within a loop by the optimization process
INTERLANGUAGE PROCEDURE <i>xxx</i>	Identifies the start of an implicitly generated ILC procedure <i>xxx</i>
METHOD OR ORDER OF CALCULATING EXPRESSIONS CHANGED	Indicates that the order of the code following was changed to optimize the object code
ON-UNIT BLOCK NUMBER <i>n</i>	Indicates the start of an ON-unit block with number <i>n</i>
ON-UNIT BLOCK END	Indicates the end of the ON-unit block
PROCEDURE <i>xxx</i>	Identifies the start of the procedure labeled <i>xxx</i>
PROCEDURE BASE	Identifies the address loaded into the base register for the procedure
PROGRAM ADDRESSABILITY REGION BASE	Identifies the address where the routine base is updated if the routine size exceeds 4096 bytes and consequently cannot be addressed from one base
PROLOGUE BASE	Identifies the start of the prolog code common to all entry points into that procedure
REAL ENTRY	Precedes the actual executable entry point for a procedure

Table 23. Comments in a PL/I object code listing (continued)

Comment	Function
STATEMENT LABEL <i>xxx</i>	Identifies the position of source program statement label <i>xxx</i>
STATEMENT NUMBER <i>n</i>	Identifies the start of code generated for statement number <i>n</i> in the source listing

In certain cases the compiler uses mnemonics to identify the type of operand in an instruction and, where applicable, follows the mnemonic by the name of a PL/I variable.

Table 24. PL/I mnemonics

Mnemonic	Explanation
A.. <i>xxx</i>	Address constant for <i>xxx</i>
ADD.. <i>xxx</i>	Aggregate descriptor for <i>xxx</i>
BASE.. <i>xxx</i>	Base address of variable <i>xxx</i>
BLOCK.. <i>n</i>	Identifier created for an otherwise unlabeled block
CL.. <i>n</i>	Compiler-generated label number <i>n</i>
D.. <i>xxx</i>	Descriptor for <i>xxx</i>
DED.. <i>xxx</i>	Data element descriptor for <i>xxx</i>
HOOK...ENTRY	Debugging tool block entry hook
HOOK...BLOCK-EXIT	Debugging tool block exit hook
HOOK...PGM-EXIT	Debugging tool program exit hook
HOOK...PRE-CALL	Debugging tool pre-call hook
HOOK...INFO	Additional pre-call hook information
HOOK...POST-CALL	Debugging tool post call hook
HOOK...STMT	Debugging tool statement hook
HOOK...IF-TRUE	Debugging tool IF true hook
HOOK...IF-FALSE	Debugging tool ELSE hook
HOOK...WHEN	Debugging tool WHEN true hook
HOOK...OTHERWISE	Debugging tool OTHERWISE true hook
HOOK...LABEL	Debugging tool label hook
HOOK...DO	Debugging tool iterative DO hook
HOOK...ALLOC	Debugging tool ALLOCATE controlled hook
WSP.. <i>n</i>	Workspace, followed by identifying number <i>n</i>
L.. <i>xxx</i>	Length of variable <i>xxx</i>
PR.. <i>xxx</i>	Pseudoregister vector slot for <i>xxx</i>
LOCATOR.. <i>xxx</i>	Locator for <i>xxx</i>
RKD.. <i>xxx</i>	Record or key descriptor for <i>xxx</i>
VO.. <i>xxx</i>	Virtual origin for <i>xxx</i> (the address where element 0 is held for a one-dimensional array, element 0,0 for a two-dimensional array, and so on)

Generating a Language Environment dump of a PL/I routine

To generate a dump of a PL/I routine, you can call either the Language Environment callable service CEE3DMP or PLIDUMP. For information about calling CEE3DMP, see “Generating a Language Environment dump with CEE3DMP” on page 35.

PLIDUMP syntax and options

PLIDUMP calls intermediate PL/I library routines, which convert most PLIDUMP options to CEE3DMP options. The following list contains PLIDUMP options and the corresponding CEE3DMP option, if applicable.

Some PLIDUMP options do not have corresponding CEE3DMP options, but continue to function as PL/I default options. The list following the syntax diagram provides a description of those options.

Note: VisualAge PL/I does not support multitasking, therefore, the PLIDUMP options that refer to multitasking do not apply to VisualAge PL/I.

PLIDUMP now conforms to National Language Support standards.

PLIDUMP can supply information across multiple Language Environment enclaves. If an application running in one enclave fetches a main procedure (an action that creates another enclave), PLIDUMP contains information about both procedures.

The syntax and options for PLIDUMP are shown below.

Syntax

►►—PLIDUMP—(—*char.-string-exp 1*—,—*char.-string-exp 2*—)————►►

char.-string-exp 1

A dump options character string consisting of one or more of the following:

- A** All. Results in a dump of all tasks including the ones in the WAIT state.
- B** BLOCKS (PL/I hexadecimal dump). Dumps the control blocks used in Language Environment and member language libraries. For PL/I, this includes the DSA for every routine on the call chain and PL/I “global” control blocks, such as Tasking Implementation Appendage (TIA), Task Communication Area (TCA), and the PL/I Tasking Control Block (PTCB). PL/I file control blocks and file buffers are also dumped if the F option is specified.
- C** Continue. The routine continues after the dump.
- E** Exit. The enclave terminates after the dump. In a multitasking environment, if PLIDUMP is called from the main task, the enclave terminates after the dump. If PLIDUMP is called from a subtask, the subtask and any subsequent tasks created from the subtask terminate after the dump. In a multithreaded environment, if PLIDUMP is called from the Initial Process Thread (IPT), the enclave terminates after the dump. If PLIDUMP is called from a non-IPT, only the non-IPT terminates after the dump.

- F** FILE INFORMATION. A set of attributes for all open files is given. The contents of the file buffers are displayed if the B option is specified.
- H** STORAGE in hexadecimal. A SNAP dump of the region is produced. A ddname of CEESNAP must be provided to direct the CEESNAP dump report.
- K** BLOCKS (when running under CICS). The Transaction Work Area is included.
- Note:** This option is not supported under VisualAge PL/I.
- NB** NOBLOCKS.
- NF** NOFILES.
- NH** NOSTORAGE.
- NK** NOBLOCKS (when running under CICS).
- NT** NOTRACEBACK.
- O** THREAD(CURRENT). Results in a dump of only the current task or current thread (the invoker of PLIDUMP).
- S** Stop. The enclave terminates after the dump. In a multitasking environment, regardless of whether PLIDUMP is called from the main task or a subtask, the enclave terminates after the dump. In a multithreaded environment, regardless of whether PLIDUMP is called from the IPT or a non-IPT, the enclave terminates after the dump (in which case there is no fixed order as to which thread terminates first).
- T** TRACEBACK. Includes a traceback of all routines on the call chain. The traceback shows transfers of control from either calls or exceptions. BEGIN blocks and ON-units are also control transfers and are included in the trace. The traceback extends backwards to the main program of the current thread.

T, F, C, and A are the default options.

char.-string-exp 2

A user-identified character string up to 80 characters long that is printed as the dump header.

PLIDUMP usage notes

If you use PLIDUMP, the following considerations apply:

- If a routine calls PLIDUMP a number of times, use a unique user-identifier for each PLIDUMP invocation. This simplifies identifying the beginning of each dump.
- In MVS or TSO, you can use ddnames of CEEDUMP, PLIDUMP, or PL1DUMP to direct dump output. If no ddname is specified, CEEDUMP is used.
- The data set defined by the PLIDUMP, PL1DUMP, or CEEDUMP DD statement should specify a logical record length (LRECL) of at least 131 to prevent dump records from wrapping.
- When you specify the H option in a call to PLIDUMP, the PL/I library issues an OS SNAP macro to obtain a dump of virtual storage. The first invocation of PLIDUMP results in a SNAP identifier of 0. For each successive invocation, the ID is increased by one to a maximum of 256, after which the ID is reset to 0.
- Support for SNAP dumps using PLIDUMP is provided only under MVS. SNAP dumps are not produced in a CICS environment.

- If the SNAP does not succeed, the CEE3DMP DUMP file displays the message:
Snap was unsuccessful
Failure to define a CEESNAP data set is the most likely cause of an unsuccessful CEESNAP.
- If the SNAP is successful, CEE3DMP displays the message:
Snap was successful; snap ID = *nnn*
where *nnn* corresponds to the SNAP identifier described above. An unsuccessful SNAP does not result in an incrementation of the identifier.
- To ensure portability across system platforms, use PLIDUMP to generate a dump of your PL/I routine.

Finding PL/I information in a dump

The following sections discuss PL/I-specific information located in the following sections of a Language Environment dump:

- Traceback
- Control Blocks for Active Routines
- Control Block Associated with the Thread
- File Status and Attributes

Traceback

Examine the traceback section of the dump, shown in Figure 104 on page 248, for condition information about your routine and information about the statement number and address where the exception occurred.

PLIDUMP was called from statement number 6 at offset +000000D6 from ERROR ON-UNIT with entry address 00020168

Information for enclave EXAMPLE

Information for thread 800000000000000

Registers on Entry to CEE3DMP:

```
PM..... 0100
GPR0..... 00000000 GPR1..... 00077448 GPR2..... 053AD9AF GPR3..... 853AD514
GPR4..... 00000001 GPR5..... 053AD314 GPR6..... 80077454 GPR7..... 00000000
GPR8..... 00000001 GPR9..... 80000000 GPR10..... 00077470 GPR11..... 000F7490
GPR12..... 0006A520 GPR13..... 000773C8 GPR14..... 80060712 GPR15..... 853F7918
FPR0..... 4D000000 00043C31 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
```

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Load	Service	Statement	Status
005359A0	CEEKKMRA	00654438	+00000748	CEEKKMRA	00654438	+00000748	CELE38			Call
006D9810	LIBRARY(PLI)	005CBE98	+000000B2	LIBRARY(PLI)	005CBE98	+000000B2	CEEPLPKA			Call
005358A0	EXAMPLE	00020080	+000001BE	ERR ON-UNIT	00020168	+000000D6	CELE38		6	Call
00535698	IBMRERPL	007CB410	+00000528	IBMRERPL	007CB410	+00000528				Call
005355B0	CEEV010	005B5000	+000000E8	CEEV010	005B5000	+000000E8				Call
006C3018	CEEHDS	005F6D00	+00000970	CEEHDS	005F6D00	+00000970				Call
00535428	IBMRERRI	007CB040	+00000254	IBMRERRI	007CB040	+00000254				Exception
00535358	EXAMPLE	00020080	+00000296	LAB1: BEGIN	00020258	+000000BE			11	Call
00535258	EXAMPLE	00020080	+000000D0	EXAMPLE	00020088	+000000C8			8	Call
005351B0	IBMRPMIA	007CABD0	+000002FA	IBMRPMIA	007CABD0	+000002FA				Call
005350C8	CEEV010	005B5000	+000001FE	CEEV010	005B5000	+000001FE				Call
00535018	CEEBEXT	005E55F0	+0000012E	CEEBEXT	005E55F0	+0000012E				Call

Condition Information for Active Routines

Condition Information for IBMRERRI (DSA address 00535428)

CIB Address: 006C33C8

Current Condition:

IBM0930S

Original Condition:

IBM0421S ONCODE=520 The SUBSCRIPTRANGE condition was raised.

Location:

Program Unit: IBMRERRI Entry: IBMRERRI Statement: Offset: +00000254

:

Figure 104. Traceback section of dump

PL/I task traceback

A task traceback table is produced for multitasking programs showing the task invocation sequence (trace). For each task, the CAA address, task variable address, event variable address, thread ID, and absolute priority appear in the traceback table. An example is shown in Figure 105 on page 249.

PLIDUMP was called from statement number 23 at offset +000000D2 from BEGIN BLOCK6 within task SUBTSK2

PL/I Task Traceback:

Task	Attached by	Thread ID	TCA Addr	EV Addr	TV Addr	Absolute Priority
SUBTSK2	SUBTSK1	03B2CB7800000003	00070708	000684F8	000684E8	000
SUBTSK1	SUBTASK	03B2C2D000000002	00066708	00034498	00034488	000
SUBTASK	TASKING	03B2BA2800000001	0005D708	00034468	00034458	056
TASKING		03B2B18000000000	00016658	000545D4	0005423C	254

Information for enclave TASKING

Information for thread 03B2CB7800000003

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Load	Service	Statement	Status
000726A8	CEEKMMRA	0394AAC0	+00000860	CEEKMMRA	0394AAC0	+00000860	CELE38			Call
0006E2E8	IBMRKDM	039D2450	+000000BA	IBMRKDM	039D2450	+000000BA	CEEPLPKA			Call
000725B8	SUBTSK1	00007760	+0000052A	BEGIN BLOCK6	00007BB8	+000000D2	CELE38		23	Call
000724F8	SUBTSK1	00007760	+0000043E	PROCA	00007B1C	+00000082			21	Call
00072430	SUBTSK1	00007760	+0000036E	SUBTSK2	00007A24	+000000AA			19	Call
000715D8	IBMUPTMM	00023920	+000000F6	IBMUPTMM	00023920	+000000F6				Call
7F6653A0		00006E38	+00000000		00006E38	+00000000				Call

Figure 105. Task traceback section

Condition information

If the dump was called from an ON-unit, the type of ON-unit is identified in the traceback as part of the entry information. For ON-units, the values of any relevant condition built-in functions (for example, ONCHAR and ONSOURCE for conversion errors) appear. In cases where the cause of entry into the ON-unit is not stated, usually when the ERROR ON-unit is called, the cause of entry appears in the condition information.

Statement number and address where error occurred

This information, which is the point at which the condition that caused entry to the ON-unit occurred, can be found in the traceback section of the dump.

If the condition occurs in compiled code, and you compiled your routine with either GOSTMT or GONUMBER, the statement numbers appear in the dump. To identify the assembler instruction that caused the error, use the traceback information in the dump to find the program unit (PU) offset of the statement number in which the error occurred. Then find that offset and the corresponding instruction in the object code listing.

Control blocks for active routines

This section shows the stack frames for all active routines, and the static storage. Use this section of the dump to identify variable values, determine the contents of parameter lists, and locate the timestamp.

Figure 106 on page 250 shows this section of the dump.

Control Blocks for Active Routines:

```
:
:
DSA for PLIDMPB: 003CA438
+000000  FLAGS.... 8025      member... 0000      BKC..... 003CA348  FWC..... 00000000  R14..... 4E020360
+000010  R15..... 805611C0  R0..... 003CA588  R1..... 000204AC  R2..... 5E0202BE  R3..... 000203B8
+000024  R4..... 003CA57C  R5..... 00000000  R6..... 00000009  R7..... 00000001  R8..... 003CA554
+000038  R9..... 003CA4F8  R10..... 00000004  R11..... 00000008  R12..... 003CA4EC  reserved. 00542280
+00004C  NAB..... 003CA588  PNAB..... 003CA588  reserved. 91E091E0 003CA348 007C8090 00627188
+000064  reserved. 003CA4E8  reserved. 005C8E10  MODE..... 0058C848  reserved. 003CA608 00620258
+000078  reserved. 003CA4E8  reserved. 003CA4EC
DYNAMIC SAVE AREA (PLIDMPB): 003CA438
+000000 003CA438 80250000 003CA348 00000000 4E020360 805611C0 003CA588 000204AC 5E0202BE | .....t.....+.-.....vh.....;...
+000020 003CA458 000203B8 003CA57C 00000000 00000009 00000001 003CA554 003CA4F8 00000004 | .....v@.....v.....u8...
+000040 003CA478 00000008 003CA4EC 00542280 003CA588 003CA588 91E091E0 003CA348 007C8090 | .....u.....vh..vhj.j...t..@..
+000060 003CA498 00627188 003CA4E8 005C8E10 0058C848 003CA608 00620258 003CA4E8 003CA4EC | ...h..uY.*....H...w.....uY..u.
+000080 003CA4B8 00627250 003CA608 00627258 000204D7 00100000 00627250 003CA4E8 00627250 | ...&.w.....P.....&.uY...&.
+0000A0 003CA4D8 003CA4EC 003CA530 003CA608 00000000 00000248 00000003 003CA4F8 00020474 | ..u...v...w.....u8...
+0000C0 003CA4F8 0004E385 9989B870 00000000 00000008 C7899393 89A297A8 8E572778 40404040 | ..Teri.....Gillispy...
+0000E0 003CA518 003CA520 40404040 00020438 00020418 40404040 8002046C 00404000 007C8004 | ..v. .... .%. ..@..
+000100 003CA538 40404040 40404040 00014040 00542460 E3C2C6C3 003CA548 00040000 D7D3C9C4 | .. ..-TBFC..v.....PLID
+000120 003CA558 E4D4D740 83819393 85844086 99969440 97999683 8584A499 8540D7D3 C9C4D4D7 | UMP called from procedure PLIDMP
+000140 003CA578 C2404040 003CA554 00250000 40404040 88004040 00542280 003CAA58 6E579B82 | B ..v..... h. ....>..b
:
:
STATIC FOR PROCEDURE PLIDMP  TIMESTAMP: 2 DEC 92 11:26:26
STARTING FROM: 000203B8
+000000 000203B8 E0000300 00020088 00020116 00020188 000201E2 00020254 000202AE 000202BE | .....h.....h...S.....
+000020 000203D8 000202BE 000202BE 000202BE 80020A38 80020A50 80020A68 80020A80 80020A98 | .....&.....q
+000040 000203F8 80020AB0 80021340 80021148 80020AC8 800213B8 800213D0 80020AE0 80020AF8 | .....H.....8
+000060 00020418 20000002 1F802800 00040008 00000000 000204C8 000F0000 000204D7 00100000 | .....H.....P...
+000080 00020438 000204F3 00100000 00000000 00040000 00000000 00250000 00020608 00110000 | ...3.....
+0000A0 00020458 00000000 00020474 91E091E0 00000005 00000009 00000001 00000003 00000000 | .....j.j.....
+0000C0 00020478 000C8000 0000000E 000C8000 000206D0 000206D0 003CA320 8002046C 000206D0 | .....t.....%
+0000E0 00020498 003CA410 8002046C 000206D0 003CA520 8002046C 003CA54C 803CA57C 80020A08 | ..u...%.....v.....v<..v@....
```

Figure 106. Control blocks for active routines section of the dump

Automatic variables

To find automatic variables, use an offset from the stack frame of the block in which they are declared. This information appears in the variable storage map generated when the MAP compiler option is in effect. If you have not used the MAP option, you can determine the offset by studying the listing of compiled code instructions.

Static variables

If your routine is compiled with the MAP option, you can find static variables by using an offset in the variable storage map. If the MAP option is not in effect, you can determine the offset by studying the listing of compiled code.

Based variables

To locate based variables, use the value of the defining pointer. Find this value by using one of the methods described above to find static and automatic variables. If the pointer is itself based, you must find its defining pointer and follow the chain until you find the correct value.

The following is an example of typical code for X BASED (P), with P AUTOMATIC:

```
58 60 D 0C8          L 6,P
:
58 E0 6 000          L 14,X
```

P is held at offset X'C8' from register 13. This address points to X.

Take care when examining a based variable to ensure that the pointers are still valid.

Area variables

Area variables are located using one of the methods described above, according to their storage class.

The following is an example of typical code: for an area variable A declared AUTOMATIC:

```
41 60 D 0F8      LA 6,A
```

The area starts at offset X'F8' from register 13.

Variables in areas

To find variables in areas, locate the area and use the offset to find the variable.

Contents of parameter lists

To find the contents of a passed parameter list, first find the register 1 value in the save area of the calling routine's stack frame. Use this value to locate the parameter list in the dump. If R1=0, no parameters passed. For additional information about parameter lists, see either *PL/I for MVS & VM Programming Guide* or *VisualAge PL/I for OS/390 Programming Guide*.

Timestamp

If the TSTAMP compiler installation option is in effect, the date and time of compilation appear within the last 32 bytes of the static internal control section. The last three bytes of the first *word* give the offset to this information. The offset indicates the end of the timestamp. Register 3 addresses the static internal control section. If the BLOCK option is in effect, the timestamp appears in the static storage section of the dump.

Control blocks associated with the thread

This section of the dump, shown in Figure 107 on page 252, includes information about PL/I fields of the CAA and other control block information.

Control Blocks Associated with the Thread:

```
CAA: 0058C848
+000000 0058C848 00000800 00542648 003CA000 0044A000 00000000 0058C858 00000000 005421A0 | .....H.....|
+000020 0058C868 00000000 00000000 00542030 00000000 00542230 007C8004 005421F8 00000000 | .....8.....|
+000040 0058C888 005421C0 00000000 006CB660 00000000 006CBB88 006C66E0 00000000 00000000 | .....%-.....%|
+000060 0058C8A8 00000000 006C6660 00000000 006CB200 006CB4A0 006CB620 006C7028 04001010 | .....%-.....%|

:
DUMMY DSA: 0058E040
+000000 FLAGS.... 0000      member... 0000      BKC..... 000095E8 FWC..... 003CA018 R14..... 40020810
+000010 R15..... 805905F0 R0..... 00000E08 R1..... 0058AE14 R2..... 000206F0 R3..... 00000002
+000024 R4..... 00000000 R5..... 00000000 R6..... 00000000 R7..... 005801E0 R8..... 000206C0
+000038 R9..... 00D44A30 R10..... 00000000 R11..... 4002073A R12..... 0058C848 reserved. 00542280
+00004C NAB..... 003CA018 PNAB..... 003CA018 reserved. 00000000 00000000 00000000 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE..... 00000000 reserved. 00000000 00000000
+000078 reserved. 00000000 reserved. 00000000
```

CEE3DMP V1 R5.0: PLIDUMP called from procedure PLIDMPB.

08/05/95 11:29:13 AM

Page: 5

```
PL/I TCA APPENDAGE: 00542030
+000000 00542030 00000000 00000000 00000000 00000030 00000000 00000000 00000000 00000000 | .....|
+000020 00542050 005421D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | ...Q.....|
+000040 00542070 00000000 00000000 00000000 00000000 0058C848 00000000 00000000 00000000 | .....H.....|
+000060 00542090 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|
```

Enclave Control Blocks:

```
EDB: 0058AD68
+000000 0058AD68 C3C5C5C5 C4C24040 80400001 0058C6B8 0058B3A8 00000000 00000000 00000000 | CEEEDB . ...F...y.....|
+000020 0058AD88 0058B0D8 0058B108 005801E0 0057F198 00000000 8057F088 0058AE14 00000000 | ...Q.....1q.....0h.....|
MEML: 0058C6B8
+000000 0058C6B8 00000000 00000000 00592030 00000000 00000000 00000000 00592030 00000000 | .....|
+000020 0058C6D8 - +00009F 0058C757      same as above
+0000A0 0058C758 00000000 00000000 0054A000 00000000 00000000 00592030 00000000 | .....|
```

File Status and Attributes:

```
ATTRIBUTES OF FILE: SYSPRINT
STREAM OUTPUT PRINT ENVIRONMENT( F BLKSIZE(80) RECSIZE(80) BUFFERS(2) )
CONTENTS OF BUFFERS
BUFFER: 007CDF60
+000000 007CDF60 40D7D3C9 C4D4D7C2 40E2A381 99A38995 87404040 40404040 40404040 40404040 | PLIDMPB Starting
+000020 007CDF80 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+000040 007CDFA0 40404040 40404040 40404040 40404040 40D7D3C9 C4D4D7C1 40E2A381 99A38995 | PLIDMPA Starting
BUFFER: 007CDFB0
+000000 007CDFB0 40D7D3C9 C4D4D7C1 40E2A381 99A38995 87404040 40404040 40404040 40404040 | PLIDMPA Starting
+000020 007CDFD0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+000040 007CDFE0 40404040 40404040 40404040 40404040 007CE031 7A958983 924BE3C5 D9C94040 |
```

File Control Blocks:

```
FILE CONTROL BLOCK (FCB): 007C8004
+000000 007C8004 00000000 00000000 0056B3CA 005CB5EC 000206D0 007C8090 00000000 00000000 | .....*. ....@.....|
+000020 007C8024 00000000 41211100 82000000 00000104 00500000 00000050 007CDF60 E3C600F4 | .....b.....&.....&@.-TF.4
+000040 007C8044 00000000 00000000 00000000 007CDF72 003E0001 003C004F 00030000 00000000 | .....@.....|.....|
+000060 007C8064 00000000 006CBB88 00000000 003CA520 00000000 00000000 00000000 00000000 | .....%. ....v.....|
DATA CONTROL BLOCK (DCB): 007C8090
+000000 007C8090 00000000 00000000 00000000 00000000 00280000 027CDF58 00504000 007D94B0 | .....@.....& ..'m.
+000020 007C80B0 40000001 84000000 00000048 007D94A0 92D5927E 00000001 0C5CB7FA 00090050 | ...d.....'m.kNk=.....*.....&
+000040 007C80D0 00000000 007D94B0 007CDFB0 007CDFB0 00000050 80000001 00000000 00000001 | .....'m.@.....&.....|
DECLARE CONTROL BLOCK (DCLCB): 000206D0
+000000 000206D0 FFFFFFFC 41201000 02D70F00 00000000 00000014 0008E2E8 E2D7D9C9 D5E30000 | .....P.....SYSPRINT..|
```

Process Control Blocks:

```
PCB: 0057F198
+000000 0057F198 C3C5C5D7 C3C24040 02020220 00000000 00000000 00000000 0057FB30 005C8E10 | CEEPCB .....*..|
+000020 0057F1B8 005C75B8 005C2288 005C1C80 00000000 00000000 00000000 0057FB18 0057FB30 | .....*..h.*.....|

MEML: 0057FB30
+000000 0057FB30 00000000 00000000 00592030 00000000 00000000 00000000 00592030 00000000 | .....|
+000020 0057FB50 - +00009F 0057FBCF      same as above
```

Figure 107. Control blocks associated with the thread section of the dump

The CAA

The address of the CAA control block appears in this section of the dump. If the BLOCK option is in effect, the complete CAA (including the PL/I implementation appendage) appears separately from the body of the dump. Register 12 addresses the CAA.

File status and attribute information

This part of the dump includes the following information:

- The default and declared attributes of all open files

- Buffer contents of all file buffers
- The contents of FCBs, DCBs, DCLCBs, IOCBs, and control blocks for the process or enclave

PL/I contents of the Language Environment trace table

Language Environment provides three PL/I trace table entry types that contain character data:

- Trace entry 100 occurs when a task is created.
- Trace entry 101 occurs when a task that contains the tasking CALL statements is terminated.
- Trace entry 102 occurs when a task that does not contain a tasking CALL statement is terminated.

The format for trace table entries 100, 101, and 102 is:

```
—>(100) NameOfCallingTask NameOfCalledTask OffsetOfCallStmt
        UserAgrPtr CalledTaskPtr TaskVarPtr EventVarPtr
        PriorityPtr CallingR2-R5 CallingR12-R14
```

```
—>(101) NameOfReturnTask ReturnerR2-R5 ReturnerR12-R14
```

```
—>(102) NameOfReturnTask
```

For more information about the Language Environment trace table format, see “Understanding the trace table entry (TTE)” on page 116.

Debugging example of PL/I routines

This section contains examples of PL/I routines and instructions for using information in the Language Environment dump to debug them. Important areas in the source code and in the dump for each routine are highlighted.

Subscript range error

Figure 108 on page 254 illustrates an error caused by an array subscript value outside the declared range. In this example, the declared array value is 10.

This routine was compiled with the options LIST, TEST, GOSTMT, and MAP. It was run with the TERMTHDACT(TRACE) option to generate a traceback for the condition.

```

15688-235 IBM SAA AD/Cycle PL/I          Ver 1 Rel 2 Mod 0          18 NOV 92   15:57:56   PAGE   1
-OPTIONS SPECIFIED
0*PROCESS  GOSTMT LIST S STG LC(100) TEST MAP;          00001000
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE: PROC  OPTIONS(MAIN);          PAGE   2
- SOURCE LISTING
- STMT
0
1  EXAMPLE: PROC  OPTIONS(MAIN);          00002000
2  DCL Array(10) Fixed bin(31);          00003000
3  DCL (I,Array_End) Fixed bin(31);      00004000
4  On error                               00005000
5  Begin;                                00006000
6  On error system;                       00007000
7  Call plidump('tbnfs','Plidump called from error ON-unit'); 00008000
8  End;                                   00009000
9  (subrg): /* Enable subscriptrange condition*/ 00010000
10 Lab11: Begin;                           00011000
11 Array_End = 20;                          00012000
12 Do I = 1 to Array_End; /* Loop to initialize array */ 00013000
13 Array(I) = 2; /* Set array elements to 2 */ 00014000
14 End;                                     00015000
15 End Lab11;                              00016000
16 End Example;                            00017000
17                                         00018000
18                                         00019000
19                                         00020000
20                                         00021000
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE: PROC  OPTIONS(MAIN);          PAGE   3
- VARIABLE STORAGE MAP
-IDENTIFIER          LEVEL          OFFSET          (HEX)  CLASS  BLOCK
I                    1            200            C8    AUTO    EXAMPLE
ARRAY_END            1            204            CC    AUTO    EXAMPLE
ARRAY                1            208            D0    AUTO    EXAMPLE
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE: PROC  OPTIONS(MAIN);          PAGE   6
- OBJECT LISTING

```

Figure 108. Example of moving a value outside an array range

Figure 109 on page 255 shows sections of the dump generated by a call to PLIDUMP.

PLIDUMP was called from statement number 6 at offset +000000D6 from ERROR ON-unit with entry address 00020168

Information for enclave EXAMPLE

Information for thread 8000000000000000

Registers on Entry to CEE3DMP:

```
PM..... 0100
GPR0..... 00000000 GPR1..... 00077448 GPR2..... 053AD9AF GPR3..... 853AD514
GPR4..... 00000001 GPR5..... 053AD314 GPR6..... 80077454 GPR7..... 00000000
GPR8..... 00000001 GPR9..... 80000000 GPR10..... 00077470 GPR11..... 000F7490
GPR12..... 0006A520 GPR13..... 000773C8 GPR14..... 80060712 GPR15..... 853F7918
FPR0..... 4D000000 00043C31 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
```

Traceback:

DSA Addr	Program Unit	PU Addr	PU Offset	Entry	E Addr	E Offset	Load	Service	Statement	Status
005359A0	CEEKMRRA	00654438	+00000748	CEEKMRRA	00654438	+00000748	CELE3B			Call
006D9810	LIBRARY(PLI)	005CBE98	+000000B2	LIBRARY(PLI)	005CBE98	+000000B2	CEEPLPKA			Call
							CELE3B			
005358A0	EXAMPLE	00020080	+000001BE	ERR ON-UNIT	00020168	+000000D6			6	Call
00535698	IBMRERPL	007CB410	+00000528	IBMRERPL	007CB410	+00000528				Call
00535580	CEEV010	005B5000	+000000E8	CEEV010	005B5000	+000000E8				Call
006C3018	CEEHDS	005F6D00	+00000970	CEEHDS	005F6D00	+00000970				Call
00535428	IBMRERRI	007CB040	+00000254	IBMRERRI	007CB040	+00000254				Exception
00535358	EXAMPLE	00020080	+00000296	LAB1: BEGIN	00020258	+000000BE			11	Call
00535258	EXAMPLE	00020080	+000000D0	EXAMPLE	00020088	+000000C8			8	Call
00535180	IBMRPMIA	007CABD0	+000002FA	IBMRPMIA	007CABD0	+000002FA				Call
005350C8	CEEV010	005B5000	+000001FE	CEEV010	005B5000	+000001FE				Call
00535018	CEEBEXT	005E55F0	+0000012E	CEEBEXT	005E55F0	+0000012E				Call

Condition Information for Active Routines

Condition Information for IBMRERRI (DSA address 00535428)

CIB Address: 006C33C8

Current Condition:

IBM0930S

Original Condition:

IBM0421S ONCODE=520 The SUBSCRIPTRANGE condition was raised.

Location:

Program Unit: IBMRERRI Entry: IBMRERRI Statement: Offset: +00000254

Control Blocks for Active Routines:

DYNAMIC SAVE AREA (EXAMPLE): 00535258

+000000	00535258	C0250000	005351B0	00000000	4E020152	00020258	00535358	00535258	4E020140	é.....+.....+..
+000020	00535278	00020350	0002053C	00535258	00535328	0002053C	00020408	00000001	00020088	...&.....h
+000040	00535298	00000005	006F3848	006D9410	00535358	00535358	91E091A0	00000000	00020408	?....._m.....j.j.....
+000060	005352B8	00000000	00000000	00000000	00000001	00535310	00000200	00000001	00000000	
+000080	005352D8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
+0000A0	005352F8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
+0000C0	00535318	00535328	000203A4	0000000B	00000014	00000002	00000002	00000002	00000002	
+0000E0	00535338	00000002	00000002	00000002	00000002	00000002	00000002	00000000	00000000	

:

Figure 109. Sections of the Language Environment dump

To debug this routine, use the following steps:

1. In the dump, PLIDUMP was called by the ERROR ON-unit in statement 6. The traceback information in the dump shows that the exception occurred following statement 11.
2. Locate the Original Condition message in the Condition Information for Active Routines section of the dump. The message is IBM0421S ONCODE=520 The SUBSCRIPTRANGE condition was raised. This message indicates that the exception occurred when an array element value exceeded the subscript range value (in this case, 10). For more information about this message, see *z/OS Language Environment Run-Time Messages*.
3. Locate statement 9 in the routine in Figure 108 on page 254. The instruction is `Array_End = 20`. This statement assigns a 20 value to the variable `Array_End`.
4. Statement 10 begins the DO-loop instruction `Do I = 1 to Array_End`. Since the previous instruction (statement 9) specified that `Array_End = 20`, the loop in statement 10 should run until `I` reaches a 20 value.

The instruction in statement 2, however, declared a 10 value for the array range. Therefore, when the I value reached 11, the SUBSCRIPTRANGE condition was raised.

The following steps provide another method for finding the value that raised the SUBSCRIPTRANGE condition.

1. Locate the offset of variable I in the variable storage map in Figure 108 on page 254. Use this offset to find the I value at the time of the dump. In this example, the offset is X'C8'.
2. Now find offset X'C8' from the start of the stack frame in Figure 109 on page 255.

The block located at this offset contains the value that exceeded the array range, X'B' or 11.

Calling a nonexistent subroutine

Figure 110 demonstrates the error of calling a nonexistent subroutine. This routine was compiled with the LIST, MAP, and GOSTMT compiler options. It was run with the TERMTHDACT(DUMP) run-time option to generate a traceback.

15688-235 IBM SAA AD/Cycle PL/I		Ver 1 Rel 2 Mod 0	25 NOV 92	13:47:13	PAGE	1
-OPTIONS SPECIFIED						
0+PROCESS GOSTMT LIST S STG LC(100) TEST MAP;				00001000		
15688-235 IBM SAA AD/Cycle PL/I		EXAMPLE1: PROC OPTIONS(MAIN);			PAGE	2
-SOURCE LISTING						
-STMT						
0						
1	EXAMPLE1: PROC OPTIONS(MAIN);			00002000		
2	DCL Prog01 entry external;			00003000		
3	On error			00004000		
4	Begin;			00005000		
5	On error system;			00006000		
6	Call plidump('tbnfs','Plidump called from error ON-unit');			00007000		
7	End;			00008000		
8	Call Prog01; /* Call external program PR0G01 */			00009000		
9	End Example1;			00010000		
10				00011000		
11				00012000		
12				00013000		
13				00014000		
15688-235 IBM SAA AD/Cycle PL/I		EXAMPLE1: PROC OPTIONS(MAIN);			PAGE	3
-STORAGE REQUIREMENTS						
-BLOCK, SECTION OR STATEMENT		TYPE	LENGTH (HEX)	DSA SIZE (HEX)		
-EXAMLE11		PROGRAM CSECT	444 1BC			
-EXAMLE12		STATIC CSECT	292 124			
-EXAMPLE1		PROCEDURE BLOCK	210 D2	192 C0		
-BLOCK 2 STMT 3		ON UNIT	232 E8	256 100		
15688-235 IBM SAA AD/Cycle PL/I		EXAMPLE1: PROC OPTIONS(MAIN);			PAGE	4
-STATIC INTERNAL STORAGE MAP						

Figure 110. Example of calling a nonexistent subroutine

Figure 111 on page 257 shows the traceback and condition information from the dump.

PLIDUMP was called from statement number 5 at offset +000000D6 from ERROR ON-unit with entry address 00020154

Information for enclave EXAMPLE1

Information for thread 800000000000000

Registers on Entry to CEE3DMP:

```
PM..... 0100
GPR0..... 00000000 GPR1..... 00077448 GPR2..... 053AD9AF GPR3..... 853AD514
GPR4..... 00000001 GPR5..... 053AD314 GPR6..... 80077454 GPR7..... 00000000
GPR8..... 00000001 GPR9..... 80000000 GPR10..... 00077470 GPR11..... 000F7490
GPR12..... 0006A520 GPR13..... 000773C8 GPR14..... 80060712 GPR15..... 853F7918
FPR0..... 4D000000 00043C31 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
```

Figure 111. Sections of the Language Environment dump (Part 1 of 2)

```
Traceback:
DSA Addr  Program Unit  PU Addr  PU Offset  Entry          E Addr  E Offset  Statement  Status
003FC708  CEEKKMRA    004AB438 +00000748 CEEKKMRA      004AB438 +00000748          Call
006D4680  LIBRARY (PLI) 004931C0 +000000B2 LIBRARY (PLI) 004931C0 +000000B2          Call
003FC608  EXAMPLE1    00020080 +000001AA ERR ON-UNIT  00020154 +000000D6      5 Call
003FC400  IBMRRERPL   006E33D8 +00000528 IBMRRERPL     006E33D8 +00000528          Call
003FC318  CEEEV010    0047C000 +0000011C CEEEV010     0047C000 +0000011C          Call
00623018  CEEHDSP     005A1D00 +000009B8 CEEHDSP      005A1D00 +000009B8          Call
003FC258  EXAMPLE1    00020080 +000000C0 EXAMPLE1     0002008C +000000B4      7 Exception
003FC1B0  IBMRPMIA    006E2BA8 +00000316 IBMRPMIA     006E2BA8 +00000316          Call
003FC0C8  CEEEV010    0047C000 +000003DE CEEEV010     0047C000 +000003DE          Call
003FC018  CEEBBEXT    005905F0 +0000012E CEEBBEXT     005905F0 +0000012E          Call

Condition Information for Active Routines
Condition Information for EXAMPLE1 (DSA address 003FC258)
CIB Address: 006233C8
Current Condition:
CEE3201S The system detected an Operations exception.
Location:
Program Unit: EXAMPLE1 Entry: EXAMPLE1 Statement: 7 Offset: +000000C0
Machine State:
ILC..... 0003  Interruption Code..... 0001
PSW..... FFE40001 CE000006
GPR0..... 003FC318 GPR1..... 00000000 GPR2..... 4E020134 GPR3..... 00020240
GPR4..... 0002036C GPR5..... 00000000 GPR6..... 003FC310 GPR7..... 0002036C
GPR8..... 000202B0 GPR9..... 0002008C GPR10..... 00567198 GPR11..... 00000003
GPR12..... 00574848 GPR13..... 003FC258 GPR14..... 4E020142 GPR15..... 00000000
```

Figure 111. Sections of the Language Environment dump (Part 2 of 2)

To understand the traceback and debug this example routine, use the following steps:

1. Find the Current Condition message in the Condition Information for Active Routines section of the dump. The message is CEE3201S. The system detected an Operation exception. For more information about this message, see *z/OS Language Environment Run-Time Messages*.

This section of the dump also provides such information as the name of the active routine and the current statement number at the time of the dump.

2. Locate statement 7 in the routine (Figure 110 on page 256). This statement calls subroutine Prog01. The message CEE3201S, which indicates an operations exception, was generated because of an unresolved external reference.
3. Check the linkage editor output for error messages.

Divide-by-zero error

Figure 112 demonstrates a divide-by-zero error. In this example, the main PL/I routine passed bad data to a PL/I subroutine. The bad data in this example is 0, and the error occurred when the subroutine SUB1 attempted to use this data as a divisor.

```

-          SOURCE LISTING
-  STMT
0
1  SAMPLE: PROC  OPTIONS(MAIN) ;                                00002000
2  On error                                                    00003000
  begin;                                                       00004000
3    On error system;          /* prevent nested error conditions */ 00005000
4    Call PLIDUMP('TBC','PLIDUMP called from error ON-unit');    00006000
5    Put Data;                /* Display variables                */ 00007000
6  End;                                                           00008000
7  DECLARE                                                       00009000
    A_number    Fixed Bin(31),                                00010000
    My_Name     Char(13),                                     00011000
    An_Array(3) Fixed Bin(31) init(1,3,5);                     00012000
8  Put skip list('Sample Starting');                             00013000
9  A_number = 0;                                                 00014000
10 My_Name = 'Tery Gillaspay';                                   00015000
11 Call Sub1(a_number, my_name, an_array);                       00016000
12 SUB1: PROC(divisor, name1, Array1);                           00017000
13   Declare                                                    00018000
    Divisor     Fixed Bin(31),                                00019000
    Name1       Char(13),                                     00020000
    Array1(3)   Fixed Bin(31);                                 00021000
14   Put skip list('Sub1 Starting');                             00022000
15   Array1(1) = Array1(2) / Divisor;                           00023000
16   Put skip list('Sub1 Ending');                               00024000
17   End SUB1;                                                  00025000
18   Put skip list('Sample Ending');                             00026000
19 End;                                                         00027000

```

Figure 112. PL/I routine with a divide-by-zero error

Since variables are not normally displayed in a PLIDUMP dump, this routine included a PUT DATA statement, which generated a listing of arguments and variables used in the routine. Figure 113 shows this output.

```

1Sample Starting
  Sub1 Starting      A_NUMBER=          0 MY_NAME='Tery Gillaspay' AN_ARRAY(1)= 1
  AN_ARRAY(2)=       3              AN_ARRAY(3)=          5;

```

Figure 113. Variables from routine SAMPLE

The routine in Figure 112 was compiled with the LIST compiler option, which generated the object code listing shown in Figure 114 on page 259.

- OBJECT LISTING

```

* STATEMENT NUMBER 15
000372 58 B0 D 0C8      L      11,200(0,13)
000376 58 40 B 004      L      4,4(0,11)
00037A 58 90 3 0AC      L      9,172(0,3)
00037E 5C 80 4 004      M      8,4(0,4)
000382 58 70 3 0CC      L      7,204(0,3)
000386 5C 60 4 004      M      6,4(0,4)
00038A 58 80 D 0C0      L      8,192(0,13)
00038E 58 60 B 000      L      6,0(0,11)
000392 5F 60 4 000      SL     6,0(0,4)
000396 58 E7 6 000      L      14,VO..ARRAY1(7)
00039A 8E E0 0 020      SRDA   14,32
00039E 5D E0 8 000      D      14,DIVISOR
0003A2 50 F9 6 000      ST     15,VO..ARRAY1(9)

```

Figure 114. Object code listing from example PL/I routine

Figure 115 shows the Language Environment dump for routine SAMPLE.

```

CEE3DMP V1 R5.0: PLIDUMP called from error ON-unit.                                02/05/95 3:17:13 PM      Page:   1

PLIDUMP was called from statement number 4 at offset +000000BE from ERROR ON-unit with entry address 0002022C

Information for enclave SAMPLE

Information for thread 8000000000000000

Registers on Entry to CEE3DMP:
PM..... 0100
GPR0..... 00000000 GPR1..... 00077448 GPR2..... 053AD9AF GPR3..... 853AD514
GPR4..... 00000001 GPR5..... 053AD314 GPR6..... 80077454 GPR7..... 00000000
GPR8..... 00000001 GPR9..... 80000000 GPR10..... 00077470 GPR11..... 000F7490
GPR12..... 0006A520 GPR13..... 000773C8 GPR14..... 80060712 GPR15..... 853F7918
FPR0..... 4D000000 00043C31 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000

Traceback:
DSA Addr  Program Unit  PU Addr  PU Offset  Entry          E Addr  E Offset  Load  Service Statement  Status
004C58A0  CEEKMMRA          00572438 +00000748 CEEKMMRA      00572438 +00000748 CELE38      Call
004BD680  LIBRARY (PLI)      00558498 +000000B2 LIBRARY (PLI) 00558498 +000000B2 CEEPLPKA     Call
CELE38
004C5778  SAMPLE            00020080 +0000026A ERR_ON-UNIT   0002022C +000000BE          4 Call
004C5570  IBMRERPL          006D7470 +00000528 IBMRERPL      006D7470 +00000528          Call
004C5488  CEEEV010          00545000 +000000E8 CEEEV010      00545000 +000000E8          Call
00675018  CEEHDSP           0059AD58 +000009DC CEEHDSP       0059AD58 +000009DC          Call
004C5388  SAMPLE            00020080 +0000039E SUB1           00020348 +000000D6          15 Exception
004C5258  SAMPLE            00020080 +0000015C SAMPLE         00020088 +00000154          11 Call
004C51B0  IBMRPMIA          006D6BB8 +000003A2 IBMRPMIA      006D6BB8 +000003A2          Call
004C50C8  CEEEV010          00545000 +000001FE CEEEV010      00545000 +000001FE          Call
004C5018  CEEBBEXT          005895F0 +0000012E CEEBBEXT      005895F0 +0000012E          Call

Condition Information for Active Routines
Condition Information for SAMPLE (DSA address 004C5388)
CIB Address: 006753C8
Current Condition:
IBM0281S A prior condition was promoted to the ERROR condition.
Original Condition:
CEE3209S The system detected a Fixed Point divide exception.
Location:
Program Unit: SAMPLE Entry: SUB1 Statement: 15 Offset: +0000039E
Machine State:
ILC..... 0002      Interruption Code..... 0009
PSW..... FFE40009 AE020422
GPR0..... 004C5488 GPR1..... 004C5460 GPR2..... 4E0203B4 GPR3..... 00020478
GPR4..... 00020534 GPR5..... 004C5258 GPR6..... 004C532C GPR7..... 00000008
GPR8..... 004C5328 GPR9..... 00000004 GPR10..... 00000003 GPR11..... 004C5320
GPR12..... 00585848 GPR13..... 004C5388 GPR14..... 00000000 GPR15..... 00000003
DSA for SUB1: 004C5388
+00000000 FLAGS..... 8025      member... 0000      BKC..... 004C5258 FWC..... 00000000 R14..... 00000000
+0000010 R15..... 00000003 R0..... 004C5488 R1..... 004C5460 R2..... 4E0203B4 R3..... 00020478
+0000024 R4..... 00020534 R5..... 004C5258 R6..... 004C532C R7..... 00000008 R8..... 004C5328
+0000038 R9..... 00000004 R10..... 00000003 R11..... 004C5320 R12..... 00585848 reserved. 004BD280
+000004C NAB..... 004C5488 PNAB..... 004C5488 reserved. 91E091E0 004C5258 00000000 00000000
+0000064 reserved. 00000000 reserved. 00000000 MODE..... 00000000 reserved. 00000000 00000200
+0000078 reserved. 00000000 reserved. 00000000

```

Figure 115. Language Environment dump from example PL/I routine (Part 1 of 2)

```

CEE3DMP V1 R5.0: PLIDUMP called from error ON-unit.                                02/05/95 3:17:13 PM      Page: 5
CIB for SUB1: 006753C8
+000000 006753C8 C39C240 00000000 00000000 010C0002 00000000 00000000 00030119 59C9C2D4 |CIB .....IBM
+000020 006753E8 00000000 00675D08 00030C89 59C3C5C5 00000000 00000004 004C5258 00545000 |.....).....i.CEE.....<...&
+000040 00675408 00000000 004C5388 00020422 00586028 0000000A 00000000 00000000 00000000 |.....<.h.....-.....
+000060 00675428 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+000080 00675448 - +00009F 00675467 same as above
+0000A0 00675468 00000000 00000000 00000000 00000000 44230000 00000000 00000000 00000000 |.....
+0000C0 00675488 00000000 00000000 004C5388 004C5388 0002041E 00000000 00000000 00000001 |.....<.h.<.h.....
+0000E0 006754A8 004C5258 0000000A 00000064 00000000 FFFFFFFC 00000000 00000000 00000000 |.<.....
+000100 006754C8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
< DYNAMIC SAVE AREA (SUB1): 004C5388
+000000 004C5388 80250000 004C5258 00000000 00000000 00000003 004C5488 004C5460 4E0203B4 |.....<.....<.h.<.-+...
+000020 004C53A8 00020478 00020534 004C5258 004C532C 00000008 004C5328 00000004 00000003 |.....<.....<.....<.....
+000040 004C53C8 004C5320 00585848 004BD280 004C5488 004C5488 91E091E0 004C5258 00000000 |.<.....K.<.h.<.hj.j.<.....
+000060 004C53E8 00000000 00000000 00000000 00000000 00000000 00000200 00000000 00000000 |.....
+000080 004C5408 00000000 C9C2D4D9 D6D7C1C1 000205A4 000F0000 00675188 004C56A0 5E5C20CA |.....IBMROPAA...u.....eh.<.;*
+0000A0 004C5428 005B88C8 00000000 004C54FC 00578198 00675250 00675258 006DB968 006DAF44 |. $h.....aq...&;.....
+0000C0 004C5448 004C5328 004C5318 004C5320 00000001 004C5460 00000003 00020510 000204D8 |.<.....<.....<.....<.-.....Q
+0000E0 004C5468 00000000 80020524 00400000 007D1004 00000000 00000000 00010000 004BD460 |.....M-
DSA for SAMPLE: 004C5258
+000000 FLAGS.... C025 member... 0000 BKC..... 004C51B0 FWC..... 00000000 R14..... 5E0201DE
+000010 R15..... 00020348 R0..... 004C5388 R1..... 00020558 R2..... 4E020166 R3..... 8
+000020 R4..... 00000005 R5..... 004C5258 R6..... 004C5330 R7..... 004C5320 R8..... 00000001
+000030 R9..... 00020088 R10..... 00000003 R11..... 00578198 R12..... 00585848 reserved. 004BD280
+00004C NAB..... 004C5388 PNAB..... 004C5388 reserved. 91E091E0 00000000 00020610 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE.... 00000001 reserved. 004C5310 00000200
+000078 reserved. 00000001 reserved. 00000000
DYNAMIC SAVE AREA (SAMPLE): 004C5258
+000000 004C5258 C0250000 004C51B0 00000000 5E0201DE 00020348 004C5388 00020558 4E020166 |.....<e.....;.....<.h.....+...
+000020 004C5278 00020478 00000005 004C5258 004C5330 004C5320 00000001 00020088 00000003 |.....<.....<.....<.....h...
+000040 004C5298 00578198 00585848 004BD280 004C5388 004C5388 91E091E0 00000000 00020610 |.aq.....K.<.h.<.hj.j.....
+000060 004C52B8 00000000 00000000 00000000 00000001 004C5310 00000200 00000001 00000000 |.....
+000080 004C52D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0000A0 004C52F8 00000000 00000000 00000000 00000000 00000000 00000000 0C010000 00000000 |.....
+0000C0 004C5318 004C533C 000D0000 004C5330 00020534 00000000 00000000 00000001 00000003 |.<.....
+0000E0 004C5338 00000005 E38599A8 40C78993 9381A297 A8000000 00000000 00000000 00000000 |.....Tery GillaspY.....
+000100 004C5358 004C5360 00000000 000204E0 000204D8 00000000 80020524 00400000 007D1004 |.<.-.....Q.....'
+000120 004C5378 00000000 00000000 00010000 004BD460 80250000 004C5258 00000000 00000000 |.....M-.....<.....
STATIC FOR PROCEDURE SAMPLE      TIMESTAMP: 15 JAN 93 15:10:05
STARTING FROM: 00020478
+000000 00020478 E00002A4 00020088 00020130 00020166 0002022C 000202A0 00020348 000203B4 |...u...h.....
+000020 00020498 000203B4 000203B4 000203B4 80020AA0 80020AB8 80020AD0 80020AE8 80020B00 |.....Y.....
+000040 000204B8 80020B18 800213A8 800211B0 80020B30 80021420 80021438 80020B48 80020B60 |.....y.....-
+000060 000204D8 20000002 1F800000 000205A4 000F0000 00000000 00000000 00000000 00020534 |.....u.....
+000080 000204F8 000205C0 000D0000 00000000 00030000 00000000 00210000 000205F1 000D0000 |.....l.....
+0000A0 00020518 000205FE 000B0000 91E091E0 00000001 00000003 00000005 00000000 00000004 |.....j.j.....
+0000C0 00020538 00000004 00000003 00000001 00000002 00020738 00020738 004C5360 80020524 |.....-.....
+0000E0 00020558 004C5388 004C5318 804C5320 00020738 00205240 004C586C 804C5898 00000000 |.<.....<.....<.....<.%.<q
+000100 00020578 80020A70 00020738 80000000 00000000 80020620 00020738 004C5460 80020524 |.....<.-.....
+000120 00020598 00020738 00000000 80020524 E2819497 938540E2 A38199A3 899587E3 8599A8A0 |.....Sample StartingTery
+000140 000205B8 C7899393 81A297A8 E2819497 938540C5 95848995 87E3C2C3 D7D3C9C4 E4D4D740 |GillaspYSample EndingTBCPLIDUMP
+000160 000205D8 83819393 85844086 99969440 85999996 9940D6D5 60A94589 A3E2A482 F140E2A3 |called from error ON-unitSubl St
+000180 000205F8 8199A389 9587E2A4 82F140C5 95848995 87000000 00000000 0C160000 0002022C |artingSubl Ending.....
+0001A0 00020618 0C960000 00000000 00020634 00020650 0002066C 00000000 00000000 85000001 |.o.....&...%.e.....
+0001C0 00020638 000204DA 000000D0 00000000 0008C16D D5E4D4C2 C5D90000 81000001 000204D8 |.....A NUMBER..a.....Q
+0001E0 00020658 000000C0 00000000 0007D4E8 6DD5C1D4 C5000000 81000101 000204DA 000000C8 |.....MY_NAME...a.....H
+000200 00020678 00000000 0008C1D5 6DC1D9D9 C1E80000 00000001 00020088 000001A4 000206BC |.....AN ARRAY.....h...u.....
+000220 00020698 00000001 00DE0002 00E20008 01200009 0128000A 012E000B 01560012 01940013 |.....S.....m.....
CEE3DMP V1 R5.0: PLIDUMP called from error ON-unit.                                02/05/95 3:17:13 PM      Page: 4
+000240 000206B8 01A40002 0002022C 00000112 000206E0 00000002 00740003 00780004 00C00005 |.u.....
+000260 000206D8 01020006 0114000C 00020348 0000012C 00020704 0000000C 006C000E 00AA000F |.....%.
+000280 000206F8 00DE0010 011C0011 011C0011 0E0E0E0E F1F540D1 C1D540F9 F34040F1 F57AF1F0 |.....15 JAN 93 15:10
+0002A0 00020718 7AF0F540 80000010 00020000 00000000 01000001 00020088 00020758 00000000 |:05 .....h.....

```

Figure 115. Language Environment dump from example PL/I routine (Part 2 of 2)

To understand the dump information and debug this routine, use the following steps:

1. Notice the title of the dump: PLIDUMP called from error ON-unit. This was the title specified when PLIDUMP was invoked, and it indicates that the ERROR condition was raised and PLIDUMP was called from within the ERROR ON-unit.
2. Locate the messages in the Condition Information section of the dump.

There are two messages. The current condition message indicates that a prior condition was promoted to the ERROR condition. The promotion of a condition occurs when the original condition is left unhandled (no PL/I ON-units are assigned to gain control). The original condition message is CEE3209S. The system detected a Fixed Point divide exception. The original condition usually indicates the actual problem. For more information about this message, see *z/OS Language Environment Run-Time Messages*.

3. In the traceback section, note the sequence of calls in the call chain. SAMPLE called SUB1 at statement 11, and SUB1 raised an exception at statement 15, PU offset X'39E'.
4. Find the statement in the listing for SUB1 that raised the ZERODIVIDE condition. If SUB1 was compiled with GOSTMT and SOURCE, find statement 15 in the source listing.

Since the object listing was generated in this example, you can also locate the actual assembler instruction causing the exception at offset X'39E' in the object listing for this routine, shown in Figure 114 on page 259. Either method shows that *divisor* was used as the divisor in a divide operation.

5. You can see from the declaration of SUB1 that *divisor* is a parameter passed from SAMPLE. Because of linkage conventions, you can infer that register 1 in the SAMPLE save area points to a parameter list that was passed to SUB1. *divisor* is the first parameter in the list.
6. In the SAMPLE DSA, the R1 value is X'20558'. This is the address of the parameter list, which is located in static storage.
7. Find the parameter list in the stack frame; the value of the first parameter is X'00000000'. Thus, the exception occurred when SAMPLE passed a 0 value used as a divisor in subroutine SUB1.

Chapter 8. Debugging under CICS

This chapter provides information for debugging under the Customer Information Control System (CICS). The following sections explain how to access debugging information under CICS, and describe features unique to debugging under CICS.

Use the following list as a quick reference for debugging information:

- Language Environment run-time messages (CESE transient data queue)
- Language Environment traceback (CESE transient data queue)
- Language Environment dump output (CESE transient data queue)
- CICS Transaction Dump (CICS DFHDMPA or DFHDMPB data set)
- Language Environment abend and reason codes (system console)
- Language Environment return codes to CICS (system console)

If the EXEC CICS HANDLE ABEND command is active and the application, or CICS, initiates an abend or application interrupt, then Language Environment does not produce any run-time messages, tracebacks, or dumps.

If EXEC CICS ABEND NODUMP is issued, then no Language Environment dumps or CICS transaction dumps are produced.

Accessing debugging information

The following sections list the debugging information available to CICS users, and describe where you can find this information.

Under CICS, the Language Environment run-time messages, Language Environment traceback, and Language Environment dump output are written to the CESE transient data queue. The transaction identifier, terminal identifier, date, and time precede the data in the queue. For detailed information about the format of records written to the transient data queue, see *z/OS Language Environment Programming Guide*.

The CESE transient data queue is defined in the CICS destination control table (DCT). The CICS macro DFHDCT is used to define entries in the DCT. See *CICS Resource Definition Guide* for a detailed explanation of how to define a transient data queue in the DCT. If you are not sure how to define the CESE transient data queue, see your system programmer.

Locating Language Environment run-time messages

Under CICS, Language Environment run-time messages are written to the CESE transient data queue. A sample Language Environment message that appears when an application abends due to an unhandled condition from an EXEC CICS command is:

```
P039UTV9 19910916145313 CEE3250C The System or User ABEND AEI0 was issued.  
P039UTV9 19910916145313          From program unit UT9CVERI at entry point UT9CVERIT  
                                +0000011E at P039UTV9 19910916145313  
                                at offset address 0006051E.
```

Locating the Language Environment traceback

Under CICS, the Language Environment traceback is written to the CESE transient data queue. Because Language Environment invokes your application routine, the Language Environment routines that invoked your routine appear in the traceback. Figure 116 on page 264 shows an example Language Environment traceback

written to the CESE transient data queue. Data unnecessary for this example has been replaced by ellipses.

```

P023T2AB 19911011084413 CEE3211S The system detected a Decimal-divide exception.
P023T2AB 19911011084413      From program unit T2AB at entry point T2AB at offset +000014A8 at address 0A9444E0.
1P023T2AB 19911011084413 CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.      08/30/01 09:23:10
P023T2AB 19911011084413
P023T2AB 19911011084413 Information for enclave T2AB
P023T2AB 19911011084413
P023T2AB 19911011084413 Information for thread 8000000000000000
P023T2AB 19911011084413
P023T2AB 19911011084413 Registers on Entry to CEE3DMP:
P023T2AB 19911011084413
P023T2AB 19911011084413 PM..... 0100
P023T2AB 19911011084413 GPR0..... 00000000 GPR1..... 00077448 GPR2..... 053AD9AF GPR3..... 853AD514
P023T2AB 19911011084413 GPR4..... 00000001 GPR5..... 053AD314 GPR6..... 80077454 GPR7..... 00000000
P023T2AB 19911011084413 GPR8..... 00000001 GPR9..... 80000000 GPR10.... 00077470 GPR11.... 000F7490
P023T2AB 19911011084413 GPR12.... 0006A520 GPR13.... 000773C8 GPR14.... 80060712 GPR15.... 853F7918
P023T2AB 19911011084413 FPR0..... 4D000000 00043C31 FPR2..... 00000000 00000000
P023T2AB 19911011084413 FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
P023T2AB 19911011084413
P023T2AB 19911011084413 Traceback:
P023T2AB 19911011084413 DSA Addr Program Unit PU Addr Entry E Addr E Offset Statement Load Mod Service Status
P023T2AB 19911011084413 0A9A5630 CEEHDSP 001DB878 c9101 05500028 +00000152
P023T2AB 19911011084413 0A9AD1D0 CEECGEX 001D6BF0 CB2C9101 055001E0 +000003D0
P023T2AB 19911011084413 0A9ABD88 T2AB 0A943038 IGZCEV5 04CF9000 +00000836
P023T2AB 19911011084413 0A9AD098 CEECRINV 001D9F90 CEECRINV 0522A708 +0000036E
P023T2AB 19911011084413 0A9AD010 CEECCICS 001C6370 CEECCICS 0001FF28 +00000456 CEECCICS UQ00568 Call
:

```

Figure 116. Language Environment traceback written to the transient data queue

Locating the Language Environment dump

Under CICS, the Language Environment dump output is written to the CESE transient data queue. For active routines, the Language Environment dump contains the traceback, condition information, variables, storage, and control block information for the thread, enclave, and process levels. Use the Language Environment dump with the CICS transaction dump to locate problems when operating under CICS.

For a sample Language Environment dump, see “Understanding the Language Environment dump” on page 43.

Using CICS transaction dump

The CICS transaction dump is generated to the DFHDMPIA or DFHDMPB data set. The offline CICS dump utility routine converts the transaction dump into formatted, understandable output.

The CICS transaction dump contains information for the storage areas and resources associated with the current transaction. This information includes the Communication Area (COMMAREA), Transaction Work Area (TWA), Exec Interface Block (EIB), and any storage obtained by the CICS EXEC commands. This information does not appear in the Language Environment dump. It can be helpful to use the CICS transaction dump with the Language Environment dump to locate problems when operating under CICS.

When the location of an error is uncertain, it can be helpful to insert EXEC CICS DUMP statements in and around the code suspected of causing the problem. This generates CICS transaction dumps close to the error for debugging reference.

For information about interpreting CICS dumps, see *CICS Problem Determination Guide*

Using CICS register and program status word contents

When a routine interrupt occurs (code = ASRA) and a CICS dump is generated, CICS formats the contents of the program status word (PSW) and the registers at the time of the interrupt. This information is also contained in the CICS trace table entry marked SSRP * EXEC* – ABEND DETECTED. For the format of the information contained in this trace entry, see *CICS Data Areas*, KERRD - KERNEL ERROR DATA.

The address of the interrupt can be found from the second word of the PSW, giving the address of the instruction following the point of interrupt. The address of the entry point of the function can be subtracted from this address. The offset compared to this listing gives the statement that causes the interrupt.

For C routines, you can find the address of the entry point in register 3.

If register 15 is corrupted, the contents of the first load module of the active enclave appear in the program storage section of the CICS transaction dump.

Using Language Environment abend and reason codes

An application can end with an abend in two ways:

- User-specified abend (that is, an abend requested by the assembler user exit or the ABTERMENC run-time option).
- Language Environment-detected unrecoverable error (in which case there is no Language Environment condition handling).

When Language Environment detects an unrecoverable error under CICS, Language Environment terminates the transaction with an EXEC CICS abend. The abend code has a number between 4000 and 4095. A write-to-operator (WTO) is performed to write a CEE1000S message to the system console. This message contains the abend code and its associated reason code. The WTO is performed only for unrecoverable errors detected by Language Environment. No WTO occurs for user-requested abends.

Although this type of abend is performed only for unrecoverable error conditions, an abend code of 4000–4095 does not necessarily indicate an internal error within Language Environment. For example, an application routine can write a variable outside its storage and corrupt the Language Environment control blocks.

Possible causes of a 4000–4095 abend are corrupted Language Environment control blocks and internal Language Environment errors. For more information about abend codes 4000–4095, see *z/OS Language Environment Run-Time Messages*. Following is a sample Language Environment abend and reason code. Abend codes appear in decimal, and reason codes appear in hexadecimal.

```
12.34.27 JOB05585 IEF450I XCEPII03 GO CEPII03 - ABEND=S000 U4094 REASON=0000002C
```

Using Language Environment return codes to CICS

When the Language Environment condition handler encounters a severe condition that is specific to CICS, the condition handler generates a CICS-specific return code. This return code is written to the system console.

Possible causes of a Language Environment return code to CICS are:

- Incorrect region size
- Incorrect DCT

- Incorrect CSD definitions

For a list of the reason codes written only to CICS, see *z/OS Language Environment Run-Time Messages*. Following is a sample of a return code that was returned to CICS

```
+DFHAP1200I
LE03CC01 A CICS request to Language Environment has failed. Reason
code '0012030'.
```

Activating Language Environment feature trace records under CICS

Activating Language Environment feature trace records under CICS will allow users to monitor and determine the activity of a transaction. By activating the feature trace records, Level 2 trace points are added inside Language Environment at these significant points:

- Event Handle
- Set anchor
- Gives R13 and parameters before call

These trace points are useful for any support personnel that needs to know what happened inside Language Environment from a CICS call.

The function will be enabled by the existing CICS transactions. A user must enable the AP domain level 2 in order to include the Language Environment trace points. For more information on activating the CICS trace, see *CICS Diagnosis Reference*.

Every time CICS calls Language Environment, the feature trace is activated under the Extended Run-Time Library Interface (ERTLI). The trace can be seen in CICS transaction dumps. Feature trace entries are formatted in a similar way to CICS trace items. There are three formats: ABBREV, SHORT & FULL. The ABBREV version just formats the heading line for each trace point and is laid out in a similar way to CICS trace entries. For example,

```
00036 1 AP 1940 APLI ENTRY START_PROGRAM NAMETEST,CEDF,FULLAPI,EXEC,NO,0678FABC,00000000 , 00000000,1,NO =000334=
00036 1 AP 1948 APLI EVENT CALL-TO-LE/370 Thread_Initialization NAMETEST =000339=
00036 1 AP 1949 APLI EVENT RETURN-FROM-LE/370 Thread_Initialization OK NAMETEST =000340=
00036 1 AP 1948 APLI EVENT CALL-TO-LE/370 Rununit_Initialization NAMETEST =000343=
00036 1 FT 1014 Lang.Env. CEEZCREN EVENT CEEEVNT-ID(PRCINIT) R13(06C00E10), 00000000 =000344=
00036 1 FT 1013 Lang.Env. CEEZCREN EVENT CEEEVNT-ID(OTPT) R13(06C00E10), 06C049B0, 07500F28, 06C0403C, 06C010B4 =000345=
00036 1 FT 1101 Lang.Env. CEECRINI EVENT SET ANCHOR R13(06C009B8), 06C06180, 00000002 =000346=
00036 1 FT 1018 Lang.Env. CEEZINV EVENT CEEEVNT-ID(ENCINIT) R13(06C06D80), 00000000, 06C0403C, 00000000, 06C041B4, 00000 =000347=
00036 1 FT 1008 Lang.Env. CEECRINV EVENT CEEEVNT-ID(MAININV) R13(06C06D80), 87500020, 00000001, 00000000, 00140050, 87500 =000348=
00036 1 AP 1948 APLI EVENT CALL-TO-LE/370 Rununit_End_Invocation NAMETEST =000386=
00036 1 AP 1949 APLI EVENT RETURN-FROM-LE/370 Rununit_End_Invocation OK NAMETEST =000387=
00036 1 AP 1948 APLI EVENT CALL-TO-LE/370 Rununit_Termination NAMETEST =000388=
00036 1 FT 1012 Lang.Env. CEEZDSEX EVENT CEEEVNT-ID(ENCTERM) R13(06C06D80), 06C0403C, 00000000 =000389=
00036 1 FT 1102 Lang.Env. CEECRTRM EVENT SET ANCHOR R13(06C009B8), 00000000 =000390=
00036 1 AP 1949 APLI EVENT RETURN-FROM-LE/370 Rununit_Termination OK NAMETEST =000391=
00036 1 AP 1948 APLI EVENT CALL-TO-LE/370 Thread_Termination =000392=
00036 1 FT 1009 Lang.Env. CEEZDSPR EVENT CEEEVNT-ID(PRCTERM) R13(06C00A80), 00000000 =000393=
00036 1 AP 1949 APLI EVENT RETURN-FROM-LE/370 Thread_Termination OK =000394=
00036 1 AP 1941 APLI EXIT START_PROGRAM/OK ....,NO,NAMETEST =000395=
```

Figure 117. CICS trace output in the ABBREV format.

The Domain Name field is replaced with a "Feature" short name (for example, Lang.Env.) and module name (for example, CEE.....) which are coded into the "Feature Trace" initialization (short name) and header formatting call (module name). See the following macro example.

The FULL version includes the heading from the ABBREV version and then dumps each captured block in Hex and Character formats. For example:

```

AP 1948 APLI EVENT CALL-TO-LE/370 - Rununit_Initialization      Program_name(NAMETEST)

TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-868218FE TIME-05:58:55.2643333923 INTERVAL-00.0000020781 =000343=
1-0000 0000001E                                     *....*
2-0000 06878DE0 00140148 0005848C 0014014C 00045A4C 00140130 0014001C 067F3CE8 *g.\.....d.....<.!<.....".Y*
0020 0678FAD8 06878F37 867F3DD0                                     *...Q.g.f".}*
3-0000 D5C1D4C5 E3C5E2E3                                     *NAMETEST*
4-0000 00000030 20000000 07500000 00001800 87500020 00000000 06C03800 00000000 *.....&.....g&.....{.....*
0020 00140050 00000000 00000000 0678FABC                                     *...&.....*

FT Lang.Env. 1014 CEEZCREN EVENT - CEEEVENT-ID(PRCINIT)      R13(06C00E10), PARS(00000000)

TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F092A0 TIME-05:58:55.2643970329 INTERVAL-00.0000636406 =000344=
0000 0000C9C2 D4404040 40404040 40404040 40404040 40404040 40404040 40404040 *..IBM*
0020 D3819587 A4818785 40C595A5 89999695 948595A3 40404040 40404040 4040F0F0 *Language Environment 00*
0040 F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3 D3819587 4BC595A5 4B000000 *00000001CEECTFMTLang.Env....*
1-0000 06C00E10 00000011 00000000 *{.....*

FT Lang.Env. 1013 CEEZCREN EVENT - CEEEVENT-ID(PTP)          R13(06C00E10), PARS(06C049B0, 07500F28, 06C0403C, 06C010B4)

TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F0A23A TIME-05:58:55.2644148454 INTERVAL-00.0000178125 =000345=
0000 0000C9C2 D4404040 40404040 40404040 40404040 40404040 40404040 40404040 *..IBM*
0020 D3819587 A4818785 40C595A5 89999695 948595A3 40404040 40404040 4040F0F0 *Language Environment 00*
0040 F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3 D3819587 4BC595A5 4B000000 *00000001CEECTFMTLang.Env....*
1-0000 06C00E10 00000004 06C049B0 07500F28 06C0403C 06C010B4 *{.....{...&...{ ..{..*

FT Lang.Env. 1101 CEECRINI EVENT - SET_ANCHOR R13(06C009B8), PARS(06C06180, 00000002)

TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F02F90 TIME-05:58:55.2644493767 INTERVAL-00.0000345312 =000346=
0000 0000C9C2 D4404040 40404040 40404040 40404040 40404040 40404040 40404040 *..IBM*
0020 D3819587 A4818785 40C595A5 89999695 948595A3 40404040 40404040 4040F0F0 *Language Environment 00*
0040 F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3 D3819587 4BC595A5 4B0092B4 *00000001CEECTFMTLang.Env..k.*
1-0000 06C009B8 06C06180 00000002 *{...{/.....*

FT Lang.Env. 1018 CEEZINV EVENT - CEEEVENT-ID(ENCINIT)      R13(06C06D80), PARS(00000000, 06C0403C, 00000000, 06C041B4, 00000000,
01000000, 00000000, 00000000)

TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F0C9B4 TIME-05:58:55.2644710798 INTERVAL-00.0000217031 =000347=
0000 000AC9C2 D4404040 40404040 40404040 40404040 40404040 40404040 40404040 *..IBM*
0020 D3819587 A4818785 40C595A5 89999695 948595A3 40404040 40404040 4040F0F0 *Language Environment 00*
0040 F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3 D3819587 4BC595A5 4B0072B4 *00000001CEECTFMTLang.Env....*
1-0000 06C06D80 00000012 00000000 06C0403C 00000000 06C041B4 00000000 01000000 *{.....{ .....{.....*
0020 00000000 00000000 *.....*
2-0000 D8C3C5E2 C9000000 00000000 00000000 D8C3C5E2 D6000000 00000000 00000000 *QCESI.....QCES0.....*
0020 D8C3C5E2 C5000000 00000000 00000000 *QCESE.....*

FT Lang.Env. 1008 CEECRINV EVENT - CEEEVENT-ID(MAININV)      R13(06C06D80), PARS(87500020, 00000001, 00000000, 00140050, 87500020)

TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F038D2 TIME-05:58:55.2645123298 INTERVAL-00.0000412500 =000348=
0000 0000C9C2 D4404040 40404040 40404040 40404040 40404040 40404040 40404040 *..IBM*
0020 D3819587 A4818785 40C595A5 89999695 948595A3 40404040 40404040 4040F0F0 *Language Environment 00*
0040 F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3 D3819587 4BC595A5 4B000000 *00000001CEECTFMTLang.Env....*
1-0000 06C06D80 0000000E 87500020 00000001 00000000 00140050 87500020 *{.....g&.....&g&..*

AP 1948 APLI EVENT CALL-TO-LE/370 - Rununit_End_Invocation    Program_name(NAMETEST)

TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-868218FE TIME-05:58:55.2670554079 INTERVAL-00.0000107187 =000386=
1-0000 00000021                                     *....*
2-0000 06878DE0 00140148 0005848C 0014014C 00045A4C 00140130 0014001C 067F3CE8 *g.....d.....<.!<.....".Y*
0020 0678FAD8 80140390                                     *...Q....*
3-0000 D5C1D4C5 E3C5E2E3                                     *NAMETEST*
4-0000 40000000 00000000 D5C1D4C5 0000036C D3F3F2F1 00000005 00000000 00000000 *.....NAME...%L321.....*
0020 00000000 001402FC 00000000 00000000 00000000 00000000 00000000 00000000 *.....*

```

Figure 118. CICS trace output in the FULL format.

The first block is used for the feature trace information. It contains the name of the off-line formatting module and the short name used in the formatted heading line. The other 6 blocks are available for user data.

The SHORT version is a cross between the ABBREV and FULL versions.

For more information about the CICS trace, see *CICS Diagnosis Reference*.

Ensuring transaction rollback

If your application does not run to normal completion and there is no CICS transaction abend, take steps to ensure that transaction rollback (the backing out of any updates made by the malfunctioning application) takes place.

There are two ways to ensure that a transaction rollback occurs when an unhandled condition of severity 2 or greater is detected:

- Use the ABTERMENC run-time option with the ABEND suboption (ABTERMENC(ABEND))
- Use an assembler user exit that requests an abend for unhandled conditions of severity 2 or greater

The IBM-supplied assembler user exit for CICS (CEEEXITA), available in the Language Environment SCEESAMP sample library, ensures that a transaction abend and rollback occur for all unhandled conditions of severity 2 or greater. For more information about the assembler user exit, see “Invoking the assembler user exit” on page 25 and *z/OS Language Environment Programming Guide*.

Finding data when Language Environment returns a nonzero return code

Language Environment does not write any messages to the CESE transient data queue. Following is the output generated when Language Environment returns a nonzero reason code to CICS and the location where the output appears:

Table 25. Finding data when Language Environment returns a nonzero return code

Output Message	Location	Issued By
DFHAC2206 14:43:54 LE03CC01 Transaction UTV2 has failed with abend AEC7. Resource backout was successful.	User's terminal	CICS
DFHAP1200I LE03CC01 A CICS request to the Language Environment has failed. Reason code '0012030'.	System console	CICS
DFHAC2236 06/05/91 14:43:48 LE03CC01 Transaction UTV2 abend AEC7 in routine UT2CVERI term P021 backout successful.	Transient data queue CSMT	CICS

Finding data when Language Environment abends internally

Language Environment does not write any messages to the CESE transient data queue. Following is the output generated when Language Environment abends internally and the location where the output appears:

Table 26. Finding data when Language Environment abends internally

Output Message	Location	Issued By
DFHAC2206 14:35:24 LE03CC01 Transaction UTV8 has failed with abend 4095. Resource backout was successful.	User's terminal	CICS
CEE1000S LE INTERNAL abend. ABCODE = 00000FFF REASON = 00001234	System console	Language Environment

Table 26. Finding data when Language Environment abends internally (continued)

Output Message	Location	Issued By
DFHAC2236 06/05/91 14:35:24 LE03CC01 Transaction UTV8 abend 4095 in routine UT8CVERI term P021 backout successful.	Transient data queue CSMT	CICS

Finding data when Language Environment abends from an EXEC CICS command

This section shows the output generated when an application abends from an EXEC CICS command and the location where the output appears.

This error assumes the use of Language Environment run-time option TERMTHDACT(MSG).

Table 27. Finding data when Language Environment abends from an EXEC CICS command

Output Message	Location	Issued By
DFHAC2206 14:35:34 LE03CC01 Transaction UTV8 has failed with abend AEI. Resource backout was successful.	User's terminal	CICS
No message.	System console	CICS
DFHAC2236 06/05/91 14:35:17 LE03CC01 Transaction UTV9 abend AEI0 in routine UT9CVERI term P021 backout successful.	Transient data queue CSMT	CICS
P021UTV9 091156 143516 CEE3250C The System or User Abend AEI0 was issued.	Transient data queue CESE	Language Environment

Displaying and modifying run-time options with the CLER transaction

The CLER transaction can be used to:

- Display the current run-time options in effect for the region.
- Modify the following subset of the region run-time options:
 - TRAP(ONIOFF)
 - TERMTHDACT(QUIETIMSGITRACEIDUMPIUAONLYIUATRACEIUADUMPIUAIMM)
 - RPTOPTS(ONIOFF)
 - RPTSTG(ONIOFF)
 - ALL31(ONIOFF)
 - CBLPSHPOP(ONIOFF)
- Write the current region run-time options to the CESE queue for printing.

The CLER transaction is conversational; it presents the user with commands for the terminal display. The run-time options that can be modified with this transaction are only in effect for the duration of the running region.

The CLER transaction must be defined in the CICS CSD (CICS System Definition file). The following definitions are required, and are in the Language Environment CEECCSD job in the SCEESAMP data set:


```
DEFINE PROGRAM(CEL4RT0) GROUP(CEE) LANGUAGE(ASSEMBLER) EXECKEY(CICS)
DEFINE MAPSET(CELCLEM) GROUP(CEE)
DEFINE MAPSET(CELCLRH) GROUP(CEE)
DEFINE TRANS(CLER) PROG(CEL4RT0) GROUP(CEE)
```

Use the CEECCSD job to activate these definitions, or you must define them dynamically with the CICS CEDA transaction.

Note: If the run-time option ALL31 is modified to OFF, the stack is forced to BELOW. A warning message, asking if want to continue, is presented on the panel. Once the stack is modified to BELOW, it will remain below for the duration of the region, even if you set ALL31 back to ON.

To send the run-time option report to the CESE queue for output display or printing, press PF10 on the panel which displays the run-time option report.

For detailed information on the use of CLER, select PF1 from the main menu that is displayed when the CLER transaction is invoked.

Part 3. Debugging Language Environment AMODE 64 applications

This part provides specific information for debugging applications written to make use of the memory address space above the 2 GB bar.

Chapter 9. Preparing your AMODE 64 application for debugging

This chapter describes options and features that you can use to prepare your AMODE 64 application for debugging. The following topics are covered:

- Compiler options for C, C++
- Language Environment run-time options
- Use of storage in routines
- Options for modifying exception handling
- Assembler user exits
- Enclave termination behavior
- Language Environment feedback codes and condition tokens

Setting compiler options

The following sections discuss language-specific compiler options important to debugging routines in Language Environment. These sections cover only the compiler options that are important to debugging. For a complete list of compiler options, refer to the appropriate HLL publications.

The use of some compiler options (such as `DEBUG`) can affect the performance of your routine. You must set these options before you compile. In some cases, you might need to remove the option and recompile your routine before delivering your application.

XL C and XL C++ compiler options for AMODE 64 applications

When compiling an application using the `LP64` compiler option, you cannot use the `TEST` compiler option. You must instead use the `DEBUG(FORMAT(DWARF))` compiler option.

For a detailed explanation of the debugging options for XL C/C++ and Inter-procedural Analysis (IPA), see *z/OS XL C/C++ User's Guide* and *z/OS XL C/C++ Programming Guide*.

Using Language Environment run-time options

There are several run-time options that affect debugging in Language Environment. The `TEST` run-time option, for example, can be used with a debugging tool to specify the level of control in effect for the debugging tool when the routine being initialized is started. The `HEAPCHK`, `TERMTHDACT`, `TRACE`, and `TRAP` options affect exception handling.

The following Language Environment run-time options affect debugging:

HEAPCHK	Determines whether additional heap check tests are performed.
INFOMSGFILTER	Filters user specified informational messages from stderr. Note: Affects only those messages generated by Language Environment and any routine that calls <code>__le_msg_get_and_write()</code> . Other routines that write to stderr, such as <code>__le_msg_write()</code> , do not have a filtering option.
PROFILE	Controls the use of an optional profiler tool, which collects performance data for the running application. When this option is in effect, the profiler is loaded and the debugger cannot be loaded. If the <code>TEST</code> option is in effect when <code>PROFILE</code> is specified, the profiler tool will not be loaded.

RPTOPTS	Causes a report to be produced which contains the run-time options in effect. See “Determining run-time options in effect” below.
RPTSTG	Generates a report of the storage used by an enclave. See “Controlling storage allocation” on page 275.
STORAGE	Specifies that Language Environment initializes all heap and stack storage to a user-specified value.
TERMTHDACT	Controls response when an enclave terminates due to an unhandled condition of severity 2 or greater.
TEST	Specifies the conditions under which a debugging tool assumes control.
TRACE	Activates Language Environment run-time library tracing and controls the size of the trace table, the type of trace, and whether the trace table should be dumped unconditionally upon termination of the application.
TRAP	When TRAP is set to ON, Language Environment traps routine interrupts and abends, and optionally prints trace information or invokes a user-written exception handling routine. With TRAP set to OFF, the operating system handles all interrupts and abends. You should generally set TRAP to ON, or your run-time results can be unpredictable.

For a more detailed discussion of these run-time options, see *z/OS Language Environment Programming Reference*.

Determining run-time options in effect

The run-time options in effect at the time the routine is run can affect routine behavior. Use RPTOPTS(ON) to generate an options report in the Language Environment message file when your routine terminates. The options report lists run-time options, and indicates where they were set.

Figure 119 shows a sample options report.

Options Report for Enclave main Wed Oct 13 19:21:04 2004
Language Environment V01 R07.00

LAST WHERE SET	OPTION

Installation default	ENVAR("")
PARMLIB(CEEPRMPV)	ENVAR("AA1="Report 1")
Installation default	FILETAG(NOAUTOCVT,NOAUTOTAG)
PARMLIB(CEEPRMPV)	HEAPCHK(ON,1,0,0,0)
Installation default	HEAPPOOLS64(OFF,8,4000,32,2000,128,700,256,350,1024,100,2048,50,3072,50,4096,50,8192,25,16384,10,32768,5,65536,5)
PARMLIB(CEEPRMPV)	HEAP64(6M,1M,KEEP,32768,32768,KEEP,4096,4096,FREE)
Installation default	INFOMSGFILTER(OFF,,,))
Installation default	IOHEAP64(1M,1M,FREE,12288,8192,FREE,4096,4096,FREE)
Invocation command	LIBHEAP64(1M,1M,FREE,32763,8192,FREE,8192,4096,FREE)
Installation default	NATLANG(ENU)
Invocation command	POSIX(ON)
Installation default	PROFILE(OFF,"")
DD:CEEOPTS	RPTOPTS(ON)
SETCEE command	RPTSTG(ON)
Installation default	STACK64(1M,1M,128M)
Installation default	STORAGE(NONE,NONE,NONE,)
Installation default	TERMTHDACT(TRACE,,96)
Installation default	NOTEST(ALL,"*","PROMPT","INSPREF")
Installation default	THREADSTACK64(OFF,1M,1M,128M)
Installation default	TRACE(OFF,4096,DUMP,LE=0)
Installation default	TRAP(ON,SPIE)

Figure 119. 64-bit options report

Controlling storage allocation

The following run-time options control storage allocation:

- HEAP64
- HEAPPOOLS64
- IOHEAP64
- LIBHEAP64
- STACK64
- THREADSTACK64

z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode provides useful tips to assist with the tuning process. Appropriate tuning is necessary to avoid performance problems.

To generate a report of the storage a routine (or more specifically, an enclave) used during its run, specify the RPTSTG(ON) run-time option. The storage report, generated during enclave termination provides statistics that can help you understand how space is being consumed as the enclave runs. If storage management tuning is desired, the statistics can help you set the corresponding storage-related run-time options for future runs.

Figure 120 on page 276 shows a sample storage report.

Storage Report for Enclave main Fri Apr 23 18:58:21 2004

Language Environment V01 R06.00

```
STACK64 statistics:
  Initial size: 1M
  Increment size: 1M
  Maximum used by all concurrent threads: 1M
  Largest used by any thread: 1M
  Number of increments allocated: 0
THREADSTACK64 statistics:
  Initial size: 1M
  Increment size: 1M
  Maximum used by all concurrent threads: 0M
  Number of increments allocated: 0
64bit User HEAP statistics:
  Initial size: 1M
  Increment size: 1M
  Total heap storage used: 34528
  Suggested initial size: 1M
  Successful Get Heap requests: 252
  Successful Free Heap requests: 218
  Number of segments allocated: 0
  Number of segments freed: 0
31bit User HEAP statistics:
  Initial size: 32768
  Increment size: 32768
  Total heap storage used (sugg. initial size): 34096
  Successful Get Heap requests: 2
  Successful Free Heap requests: 1
  Number of segments allocated: 2
  Number of segments freed: 0
24bit User HEAP statistics:
  Initial size: 4096
  Increment size: 4096
  Total heap storage used (sugg. initial size): 4656
  Successful Get Heap requests: 2
  Successful Free Heap requests: 1
  Number of segments allocated: 2
  Number of segments freed: 1
64bit Library HEAP statistics:
  Initial size: 1M
  Increment size: 1M
  Total heap storage used: 90144
  Suggested initial size: 1M
  Successful Get Heap requests: 25
  Successful Free Heap requests: 8
  Number of segments allocated: 0
  Number of segments freed: 0
31bit Library HEAP statistics:
  Initial size: 16384
  Increment size: 8192
  Total heap storage used (sugg. initial size): 0
  Successful Get Heap requests: 0
  Successful Free Heap requests: 0
  Number of segments allocated: 0
  Number of segments freed: 0
```

Figure 120. 64-bit storage report (Part 1 of 2)

```

24bit Library HEAP statistics:
  Initial size:                        8192
  Increment size:                     4096
  Total heap storage used (sugg. initial size): 0
  Successful Get Heap requests:       0
  Successful Free Heap requests:      0
  Number of segments allocated:       0
  Number of segments freed:           0
64bit I/O HEAP statistics:
  Initial size:                       1M
  Increment size:                     1M
  Total heap storage used:             0
  Suggested initial size:             1M
  Successful Get Heap requests:       0
  Successful Free Heap requests:      0
  Number of segments allocated:       0
  Number of segments freed:           0
31bit I/O HEAP statistics:
  Initial size:                       12288
  Increment size:                     8192
  Total heap storage used (sugg. initial size): 3080
  Successful Get Heap requests:       23
  Successful Free Heap requests:      16
  Number of segments allocated:       1
  Number of segments freed:           0
24bit I/O HEAP statistics:
  Initial size:                       4096
  Increment size:                     4096
  Total heap storage used (sugg. initial size): 2640
  Successful Get Heap requests:       8
  Successful Free Heap requests:      0
  Number of segments allocated:       1
  Number of segments freed:           0
  Largest number of threads concurrently active: 2
End of Storage Report

```

Figure 120. 64-bit storage report (Part 2 of 2)

For storage statistics and heap storage statistics, see *z/OS Language Environment Programming Reference* in topic “Storage statistics for AMODE 64 applications”.

HeapPools storage statistics

The HEAPPOOLS run-time option (for C/C++ applications only) controls usage of the heap pools storage algorithm at the enclave level. The heap pools algorithm allows for the definition of one to twelve heap pools, each consisting of a number of storage cells of a specified length. For further details regarding HeapPools storage statistics in the storage report, see “HeapPools storage statistics” on page 405.

Modifying exception handling behavior

Setting the exception handling behavior of your routine affects the response that occurs when the routine encounters an error.

You can modify exception handling behavior in the following ways:

- Application program interfaces (API)
- User-written exception handlers
- POSIX functions (used to specifically set signal actions and signal masks)

Language Environment application program interfaces (API)

You can use the following APIs to modify exception handling:

<code>__cabend()</code>	Terminates an enclave using an abend.
<code>__le_cib_get()</code>	Returns a pointer to a condition information block (CIB) associated with a given condition token. The CIB contains detailed information about the condition.
<code>__set_exception_handler()</code>	Activates a routine to handle an exception.
<code>__reset_exception_handler()</code>	Removes handling of an exception by any routine.

Language Environment run-time options

These Language Environment run-time options can affect your routine's exception handling behavior:

TERMTHDACT	Sets the level of information that is produced when a condition of severity 2 or greater remains unhandled within the enclave. The possible parameter settings for different levels of information are: • QUIET for no information • MSG for message only • TRACE for message and a traceback • DUMP for message, traceback, and Language Environment dump • UAONLY for message and a system dump of the user address space • UATRACE for message, Language Environment dump with traceback information only, and a system dump of the user address space • UADUMP for message, traceback, Language Environment dump, and system dump • UAIMM for a system dump of the user address space of the original abend or program interrupt prior to the Language Environment condition manager processing the condition.
TRAP(ON)	Fully enables the Language Environment exception handler. This causes the Language Environment exception handler to intercept error conditions and routine interrupts. When TRAP(ON, NOSPIE) is specified, Language Environment handles all program interrupts and abends through an ESTAE. Use this feature when you do not want Language Environment to issue an ESPIE macro. During normal operation, you should use TRAP(ON) when running your applications.
TRAP(OFF)	Disables the Language Environment condition handler from handling abends and program checks/interrupts. ESPIE is not issued with TRAP(OFF). Specify TRAP(OFF) when you do not want Language Environment to issue an ESPIE. When TRAP(OFF), TRAP(OFF,SPIE), or TRAP(OFF,NOSPIE) is specified and either a program interrupt or abend occurs, the user exit for termination is ignored. TRAP(OFF) can cause several unexpected side effects. It is not supported in AMODE 64 production execution. For further information, see the TRAP run-time option in <i>z/OS Language Environment Programming Reference</i>.

Customizing exception handlers

User-written exception handlers permit you to customize exception handling for certain conditions. You can register a user-written exception handler for the current stack frame by using the `__set_exception_handler()` API.

For more information about user-written exception handlers and the Language Environment condition manager, see *z/OS XL C/C++ Programming Guide*.

Using condition information

If a condition that might require attention occurs while an application is running, Language Environment builds a condition token. The condition token contains 16 bytes (128 bits) of information about the condition that Language Environment or your routines can use to respond appropriately. Each condition is associated with a single Language Environment run-time message.

You can use this condition information in two ways:

- To specify the feedback code parameter when calling Language Environment services (see “Using the feedback code parameter”).
- To code a symbolic feedback code in a user-written exception handler (see “Using the symbolic feedback code” on page 281).

Using the feedback code parameter

The feedback code is an optional parameter of the Language Environment APIs. For C/C++ applications, this parameter is optional. For more information about **feedback codes** and condition tokens, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

When you provide the feedback code (**fc**) parameter, the API in which the condition occurs sets the feedback code to a specific value called a condition token.

The condition token does not apply to asynchronous signals. For a discussion of the distinctions between synchronous signals and asynchronous signals with POSIX(ON), see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

When you do not provide the **fc** parameter, any nonzero condition is signaled and processed by Language Environment exception handling routines. If you have registered a user-written exception handler, Language Environment passes control to the handler, which determines the next action to take. If the condition remains unhandled, Language Environment writes a message to stderr. The message is the translation of the condition token into English (or another supported national language).

Language Environment provides APIs that can be used to convert condition tokens to routine variables, messages, or signaled conditions. The following table lists these Language Environment APIs and their functions.

<code>__le_msg_write()</code>	Writes a message string to stderr
<code>__le_msg_get_and_write()</code>	Takes a message associated with a condition and writes it to stderr
<code>__le_msg_get()</code>	Retrieves, formats, and stores message data for a condition
<code>__le_msg_add_insert()</code>	Creates a message insert

For more information on these APIs, see *z/OS XL C/C++ Programming Guide*.

There are two types of condition tokens. Case 1 condition tokens contain condition information, including the Language Environment message number. All Language Environment APIs and most application routines use case 1 condition tokens. Case 2 condition tokens contain condition information and a user-specified class and cause code. Application routines, user-written exception handlers, assembler user exits, and some operating systems can use case 2 condition tokens.

0	-	31	32 - 33	34 - 36	37 - 39	40 - 63	64 - 127
Condition_ID			Case Number	Severity Number	Control Code	Facility_ID	ISI

For Case 1 condition tokens,
Condition_ID is:

0 - 15	16 - 31
Severity Number	Message Number

For Case 2 condition tokens,
Condition_ID is:

0 - 15	16 - 31
Class Code	Cause Code

A symbolic feedback code represents the first 8 bytes of a condition token. It contains the Condition_ID, Case Number, Severity Number, Control Code, and Facility_ID, whose bit offsets are indicated.

Figure 121. Language Environment condition token

For example, in the condition token: X'0003032D 59C3C5C5 00000000 00000000'

- X'0003' is severity.
- X'032D' is message number 813.
- X'59' are hexadecimal flags for case, severity, and control.
- X'C3C5C5' is the CEE facility ID.
- X'00000000 00000000' is the instance specific information (ISI). (In this case, no ISI was provided.)

If a Language Environment traceback or dump is generated while a condition token is being processed or when a condition exists, Language Environment writes the run-time message to the condition section of the traceback or dump. If a condition is detected when a Language Environment API is invoked without a feedback code, the condition token is passed to the Language Environment condition manager. If a condition is severity 0 or 1, Language Environment resumes without issuing a message. For conditions of severity 2 or greater, Language Environment issues a message and terminates. For a list of Language Environment run-time messages and corrective information, see *z/OS Language Environment Run-Time Messages*.

If a second condition is raised while Language Environment is attempting to handle a condition, the message CEE0374C CONDITION = <message no.> is displayed using a write-to-operator (WTO). The message number in the CEE0374C message indicates the original condition that was being handled when the second condition was raised. This can happen when a critical error is signaled (for example, when internal control blocks are damaged).

If the output for this error message appears several times in sequence, the conditions appear in order of occurrence. Correcting the earliest condition can cause your application to run successfully.

Using the symbolic feedback code

The symbolic feedback code represents the first 8 bytes of a 16-byte condition token. You can think of the symbolic feedback code as the nickname for a condition. As such, the symbolic feedback code can be used in user-written exception handlers to screen for a given condition, even if it occurs at different locations in an application.

For more details on symbolic feedback codes, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

Chapter 10. Classifying AMODE 64 application errors

This chapter describes errors that commonly occur in Language Environment AMODE 64 applications. It also explains how to use run-time messages and abend codes to obtain information about errors in your application.

Identifying problems in routines

The following sections describe how you can identify errors in Language Environment routines. Included are common error symptoms and solutions.

Language Environment module names

You can identify Language Environment-supplied module elements by any of the following three-character prefixes:

- CEE (Language Environment)
- CEL (Language Environment)
- EDC (C/C++)

Module elements or text files with other prefixes are not part of the Language Environment product for AMODE 64 applications.

Common errors in routines

These common errors have simple solutions:

- If you receive abend U4093, reason X'224' (548 decimal), then make sure you use MEMLIMIT to allow access to above the 2 GB Bar.
- If you do not have enough virtual storage, increase your region size or decrease your storage usage (stack size) by using the storage-related run-time options and callable services. (See “Controlling storage allocation” on page 275 for information about using storage in routines.)
- If you do not have enough disk space, increase your disk allocation.
- If executable files are not available, check your executable library to ensure that they are defined. For example, check your STEPLIB or JOBLIB definitions.

If your error is not caused by any of the items listed above, examine your routine or routines for changes since the last successful run. If there have been changes, review these changes for errors that might be causing the problem. One way to isolate the problem is to branch around or comment out recent changes and rerun the routine. If the run is successful, the error can be narrowed to the scope of the changes.

Changes in optimization levels, addressing modes, and input/output file formats can also cause unanticipated problems in your routine.

In most cases, generated condition tokens or run-time messages point to the nature of the error. The run-time messages offer the most efficient corrective action. To help you analyze errors and determine the most useful method to fix the problem, Table 28 on page 284 lists common error symptoms, possible causes, and programmer responses.

Table 28. Common error symptoms, possible causes, and programmer responses

Error Symptom	Possible Cause	Programmer Response
Numbered run-time message appears	Condition raised in routine	For any messages you receive, read the Programmer Response. For information about message structure, see “Interpreting run-time messages” below.
User abend code < 4000	a) A non-Language Environment abend occurred b) The assembler user exit requested an abend for an unhandled condition of severity ≥ 2	See the Language Environment abend codes in <i>z/OS Language Environment Run-Time Messages</i> . Check for a subsystem-generated abend or a user-specified abend.
User abend code ≥ 4000	a) Language Environment detected an error and could not proceed b) An unhandled software-raised condition occurred c) The assembler user exit requested an abend for an unhandled condition of severity 4	For any abends you receive, read the appropriate explanation listed in the abend codes section of <i>z/OS Language Environment Run-Time Messages</i> .
System abend with TRAP(OFF)	Cause depends on type of malfunction	Respond appropriately. Refer to the messages and codes book of the operating system.
System abend with TRAP(ON)	System-detected error	Refer to the messages and codes book of the operating system.
No response (wait/loop)	Application logic failure	Check routine logic.
Unexpected message (message received was not from most recent service)	Condition caused by something related to current service	Generate a traceback using <code>cdump()</code> or <code>ctrace()</code> .
Incorrect output	Incorrect file definitions, storage overlay, incorrect routine mask setting, references to uninitialized variables, data input errors, or application routine logic error	Correct the appropriate parameters.
No output	Incorrect ddname or file definitions	Correct the appropriate parameters.
Nonzero return code from enclave	The return code was issued by the application routine	Check the application for the meaning of the return code.

Interpreting run-time messages

The first step in debugging your routine is to look up any run-time messages. Run-time messages are written to the C `stderr` stream.

Run-time messages provide users with additional information about a condition, and possible solutions for any errors that occurred. They can be issued by Language Environment common routines or language-specific run-time routines and contain a message prefix, message number, severity code, and descriptive text.

In the following example Language Environment message:

CEE3206S The system detected a specification exception (System Completion Code=0C6).

- The message prefix is CEE.
- The message number is 3206.
- The severity code is S.

- The message text is “The system detected a specification exception (System Completion Code=0C6)”.

Language Environment messages can appear even though you made no explicit calls to Language Environment services. C/C++ run-time library routines commonly use the Language Environment services. This is why you can see Language Environment messages even when the application routine does not directly call common run-time services.

Message prefix

The message prefix indicates the Language Environment component that generated the message. The message prefix is the first three characters of the message number and is also the facility ID in the condition token. See the following table for more information about Language Environment run-time messages.

Message Prefix	Language Environment Component
CEE	Common run time
EDC	C/C++ run time

The messages for the various components can be found in *z/OS Language Environment Run-Time Messages*.

Message number

The message number is the 4-digit number following the message prefix. Leading zeros are inserted if the message number is less than four digits. It identifies the condition raised and references additional condition and programmer response information.

Severity code

The severity code is the letter following the message number and indicates the level of attention called for by the condition. Messages with severity “I” are informational messages and do not usually require any corrective action. In general, if more than one run-time message appears, the first noninformational message indicates the problem. For a complete list of severity codes, severity values, condition information, and default actions, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

Message text

The message text provides a brief explanation of the condition.

Understanding abend codes

Under Language Environment, abnormal terminations generate abend codes. There are two types of abend codes: 1) user abends (Language Environment and user-specified) and 2) system abends. User abends follow the format of *Udddd*, where *dddd* is a decimal user abend code. System abends follow the format of *Shhh*, where *hhh* is a hexadecimal abend code. Language Environment abend codes are usually in the range of 4000 to 4095. However, some subsystem abend codes can also fall in this range. User-specified abends use the range of 0 to 3999.

Example abend codes are:

User (Language Environment) abend code:U4041
User-specified abend code:U0005
System abend code:S80A

The Language Environment API `__cabend()` terminates your application with an abend. You can set the `clean_up` parameter value to determine how the abend is processed and how Language Environment handles the raised condition. For more information about `__cabend()` and `clean_up`, see *z/OS XL C/C++ Run-Time Library Reference*.

User abends

If you receive a Language Environment abend code, see *z/OS Language Environment Run-Time Messages* for a list of abend codes, error descriptions, and programmer responses.

System abends

If you receive a system abend code, look up the code and the corresponding information in the publications for the system you are using.

When a system abend occurs, the operating system can generate a system dump. System dumps are written to ddname SYSMDUMP, SYSABEND, or SYSUDUMP. System dumps show the memory state at the time of the condition. See “Generating a system dump” on page 306 for more information about system dumps.

Chapter 11. Using Language Environment AMODE 64 debugging facilities

This chapter describes methods of debugging AMODE 64 routines in Language Environment. Currently, most problems in Language Environment and member language routines can be determined through the use of a debugging tool or through information provided in the Language Environment dump.

Debugging tools

You can use **dbx** to debug Language Environment applications. *z/OS UNIX System Services Command Reference* has information on **dbx** subcommands, while *z/OS UNIX System Services Programming Tools* contains usage information.

Language Environment dumps

The following sections provide information about using the Language Environment dump service, and describe the contents of the Language Environment dump.

Generating a Language Environment dump with TERMTHDACT

The TERMTHDACT run-time option produces a dump during program checks or abnormal terminations. You must use TERMTHDACT(DUMP) in conjunction with TRAP(ON) to generate a Language Environment dump.

You can use TERMTHDACT to produce a traceback, Language Environment dump, or user address space dump when a thread ends abnormally because of an unhandled condition of severity 2 or greater. If this is the last thread in the process, the enclave goes away. A thread terminating in a non-POSIX environment is analogous to an enclave terminating. For information on enclave termination, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

The TERMTHDACT suboptions QUIET, MSG, TRACE, DUMP, UAONLY, UATRACE, UADUMP, and UAIMM control the level of information available. Following are the suboptions, the levels of information produced, and the destination of each.

Table 29. TERMTHDACT suboptions, level of information, and destinations

Suboption	Level of Information	Destination
QUIET	No information	No destination.
MSG	Message	Stderr
TRACE	Message and Language Environment dump containing only a traceback	Message goes to stderr. Traceback goes to CEEDUMP file.
DUMP	Message and complete Language Environment dump	Message goes to stderr. Language Environment dump goes to CEEDUMP file.

Table 29. TERMTHDACT suboptions, level of information, and destinations (continued)

Suboption	Level of Information	Destination
UAONLY	SYSMDUMP, SYSABEND dump, or SYSUDUMP depending on the DD card used in the JCL in z/OS. You will get a system dump of your user address space if the appropriate DD statement is used. Note: A Language Environment dump is not generated.	Language Environment generates a U4039 abend which allows a system dump of the user address space to be generated. For z/OS, the system dump is written to the ddname specified.
UATRACE	Message, Language Environment dump containing only a traceback, and a system dump of the user address space	Message goes to stderr. Traceback goes to CEEDUMP file. Language Environment generates a U4039 abend which allows a system dump of the user address space to be generated. For z/OS, the system dump is written to the ddname specified.
UADUMP	Message, Language Environment dump, and SYSMDUMP, SYSABEND dump, or SYSUDUMP depending on the DD card used in the JCL in z/OS.	Message goes to stderr. Language Environment dump goes to CEEDUMP file. Language Environment generates a U4039 abend which allows a system dump of the user address space to be generated. For z/OS, the system dump is written to the ddname specified.
UAIMM	Language Environment generates a system dump of the original abend/program interrupt of the user address space. You will get a system dump of your user address space if the appropriate DD statement is used. After the dump is taken by the operating system, Language Environment condition manager continues processing.	Message goes to stderr. User address space dump goes to ddname specified for z/OS.

The TRACE and UATRACE suboptions of TERMTHDACT use these dump options:

- ENCLAVE(ALL)
- NOENTRY
- CONDITION
- TRACEBACK
- THREAD(ALL)
- NOBLOCKS
- NOSTORAGE
- VARIABLES
- FILES
- STACKFRAME(ALL)
- PAGESIZE(60)
- FNAME(CEEDUMP)
- GENOPTS
- REGSTOR(96)

The DUMP and UADUMP suboptions of TERMTHDACT use these dump options:

- ENCLAVE(ALL)
- NOENTRY

- CONDITION
- TRACEBACK
- THREAD(CURRENT)
- BLOCKS
- STORAGE
- VARIABLES
- FILES
- STACKFRAME(ALL)
- PAGESIZE(60)
- FNAME(CEEDUMP)
- GENOPTS
- REGSTOR(96)

Considerations for setting TERMTHDACT options

• z/OS UNIX Considerations

- The TERMTHDACT option applies when a thread terminates abnormally. Abnormal termination of a single thread causes termination of the entire enclave. If an unhandled condition of severity 2 or higher percolates beyond the first routine's stack frame, the enclave terminates abnormally.
- If an enclave terminates due to a POSIX default signal action, then TERMTHDACT applies to conditions that result from software signals, program checks, or abends.
- If running under a shell and Language Environment generates a system dump, then a core dump is generated to a file based on the kernel environment variable, `_BPXK_MDUMP`.

• Preinitialized Environments for Authorized Programs Considerations

- The TERMTHDACT suboptions TRACE, DUMP, UADUMP, UATRACE are overridden to UAONLY.
- For UAONLY, a U4039 abend is generated and an SVC dump of the U4039 abend with the title:
`COMPON=CEL,COMPID=568819801,ISSUER=CELAFRR ,MODULE=CELAEICT+????,`
`ABEND=U4039,REASON=00000000`
is taken.
- For UAIMM, an SVC dump of the original abend/program interrupt with a title like:
`COMPON=CEL,COMPID=568819801,ISSUER=CELAFRR ,MODULE=CELAEICT+????,`
`ABEND=S00C9,REASON=00000000`
(where the ABEND and REASON values are those of the original abend/program interrupt) is taken.

For more information about the TERMTHDACT run-time option, see *z/OS Language Environment Programming Reference*.

Generating a Language Environment dump with language-specific functions

C/C++ routines can use the functions `cdump()`, `csnap()`, and `ctrace()` to produce a Language Environment dump. For more information on these functions, see “Generating a Language Environment dump of a C/C++ routine” on page 381.

Understanding the Language Environment dump

The Language Environment dump service generates output of data and storage from the Language Environment run-time environment on an enclave basis. This output contains the information needed to debug most basic routine errors.

Figure 124 on page 294 illustrates a dump for enclave main. The example shows full use of the TERMTHDACT dump options. Ellipses are used to summarize some sections of the dump and information regarding unhandled conditions may not be present at all. Sections of the dump are numbered to correspond with the descriptions given in “Sections of the Language Environment dump” on page 303.

The CEE3DMP was generated by the C program CELQSAMP shown in Figure 122 on page 291. CELQSAMP uses the DLL CELQDLL shown in Figure 123 on page 293.

```

#pragma options(SERVICE("1.2.d"),NOOPT)
#pragma runopts(TERMTHDACT(UADUMP),POSIX(ON))
#pragma runopts(TRACE(ON,1M,NODUMP,LE=1),HEAPCHK(ON))
#pragma runopts(RPTSTG(ON))
#define _OPEN_THREADS
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <dll.h>

typedef void* FUNC(void *);

pthread_mutex_t      mut;
pthread_t            thread[2];
int                  threads_joined = 0;
char *               t1 = "Thread 1";
char *               t2 = "Thread 2";
/*****
/*  thread_func: Invoked via pthread_create.
*****/
void *thread_func(void *parm)
{
    printf(">>> Thread_func: %s locking mutex\n",parm);
    pthread_mutex_lock(&mut);
    pthread_mutex_unlock(&mut);
    printf(">>> Thread_func: %s exiting\n",parm);
    pthread_exit(NULL);
}
/*****
/*  Start of Main function.
*****/
main()
{
    dllhandle *       handle;
    FUNC *            fp;
    FILE*             fp1;
    FILE*             fp2;

    printf("Load DLL...\n");
    handle = dllload("CELQDLL");
    if (handle == NULL) {
        perror("Could not load DLL CELQDLL");
        exit(106);
    }

    printf("Query DLL...\n");
    fp = (FUNC *)dllqueryfn(handle,"div_zero");
    if (fp == NULL) {
        perror("Could not find thread_func");
        exit(107);
    }

    printf("Init MUTEX...\n");
    if (pthread_mutex_init(&mut, NULL) == -1) {
        perror("Init of mut failed");
        exit(101);
    }
}

```

Figure 122. The C program CELQSAMP (Part 1 of 2)

```

printf("Lock Mutex Lock...\n");
if (pthread_mutex_lock(&mut) == -1) {
    perror("Lock of mut failed");
    exit(102);
}

printf("Create 1st thread...\n");
if (pthread_create(&thread[0],NULL,thread_func,(void *)t1) == -1) {
    perror("Could not create thread #1");
    exit(103);
}

printf("Create 2nd thread...\n");
if (pthread_create(&thread[1],NULL,thread_func,(void *)t2) == -1) {
    perror("Could not create thread #2");
    exit(104);
}
printf("Write to some files...\n");
fp1 = fopen("myfile.data", "w");
if (!fp1) {
    perror("Could not open myfile.data for write");
    exit(109);
}

fprintf(fp1, "record 1\n");
fprintf(fp1, "record 2\n");
fprintf(fp1, "record 3\n");

fp2 = fopen("memory.data", "wb,type=memory");
if (!fp2) {
    perror("Could not open memory.data for write");
    exit(112);
}

fprintf(fp2, "some data");
fprintf(fp2, "some more data");
fprintf(fp2, "even more data");

printf("Call div_zero...\n");
fp(NULL);

printf("Error -- Should not get here\n");
exit(110);
}

```

Figure 122. The C program CELQSAMP (Part 2 of 2)

```

/* DLL containing div_zero */
#pragma options(SERVICE("1.4.f"),NOOPT)
#pragma export(div_zero)
#include <stdio.h>
#include <stdlib.h>
char wsa_array[11] = { 'C','E','L','Q','D','L','L',' ','W','S','A'};
/*****
/* div_zero: Cause divide by zero exception */
*****/
void *div_zero(void *parm)
{
    int i = 0;

    printf("Divide by zero...\n");
    i = 1/i;
    printf("Error -- Should not get here. i=%d\n",i);
    exit(110);
}

```

Figure 123. The C DLL CELQDLL

For easy reference, the sections of the following dump are numbered to correspond with the descriptions in “Sections of the Language Environment dump” on page 303.

[2]Information for enclave main

[3]Information for thread 2547D8200000000

[4]Traceback:

DSA	Entry	E Offset	Load Mod	Program Unit	Service	Status
00000001	CEEHDS	+00000000	CELQLIB	CEEHDS	HLE7709	Call
00000002	CEEOSIG	+00000956	CELQLIB	CEEOSIG	HLE7709	Call
00000003	CELQHROD	+00000256	CELQLIB	CELQHROD	HLE7709	Call
00000004	CEEOSIG	-1CF02BFC	CELQLIB	CEEOSIG	HLE7709	Call
00000005	CELQHROD	+00000256	CELQLIB	CELQHROD	HLE7709	Call
00000006	div_zero	+00000030	CELQDLL		1.4.f	Exception
00000007	main	+000002DC	CELQSAMP		1.2.d	Call
00000008	CELQINIT	+00001146	CELQLIB	CELQINIT	HLE7709	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Attributes
00000001	00000001082FAAC0	000000002517A8D8	000000002517A8D8	00000000	20040312	XPLINK EBCDIC POSIX Floating Point
00000002	00000001082FD3E0	0000000025203768	0000000025203768	00000956	20040312	XPLINK EBCDIC POSIX Floating Point
00000003	00000001082FDD00	0000000025118500	0000000025118500	00000256	20040312	XPLINK EBCDIC POSIX Floating Point
00000004	00000001082FE020	00000000251FCE40	00000000251FCE40	1CF02BF0	20040312	XPLINK EBCDIC POSIX Floating Point
00000005	00000001082FEE40	0000000025118500	0000000025118500	00000256	20040312	XPLINK EBCDIC POSIX Floating Point
00000006	00000001082FF080	00000000255AB660	0000000000000000	*****	20040408	XPLINK EBCDIC POSIX IEEE
00000007	00000001082FF180	00000000251000C8	0000000000000000	*****	20040408	XPLINK EBCDIC POSIX IEEE
00000008	00000001082FF280	0000000025103010	0000000025103010	00001146	20040312	XPLINK EBCDIC POSIX Floating Point

[5]Condition Information for Active Routines

Condition Information for (DSA address 00000001082FF080)

CIB Address: 00000001082FBD00

Current Condition:

CEE0198S The termination of a thread was signaled due to an unhandled condition.

Original Condition:

CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

Location:

Program Unit: Entry: div_zero Statement: Offset: +00000030

Machine State:

ILC..... 0004 Interruption Code..... 0009

PSW..... 0785240180000000 00000000255AB694

GPR0..... 0000000000000000 GPR1..... 0000000000000001 GPR2..... 00000001082FF028 GPR3..... 0000000000000012

GPR4..... 00000001082FF080 GPR5..... 00000000000000F0 GPR6..... 000000002548EBA8 GPR7..... 00000000255AB686

GPR8..... 00000000255AB6C4 GPR9..... 0000000108301140 GPR10..... 00000000255AB6D8 GPR11..... 0000000108948F10

GPR12..... 00000001000039D0 GPR13..... 0000000000006F48 GPR14..... 000000007F7864E8 GPR15..... 000000000000001F

FPC..... 00000000

FPR0..... 43276323 A99240CD FPR1..... 3FE6A09E 667F3BCD

FPR2..... 3FF20DD7 50429B6D FPR3..... 34100000 00000000

FPR4..... 3FF6A09E 667F3BCD FPR5..... 00000000 00000000

FPR6..... 3FD45F30 6DC9C883 FPR7..... 00000000 00000000

FPR8..... 00000000 00000000 FPR9..... 00000000 00000000

FPR10..... 00000000 00000000 FPR11..... 00000000 00000000

FPR12..... 00000000 00000000 FPR13..... 00000000 00000000

FPR14..... 00000000 00000000 FPR15..... 00000000 00000000

Storage dump near condition, beginning at location(00000000255AB680)

+0000 00000000255AB680 B904001A 0D760700 A7090001 8E000020 |.....x.....|

+0010 00000000255AB690 5D008000 B9140021 EB569000 00044110 |).....|

[6]Parameters, Registers, and Variables for Active Routines:

div_zero (DSA address 00000001082FF080):

DOWNSTACK DSA

Saved Registers:

GPR0..... ***** GPR1..... ***** GPR2..... ***** GPR3..... *****

GPR4..... 00000001082FF080 GPR5..... BBBB BBBB BBBB BBBB GPR6..... 0000000025118500 GPR7..... 00000000255AB686

GPR8..... 00000000255AB6C4 GPR9..... 0000000108301140 GPR10..... 00000000255AB6D8 GPR11..... 0000000108948F10

GPR12..... 4040404040404040 GPR13..... 4040404040404040 GPR14..... 4040404040404040 GPR15..... 4040404040404040

GPREG STORAGE:

Storage around GPR0 is invalid.

Storage around GPR1 is invalid.

Storage around GPR2 is invalid.

Storage around GPR3 is invalid.

Storage around GPR4 (00000001082FF080)

+0800 00000001082FF880 00000001 082FF180 F0F0F0F0 F0F0F0F0 |.....1.00000000|

+0810 00000001082FF890 00000000 255AB660 00000000 251003A6 |.....!-.....w|

+0820 00000001082FF8A0 00000000 25100460 00000001 08300070 |.....!-.....|

+0830 00000001082FF8B0 00000000 255A89A8 00000001 08948F10 |.....!iy.....m..|

+0840 00000001082FF8C0 00000001 000039D0 00000000 00006F48 |.....!.....?..|

+0850 00000001082FF8D0 00000000 7F7864E8 00000000 0000001F |.....!.....Y.....|

:

Figure 124. Example dump using CEE3DMP (Part 1 of 10)


```

Storage around GPR15(4040404040404040):
-0020 4040404040404020 Inaccessible storage.
-0010 4040404040404030 Inaccessible storage.
+0000 4040404040404040 Inaccessible storage.
+0010 4040404040404050 Inaccessible storage.
+0020 4040404040404060 Inaccessible storage.
+0030 4040404040404070 Inaccessible storage.
main (DSA address 00000001082FF180):
DOWNSTACK DSA
Saved Registers:
GPR0..... ***** GPR1..... ***** GPR2..... ***** GPR3..... *****
GPR4..... ***** GPR5..... ***** GPR6..... 00000000255AB660 GPR7..... 00000000251003A6
GPR8..... 0000000025100460 GPR9..... 0000000108300070 GPR10..... 00000000255A89A8 GPR11..... *****
GPR12..... ***** GPR13..... ***** GPR14..... ***** GPR15..... *****
GPREG STORAGE:
Storage around GPR0 is invalid.
Storage around GPR1 is invalid.
:
:
CELQINIT (DSA address 00000001082FF280):
DOWNSTACK DSA
Saved Registers:
GPR0..... ***** GPR1..... ***** GPR2..... ***** GPR3..... *****
GPR4..... ***** GPR5..... ***** GPR6..... 00000000251000C8 GPR7..... 0000000025104158
GPR8..... 0000000000006FE0 GPR9..... 0000000025101598 GPR10..... 00000000251011F0 GPR11..... 00000001089491B0
GPR12..... ***** GPR13..... ***** GPR14..... ***** GPR15..... *****
GPREG STORAGE:
Storage around GPR0 is invalid.
Storage around GPR1 is invalid.
:
:
[7]Control Blocks for Active Routines:
DSA for CEEHDSR: 00000001082FB2C0
+000000 R4..... 00000001082FD3E0 R5..... 000000002517EF0C R6..... 000000002517A8D8
+000018 R7..... 00000000252040C0 R8..... 0000000108913470 R9..... 0000000000000005
+000030 R10..... 00000001082FE3DF R11..... 00000001082FE0CC R12..... 0000000108913470
+000048 R13..... 00000001082FE680 R14..... 0000000025203FC0 R15..... 00000000251000B0
+000060 reserved. 0000000000000000 reserved. 0000000000000000 HPTRAN... 0000000000000000
+000078 reserved. 0000000000000000
DSA for CEEOSIGJ: 00000001082FDBE0
+000000 R4..... 00000001082FDDE0 R5..... 000000002520508C R6..... 0000000025203768
+000018 R7..... 0000000025118758 R8..... 00000000255A4AF8 R9..... 00000000255A4BF0
+000030 R10..... 00000000255A4A48 R11..... 0000000000000020 R12..... 0000000100006160
+000048 R13..... 00000001082FE680 R14..... 00000000252020B4 R15..... 0000000000000003
+000060 reserved. 00000001082FDEE8 reserved. 00000001082FDF70 HPTRAN... 00000001082FDF60
+000078 reserved. 0000000100006160
DSA for CELQHROD: 00000001082FE5E0
+000000 R4..... E3D9C1D5E3D9C1D5 R5..... CCCCCCCCCCCCCCCC R6..... 0000000025118500
+000018 R7..... 0000000000000000 R8..... 0000000000000008 R9..... 0000000001049CA8
+000030 R10..... 00000001082FF01F R11..... 0000000119500050 R12..... 0000000100006160
+000048 R13..... 00000001082FF6E0 R14..... 00000000251FEB90 R15..... 00000000008DFC48
+000060 reserved. 0000000100006160 reserved. 00000001082FF6E0 HPTRAN... 00000001082FE720
+000078 reserved. 00000000026744E6
DSA for CELQHROD: 00000001082FE720
+000000 EYE..... 64INTRPT TRTYPE... 00000010F6F4C9D5 reserved. 19400050F6F4C9D5 reserved. 0000000100008B30
+000018 reserved. 00000000255A89C8 reserved. 00000001082FED70 reserved. 0000000100006160
+000030 TRANEP... 0000000025118500 TR_R0... 00000001000074F8 TR_R1... 00000000000079B0
+000048 TR_R2... 0000000000000004 TR_R3... 0000000F2550B4F0 TR_R4... 00000001082FE020
+000060 TR_R5... 00000001082FEBD0 TR_R6... 00000001082FEBD8 TR_R7... 0000000000000003
+000078 TR_R8... 000000007F786620 TR_R9... 00000001082FEBD0 TR_R10... 00000001082FEBD8
+000090 TR_R11... 00000001082FE7D8 TR_R12... 000000002545257A TR_R13... 0000000000000000
+0000A8 TR_R14... 0000000000000000 TR_R15... 0000000100006160 reserved. 0000000000000000
+0000C0 ROND_DSA. 00000001082FDDE0 ROND_R13. 00000000255A49B8 ROND_R14. 00000000252020B4

```

Figure 124. Example dump using CEE3DMP (Part 2 of 10)

```

DSA for CEEOSIGG: 00000001082FE820
+000000 R4..... 00000001082FEE40
+000018 R7..... 0000000025118758
+000030 R10..... 00000000255A4148
+000048 R13..... 00000001082FF6E0
+000060 reserved. 00000001082FE380
+000078 reserved. 00000000251DAE54
DSA for CELQHRD: 00000001082FF640
+000000 R4..... E3D9C1D5E3D9C1D5
+000018 R7..... 0000000000000000
+000030 R10..... 00000000255AB6D8
+000048 R13..... 0000000000006F48
+000060 reserved. 00000000255AB686
+000078 reserved. 3FF6A09E667F3BCD
DSA for CELQHRD: 00000001082FF780
+000000 EYE..... 64INTRPT TRTYPE... 00000010F6F4C9D5 reserved. F0F0F0F0F6F4C9D5 reserved. F0F0F0F1F240C5D9
+000018 reserved. D9D5D67EF0F0F0F0 reserved. F0F0F0F040C5D9D9 reserved. D5D6F27EF0F0F0F0
+000030 TRANEP... 0000000025118500 TR_R0.... 00000001082FF748 TR_R1.... 0000000700000000
+000048 TR_R2.... 0000000000000040 TR_R3.... 0000000025490118 TR_R4.... 00000001082FF080
+000060 TR_R5.... 0000000000000012 TR_R6.... 40F0F0F0F0F0F0F0 TR_R7.... 00000000255AB686
+000078 TR_R8.... F204404040404040 TR_R9.... 000000000000006F0 TR_R10... 0000000025496F23
+000090 TR_R11... 0000000000000000 TR_R12... 00000001082FF712 TR_R13... 0000000000000000
+0000A8 TR_R14... 00000000255AB6D8 TR_R15... 00000001082FF680 reserved. 0000000100006160
+0000C0 ROND_DSA. 00000001082FEE40 ROND_R13. 00000000255A4078 ROND_R14. 000000002518748A
DSA for div_zero: 00000001082FF880
+000000 R4..... 00000001082FF180
+000018 R7..... 00000000251003A6
+000030 R10..... 00000000255A89A8
+000048 R13..... 0000000000006F48
+000060 reserved. 0000000000000011
+000078 reserved. F19F000000000000
CIB for div_zero(00000001082FBD0F)
+0000 00000001082FBD0F C3C9C240 00000000 00000000 00000000 | CIB .....
+0010 00000001082FBE00 00000000 00000000 01900004 00000000 | .....
+0020 00000001082FBE10 00000000 00000000 000300C6 59C3C5C5 | .....F.CEE
+0030 00000001082FBE20 00000000 00000000 00000001 082F8F80 | .....
+0040 00000001082FBE30 00030C89 59C3C5C5 00000000 00000003 | ...i.CEE.....
+0050 00000001082FBE40 00000004 00000000 00000001 082FF180 | .....1.
+0060 00000001082FBE50 00000000 2517F4E8 00000000 00000000 | .....4Y.....
+0070 00000001082FBE60 00000001 082FF080 00000000 255AB694 | .....0.....!m
+0080 00000001082FBE70 00000001 00005648 00000003 00000000 | .....
+0090 00000001082FBE80 00000001 082FF080 01010000 00000000 | .....0.....
+00A0 00000001082FBE90 00000000 00000000 00000000 00000000 | .....
+00B0 00000001082FBEA0 - +0000FF 00000001082FBEEF | same as above
+0100 00000001082FBEF0 48220400 00000000 940C9000 00000009 | .....m.....
+0110 00000001082FBF00 00000000 00000000 00000000 251006C8 | .....H
+0120 00000001082FBF10 00000001 082FF080 00000001 082FF080 | .....0.....
+0130 00000001082FBF20 00000000 255AB690 00000000 00000000 | .....!.....
+0140 00000001082FBF30 00000000 00000000 00000067 00000000 | .....
+0150 00000001082FBF40 00000001 082FF180 00000003 00000008 | .....1.....
+0160 00000001082FBF50 00000014 00000004 00000000 00000000 | .....
+0170 00000001082FBF60 00000000 00000000 00000008 00000000 | .....
+0180 00000001082FBF70 00000001 00005868 00000000 00000000 | .....
DSA for main: 00000001082FF980
+000000 R4..... 00000001082FF280
+000018 R7..... 0000000025104158
+000030 R10..... 00000000251011F0
+000048 R13..... 0000000000006F48
+000060 reserved. 0000000000000000
+000078 reserved. 0000000000000000
DSA for CELQINIT: 00000001082FFA80
+000000 R4..... 00000001082FF760
+000018 R7..... 00000000251042FE
+000030 R10..... 0000000000000000
+000048 R13..... 0000000000000000
+000060 reserved. 0000000000000000
+000078 reserved. 0000000000000000
R5..... 00000000251FFCA8
R8..... 00000000255A41D0
R11..... 0000000000000020
R14..... 000000002518748A
reserved. 00000000251EBA58
R6..... 00000000251FCE40
R9..... 00000000255A41D8
R12..... 0000000100006160
R15..... 00000000255AB686
HPTRAN... 00000000251E22A8
R5..... CCCCCCCCCCCCCCCC
R8..... 00000000255AB6C4
R11..... 0000000108948F10
R14..... 000000007F7864E8
reserved. 43276323A99240CD
R6..... 0000000025118500
R9..... 0000000108301140
R12..... 00000001000039D0
R15..... 000000000000001F
HPTRAN... 00000001082FF780
R5..... F0F0F0F0F0F0F0F0
R8..... 0000000025100460
R11..... 0000000108948F10
R14..... 000000007F7864E8
reserved. F9F0F0F0F0F0F0F0
R6..... 00000000255AB660
R9..... 0000000108300070
R12..... 00000001000039D0
R15..... 000000000000001F
HPTRAN... F00F0F0F0F0F0F0F1

```

Figure 124. Example dump using CEE3DMP (Part 3 of 10)

[8]Storage for Active Routines:

```

DSA frame(00000001082FAAC0)
+0800 00000001082FB2C0 00000001 082FD3E0 00000000 2517EF0C |.....L.....|
+0810 00000001082FB2D0 00000000 2517A8D8 00000000 252040C0 |.....yQ.....|
+0820 00000001082FB2E0 00000001 08913470 00000000 00000005 |.....j.....|
+0830 00000001082FB2F0 00000001 082FE3DF 00000001 082FE0CC |.....T.....|
+0870 00000001082FB330 - +00087F 00000001082FB33F      same as above

:
+2540 00000001082FD000 - +00310F 00000001082FDBCf      same as above
+3110 00000001082FDBD0 00000000 25213A80 00000000 25210E6E |.....>|
DSA frame(00000001082FD3E0)
+0800 00000001082FDBE0 00000001 082FDDE0 00000000 2520508C |.....&.....|
+0810 00000001082FDBF0 00000000 25203768 00000000 25118758 |.....g.....|
+0820 00000001082FDC00 00000000 255A4AF8 00000000 255A4BF0 |.....!8.....!0|

:
+1180 00000001082FE560 00000000 00000000 00000000 00000000 |.....|
+1190 00000001082FE570 - +0011FF 00000001082FE5DF      same as above

DSA frame(00000001082FDDE0)
+0800 00000001082FE5E0 E3D9C1D5 E3D9C1D5 CCCCCCCC CCCCCCCC |TRANTRAN.....|

DSA frame(00000001082FE020)
+0800 00000001082FE820 00000001 082FEE40 00000000 251FFCA8 |.....y.....|
+0810 00000001082FE830 00000000 251FCE40 00000000 25118758 |.....g.....|
+0820 00000001082FE840 00000000 255A41D0 00000000 255A41D8 |.....!.....!Q|
+0830 00000001082FE850 00000000 255A4148 00000000 00000020 |.....!.....|
+0840 00000001082FE860 00000001 00006160 00000001 082FF6E0 |...../-.....6|

:
+1600 00000001082FF620 00000000 25100460 00000001 08300070 |.....-.....|
+1610 00000001082FF630 00000000 255A89A8 00000001 08948F10 |.....!iy.....m..|

DSA frame(00000001082FEE40)
+0800 00000001082FF640 E3D9C1D5 E3D9C1D5 CCCCCCCC CCCCCCCC |TRANTRAN.....|
DSA frame(00000001082FF080)
+0800 00000001082FF880 00000001 082FF180 F0F0F0F0 F0F0F0F0 |.....1.00000000|
+0810 00000001082FF890 00000000 255AB660 00000000 251003A6 |.....!-.....w|
+0820 00000001082FF8A0 00000000 25100460 00000001 08300070 |.....-.....|
+0830 00000001082FF8B0 00000000 255A89A8 00000001 08948F10 |.....!iy.....m..|

:
+08B0 00000001082FF930 00000001 082FF780 00000001 00006160 |.....7...../-|
+08C0 00000001082FF940 00000000 00006F48 00000000 7F7864E8 |.....?.....".Y|
+08D0 00000001082FF950 00000000 0000001F 00000000 00000000 |.....|
+08E0 00000001082FF960 00000000 00000000 00000000 00000000 |.....|
+08F0 00000001082FF970 - +0008FF 00000001082FF97F      same as above

DSA frame(00000001082FF180)
+0800 00000001082FF980 00000001 082FF280 00000000 00000000 |.....2.....|
+0810 00000001082FF990 00000000 251000C8 00000000 25104158 |.....H.....|
+0820 00000001082FF9A0 00000000 00006FE0 00000000 25101598 |.....?.....q|
+0830 00000001082FF9B0 00000000 251011F0 00000001 08949180 |.....0.....mj|
+0840 00000001082FF9C0 00000001 000039D0 00000000 00006F48 |.....?.....|
+0850 00000001082FF9D0 00000000 7F7864E8 00000000 0000001F |.....".Y.....|
+0860 00000001082FF9E0 00000000 00000000 00000000 00000000 |.....|
+0870 00000001082FF9F0 - +00087F 00000001082FF9FF      same as above
+0880 00000001082FFA00 00000000 25100653 00000000 255A89C8 |.....!iH.....|
+0890 00000001082FFA10 00000000 0000000E 00000000 2510046C |.....%.....|
+08A0 00000001082FFA20 00000000 00000000 00000000 00000000 |.....|
+08B0 00000001082FFA30 - +0008FF 00000001082FFA7F      same as above

DSA frame(00000001082FF280)
+0800 00000001082FFA80 00000001 082FF760 00000000 00000000 |.....7-.....|
+0810 00000001082FFA90 00000000 25103010 00000000 251042FE |.....|
+0820 00000001082FFAA0 00000000 00000000 00000000 00000000 |.....|
+0830 00000001082FFAB0 - +00087F 00000001082FFAFF      same as above
+0880 00000001082FFB00 00000001 082FFD04 00000001 00006160 |...../-.....|
+0890 00000001082FFB10 00000001 082FF760 00000000 00000000 |.....7-.....|
+08A0 00000001082FFB20 00000000 00000000 00000000 00000000 |.....|
+08B0 00000001082FFB30 - +0009FF 00000001082FFC7F      same as above

:

```

Figure 124. Example dump using CEE3DMP (Part 4 of 10)

```

+0B10 00000001082FFD90 - +000B4F 00000001082FFDCF same as above
+0B50 00000001082FFDD0 00000000 00000000 00000001 08300070 .....
+0B60 00000001082FFDE0 00000000 00000000 00000000 00000000 .....
+0B70 00000001082FFDF0 - +000BAF 00000001082FFE2F same as above
+0BB0 00000001082FFE30 00000001 08300070 00000000 00000003 .....
+0BC0 00000001082FFE40 00000001 089491C0 00000001 08943410 .....mj.....m..
+0BD0 00000001082FFE50 00000001 082FF280 00000000 25104240 .....2.....
+0BE0 00000001082FFE60 00000000 251000C8 00000000 25120158 .....H.....
+0BF0 00000001082FFE70 00000000 00000000 00000000 00000000 .....
+0C00 00000001082FFE80 - +000CDF 00000001082FFF5F same as above

[9]Control Blocks Associated with the Thread:
CAA(0000000100006160)
+0000 0000000100006160 00000000 00000000 00000000 00000000 .....
+0010 0000000100006170 - +0002AF 000000010000640F same as above
+02B0 0000000100006410 00008000 00000000 00000000 00000000 .....
+02C0 0000000100006420 00000000 00000000 00000000 00000000 .....
+02D0 0000000100006430 - +00030F 000000010000646F same as above
+0310 0000000100006470 00000001 00009FB0 00000000 00000000 .....
+0320 0000000100006480 00000001 00003C28 00000001 00000000 .....

:
+03F0 0000000100006550 00000000 00000000 00000000 2515E28 .....
+0400 0000000100006560 00000000 00000000 00000000 00000000 .....
+0410 0000000100006570 00000001 00006578 00000001 00000000 .....
+0420 0000000100006580 00000000 00000000 00000000 00000000 .....
+0430 0000000100006590 - +0004BF 000000010000661F same as above

Thread Synchronization Queue Element (SQEL) (00000000255A3340)
+0000 00000000255A3340 00000000 00000000 00000000 00000000 .....
+0010 00000000255A3350 00000000 00000000 00000001 08301190 .....
+0020 00000000255A3360 00000000 00000000 00000001 08911438 .....j..
+0030 00000000255A3370 00000000 00000000 00000001 00006160 ...../-
+0040 00000000255A3380 00000000 00000000 00000000 00000000 .....
+0050 00000000255A3390 - +00006F 00000000255A33AF same as above

DUMMY DSA: 00000001082FFF60
+000000 R4..... 0000000000000000 R5..... 0000000000000000 R6..... 0000000000000000
+000018 R7..... 1111111111111111 R8..... 0000000000000000 R9..... 0000000000000000
+000030 R10..... 0000000000000000 R11..... 0000000000000000 R12..... 0000000000000000
+000048 R13..... 0000000000000000 R14..... 0000000000000000 R15..... 0000000000000000
+000060 reserved. 0000000000000000 reserved. 0000000000000000 HPTRAN... 0000000000000000
+000078 reserved. 0000000000000000

[3]Information for thread 2548146000000001

[4]Traceback:
DSA Entry E Offset Load Mod Program Unit Service Status
00000001 CEEOPML2 +00000000 CELQLIB CEEOPML2 HLE7709 Call
00000002 @@WRAP@TRACE+000001B4 CELQLIB Call
00000003 thread_func +00000036 CELQSAMP 1.2.d Call
00000004 CELQPCMM +00000E56 CELQLIB CELQPCMM HLE7709 Call

DSA DSA Addr E Addr PU Addr PU Offset Comp Date Attributes
00000001 0000000110BFEB60 00000000251DF5D8 00000000251DF5D8 00000000 20040312 XPLINK EBCDIC POSIX Floating Point
00000002 0000000110BFEF40 000000002548EBA8 0000000000000000 00000000 20040312 XPLINK EBCDIC POSIX Floating Point
00000003 0000000110BFF2C0 00000000251003E8 0000000000000000 00000000 20040408 XPLINK EBCDIC POSIX IEEE
00000004 0000000110BFF3C0 0000000025124FC0 0000000025124FC0 00000E56 20040312 XPLINK EBCDIC POSIX Floating Point

GPR0.... 0000000000000001 GPR1.... 00000000255B5AC8 GPR2.... 00000001089102A0 GPR3.... 00000000255B5AC8
GPR4.... 0000000110BFEB60 GPR5.... 00000000251E0DF4 GPR6.... 00000000251E22A8 GPR7.... 00000000251E07CC
GPR8.... 00000000DAA4A538 GPR9.... 0000000110E013C8 GPR10.... 0000000110BFF7C0 GPR11.... 0000000110BFF8C0
GPR12.... 0000000110E013C8 GPR13.... 000000007F661000 GPR14.... 0000000100002468 GPR15.... 00000001808C4110

GPREG STORAGE:
Storage around GPR0 (0000000000000001)
-0001 0000000000000000 Inaccessible storage.
+000F 0000000000000010 Inaccessible storage.

:

```

Figure 124. Example dump using CEE3DMP (Part 5 of 10)

```

[7]]Control Blocks for Active Routines:
DSA for CEEOPML2: 0000000110BFF360
+000000 R4..... 0000000110E013C8      R5..... 0000000100008B30      R6..... 00000000251DF5D8
+000018 R7..... 000000002548ED5E      R8..... 0000000110BFFA00      R9..... 0000000000000000
+000030 R10..... 0000000110BFF7C0      R11..... 0000000110BFF8C0      R12..... 0000000110E013C8
:

[8]Storage for Active Routines:
DSA frame(0000000110BFE60)
+0800 0000000110BFF360 00000001 10E013C8 00000001 00008B30 |.....H.....|
+0810 0000000110BFF370 00000000 251DF5D8 00000000 2548ED5E |.....5Q.....;|
+0820 0000000110BFF380 00000001 10BFFA00 00000000 00000000 |.....|
+0830 0000000110BFF390 00000001 10BFF7C0 00000001 10BFF8C0 |.....7.....8.|
:

[9]Control Blocks Associated with the Thread:
CAA(0000000110E013C8)
+0000 0000000110E013C8 00000000 00000000 00000000 00000000 |.....|
+0010 0000000110E013D8 - +0002AF 0000000110E01677      same as above
+02B0 0000000110E01678 00008000 00000000 00000000 00000000 |.....|
:

[3]Information for thread 2548237000000002

[4]Traceback:
DSA      Entry      E Offset Load Mod Program Unit Service Status
00000001 CEEOPML2      +00000000 CELQLIB CEEOPML2      HLE7709 Call
00000002 @@WRAP@TRACE+000001B4 CELQLIB
00000003 thread_func +00000036 CELQSAMP      1.2.d Call
00000004 CELQPCMM      +00000E56 CELQLIB CELQPCMM      HLE7709 Call

DSA      DSA Addr      E Addr      PU Addr      PU Offset      Comp Date Attributes
00000001 000000001190FEB60 000000000251DF5D8 000000000251DF5D8 00000000      20040312 XPLINK EBCDIC POSIX Floating Point
00000002 000000001190FEF40 0000000002548EBA8 0000000000000000 *****      20040312 XPLINK EBCDIC POSIX Floating Point
00000003 000000001190FF2C0 000000000251003E8 0000000000000000 *****      20040408 XPLINK EBCDIC POSIX IEEE
00000004 000000001190FF3C0 00000000025124FC0 00000000025124FC0 000000E56      20040312 XPLINK EBCDIC POSIX Floating Point

GPR0..... 0000000000000001 GPR1..... 00000000255B5580 GPR2..... 00000001089102A0 GPR3..... 00000000255B5580
GPR4..... 00000001190FEB60 GPR5..... 00000000251E0DF4 GPR6..... 00000000251E22A8 GPR7..... 00000000251E07CC
GPR8..... 00000000DAA4AA80 GPR9..... 00000001193013C8 GPR10..... 00000001190FF7C0 GPR11..... 00000001190FF8C0
GPR12..... 00000001193013C8 GPR13..... 000000007F669000 GPR14..... 0000000100002468 GPR15..... 00000001808C4198

GPREG STORAGE:
Storage around GPR0 (0000000000000001)
-0001 0000000000000000 Inaccessible storage.
+000F 0000000000000010 Inaccessible storage.
:

[7]]Control Blocks for Active Routines:
DSA for CEEOPML2: 00000001190FF360
+000000 R4..... 00000001193013C8      R5..... 0000000100008B30      R6..... 00000000251DF5D8
+000018 R7..... 000000002548ED5E      R8..... 00000001190FFA00      R9..... 0000000000000000
+000030 R10..... 00000001190FF7C0      R11..... 00000001190FF8C0      R12..... 00000001193013C8
+000048 R13..... 000000007F669000      R14..... 000000007F66D2E8      R15..... 00000001190FFB40
+000060 reserved. 00000001083012D0      reserved. 0000000000000000      HPTRAN... 0000000000000000
+000078 reserved. 0000000000000000
DSA for @@WRAP@TRACE: 00000001190FF740
+000000 R4..... 00000001190FF2C0      R5..... 0000000000002050      R6..... 000000002548EBA8
+000018 R7..... 0000000025100420      R8..... 0000000025100460      R9..... 000000002510046C
+000030 R10..... 0000000108300070      R11..... 00000001190FFEE0      R12..... 000000007F66D2E8
+000048 R13..... 000000007F669000      R14..... 000000007F66D2E8      R15..... 0000000120000000
+000060 reserved. 00000001190FF908      reserved. 00000001190FF910      HPTRAN... 00000001190FF918
+000078 reserved. 00000001190FF920
DSA for thread_func: 00000001190FFAC0

```

Figure 124. Example dump using CEE3DMP (Part 6 of 10)

```

DSA for thread_func: 00000001190FFAC0
+000000 R4..... 000000010894B6F0
+000018 R7..... 0000000025125E18
+000030 R10..... 0000000000000080
+000048 R13..... 00000001190FFE10
+000060 reserved. 000000010894B9F0
+000078 reserved. 0000000025A8DC8
DSA for CELQPCMM: 00000001190FFBC0
+000000 R4..... 00000001190FF760
+000018 R7..... 00000000251264AC
+000030 R10..... 0000000000000080
+000048 R13..... 000000007F669000
+000060 reserved. 0000000000000000
+000078 reserved. 0000000000000000
R5..... 00000001190FFBA0
R8..... 0000000119301FA0
R11..... 0000000119301750
R14..... 00000000251189B4
reserved. 0000000000000000
R6..... 00000000251003E8
R9..... 000000007F6690D0
R12..... 000000007F668860
R15..... 00000000253F44F0
HPTRAN... 000000010894B8BC
R5..... 0000000000000000
R8..... 0000000119301FA0
R11..... 00000001190FFEE0
R14..... 000000007F66D2E8
reserved. 0000000000000000
R6..... 0000000025124FC0
R9..... 000000007F6690D0
R12..... 000000007F66D2E8
R15..... 0000000120000000
HPTRAN... 0000000000000000

```

[8]Storage for Active Routines:

```

DSA frame(00000001190FEB60)
+0800 00000001190FF360 00000001 193013C8 00000001 00008B30 |.....H.....|
+0810 00000001190FF370 00000000 251DF5D8 00000000 2548ED5E |.....50.....|
+0820 00000001190FF380 00000001 190FFA00 00000000 00000000 |.....|
:
:

```

[9]Control Blocks Associated with the Thread:

```

CAA(00000001193013C8)
+0000 00000001193013C8 00000000 00000000 00000000 00000000 |.....|
+0010 00000001193013D8 - +0002AF 0000000119301677 same as above
:
:

```

[10]Enclave Control Blocks:

```

EDB(00000001000039D0)
+0000 00000001000039D0 C3C5C5C5 C4C24040 00000000 00000000 |CEEEDB .....|
+0010 00000001000039E0 00000000 00000000 00000000 00000000 |.....|
+0020 00000001000039F0 - +0000FF 0000000100003ACF same as above
+0100 0000000100003AD0 97000100 00000000 00000001 00004F40 |p.....|
+0110 0000000100003AE0 00000001 00004458 00000000 00000000 |.....|
+0120 0000000100003AF0 00000001 00002468 00000001 0000EE70 |.....|
:
:
+0170 0000000100003B40 00000000 00006FE0 00000000 00000000 |.....?.....|
+0180 0000000100003B50 00000000 00000200 00000001 000031A8 |.....y.....|
+0190 0000000100003B60 00000001 00003B58 00000000 00000000 |.....|
+01A0 0000000100003B70 00000000 00000000 00000000 00000000 |.....|
+01B0 0000000100003B80 - +0001FF 0000000100003BCF same as above
MEML(0000000100004F40)
+0000 0000000100004F40 00000000 00000000 00000000 00000000 |.....|
+0010 0000000100004F50 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+0020 0000000100004F60 00000000 00000000 00000000 00000000 |.....|
+0030 0000000100004F70 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+0040 0000000100004F80 00000000 00000000 00000000 00000000 |.....|
+0050 0000000100004F90 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
:
:
+0130 0000000100005070 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+0140 0000000100005080 00000000 00000000 00000000 00000000 |.....|
+0150 0000000100005090 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+0160 00000001000050A0 00000000 00000000 00000000 00000000 |.....|
+0170 00000001000050B0 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+0180 00000001000050C0 00000000 00000000 00000000 00000000 |.....|
+0190 00000001000050D0 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+01A0 00000001000050E0 00000000 00000000 00000000 00000000 |.....|

```

Figure 124. Example dump using CEE3DMP (Part 7 of 10)

```

Mutex and Condition Variable Blocks (MCVB+MHT+CHT)(00000001089100B8)
+0000 00000001089100B8 00000000 00011E78 00000001 08910100 .....j..
+0010 00000001089100C8 000007F0 00007F00 00000000 00000000 ...0..."
+0020 00000001089100D8 00000001 0894A0B0 00000001 08910900 ....m.....j..
+0030 00000001089100E8 000001F0 00001F00 00000000 00000000 ...0.....
+0040 00000001089100F8 00000001 0894A0F0 00000000 00000000 ....m.0.....
+0050 0000000108910108 00000001 083018D0 00000000 00000000 .....
+0060 0000000108910118 00000001 08301990 00000000 00000000 .....

:
+0210 00000001089102C8 - +0003FF 00000001089104B7 same as above
+0400 00000001089104B8 00000001 08301BB0 00000000 00000000 .....
+0410 00000001089104C8 00000000 00000000 00000000 00000000 .....
+0420 00000001089104D8 - +00045F 0000000108910517 same as above

:
+07F0 00000001089108A8 - +00087F 0000000108910937 same as above
+0880 0000000108910938 00000001 0894A030 00000000 00000000 ....m.....
+0890 0000000108910948 00000001 0894A070 00000000 00000000 ....m.....
+08A0 0000000108910958 00000000 00000000 00000000 00000000 .....
+08B0 0000000108910968 - +0009EF 0000000108910AA7 same as above
+09F0 0000000108910AA8 00000001 0894A130 00000000 00000000 ....m.....
+0AA0 0000000108910AB8 00000000 00000000 00000000 00000000 .....
+0A10 0000000108910AC8 - +000A4F 0000000108910B07 same as above

Thread Synchronization Enclave Latch Table (EPALT)(0000000108910B00)
+0000 0000000108910B00 00000000 00000000 00000000 00000000 .....
+0010 0000000108910B10 - +00015F 0000000108910C5F same as above
+0160 0000000108910C60 00000000 00000000 DAA5CCC0 00000000 .....V.....
+0170 0000000108910C70 00000000 255A3340 00000001 08911118 .....!. ....j..
+0180 0000000108910C80 00000000 00000000 00000000 00000000 .....

:
+0640 0000000108911140 - +00092F 000000010891142F same as above
+0930 0000000108911430 00000000 00000000 DAA5CCC0 00000000 .....V.....
+0940 0000000108911440 00000000 255A3340 00000001 08910C68 .....!. ....j..
+0950 0000000108911450 00000000 00000000 00000000 00000000 .....
+0960 0000000108911460 - +0009FF 00000001089114FF same as above
+0AA0 0000000108911500 DAA5CCC0 00000000 00000000 255A3340 .v.....!.
+0A10 0000000108911510 00000001 08910D30 00000000 00000000 .....j.....
+0A20 0000000108911520 00000000 00000000 00000000 00000000 .....
+0A30 0000000108911530 - +000ADF 00000001089115DF same as above
+0AE0 00000001089115E0 00000000 251B1C80 00000000 00000000 .....
+0AF0 00000001089115F0 00000000 00000000 00000000 00000000 .....
+0B00 0000000108911600 - +0013FF 0000000108911EFF same as above

DLL Information:
WSA Addr      Module Addr      Thread ID      Use Count      Name
0000000108300050      00000000255AB000      2547D82000000000      00000001      main
0000000108301130      00000000255AB000      2547D82000000000      00000001      CELQDLL

HEAPCHK Option Control Block (HCOP)(0000000108923390)
+0000 0000000108923390 C8C3D6D7 00000048 00000001 00000000 HCOP.....
+0010 00000001089233A0 00000000 00000000 00000000 00000000 .....
+0020 00000001089233B0 00000001 089434D0 00000001 089233D8 ....m.....k.Q
+0030 00000001089233C0 00000000 00000000 00000000 00000000 .....
+0040 00000001089233D0 00000000 00000000 C8C3C6E3 00004000 .....HCFT..

HEAPCHK Element Table (HCEL) for Heapid 0000000100100138 :
Header(00000001089434D0)
+0000 00000001089434D0 C8C3C5D3 00000000 00000000 00000000 HCEL.....
+0010 00000001089434E0 00000000 00000000 00000001 00100138 .....
+0020 00000001089434F0 000001F4 00000011 00000011 00000000 ...4.....
Address      Seg Address      Length

Table(0000000108943500)
+0000 0000000108943500 00000001 08300040 00000001 08300000 00000000 00000160 00000000 00000000
+0020 0000000108943520 00000001 083001A0 00000001 08300000 00000000 00000220 00000000 00000000
+0040 0000000108943540 00000001 083003C0 00000001 08300000 00000000 00000840 00000000 00000000
+0060 0000000108943560 00000001 08300C00 00000001 08300000 00000000 000004A0 00000000 00000000
+0080 0000000108943580 00000001 083010A0 00000001 08300000 00000000 00000080 00000000 00000000
+00A0 00000001089435A0 00000001 08301120 00000001 08300000 00000000 00000060 00000000 00000000
+00C0 00000001089435C0 00000001 08301180 00000001 08300000 00000000 00000080 00000000 00000000
+00E0 00000001089435E0 00000001 08301200 00000001 08300000 00000000 00000800 00000000 00000000
+0100 0000000108943600 00000001 08301A00 00000001 08300000 00000000 00000080 00000000 00000000
+0120 0000000108943620 00000001 08301A80 00000001 08300000 00000000 00000080 00000000 00000000
+0140 0000000108943640 00000001 08301B00 00000001 08300000 00000000 00000080 00000000 00000000
+0160 0000000108943660 00000001 08301B80 00000001 08300000 00000000 00000020 00000000 00000000
+0180 0000000108943680 00000001 08301BA0 00000001 08300000 00000000 00000080 00000000 00000000
+01A0 00000001089436A0 00000001 08301C20 00000001 08300000 00000000 00000020 00000000 00000000
+01C0 00000001089436C0 00000001 08301C40 00000001 08300000 00000000 00000080 00000000 00000000
+01E0 00000001089436E0 00000001 08301CC0 00000001 08300000 00000000 00000020 00000000 00000000
+0200 0000000108943700 00000001 08301CE0 00000001 08300000 00000000 00000080 00000000 00000000

```

Figure 124. Example dump using CEE3DMP (Part 8 of 10)

Language Environment Trace Table:

Most recent trace entry is at displacement: 002080

Displacement	Trace Entry in Hexadecimal				Trace Entry in EBCDIC
+000000	Time 15.44.01.605707	Date 2004.04.20	Thread ID...	2547D82000000000	
+000010	Member ID.... 03	Flags..... 000000	Entry Type.....	00000005	
+000018	94818995 40404040 40404040 40404040	40404040 40404040 40404040 40404040			main
+000038	60606E4D F0F0F6C6 5D409799 8995A386	4D5D4040 40404040 40404040 40404040			-->(006F) printf()
+000058	40404040 40404040 40404040 40404040	40404040 40404040 40404040 40404040			
+000078	40404040 40404040				
+000080	Time 15.44.01.759312	Date 2004.04.20	Thread ID...	2547D82000000000	
+000090	Member ID.... 03	Flags..... 000000	Entry Type.....	00000006	
+000098	4C60604D F0F0F6C6 5D40D9F1 7EF0F0F0	F0F0F0F0 F0F0F0F0 F0F0F0F0 F040D9F2			<--(006F) R1=0000000000000000 R2
+000088	7EF0F0F0 F0F0F0F0 F0F2F5F4 F8C5C4F7	F840D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0			=000000002548ED78 R3=0000000000
+0000D8	F0F0F0F0 C340C5D9 D9D5D67E F0F0F0F0	F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0			0000C ERRNO=00000000 ERRNO2=0000
+0000F8	F0F0F0F0 00000000				0000....
+000100	Time 15.44.01.759321	Date 2004.04.20	Thread ID...	2547D82000000000	
+000110	Member ID.... 03	Flags..... 000000	Entry Type.....	00000005	
+000118	94818995 40404040 40404040 40404040	40404040 40404040 40404040 40404040			main
+000138	60606E4D F0F1F7F0 5D408493 93939681	844D5D40 40404040 40404040 40404040			-->(0170) d11load()
+000158	40404040 40404040 40404040 40404040	40404040 40404040 40404040 40404040			
+000178	40404040 40404040				
+000180	Time 15.44.01.783560	Date 2004.04.20	Thread ID...	2547D82000000000	
+000190	Member ID.... 03	Flags..... 000000	Entry Type.....	00000006	
+000198	4C60604D F0F1F7F0 5D40D9F1 7EF0F0F0	F0F0F0F0 F1F0F8F2 C6C6F4F6 F040D9F2			<--(0170) R1=00000001082FF460 R2
+0001B8	7EF0F0F0 F0F0F0F0 F0F2F5F4 F8C5C4F7	F840D9F3 7EF0F0F0 F0F0F0F0 F1F0F8F9			=000000002548ED78 R3=00000001089
+0001D8	F4F8C6F3 F040C5D9 D9D5D67E F0F0F0F0	F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0			48F30 ERRNO=00000000 ERRNO2=0000
+0001F8	F0F0F0F0 00000000				0000....
+000200	Time 15.44.01.783566	Date 2004.04.20	Thread ID...	2547D82000000000	
+000210	Member ID.... 03	Flags..... 000000	Entry Type.....	00000005	
+000218	94818995 40404040 40404040 40404040	40404040 40404040 40404040 40404040			main
+000238	60606E4D F0F0F6C6 5D409799 8995A386	4D5D4040 40404040 40404040 40404040			-->(006F) printf()
+000258	40404040 40404040 40404040 40404040	40404040 40404040 40404040 40404040			
+000278	40404040 40404040				

[11]Run-Time Options Report:

LAST WHERE SET	OPTION
Installation default	ENVAR("")
Installation default	FILETAG(NOAUTOCVT,NOAUTOTAG)
Programmer default	HEAPCHK(ON,1,0,0,0)
Installation default	HEAPPOLLS64(OFF,8,4000,32,2000,128,700,256,350,1024,100,2048,50,3072,50,4096,50,8192,25,16384,10,32768,5,65536,5)
Installation default	HEAP64(1M,1M,KEEP,32768,32768,KEEP,4096,4096,FREE)
Installation default	INFMSGFILTER(OFF,,,))
Installation default	IOHEAP64(1M,1M,FREE,12288,8192,FREE,4096,4096,FREE)
Installation default	LIBHEAP64(1M,1M,FREE,16384,8192,FREE,8192,4096,FREE)
Installation default	NATLANG(ENU)
Invocation command	POSIX(ON)
Installation default	PROFILE(OFF,"")
Installation default	RPTOPTS(OFF)
Programmer default	RPTSTG(ON)
Installation default	STACK64(1M,1M,128M)
Installation default	STORAGE(NONE,NONE,NONE,)
Programmer default	TERMTHDACT(UADUMP,,96)
Installation default	NOTEST(ALL,"*","PROMPT","INSPREF")
Installation default	THREADTACK64(OFF,1M,1M,128M)
Programmer default	TRACE(ON,1048576,NODUMP,LE=1)
Installation default	TRAP(ON,SPIE)

Figure 124. Example dump using CEE3DMP (Part 9 of 10)

[12]Process Control Blocks:

```
PCB(0000000100002468)
+0000 0000000100002468 C3C5C5D7 C3C24040 00000000 00000000 |CEEPCB .....|
+0010 0000000100002478 00000000 00000000 00000000 00000000 |.....|
+0020 0000000100002488 - +0000FF 0000000100002567 |same as above|
+0100 0000000100002568 03030208 00000000 00000000 00000000 |.....|
+0110 0000000100002578 00000001 00002810 00000000 00000000 |.....|
+0120 0000000100002588 00000000 00000000 00000000 00000000 |.....|
+0130 0000000100002598 00000000 00000000 00000001 00002210 |.....|
+0140 00000001000025A8 7F800000 00000000 00000000 00000000 |".....|
+0150 00000001000025B8 00000000 00000000 00000000 00000000 |.....|
+0160 00000001000025C8 00000000 251EEAE0 00000000 00000000 |.....|
+0170 00000001000025D8 00000000 00000000 00000000 00000000 |.....|
+0180 00000001000025E8 - +0001BF 0000000100002627 |same as above|
MEML(0000000100002810)
+0000 0000000100002810 00000000 00000000 00000000 00000000 |.....|
+0010 0000000100002820 - +00005F 000000010000286F |same as above|
+0060 0000000100002870 00000001 00006CD0 00000000 00000000 |.....%.....|
+0070 0000000100002880 00000000 00000000 00000000 00000000 |.....|
+0080 0000000100002890 - +0001AF 00000001000029BF |same as above|
Thread Synchronization Process Latch Table (PPALT)(0000000108911F00)
+0000 0000000108911F00 DAA5CCC0 00000000 00000000 255A3340 |.v.....!.|
+0010 0000000108911F10 00000001 08911050 00000000 00000000 |....j.&.....|
+0020 0000000108911F20 00000000 00000000 00000000 00000000 |.....|
+0030 0000000108911F30 - +00009F 0000000108911F9F |same as above|
+00A0 0000000108911FA0 DAA5CCC0 00000000 00000000 255A3340 |.v.....!.|
+00B0 0000000108911FB0 00000001 08911500 00000000 00000000 |....j.....|
+00C0 0000000108911FC0 00000000 00000000 00000000 00000000 |.....|
+00D0 0000000108911FD0 - +0013FF 00000001089132FF |same as above|
```

Figure 124. Example dump using CEE3DMP (Part 10 of 10)

Sections of the Language Environment dump

The sections of the dump listed here appear independently of the Language Environment-conforming languages used.

[1] Page Heading

The page heading section appears on the top of each page of the dump and contains:

- CEE3DMP identifier
- Title

For dumps generated as a result of an unhandled condition, the title is “Condition processing resulted in the Unhandled condition.”

- Product abbreviation of Language Environment
- Version number
- Release number
- Date
- Time
- Page number

[2] Enclave Information

These sections show information that is specific to an enclave.

[2] Enclave Identifier

This statement names the enclave for which information in the dump is provided.

[3] - [9] Thread Information

These sections show information that is specific to a thread. When multiple threads are dumped, these sections will appear for each thread.

[3] Information for thread

This section shows the system identifier for the thread. Each thread has a unique identifier.

[4] Traceback

For all active routines in a particular thread, the traceback section shows routine information in two part. The first part contains:

- DSA number
A number assigned to the information for this active routine by dump processing. The number is only used to associate information from the first part of the traceback with information in the second part of the traceback.
- Entry
For C/C++ routines, this is the function name. If a function name or entry point was not specified for a particular routine, then the string `'** NoName **'` will appear.
- Entry point offset
- Load module
- Program unit
The primary entry point of the external procedure. For C routines, this is the compile unit name. For Language Environment-conforming assemblers, this is either the ENTNAME = value on the CELQPRLG macro.
- Service level
The latest service level applied to the compile unit (for example, for IBM products, it would be the PTF number).
- Status
Routine status can be call or exception.

The second part contains:

- DSA number
A number assigned to the information for this active routine by dump processing. The number is only used to associate information from the first part of the traceback with information in the second part of the traceback.
- Stack frame (DSA) address
- Entry point address
- Program unit address
- Program unit offset
The offset of the last instruction to run in the routine. If the offset is a negative number, zero, or a very large positive number, the routine associated with the offset probably did not allocate a save area or the routine could have been called using SVC-assisted linkage. Adding the program unit address to the offset gives you the location of the current instruction in the routine. This offset is from the starting address of the routine.
- Compile Date
- Attributes

The attributes of the compile unit including whether character data is being treated as EBCDIC or ASCII and whether floating point data is being treated as IEEE or hexadecimal.

[5] Condition Information for Active Routines

This section displays the following information for all conditions currently active on the call chain:

- Statement showing failing routine and stack frame address of routine
- Condition information block (CIB) address
- Current condition, in the form of a Language Environment message for the condition raised or a Language Environment abend code, if the condition was caused by an abend
- Location
 - For the failing routine, this is the program unit, entry routine, and offset.
- Machine state, which shows:
 - Instruction length counter (ILC)
 - Interruption code
 - *Program status word (PSW)*
 - Contents of GPRs 0–15. Contents of floating point content register (FPC) and floating point registers FPR 0-15.
 - Storage dump near condition (2 hex-bytes of storage near the PSW)

These values are the current values at the time the condition was raised.

[6] Parameters, Registers, and Variables for Active Routines

For each active routine, this section shows:

- Routine name and stack frame address
- Saved registers
 - This lists the contents of GPRs 0–15 at the time the routine received control. The saved registers are those saved by the DSA-owning routine on entry. Register 7 is the return address back to the caller of the DSA-owning routine. Register 6 may be the entry point of the DSA-owning routine. (This is not true when the Branch Relative and Save instruction is used to implement the call. The non-volatile floating-point registers that are saved in the stack frame. The registers are only displayed if the program owning the stack frame saved them. Dashes are displayed in the registers when the register values are not saved.
- Storage pointed to by the saved registers
 - Treating the saved contents of each register as an address, 32 bytes before and 64 bytes after the address shown.

[7] Control Blocks for Active Routines

For each active routine controlled by the STACKFRAME option, this section lists contents of related control blocks. The Language Environment-conforming language determines which language-specific control blocks appear. The possible control blocks are:

- Stack frame
- Condition information block
- Language-specific control blocks

[8] Storage for Active Routines

This displays local storage for each active routine. The storage is dumped in hexadecimal, with EBCDIC translations on the right side of the page. There can be other information, depending on the language used. For C/C++ routines, this is the stack frame storage.

[9] Control Blocks Associated with the Thread

This section lists the contents of the Language Environment common anchor area (CAA), thread synchronization queue element (SQEL) and dummy stack frame. Other language-specific control blocks can appear in this section.

[10] Enclave Control Blocks

This section lists the contents of the Language Environment enclave data block (EDB) and enclave member list (MEML). The information presented may vary depending on which run-time options are set.

- If the POSIX run-time option is set to ON, this section lists the contents of the mutex and condition variable control blocks, the enclave level latch table, and the thread synchronization trace block and trace table.
- If DLLs have been loaded, this section shows information for each DLL including the DLL name, load address, use count, writeable static area (WSA) address, and the thread id of the thread that loaded the DLL.
- If the HEAPCHK run-time option is set to ON, this section shows the contents of the HEAPCHK options control block (HCOP) and the HEAPCHK element tables (HCEL). A HEAPCHK element table contains the location and length of all allocated storage elements for a heap in the order that they were allocated.
- When the *call-level* suboption of the HEAPCHK run-time option is set, any unfreed storage, which would indicate a storage leak, would be displayed in this area. The traceback could then be used to identify the program which did not free the storage.
- If the TRACE run-time option is set to ON, this section shows the contents of the Language Environment trace table.

Other language-specific control blocks can appear in this section.

[11] Run-Time Options Report

This section lists the Language Environment run-time options in effect when the routine was executed.

[12] Process Control Blocks

This section lists the contents for the Language Environment process control block (PCB), process member list (MEML), and if the POSIX run-time option is set to ON, the process level latch table. Other language-specific control blocks can appear in this section.

Generating a system dump

A system dump contains the storage information needed to diagnose errors. You can use Language Environment to generate a system dump through any of the following methods:

TERMTHDACT(UAONLY, UATRACE, or UADUMP)

You can use these run-time options, with TRAP(ON), to generate a system dump if an

unhandled condition of severity 2 or greater occurs. For further details regarding the level of dump information produced by each of the TERMTHDACT suboptions, see “Generating a Language Environment dump with TERMTHDACT” on page 287.

TRAP(ON,NOSPIE) TERMTHDACT(UAImm)

TRAP(ON,NOSPIE) TERMTHDACT(UAImm) generates a system dump of the user address space of the original abend or program interrupt prior to the Language Environment condition manager processing the condition.

Abend Codes in Initialization Assembler User Exit

Abend codes listed in the initialization assembler user exit are passed to the operating system. The operating system can then generate a system dump.

`__cabend()`

You can use the `__cabend()` API to cause the operating system to handle an abend.

Refer to system or subsystem documentation for detailed system dump information.

The method for generating a system dump varies for each of the Language Environment run-time environments. The following sections describe the recommended steps needed to generate a system dump in batch and z/OS UNIX shell run-time environments. Other methods may exist, but these are the recommended steps for generating a system dump.

For details on setting Language Environment run-time options, see *z/OS Language Environment Programming Guide*.

Steps for Generating a system dump in a z/OS UNIX shell

Perform the following steps to generate a system dump from a z/OS UNIX shell:

1. Specify where to write the system dump
 - To write the system dump to a z/OS data set, issue the command:
`export _BPXK_MDUMP=filename`

where *filename* is a fully qualified data set name with DCB information: LRECL=4160, BLKSIZE=4160, and RECFM=FBS.

Example:

```
export _BPXK_MDUMP=h1q.mydump
```

- To write the system dump to an HFS file, issue the command:
`export _BPXK_MDUMP=filename`

where *filename* is a fully qualified HFS filename.

Example:

```
export _BPXK_MDUMP=/tmp/mydump.dmp
```

2. Specify Language Environment run-time options:

```
export _CEE_RUNOPTS="termthdact(suboption)"
```

where *suboption* = UAONLY, UADUMP, UATRACE, or UAImm. If UAImm is set, TRAP(ON,NOSPIE) must also be set. The TERMTHDACT suboption determines

the level of detail of the Language Environment formatted dump. For further details regarding the TERMTHDACT suboptions, see “Generating a Language Environment dump with TERMTHDACT” on page 287.

3. Rerun the program.

When you are done, the system dump is written to the data set name or HFS file name specified.

For additional BPXK_MDUMP information see *z/OS UNIX System Services Command Reference*.

Note: You can also specify the signal SIGDUMP on the kill command to generate a system dump of the user address space. For more information regarding the SIGDUMP signal, see *z/OS UNIX System Services Command Reference*.

Formatting and analyzing system dumps

You can use the Interactive Problem Control System (IPCS) to format and analyze system dumps. Language Environment provides an IPCS Verbexit LEDATA that can be used to format Language Environment control blocks.

For more information on using IPCS, refer to *z/OS MVS IPCS User's Guide*.

Preparing to use the Language Environment support for IPCS

Guidelines: Use the following guidelines before you use IPCS to format Language Environment control blocks:

- Ensure that your IPCS job can find the CEEIPCSP member.

IPCS provides an exit control table with imbed statements to enable other products to supply exit control information. The IPCS default table, BLSCECT, normally in the SYS1.PARMLIB library, has the following entry for Language Environment:

```
IMBED MEMBER(CEEIPCSP) ENVIRONMENT(IPCS)
```

The Language Environment-supplied CEEIPCSP member, installed in the SYS1.PARMLIB library, contains the Language Environment-specific entries for the IPCS exit control table.

- Provide an IPCSPARM DD statement to specify the libraries containing the IPCS control tables.

Example:

```
//IPCSPARM DD DSN=SYS1.PARMLIB,DISP=SHR
```

- Ensure that your IPCS job can find the Language Environment-supplied ANALYZE exit routines installed in the SYS1.MIGLIB library.
- To aid in debugging system or address space hang situations, Language Environment mutexes, latches and condition variables can be displayed if the CEEIPSCP member you are using is updated to identify the Language Environment ANALYZE exit, by including the following statement:

```
EXIT EP(CEEEANLZ) ANALYZE
```

Language Environment IPCS Verbexit – LEDATA

Use the LEDATA Verbexit to format data for Language Environment. This Verbexit provides information about the following topics:

- A summary of Language Environment at the time of the dump
- Run-time Options

- Storage Management Control Blocks
- Condition Management Control Blocks
- Message Handler Control Blocks
- C/C++ Control Blocks

Format

Syntax

VERBEXIT LEDATA ['parameter[,parameter]...']

- Report Type Parameters:
 - [**AUTH**]
 - [**NTHREADS**(*value*)]
 - [**SUM**]
 - [**HEAP** | **STACK** | **SM**]
 - [**HPT**(*value*)]
 - [**CM**]
 - [**MH**]
 - [**CEEDUMP**]
 - [**ALL**]
- Data Selection Parameters:
 - [**DETAIL** | **EXCEPTION**]
- Control Block Selection Parameters:
 - [**CAA**(*caa-address*)]
 - [**DSA**(*dsa-address*)]
 - [**TCB**(*tcb-address*)]
 - [**ASID**(*address-space-id*)]
 - [**LAA**(*laa-address*)]

Parameters

Report type parameters

Use these parameters to select the type of report. If you omit these parameters, the default is SUMMARY.

Address space report types: Use these parameters to select a report that shows the Language Environment activity for an address space. Only one of these reports may be specified.

NTHREADS(*value*)

Requests a report that shows the traceback for the TCBs in the address space. *value* is the number of TCBs for which the traceback will be displayed. If *value* is specified as asterisk (*), all TCBs will be displayed. The LAA, CAA, or TCB parameter can be used to limit the display to only TCBs that are part of the same enclave.

AUTH

Requests a report on all Preinitialized Environments for Authorized Programs control blocks for the address space. NTHREADS is ignored when AUTH is specified.

Thread specific report types: Use these parameters to select reports that show Language Environment activity for a specific TCB. These report types are ignored if AUTH or NTHREADS is specified. You can specify as many of these reports as you wish.

SUMmary

Requests a summary of the Language Environment at the time of the dump. The following information is included:

- TCB address
- Address Space Identifier
- Language Environment Release
- Active members
- Formatted CAA, PCB, RCB, EDB, LAA and LCA
- Run-time Options in effect

HEAP | STACK | SM

HEAP

Requests a report on Storage Management control blocks pertaining to HEAP storage, as well as a detailed report on heap segments. The detailed report includes information about the free storage tree in the heap segment, and information about each allocated storage element.

Note: Language Environment does not provide support for alternative Vendor Heap Manager (VHM) data.

STACK

Requests a report on Storage Management control blocks pertaining to STACK storage.

SM

Requests a report on Storage Management control blocks. This is the same as specifying both HEAP and STACK.

HPT(value)

Requests that the heappools trace (if available) be formatted. If the value is 0 or *, the trace for every heappools poolid is formatted. If the value is a single number (1-12) the trace for the specific heappools poolid is formatted.

CM

Requests a report on Condition Management control blocks.

MH

Requests a report on Message Handler control blocks.

CEEDump

Requests a CEEDUMP-like report. Currently this includes the traceback, the Language Environment trace, and thread synchronization control blocks at process, enclave and thread levels.

ALL

Requests all above reports, as well as C/C++ reports.

Data selection parameters

Data selection parameters limit the scope of the data in the report. If no data selection parameter is selected, the default is DETAIL.

DETail

Requests formatting all control blocks for the selected components. Only significant fields in each control block are formatted.

Note: For the Heap and Storage Management Reports, the DETAIL parameter will provide a detailed heap segment report for each heap segment in the dump. The detailed heap segment report includes information on the free storage tree in the heap segments, and all allocated storage elements. This report will also identify problems detected in the heap

management data structures. For more information about the Heap Reports, see “Understanding the HEAP LEDATA output” on page 326.

EXception

Requests validating all control blocks for the selected components. Output is only produced naming the control block and its address for the first control block in a chain that is invalid. Validation consists of control block header verification at the very least.

Note: For the Summary, CEEDUMP, C/C++ reports, the EXCEPTION parameter has not been implemented. For these reports, DETAIL output is always produced.

Control block selection parameters

Use these parameters to select the control blocks used as the starting points for formatting.

CAA(caa-address)

specifies the address of the CAA. If not specified, the CAA address is obtained from the LAA.

DSA(dsa-address)

specifies the address of the DSA. If not specified, the DSA address may be obtained from the TCB or the IPCS symbol REGGEN.

TCB(tcb-address)

specifies the address of the TCB. If not specified, the TCB address may be obtained from the CAA or the CVT.

LAA(laa-address)

specifies the address of the LAA. If not specified, the LAA address may be obtained from the TCB or the PSA.

ASID(address-space-id)

specifies the hexadecimal address space id. If not specified, the IPCS default address space id is used. This parameter is not needed when the dump only has one address space.

Understanding the Language Environment IPCS Verbexit LEDATA output

The Language Environment IPCS Verbexit LEDATA generates formatted output of the Language Environment run-time environment control blocks from a system dump. Figure 125 on page 312 illustrates the output produced when the LEDATA Verbexit is invoked with the ALL parameter. The system dump being formatted was obtained by specifying the TERMTHDACT(UADUMP) run-time option when running the program CELQSAMP in Figure 122 on page 291. “Sections of the Language Environment LEDATA Verbexit formatted output” on page 322 describes the information contained in the formatted output. Ellipses are used to summarize some sections of the dump.

For easy reference, the sections of the following dump are numbered to correspond with the descriptions in “Sections of the Language Environment LEDATA Verbexit formatted output” on page 322.

```

IP VERBEXIT LEDATA 'ALL'
*****
64 BIT LANGUAGE ENVIRONMENT DATA
*****

Language Environment Product 04 V01 R06.00

[1] Information for enclave main

[2] Information for thread 254814600000001
TCB Address: 008DD8D8
CAA Address: 0000000100006160
PCB Address: 0000000100002468

[3] Registers and PSW:
GPR0..... 0000000084000000 GPR1..... 0000000084000FC7 GPR2..... 00000001082FBD0F GPR3..... 3FF6A09E667F3BCD
GPR4..... 00000001082FA8C0 GPR5..... 00000000251A7488 GPR6..... 00000000251A7390 GPR7..... 00000000251A1140
GPR8..... 00000001082FBD0F GPR9..... 00000000251A742A GPR10..... 00000001082FBABF GPR11..... 00000001082FBB08
GPR12..... 00000001082FBB08 GPR13..... 00000001082FE680 GPR14..... 00000000251A1040 GPR15..... 0000000000000000
PSW..... 078D1401 A51A742A

[4] Traceback:
DSA      Entry      E Offset  Load Mod  Program Unit  Service  Status
00000001 CEEHSDMP +0000009A SUBPOOL2 CEEHSDMP      Call
00000002 CEEHSDSP +0000369E SUBPOOL2 CEEHSDSP      Call
00000003 CEEOSIGJ +00000956 SUBPOOL2 CEEOSIGJ      Call
00000004 CELQHROD +00000256 SUBPOOL2 CELQHROD      Call
00000005 CEEOSIGG +00000000 SUBPOOL2 CEEOSIGG      Call
00000006 CELQHROD +00000256 SUBPOOL2 CELQHROD      Call
00000007 div_zero +00000030 SUBPOOL2      Exception
00000008 main +000002DC SUBPOOL2      Call
00000009 CELQINIT +00001146 SUBPOOL2 CELQINIT      Call

DSA      DSA Addr      E Addr      PU Addr      PU Offset
00000001 00000001082FA8C0 00000000251A7390 00000000251A7390 +0000009A
00000002 00000001082FAAC0 000000002519DAA0 000000002519DAA0 +0000369E
00000003 00000001082FD3E0 00000000252272D0 00000000252272D0 +00000956
00000004 00000001082FDDE0 000000002513B6C8 000000002513B6C8 +00000256
00000005 00000001082FE020 00000000252209A8 00000000252209A8 +00000000
00000006 00000001082FEE40 000000002513B6C8 000000002513B6C8 +00000256
00000007 00000001082FF080 00000000255CE660 0000000000000000 +255CE690
00000008 00000001082FF180 00000000251010C8 0000000000000000 +251013A4
00000009 00000001082FF280 0000000025127010 0000000025127010 +00001146

[5] Control Blocks Associated with the Thread:
Thread Synchronization Queue Element (SQEL): 00000000255C7340
+000000 00000000255C7340 00000000 00000000 00000000 00000000 .....
+000010 00000000255C7350 00000000 00000000 00000001 08301190 .....
+000020 00000000255C7360 00000000 00000000 00000000 00000000 .....
+000030 00000000255C7370 00000000 00000000 00000001 00006160 ...../-
+000040 00000000255C7380 00000000 00000000 00000000 00000000 .....
+000050 00000000255C7390 - +000000 00000000255C73AF same as above

[6] Enclave Control Blocks:
Mutex and Condition Variable Blocks (MCVB+MHT+CHT): 00000001089100B8
+000000 00000001089100B8 00000000 00011E78 00000001 08910100 .....j..
+000010 00000001089100C8 000007F0 00007F00 00000000 00000000 ...0.."
+000020 00000001089100D8 00000001 0894A090 00000001 08910900 ...m.....j..
+000030 00000001089100E8 000001F0 00001F00 00000000 00000000 ...0.....
+000040 00000001089100F8 00000001 0894A0D0 00000000 00000000 ...m.....
+000050 0000000108910108 00000001 08301A10 00000000 00000000 .....
+000060 0000000108910118 00000000 00000000 00000000 00000000 .....
+000070 0000000108910128 00000001 08301A90 00000000 00000000 .....
+000080 0000000108910138 00000000 00000000 00000000 00000000 .....
+000090 0000000108910148 - +000000 0000000108910187 same as above

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 1 of 11)

```

+0000D0 0000000108910188 00000001 08301450 00000000 00000000 | .....&.....|
+0000E0 0000000108910198 00000001 08301510 00000000 00000000 | .....|
+0000F0 00000001089101A8 00000001 083015D0 00000000 00000000 | .....|
+000100 00000001089101B8 00000001 08301690 00000000 00000000 | .....|
+000110 00000001089101C8 00000001 08301750 00000000 00000000 | .....&.....|
+000120 00000001089101D8 00000001 08301810 00000000 00000000 | .....|
+000130 00000001089101E8 00000001 083018D0 00000000 00000000 | .....|
+000140 00000001089101F8 00000001 08301990 00000000 00000000 | .....|
+000150 0000000108910208 00000000 00000000 00000000 00000000 | .....|
+000160 0000000108910218 - +000000 00000001089102A7 same as above
+0001F0 00000001089102A8 00000001 08301190 00000000 00000000 | .....|
+000200 00000001089102B8 00000000 00000000 00000000 00000000 | .....|
+000210 00000001089102C8 - +000000 00000001089104B7 same as above
+000400 0000000108910488 00000001 08301BB0 00000000 00000000 | .....|
+000410 00000001089104C8 00000000 00000000 00000000 00000000 | .....|
+000420 00000001089104D8 - +000000 0000000108910517 same as above
+000460 0000000108910518 00000001 08301C50 00000000 00000000 | .....&.....|
+000470 0000000108910528 00000000 00000000 00000000 00000000 | .....|
+000480 0000000108910538 - +000000 0000000108910857 same as above
+0007A0 0000000108910858 00000001 08301210 00000000 00000000 | .....|
+0007B0 0000000108910868 00000001 083012D0 00000000 00000000 | .....|
+0007C0 0000000108910878 00000001 08301390 00000000 00000000 | .....|
+0007D0 0000000108910888 00000001 08301B10 00000000 00000000 | .....|
+0007E0 0000000108910898 00000000 00000000 00000000 00000000 | .....|
+0007F0 00000001089108A8 - +000000 0000000108910917 same as above
+000860 0000000108910918 00000001 0894A010 00000000 00000000 | ....m.....|
+000870 0000000108910928 00000001 0894A050 00000000 00000000 | ....m.&.....|
+000880 0000000108910938 00000000 00000000 00000000 00000000 | .....|
+000890 0000000108910948 - +000000 0000000108910AA7 same as above
+0009F0 0000000108910AA8 00000001 0894A110 00000000 00000000 | ....m.....|
+000A00 0000000108910AB8 00000000 00000000 00000000 00000000 | .....|
+000A10 0000000108910AC8 - +000000 0000000108910B07 same as above
Thread Synchronization Enclave Latch Table (EPALT): 0000000108910B00
+000000 0000000108910B00 00000000 00000000 00000000 00000000 | .....|
+000010 0000000108910B10 - +000000 00000001089115DF same as above
+000AE0 00000001089115E0 00000000 251D57E8 00000000 00000000 | .....Y.....|
+000AF0 00000001089115F0 00000000 00000000 00000000 00000000 | .....|
+000B00 0000000108911600 - +000000 0000000108911EFF same as above

HEAPCHK Option Control Block (HCOP): 0000000108923390
+000000 0000000108923390 C8C3D6D7 00000048 00000001 00000000 | HCOP.....|
+000010 00000001089233A0 00000000 00000000 00000000 00000000 | .....|
+000020 00000001089233B0 00000001 089434D0 00000001 089233D8 | ....m.....k.Q
+000030 00000001089233C0 00000000 00000000 00000000 00000000 | .....|
+000040 00000001089233D0 00000000 00000000 C8C3C6E3 00004000 | .....HCFT..|

HEAPCHK Element Table (HCEL) for Heapid 00000001 :
Header: 00000001089434D0
+000000 00000001089434D0 C8C3C5D3 00000000 00000000 00000000 | HCEL.....|
+000010 00000001089434E0 00000000 00000000 00000001 00100138 | .....|
+000020 00000001089434F0 000001F4 0000000F 00000000F 00000000 | ...4.....|
      Address      Seg Addr      Length
Table: 0000000108943500
+000000 00000001 08300040 00000001 08300000 00000000 00000160 00000000 00000000 | .....-.....|
+000020 00000001 083001A0 00000001 08300000 00000000 00000220 00000000 00000000 | .....|
+000040 00000001 083003C0 00000001 08300000 00000000 00000840 00000000 00000000 | .....|
+000060 00000001 08300C00 00000001 08300000 00000000 000004A0 00000000 00000000 | .....|
+000080 00000001 083010A0 00000001 08300000 00000000 00000080 00000000 00000000 | .....|
+0000A0 00000001 08301120 00000001 08300000 00000000 00000060 00000000 00000000 | .....|
+0000C0 00000001 08301180 00000001 08300000 00000000 00000080 00000000 00000000 | .....|
+0000E0 00000001 08301200 00000001 08300000 00000000 00000800 00000000 00000000 | .....|
+000100 00000001 08301A00 00000001 08300000 00000000 00000080 00000000 00000000 | .....|

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 2 of 11)

```

+000120 00000001 08301A80 00000001 08300000 00000000 00000080 00000000 00000000 .....
+000140 00000001 08301B00 00000001 08300000 00000000 00000080 00000000 00000000 .....
+000160 00000001 08301B80 00000001 08300000 00000000 00000020 00000000 00000000 .....
+000180 00000001 08301BA0 00000001 08300000 00000000 00000080 00000000 00000000 .....
+0001A0 00000001 08301C20 00000001 08300000 00000000 00000020 00000000 00000000 .....
+0001C0 00000001 08301C40 00000001 08300000 00000000 00000080 00000000 00000000 .....

```

[7] Language Environment Trace Table:

Most recent trace entry is at displacement: 002080

Displacement	Trace Entry in Hexadecimal	Trace Entry in EBCDIC
+000000	Time 22.02.30.389659 Date 2004.04.08 Thread ID... 2548146000000001	
+000010	Member ID... 03 Flags..... 000000 Entry Type..... 00000005	
+000018	94818995 40404040 40404040 40404040 40404040 40404040 40404040 40404040	main
+000038	60606E4D F0F0F6C6 5D409799 8995A386 4D5D4040 40404040 40404040 40404040	-->(006F) printf()
+000058	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000078	40404040 40404040	
+000080	Time 22.02.30.389724 Date 2004.04.08 Thread ID... 2548146000000001	
+000090	Member ID... 03 Flags..... 000000 Entry Type..... 00000006	
+000098	4C60604D F0F0F6C6 5D40D9F1 7EF0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0 F040D9F2	<--(006F) R1=0000000000000000 R2
+0000B8	7EF0F0F0 F0F0F0F0 F0F2F5F4 C2F2F8C5 F040D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0	=00000000254828E0 R3=000000000000
+0000D8	F0F0F0F0 C340C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	0000C ERRNO=00000000 ERRNO2=0000
+0000F8	F0F0F0F0 00000000	0000....
+000100	Time 22.02.30.389725 Date 2004.04.08 Thread ID... 2548146000000001	
+000110	Member ID... 03 Flags..... 000000 Entry Type..... 00000005	
+000118	94818995 40404040 40404040 40404040 40404040 40404040 40404040 40404040	main
+000138	60606E4D F0F1F7F0 5D408493 93939681 844D5D40 40404040 40404040 40404040	-->(0170) dllload()
+000158	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000178	40404040 40404040	
+000180	Time 22.02.30.409904 Date 2004.04.08 Thread ID... 2548146000000001	
+000190	Member ID... 03 Flags..... 000000 Entry Type..... 00000006	
+000198	4C60604D F0F1F7F0 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F8F2 C6C6F4F6 F040D9F2	<--(0170) R1=00000001082FF460 R2
+0001B8	7EF0F0F0 F0F0F0F0 F0F2F5F4 C2F2F8C5 F040D9F3 7EF0F0F0 F0F0F0F0 F1F0F8F9	=00000000254828E0 R3=00000001089
+0001D8	F4F8C6F1 F040C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	48F10 ERRNO=00000000 ERRNO2=0000
+0001F8	F0F0F0F0 00000000	0000....
+000200	Time 22.02.30.409906 Date 2004.04.08 Thread ID... 2548146000000001	
+000210	Member ID... 03 Flags..... 000000 Entry Type..... 00000005	
+000218	94818995 40404040 40404040 40404040 40404040 40404040 40404040 40404040	main
+000238	60606E4D F0F0F6C6 5D409799 8995A386 4D5D4040 40404040 40404040 40404040	-->(006F) printf()
+000258	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000278	40404040 40404040	
+000280	Time 22.02.30.409938 Date 2004.04.08 Thread ID... 2548146000000001	
+000290	Member ID... 03 Flags..... 000000 Entry Type..... 00000006	
+000298	4C60604D F0F0F6C6 5D40D9F1 7EF0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0 F040D9F2	<--(006F) R1=0000000000000000 R2
+0002B8	7EF0F0F0 F0F0F0F0 F0F2F5F4 C2F2F8C5 F040D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0	=00000000254828E0 R3=000000000000
+0002D8	F0F0F0F0 C440C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	0000D ERRNO=00000000 ERRNO2=0000
+0002F8	F0F0F0F0 00000000	0000....
+000300	Time 22.02.30.409939 Date 2004.04.08 Thread ID... 2548146000000001	
+000310	Member ID... 03 Flags..... 000000 Entry Type..... 00000005	
+000318	94818995 40404040 40404040 40404040 40404040 40404040 40404040 40404040	main
+000338	60606E4D F0F1F6C4 5D408493 9398A485 99A88695 4D5D4040 40404040 40404040	-->(016D) dllqueryfn()
+000358	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000378	40404040 40404040	

:

[8] Process Control Blocks:

Thread Synchronization Process Latch Table (PPALT): 0000000108911F00

```

+000000 0000000108911F00 00000000 00000000 00000000 00000000 .....
+000010 0000000108911F10 - +000000 00000001089132FF same as above

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 3 of 11)

```

[9] TCB: 008DD8D8          LE Level: 12          ASID: 0021

[10] Active Members: C/C++

[11] CEELAA: 000000007F786848
+000000 EYE_CATCHER:LAA VERSION:00000001 PREVIOUS:00000000
+00000C NEXT:00000000 STCB:7F786430 FLOOR31:FFBAD000
+00001C OVERFLOW31:00000000 GTAB31:00000000 LCA31:00000000
+000028 TRT31:00000000 FLOOR64:00000001 08200000
+000048 OVERFLOW64:00000000 25144A70 GTAB64:00000001 0871AF80
+000058 LCA64:00000001 00000000 TRT64:00000001 00000030
+0000C8 FLAG1:A1 FLAG2:00 CB_STG64:00000001 00000000
+0000F8 CB_STG31:00000000 255C7340 SvcVec31:00000000
+000104 SANC31:00000000 CurKey31:00 SvcVec64:00000000 00000000
+000120 SANC64:00000001 00100000 CurKey64:80
+000130 ENSQ64:00000001 00100108 LHEAP64ID:00000001 001002A8
+000140 LHEAP31ID:00000000 00000000 IPTLAA:7F786848
+00014C MSTRLAA:00000000

[12] CEELCA: 0000000100000000
+000000 EYE_CATCHER:LCA VERSION:00000001 CAA:00000001 00006160
+000010 DIA:255C75C0 LAA:7F786848 SHUNT:00000000 00000000
+000028 QINIT:00000000 25127010 TLC:00000000 2517F440

[13] CEECAA: 0000000100006160
+0002B2 INVAR:8000 FLAG0:00 TORC:00000000 FLAG2:30 LEVEL:12
+000365 _PM:04 DMC:00000000 00000000 CD:00000000
+000374 RS:00000000 ERR:00000001 082FBDF0
+000380 DDSA:00000001 082FF760 EDB:00000001 000039D0
+000390 PCB:00000001 00002468 EYEPTR:00000001 00006148
+0003A0 CAA:00000001 00006160 SHAB:00000000 00000000
+0003B0 PRGCK:00000000 00000000 URC:00000000
+0003C0 PICICB:00000000 00000000 SIGSCTR:00000000
+0003CC SIGSFLG:08000000 THDID:25481460 00000001
+0003D8 RCB:00000001 00002210 MEMBR:00000001 00006B18
+0003E8 SIGNAL_STATUS:00000000 00000008
+0003F0 HCOM_REG14:00000000 00000000 EDCHPXV:00000000 25175FF0
+000400 THREADHEAPID:00000000 00000000 SYS_RTNCODE:00000000
+00040C SYS_RSNCODE:00000000 SIGNGPTR:00000001 00006578
+000418 SIGNG:00000001 AB_STATUS:00 STACKDIRECTION:00
+000420 AB_GRP:00000000 00000000 AB_ICD1:00000000 00000000
+000430 AB_ABCC:00000000 00000000 AB_CRC:00000000 00000000
+000440 USERRTN:00000000 00000000 QINITDSA:00000001 082FF280
+0004E8 IFLAG:0008 TRMRSN:00 DEVH:00000000 00000000
+0004F8 PtatPtr:00000000 00000000 SQLADDR:00000000 255C7340
+000510 VBA:00000000 00000000 TCS:00000000 00000000
+000520 CONDWAITDSAREG:00000000 00000000 THDSTATUS:00000000 00000000
+000570 FBTK:00000000 00000000 00000000 00000000
+000580 PTXLPTR:00000000 00000000 TICB_PTR:00000001 000050F8
+000598 FWD_CHAIN:00000001 10E013C8 BKWD_CHAIN:00000001 193013C8
+0007E8 TCB:008DD8D8

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 4 of 11)

```

[14] CEEPCB: 0000000100002468
+000000 EYE:CEPCB          SYSTM:03   HRDWR:03   SBSYS:03   FLAG2:08
+000108 DBGHE:00000000 00000000   DMBR:00000001 00002810
+000118 ZL0D:00000000 00000000   ZDEL:00000000 00000000
+000128 ZGETST:00000000 00000000   ZFREEST:00000000 00000000
+000138 RCB:00000001 00002210   OMVS_LEVEL:7F800000
+000148 CHAIN:00000000 00000000   PRFEH:00000000 00000000
+000158 FLAG6:00   QFEXIT:00000000 25212648   ABENDCODE:00000000
+0001C4 REASON:00000000   F3456:00000040   MEML:00000001 000027E8
+0001D8 MEMBR:00000001 00002810   LEHL:00000001 00001560
+0001E8 CMXB:00000001 00001730   STV:02   PM_BYTE:00   FLAG1:00
+0001F8 ISA:00000001 00000000   ISA_SIZE:0001AF80
+000204 SRV_COUNT:00000000   SRV_UWORD:00000000 00000000
+000210 WORKAR:00000000 00000000   LOAD:00000000 255A8810
+000220 DELETE:00000000 255A8800   GETSTOR:00000000 00000000
+000230 FREESTOR:00000000 00000000   EXCEPT:00000000 00000000
+000240 ATTN:00000000 00000000   MSGS:00000000 00000000
+000250 ABEND:00000000 00000000   MSGOU:00000000 00000000
+000260 GLAT:00000000 255A87F0   RLAT:00000000 255A87E0
+000270 ELAT:00000000 255A87D0   IPTQ:00000000 255A87C0
+000280 IENV:00000000 255A87B0   LODTYP:FFFFFFFF   LEVEL:FFFFFFFF
+000290 LLTPTR:00000001 00000738   RC:00000000   REASON:00000000
+0002A0 RC_MOD:00000000   FBTOK:00000000 00000000 00000000 00000000
+0002B8 PPALTADDR:00000001 08911F00   PPALTSIZE:00001400
+0002C8 PICB:00000001 00002448   XPLINKFLAGS:80
+0002D8 QICB:00000001 0000ECD0   CALLERS_SAVE:00000000 0000CF68

CEEMEML: 0000000100002810
+000000 MEMLDEF:.....   EXIT:00000000 00000000
+000010 LLVTL:00000000 00000000

[15] CEERCB: 0000000100002210
+000000 EYE:CEERCB          SYSTEM:03   HARDWARE:03   SUBSYS:03
+000043 FLAGS:00   DMBR:00000001 00002810   VERSION_ID:04010600
+000058 PCB_CHAIN:00000001 00002468

[16] CEEEDB: 00000001000039D0
+000000 EYE:CEEEDB          FLAG:97   BIPM:00   CREATOR_ID:01
+000108 BMEMBR:00000001 00004F40   OPTCB:00000001 00004458
+000118 USER_RC:00000000   RSN_CODE:00000000
+000120 PCB:00000001 00002468   APPL_STR:00000000 00000000
+000138 ENVAR:00000001 00002FA0   ENV_ANCHOR:00000001 00003B08
+000148 BOTRB:00000001 08910090   CAACHAIN:00000001 00006160
+000158 FLAGSI:82000000   THDSACT:00000000 00000003   DCB:00000000
+000170 INPUT_R1:00000000 00009A08   LAST_RB:00000000
+00017C LAST_RBCT:00000000   ENV_LGTH:00000000 00000200
+000188 ENVAR_A:00000001 000031A8   ENV_ANCHOR_A:00000001 00003B58

CEEMEML: 0000000100004F40
+000000 MEMLDEF:.....   EXIT:00000000 00000000
+000010 LLVTL:FFFFFFFF FFFFFFFF

```

[17]Language Environment Run-Time Options in effect.

```

LAST WHERE SET      Override  OPTIONS
*****
INSTALLATION DEFAULT  OVR      ENVAR("")
INSTALLATION DEFAULT  OVR      FILETAG(NOAUTOCVT,NOAUTOTAG)
PROGRAMMER DEFAULT    OVR      HEAPCHK(ON,00000001,00000000,00000000,
                                00000000)
INSTALLATION DEFAULT  OVR      HEAPPOLLS64(OFF,
                                00000008,00004000,
                                00000032,00002000,
                                00000128,00000700,
                                00000256,00000350,
                                00001024,00000100,
                                00002048,00000050)
                                00003072,00000050)
                                00004096,00000050)
                                00008192,00000025)
                                00016384,00000010)
                                00032768,00000005)
                                00065536,00000005)

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 5 of 11)

INSTALLATION DEFAULT	OVR	HEAP64(000000000000001,000000000000001, KEEP,00032768,00032768,KEEP, 000004096,00004096,FREE)
INSTALLATION DEFAULT	OVR	INFOMSGFILTER(OFF)
INSTALLATION DEFAULT	OVR	IOHEAP64(000000000000001,000000000000001, FREE,00012288,00000081,FREE, 00004096,00004096,FREE)
INSTALLATION DEFAULT	OVR	LIBHEAP64(000000000000001,000000000000001, FREE,00016384,00008192,FREE, 00008192,00004096,FREE)
INSTALLATION DEFAULT	OVR	NATLANG(ENU)
PROGRAMMER DEFAULT	OVR	POSIX(ON)
INSTALLATION DEFAULT	OVR	PROFILE(OFF,"")
INSTALLATION DEFAULT	OVR	RPTOPTS(OFF)
PROGRAMMER DEFAULT	OVR	RPTSTG(ON)
INSTALLATION DEFAULT	OVR	STACK64(000000000000001,000000000000001, 000000000000128)
INSTALLATION DEFAULT	OVR	STORAGE(NONE,NONE,NONE)
PROGRAMMER DEFAULT	OVR	TERMTHDACT(UADUMP,CESE,00000096)
INSTALLATION DEFAULT	OVR	NOTEST(ALL,*,PROMPT,INSPREF)
INSTALLATION DEFAULT	OVR	THREADSTACK64(OFF,000000000000001, 000000000000001, 000000000000128)
PROGRAMMER DEFAULT	OVR	TRACE(ON,01048576,NODUMP,LE=00000001)
INSTALLATION DEFAULT	OVR	TRAP(ON,SPIE)

[18] Heap Storage Control Blocks

```

ENSQ: 0000000100100108
+000000 EYE_CATCHER:ENSQ HEAPALLOC_VAL:00000000
+000008 HEAPFREE_VAL:00000000 DSAALLOC_VAL:00000000 FLAGS1:80000000
+000014 IPT_TOKEN:00000084 00000001 00000010 008DD8D8
+000024 HEAPLOCKWORD:00000000 RPT_STOR:00000001 00100410
+000030 UHEAP64:C8D7C3D8 00000000 00000001 001005A0 00000001 001005A0 00000000 00000001 00
+000060 LHEAP64:C8D7C3D8 00000000 00000001 001005D0 00000001 00100600 00000000 00000001 00
+000090 UHEAP31:C8D7C3D8 00000000 00000001 00100198 00000001 00100198 00000000 00008000 00
+0000C0 LHEAP31:C8D7C3D8 00000000 00000001 001001C8 00000001 001001C8 00000000 00004000 00
+0000F0 UHEAP24:C8D7C3D8 00000000 00000001 001001F8 00000001 001001F8 00000000 00001000 00
+000120 LHEAP24:C8D7C3D8 00000000 00000001 00100228 00000001 00100228 00000000 00002000 00
+000174 IPT_TCB:008DD8D8 HEAPCHK:00000000 00000000
+000188 STSB:00000000 00000000 SASB:00000000 00000000
+000198 TOKEN:7F786848 00000000
+0001A0 THDLHEAP64:C8D7C3D8 00000000 00000001 00100720 00000001 00100720 00000000 00000001 00
+0001D0 IOHEAP64:C8D7C3D8 00000000 00000001 001006F0 00000001 001006F0 00000000 00000001 00
+000200 IOHEAP31:C8D7C3D8 00000000 00000001 00100630 00000001 00100630 00000000 00003000 00
+000230 IOHEAP24:C8D7C3D8 00000000 00000001 00100660 00000001 00100660 00000000 00001000 00
+000260 SM_CELL_BLOCK:00000001 00100480 SPDE:00000001 001006C0

```

User Heap64 Control Blocks

```

HPCQ: 0000000100100138
+000000 EYE_CATCHER:HPCQ FIRST:00000001 001005A0
+000010 LAST:00000001 001005A0 INITSIZE:00000000 00000001
+000020 INCRSIZE:00000000 00000001 OPTIONS:80000000

HPSQ: 00000001000037E8
+000000 BYTES_ALLOC:00000000 00001C80
+000008 CURR_ALLOC:00000000 00001C80 GET_REQ:00000000 0000000F
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000000
+000028 FREEMAINS:00000000 00000000

THANQ: 00000001001005A0
+000000 EYE_CATCHER:THNQ FLAGS:80000000 NEXT:00000001 00100138
+000010 PREV:00000001 00100138 HEAPID:00000001 00100138
+000020 SEGMENT:00000001 08300000 SEG_LEN:00000000 00100000

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 6 of 11)

```

HANQ: 0000000108300000
+000000 EYE_CATCHER:HANQ  FLAGS:80000000  HEAPID:00000001 00100138
+000020 SEGMENT:00000001 08300000  ROOT:00000001 08301CC0
+000030 SEG_LEN:00000000 00100000  ROOT_LEN:00000000 000FE340

```

This is the last heap segment in the current heap

Free Storage Tree for Heap Segment 00000001

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	0000000108301CC0	00000000000FE340	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000108300000 LEN(X'000000000100000') ASID(X'0021')

```

0000000108300040: Allocated storage element, length=000000000000160.
To display: IP LIST 0000000108300040 LEN(X'000000000000160') ASID(X'0021')
0000000108300050: C36DE6E2 C1F6F440 40404040 40404040 00000001 08300070 00000000 251013E8 |C_WSA64 .....Y|

00000001083001A0: Allocated storage element, length=000000000000220.
To display: IP LIST 00000001083001A0 LEN(X'000000000000220') ASID(X'0021')
00000001083001B0: 00000001 083003D0 00000001 083005B8 00000001 083005F5 00000001 08300632 |.....5.....|

00000001083003C0: Allocated storage element, length=000000000000840.
To display: IP LIST 00000001083003C0 LEN(X'000000000000840') ASID(X'0021')
00000001083003D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

0000000108300C00: Allocated storage element, length=0000000000004A0.
To display: IP LIST 0000000108300C00 LEN(X'0000000000004A0') ASID(X'0021')
0000000108300C10: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

00000001083010A0: Allocated storage element, length=000000000000080.
To display: IP LIST 00000001083010A0 LEN(X'000000000000080') ASID(X'0021')
00000001083010B0: C3C4D3D3 00000000 00000000 00000000 C0000000 00000000 00000000 00000000 |CDLL.....|

0000000108301120: Allocated storage element, length=000000000000060.
To display: IP LIST 0000000108301120 LEN(X'000000000000060') ASID(X'0021')
0000000108301130: C36DE6E2 C1F6F440 40404040 40404040 00000000 000006F0 00000000 254B2710 |C_WSA64 .....0.....|

0000000108301180: Allocated storage element, length=000000000000080.
To display: IP LIST 0000000108301180 LEN(X'000000000000080') ASID(X'0021')
0000000108301190: 00000000 00000000 00000001 083001B8 10008000 00000000 00000000 255C7340 |.....h.....*. |

0000000108301200: Allocated storage element, length=000000000000800.
To display: IP LIST 0000000108301200 LEN(X'000000000000800') ASID(X'0021')
0000000108301210: 00000000 00000000 00000001 08948FD8 70004000 00000000 00000000 00000000 |.....m.Q.. ....|

0000000108301A00: Allocated storage element, length=000000000000080.
To display: IP LIST 0000000108301A00 LEN(X'000000000000080') ASID(X'0021')
0000000108301A10: 00000000 00000000 00000001 089490B8 30004000 00000000 00000000 00000000 |.....m.h.. ....|

0000000108301A80: Allocated storage element, length=000000000000080.
To display: IP LIST 0000000108301A80 LEN(X'000000000000080') ASID(X'0021')
0000000108301A90: 00000000 00000000 00000001 089490A0 30004000 00000000 00000000 00000000 |.....m.... ....|

0000000108301B00: Allocated storage element, length=000000000000080.
To display: IP LIST 0000000108301B00 LEN(X'000000000000080') ASID(X'0021')
0000000108301B10: 00000000 00000000 00000001 00003F98 30004000 00000000 00000000 00000000 |.....q.. ....|

0000000108301B80: Allocated storage element, length=000000000000020.
To display: IP LIST 0000000108301B80 LEN(X'000000000000020') ASID(X'0021')
0000000108301B90: 00000001 08301BB0 00000000 00000000 |.....|

0000000108301BA0: Allocated storage element, length=000000000000080.

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 7 of 11)

```

To display: IP LIST 0000000108301BA0 LEN(X'0000000000000080') ASID(X'0021')
0000000108301BB0: 00000000 00000000 00000001 08301B90 70004000 00000000 00000000 00000000 |.....|

0000000108301C20: Allocated storage element, length=0000000000000020.
To display: IP LIST 0000000108301C20 LEN(X'0000000000000020') ASID(X'0021')
0000000108301C30: 00000001 08301C50 00000000 00000000 |.....&.....|

0000000108301C40: Allocated storage element, length=0000000000000080.
To display: IP LIST 0000000108301C40 LEN(X'0000000000000080') ASID(X'0021')
0000000108301C50: 00000000 00000000 00000001 08301C30 70004000 00000000 00000000 00000000 |.....|

0000000108301CC0: Free storage element, length=000000000000FE340.
To display: IP LIST 0000000108301CC0 LEN(X'000000000000FE340') ASID(X'0021')

Summary of analysis for Heap Segment 0000000108300000:
Amounts of identified storage: Free:000FE340      Allocated:00001C80      Total:000FFFC0
Number of identified areas  : Free:      1  Allocated:    15  Total:    16
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
This is the last heap segment in the current heap.

:

[19] Stack Storage Control Blocks

SANC: 0000000100100000
+000000 EYE_CATCHER:SANC VERSION:0001 LENGTH:0100
+000008 SEGMENT_SIZE:00000000 00000081
+000010 ACTIVE_STACK:00000001 00200000 BOS:00000001 082FFFE0
+000020 INIT_SIZE:00000000 00000001 INCR_SIZE:00000000 00000001
+000030 USER_STACK:00000001 00200000 USER_BOS:00000001 082FFFE0
+000040 USER_FLOOR:00000001 08200000
+000048 RESERVE_STACK:00000001 08500000 SDCB:00000000 00000000
+000060 PTDATA:00000000 00000000

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 8 of 11)

DSA backchain

```

DSA: 00000001082FA8C0
+000800 HPR4:00000001 082FAAC0 HPR5:00000000 251A7488
+000810 HPR6:00000000 251A7390 HPR7:00000000 251A1140
+000820 HPR8:00000001 082FBD00 HPR9:00000001 00000003
+000830 HPR10:00000001 082FBABF HPR11:00000001 082FBB08
+000840 HPR12:00000001 082FCABE HPR13:00000001 082FE680
+000850 HPR14:00000000 251A1040 HPR15:00000000 251A21DC
+000860 HPHKSAV:00000001 082FB268 HPTRAN:00000001 082FC6D8
+000878 HPRENT:00000001 082FC6D8

```

Contents of DSA at Location : 00000001082FB0C0

```

+000000 00000001082FB0C0 00000001 082FAAC0 00000000 251A7488 .....h
+000010 00000001082FB0D0 00000000 251A7390 00000000 251A1140 .....
+000020 00000001082FB0E0 00000001 082FBD00 00000001 00000003 .....0.
+000030 00000001082FB0F0 00000001 082FBABF 00000001 082FBB08 .....
+000040 00000001082FB100 00000001 082FCABE 00000001 082FE680 .....W.
+000050 00000001082FB110 00000000 251A1040 00000000 251A21DC .....
+000060 00000001082FB120 00000001 082FB268 00000000 251A2C00 .....
+000070 00000001082FB130 00000001 082FC6D8 00000001 082FC6D8 .....FQ.....FQ
+000080 00000001082FB140 0C890000 00000000 00000000 00000000 .i.....
+000090 00000001082FB150 0001082F B0110000 0000251C 9C280400 .....
+0000A0 00000001082FB160 005B0000 000A0003 0F0ED740 40000000 .$......P ...
+0000B0 00000001082FB170 082FD3E0 251A0000 0000255D 40880000 ..L.....) h..
+0000C0 00000001082FB180 0000255D 40DF0000 00020000 25127010 ...) .....
+0000D0 00000001082FB190 082F00C5 D5E483A4 00000000 00000001 ...ENUcu.....
+0000E0 00000001082FBA0 082FB269 0000005B 0008C3C5 C5F3F2F0 .....$.CEE320
+0000F0 00000001082FB1B0 F9E240E3 888540A2 A8A2A385 94408485 9S The system de

```

```

DSA: 00000001082FAAC0
+000800 HPR4:00000001 082FD3E0 HPR5:00000000 251A20D4
+000810 HPR6:00000000 2519DAA0 HPR7:00000000 25227C28
+000820 HPR8:00000001 08913470 HPR9:00000000 00000005
+000830 HPR10:00000001 082FE3DF HPR11:00000001 082FE0CC
+000840 HPR12:00000001 08913470 HPR13:00000001 082FE680
+000850 HPR14:00000000 25227B28 HPR15:00000000 251000B0
+000860 HPHKSAV:00000000 00000000 HPTRAN:00000000 00000000
+000878 HPRENT:00000000 00000000

```

Contents of DSA at Location : 00000001082FB2C0

```

+000000 00000001082FB2C0 00000001 082FD3E0 00000000 251A20D4 .....L.....M
+000010 00000001082FB2D0 00000000 2519DAA0 00000000 25227C28 .....@.
+000020 00000001082FB2E0 00000001 08913470 00000000 00000005 .....J.....
+000030 00000001082FB2F0 00000001 082FE3DF 00000001 082FE0CC .....T.....
+000040 00000001082FB300 00000001 08913470 00000001 082FE680 .....J.....W.
+000050 00000001082FB310 00000000 25227B28 00000000 251000B0 .....#.....
+000060 00000001082FB320 00000000 00000000 00000000 00000000 .....
+000070 00000001082FB330 - +000000 00000001082FB33F same as above
+000080 00000001082FB340 00000001 082FBB08 00000000 00000000 .....
+000090 00000001082FB350 00000001 00000003 00000000 00000000 .....
+0000A0 00000001082FB360 00000001 082FC180 00000001 082FBF80 .....A.....
+0000B0 00000001082FB370 00000001 082FC380 00000000 00000000 .....C.....
+0000C0 00000001082FB380 00000000 00000000 00000000 00000000 .....
+0000D0 00000001082FB390 - +000000 00000001082FB3BF same as above

```

```

DSA: 00000001082FD3E0
+000800 HPR4:00000001 082FDDE0 HPR5:00000000 25228BF4
+000810 HPR6:00000000 252272D0 HPR7:00000000 2513B920
+000820 HPR8:00000000 255C8AF8 HPR9:00000000 255C8BF0
+000830 HPR10:00000000 255C8A48 HPR11:00000000 00000020
+000840 HPR12:00000001 00006160 HPR13:00000001 082FE680
+000850 HPR14:00000000 25225C1C HPR15:00000000 00000003
+000860 HPHKSAV:00000001 082FDDE8 HPTRAN:00000001 082FDF60
+000878 HPRENT:00000001 00006160

```

:

Figure 125. Example of formatted output from LEDATA Verbexit (Part 9 of 11)

[20] Condition Management Control Blocks

```

HCOM: 00000000255C7878
+000000 PICA_AREA:00000000 00000000 EYES:HCOM SIZE:2780 LEVEL:0001
+000010 CAA_PTR1:00000001 00006160 CVTDCB:9B FLAG1:6010C000
+000020 EXIT_STK:00000000 00000000 RSM_PTR:00000000 00000000
+000030 HDLL_STK:00000000 00000000 SRP_TOKEN:00000000 00000000
+000040 CURR_STK:00000000 00000000 CIBH:00000001 082FCFD0
+000050 SPIE_TOKEN:00000000 DMCP:00000000 00000000
+0000F0 4083_DSA:00000000 00000000

CIBH: 00000001082FCFD0
+000000 EYE:CIBH BACK:00000001 00005100 FWD:00000000 00000000
+000018 LEVEL:0002 SIZE:0C00 PTR_CIB:00000000 00000000
+000028 FLAG1:00000000 HDLQ:00000000 00000000 STATE:00000000
+000040 PRM_DESC:00000000 00000000 PRM_PREFIX:00000000 00000000
+000050 PRM_LIST:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
+000070 PARM_DESC:00000000 00000000 PARM_PREFIX:00000000 00000000
+000080 PARM_LIST:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
+0000A0 FUNC_CODE:00000000 SIZ:0000 VER:0000 FLAG5:00
+0000A9 FLAG6:00 FLAG7:00 FLAG8:00 FLAG1:00 FLAG2:00
+0000AE FLAG3:00 FLAG4:00 ABCD:00000000 ABRC:00000000
+0000B8 OLD_COND64:00000000 00000000 OLD_MIB:00000000 00000000
+0000C8 COND64:00000000 00000000 MIB:00000000 00000000
+0000D8 PL:00000000 00000000 SV2:00000000 00000000
+0000E8 SV1:00000000 00000000 INT:00000000 00000000 MID:00000000
+000100 HDL_SF:00000000 00000000 HDL_EPT:00000000 00000000
+000110 HDL_RST:00000000 00000000 RSM_SF:00000000 00000000
+000120 RSM_PT:00000000 00000000 RSM_MACH:00000000 00000000
+000130 SUSPEND_EH:00000000 00000000 COND_DFT:00000000
+000140 Q_DATA_TOKEN:00000000 00000000 FDBK:00000000 00000000
+000150 ABNAME:.....
Machine State
+000348 MCH_EYE:....
+000350 MCH_GPR00:00000000 00000000 MCH_GPR01:00000000 00000000
+000360 MCH_GPR02:00000000 00000000 MCH_GPR03:00000000 00000000
+000370 MCH_GPR04:00000000 00000000 MCH_GPR05:00000000 00000000
+000380 MCH_GPR06:00000000 00000000 MCH_GPR07:00000000 00000000
+000390 MCH_GPR08:00000000 00000000 MCH_GPR09:00000000 00000000
+0003A0 MCH_GPR10:00000000 00000000 MCH_GPR11:00000000 00000000
+0003B0 MCH_GPR12:00000000 00000000 MCH_GPR13:00000000 00000000
+0003C0 MCH_GPR14:00000000 00000000 MCH_GPR15:00000000 00000000
+0003D0 MCH_PSW:00000000 00000000 00000000 00000000 MCH_ILC:0000
+0003E2 MCH_IC1:00 MCH_IC2:00 MCH_PFT:00000000 00000000
+0003F0 MCH_FLT_0:00000000 00000000 00000000 00000000
+000400 MCH_FLT_2:00000000 00000000 00000000 00000000
+000410 MCH_FLT_4:00000000 00000000 00000000 00000000
+000420 MCH_FLT_6:00000000 00000000 00000000 00000000
+0004B8 MCH_EXT:00000000 00000000

CIBH: 0000000100005100
+000000 EYE:CIBH BACK:00000000 00000000 FWD:00000001 082FCFD0
+000018 LEVEL:0002 SIZE:0C00 PTR_CIB:00000001 082FBD0
+000028 FLAG1:C12B0000 HDLQ:00000000 00000000 STATE:00000000
+000040 PRM_DESC:00000000 00000000 PRM_PREFIX:00000000 00000000
+000050 PRM_LIST:00000001 082FBE18 00000001 082FBF40 00000001 082FBF50 00000001 00005850
+000070 PARM_DESC:00000000 00000000 PARM_PREFIX:00000000 00000000
+000080 PARM_LIST:00000001 082FBF38 00000001 082FBD0 00000001 082FBF50 00000001 00005850
+0000A0 FUNC_CODE:00000067 SIZ:0190 VER:0004 FLAG5:48
+0000A9 FLAG6:22 FLAG7:04 FLAG8:00 FLAG1:00 FLAG2:00
+0000AE FLAG3:00 FLAG4:05 ABCD:940C9000 ABRC:00000009
+0000B8 OLD_COND64:00030C89 59C3C5C5 OLD_MIB:00000000 00000000
+0000C8 COND64:00030C89 59C3C5C5 MIB:00000000 00000000
+0000D8 PL:00000000 251016C8 SV2:00000001 082FF080
+0000E8 SV1:00000001 082FF080 INT:00000000 255CE690 MID:00000003

```

Figure 125. Example of formatted output from LEDATA Verbexit (Part 10 of 11)

```

+000100 HDL_SF:00000001 082FF280 HDL_EPT:00000000 251A26B0
+000110 HDL_RST:00000000 00000000 RSM_SF:00000001 082FF080
+000120 RSM_PT:00000000 255CE694 RSM_MACH:00000001 00005648
+000130 SUSPEND_EH:00000000 00000000 COND_DFT:00000003
+000140 Q_DATA_TOKEN:00000000 00000000 FDBK:00000000 00000000
+000150 ABNAME:.....
Machine State
+000348 MCH_EYE:ZMCH
+000350 MCH_GPR00:00000000 00000000 MCH_GPR01:00000000 00000001
+000360 MCH_GPR02:00000000 255CE6EA MCH_GPR03:00000000 00000012
+000370 MCH_GPR04:00000001 082FF080 MCH_GPR05:00000000 000006F0
+000380 MCH_GPR06:00000000 254B2710 MCH_GPR07:00000000 255CE686
+000390 MCH_GPR08:00000000 255CE6C4 MCH_GPR09:00000001 08301140
+0003A0 MCH_GPR10:00000000 255CE6D8 MCH_GPR11:00000001 08948EF0
+0003B0 MCH_GPR12:00000001 000039D0 MCH_GPR13:00000000 0000CF68
+0003C0 MCH_GPR14:00000000 7F786848 MCH_GPR15:00000000 0000001F
+0003D0 MCH_PSW:07852401 80000000 00000000 255CE694 MCH_ILC:0004
+0003E2 MCH_IC1:00 MCH_IC2:09 MCH_PFT:00000000 00000000
+0003F0 MCH_FLT_0:4327614B 706A9AB9 3FF20DD7 50429B6D
+000400 MCH_FLT_2:3FF6A09E 667F3BCD 3FD45F30 6DC9C883
+000410 MCH_FLT_4:3FE6A09E 667F3BCD 34100000 00000000
+000420 MCH_FLT_6:00000000 00000000 00000000 00000000
+0004B8 MCH_EXT:00000000 00000000

CIB: 00000001082FBDFO
+000000 EYE:CIB BACK:00000000 00000000 FWD:00000000 00000000
+000018 SIZE:0190 VERSION:0004 PLAT_ID:00000000
+000028 COND:000300C6 59C3C5C5 MIB:00000000 00000000
+000038 MACHINE:00000001 082F8BF80 OLD_COND_64:00030C89 59C3C5C5
+000048 OLD_MIB:00000000 00000000 FLG_1:00 FLG_2:00 FLG_3:00
+000053 FLG_4:04 HDL_SF:00000001 082FF180
+000060 HDL_EPT:00000000 251A26B0 HDL_RST:00000000 00000000
+000070 RSM_SF:00000001 082FF080 RSM_PT:00000000 255CE694
+000080 RSM_MACH:00000001 00005648 COND_DEFAULT:00000003
+000090 PH_CALLEE_SF:00000001 082FF080 HDL_SF_FMT:01
+000099 PH_CALLEE_FMT:01 VSR:00000000 00000000
+0000D8 VSTOR:00000000 00000000 VRPSA:00000000 00000000
+0000E8 MCB:00000000 00000000 MRN:..... MFLAG:00 FLG_5:48
+000101 FLG_6:22 FLG_7:04 FLG_8:00 ABCD:940C9000
+00010C ABRC:00000009 ABNAME:..... PL:00000000 251016C8
+000120 SV2:00000001 082FF080 SV1:00000001 082FF080
+000130 INT:00000000 255CE690 Q_DATA_TOKEN:00000000 00000000
+000140 FDBK:00000000 00000000 FUN:00000067
+000150 TOKEN:00000001 082FF180 MID:00000003 STATE:00000000
+000160 RTCC:00000014 PPAV:00000004 AB_TERM_EXIT:.....
+000170 SDWA:00000000 00000000 SIGN0:00000008
+000180 PPSD:00000001 00005868

```

[21] Message Processing Control Blocks

```

CMXB: 0000000100001730
+000000 EYE:CMXB SIZE:0128 FLAGS:8000 DHEAD1:00000000 00000000
+000010 DHEAD2:00000000 00000000

TMXB: 0000000100005EA0
+000000 EYE:TMXB MIB_CHAIN_PTR:00000001 00005EE8

MGF: 0000000100005EE8
+000000 EYE:CMIB PREV:00000000 00000001
+00000C NEXT:00005EE8 00000001

```

Exiting Language Environment Data

Figure 125. Example of formatted output from LEDATA Verbexit (Part 11 of 11)

Sections of the Language Environment LEDATA Verbexit formatted output

The sections of the output listed here appear independently of the Language Environment-conforming languages used.

[1] - [8]CEEDUMP Formatted Control Blocks

These sections are included when the CEEDUMP parameter is specified on the LEDATA invocation.

[1] - [4] NTHREADS data

These sections are also included, once for each thread, when the NTHREADS() parameter is specified on the LEDATA invocations. For a description of NTHREADS, see “Report type parameters” on page 309.

[1] Enclave Identifier

This statement names the enclave for which information is provided.

[2] Information for thread

This section shows the system identifier for the thread. Each thread has a unique identifier.

[3] Registers and PSW

This section displays the register and program status word (PSW) values that were used to create the traceback. These values may come from the TCB, the RTM2 work area, a linkage stack entry or output from the BPXGMSTA service. This section is not displayed when the DSA() parameter is specified on the LEDATA invocation.

The second part contains:

- DSA number
A number assigned to the information for this active routine by LEDATA. The number is only used to associate information from the first part of the traceback with information in the second part of the traceback.
- Stack frame (DSA) address
- Entry point address
- Program unit address
- Program unit offset
The offset of the last instruction to run in the routine. If the offset is a negative number, zero, or a very large positive number, the routine associated with the offset probably did not allocate a save area, or the routine could have been called using SVC-assisted linkage. Adding the program unit address to the offset gives you the location of the current instruction in the routine. This offset is from the starting address of the routine.

[4] Traceback

For all active routines in a particular thread. The traceback section shows routine information in two parts. The first part contains:

- DSA number
A number assigned to the information for this active routine by LEDATA. The number is only used to associate information from the first part of the traceback with information in the second part of the traceback.
- Entry
For C/C++ routines, this is the function name. If a function name or entry point was not specified for a particular routine, then the string *** NoName *** will appear.
- Entry point offset
- Load module

- Program unit
The primary entry point of the external procedure. For C routines, this is the compile unit name. For Language Environment-conforming assemblers, this is the ENTNAME = value on the CELQPRLG macro.
- Stack frame (DSA) address
- Program unit address
- Program unit offset
The offset of the last instruction to run in the routine. If the offset is a negative number, zero, or a very large positive number, the routine associated with the offset probably did not allocate a save area, or the routine could have been called using SVC-assisted linkage. Adding the program unit address to the offset gives you the location of the current instruction in the routine. This offset is from the starting address of the routine.
- Service level
The latest service level applied to the compile unit (for example, for IBM products, it would be the PTF number).
- Status
Routine status can be call, exception, or running.

The second part contains:

- DSA number
A number assigned to the information for this active routine by LEDATA. The number is only used to associate information from the first part of the traceback with information in the second part of the traceback.
- Stack frame (DSA) address
- Entry point address
- Program unit address
- Program unit offset
The offset of the last instruction to run in the routine. If the offset is a negative number, zero, or a very large positive number, the routine associated with the offset probably did not allocate a save area, or the routine could have been called using SVC-assisted linkage. Adding the program unit address to the offset gives you the location of the current instruction in the routine. This offset is from the starting address of the routine.

[5] Control Blocks Associated with the Thread

This section lists the contents of the thread synchronization queue element (SQEL).

[6] Enclave Control Blocks

If the POSIX run-time option was set to ON, this section lists the contents of the mutex and condition variable control blocks, the enclave level latch table, and the thread synchronization trace block and trace table. If the HEAPCHK run-time option is set to ON, this section lists the contents of the HEAPCHK options control block (HCOP) and the HEAPCHK element tables (HCEL). A HEAPCHK element table contains the location and length of all allocated storage elements for a heap in the order that they were allocated.

[7] Language Environment Trace Table

If the TRACE run-time option was set to ON, this section shows the contents of the Language Environment trace table.

[8] Process Control Blocks

If the POSIX run-time option was set to ON, this section lists the contents of the process level latch table.

[9] - [17] Summary

These sections are included when the SUMMARY parameter is specified on the LEDATA invocation.

[9] Summary Header

The summary header section contains:

- Address of Thread control block (TCB)
- Release number
- Address Space ID (ASID)

[10] Active Members List

This list of active members is extracted from the enclave member list (MEML).

[11] CEELAA

This section formats the contents of the Language Environment library anchor area (LAA). Refer to *z/OS Language Environment Vendor Interfaces* for a description of the fields in the LAA.

[12] CEELCA

This section formats the contents of the Language Environment library control area (LCA). Refer to *z/OS Language Environment Vendor Interfaces* for a description of the fields in the LCA.

[13] CEECAA

This section formats the contents of the Language Environment common anchor area (CAA). Refer to *z/OS Language Environment Vendor Interfaces* for a description of the fields in the CAA.

[14] CEEPCB

This section formats the contents of the Language Environment process control block (PCB), and the process level member list.

[15] CEERCB

This section formats the contents of the Language Environment region control block (RCB).

[16] CEEEDB

This section formats the contents of the Language Environment enclave data block (EDB), and the enclave level member list.

[17] Run-Time Options

This section lists the run-time options in effect at the time of the dump, and indicates where they were set.

[18] Heap Storage Control Blocks

This section is included when the HEAP or SM parameter is specified on the LEDATA invocation.

This section formats the Enclave-level storage management control block (ENSQ) and for each different type of heap storage:

- Heap control block (HPCQ)
- Chain of heap anchor blocks (HANQ). A HANQ immediately precedes each segment of heap storage.

This section includes a detailed heap segment report for each segment in the dump. For more information about the detailed heap segment report, see “Understanding the HEAP LEDATA output.”

[19] Stack Storage Control Blocks

This section is included when the STACK or SM parameter is specified on the LEDATA invocation.

This section formats:

- Stack anchor (SANC)
- Chain of dynamic save areas (DSA)

[20] Condition Management Control Blocks

This section is included when the CM parameter is specified on the LEDATA invocation.

This section formats the chain of Condition Information Block Headers (CIBH) and Condition Information Blocks. The Machine State Information Block is contained with the CIBH starting with the field labeled MCH_EYE.

[21] Message Processing Control Blocks

This section is included when the MH parameter is specified on the LEDATA invocation.

Understanding the HEAP LEDATA output

The Language Environment IPCS Verbexit LEDATA generates a detailed heap segment report when the HEAP option is used with the DETAIL option, or when the SM,DETAIL option is specified. The detailed heap segment report is useful when trying to pinpoint damage because it provides very specific information. The report describes the nature of the damage, and specifies where the actual damage occurred. The report can also be used to diagnose storage leaks, and to identify heap fragmentation. Figure 126 on page 327 illustrates the output produced by specifying the HEAP option. “Heap report sections of the LEDATA output” on page 336 describes the information contained in the formatted output.

For easy reference, the sections of the dump are numbered to correspond with the description of each section that follows. Ellipses are used to summarize some sections of the dump.

Note: Language Environment does not provide support for alternative Vendor Heap Manager (VHM) data. LEDATA Verbexit will state that an alternative VHM is in use.

```

IP VERBEXIT LEDATA 'HEAP'
*****
64 BIT LANGUAGE ENVIRONMENT DATA
*****

Language Environment Product 04 V01 R06.00

Heap Storage Control Blocks

  ENSQ: 0000000100100108
+000000 EYE_CATCHER:ENSQ  HEAPALLOC_VAL:00000000
+000008 HEAPFREE_VAL:00000000  DSAALLOC_VAL:00000000  FLAGS1:80000000
+000014 IPT_TOKEN:00000084 00000001 00000010 008DD8D8
+000024 HEAPLOCKWORD:00000000  RPT_STOR:00000001 00100410
+000030 UHEAP64:C8D7C3D8 00000000 00000001 001005A0 00000001 001005A0 00000000 00000001 00
+000060 LHEAP64:C8D7C3D8 00000000 00000001 001005D0 00000001 00100600 00000000 00000001 00
+000090 UHEAP31:C8D7C3D8 00000000 00000001 00100198 00000001 00100198 00000000 00008000 00
+0000C0 LHEAP31:C8D7C3D8 00000000 00000001 001001C8 00000001 001001C8 00000000 00004000 00
+0000F0 UHEAP24:C8D7C3D8 00000000 00000001 001001F8 00000001 001001F8 00000000 00001000 00
+000120 LHEAP24:C8D7C3D8 00000000 00000001 00100228 00000001 00100228 00000000 00002000 00
+000174 IPT_TCB:008DD8D8  HEAPCHK:00000000 00000000
+000188 STSB:00000000 00000000  SASB:00000000 00000000
+000198 TOKEN:7F786848 00000000
+0001A0 THDLHEAP64:C8D7C3D8 00000000 00000001 00100720 00000001 00100720 00000000 00000001 00
+0001D0 IOHEAP64:C8D7C3D8 00000000 00000001 001006F0 00000001 001006F0 00000000 00000001 00
+000200 IOHEAP31:C8D7C3D8 00000000 00000001 00100630 00000001 00100630 00000000 00003000 00
+000230 IOHEAP24:C8D7C3D8 00000000 00000001 00100660 00000001 00100660 00000000 00001000 00
+000260 SM_CELL_BLOCK:00000001 00100480  SPDE:00000001 001006C0

User Heap64 Control Blocks

  HPCQ: 0000000100100138
+000000 EYE_CATCHER:HPCQ  FIRST:00000001 001005A0
+000010 LAST:00000001 001005A0  INITSIZE:00000000 00000001
+000020 INCRSIZE:00000000 00000001  OPTIONS:80000000

  HPSQ: 00000001000037E8
+000000 BYTES_ALLOC:00000000 00001C80
+000008 CURR_ALLOC:00000000 00001C80  GET_REQ:00000000 0000000F
+000018 FREE_REQ:00000000 00000000  GETMAINS:00000000 00000000
+000028 FREEMAINS:00000000 00000000

  THANQ: 00000001001005A0
+000000 EYE_CATCHER:THNQ  FLAGS:80000000  NEXT:00000001 00100138
+000010 PREV:00000001 00100138  HEAPID:00000001 00100138
+000020 SEGMENT:00000001 08300000  SEG_LEN:00000000 00100000

```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 1 of 10)

```

HANQ: 0000000108300000
+000000 EYE_CATCHER:HANQ  FLAGS:80000000  HEAPID:00000001 00100138
+000020 SEGMENT:00000001 08300000  ROOT:00000001 08301CC0
+000030 SEG_LEN:00000000 00100000  ROOT_LEN:00000000 000FE340

```

This is the last heap segment in the current heap

[1]Free Storage Tree for Heap Segment 00000001

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	0000000108301CC0	00000000000FE340	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

[2]Map of Heap Segment 00000001

```

To display entire segment: IP LIST 0000000108300000 LEN(X'000000000100000') ASID(X'0021')

0000000108300040: Allocated storage element, length=0000000000000160.
To display: IP LIST 0000000108300040 LEN(X'0000000000000160') ASID(X'0021')
0000000108300050: C3DE6E2 C1F6F440 40404040 40404040 00000001 08300070 00000000 251013E8 |C_WSA64 .....Y|

00000001083001A0: Allocated storage element, length=0000000000000220.
To display: IP LIST 00000001083001A0 LEN(X'0000000000000220') ASID(X'0021')
00000001083001B0: 00000001 083003D0 00000001 083005B8 00000001 083005F5 00000001 08300632 |.....5.....|

00000001083003C0: Allocated storage element, length=0000000000000840.
To display: IP LIST 00000001083003C0 LEN(X'0000000000000840') ASID(X'0021')
00000001083003D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

0000000108300C00: Allocated storage element, length=00000000000004A0.
To display: IP LIST 0000000108300C00 LEN(X'00000000000004A0') ASID(X'0021')
0000000108300C10: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

00000001083010A0: Allocated storage element, length=0000000000000080.
To display: IP LIST 00000001083010A0 LEN(X'0000000000000080') ASID(X'0021')
00000001083010B0: C3C4D3D3 00000000 00000000 00000000 C0000000 00000000 00000000 00000000 |CDLL.....|

0000000108301120: Allocated storage element, length=0000000000000060.
To display: IP LIST 0000000108301120 LEN(X'0000000000000060') ASID(X'0021')
0000000108301130: C3DE6E2 C1F6F440 40404040 40404040 00000000 000000F0 00000000 254B2710 |C_WSA64 .....0.....|

0000000108301180: Allocated storage element, length=0000000000000080.
To display: IP LIST 0000000108301180 LEN(X'0000000000000080') ASID(X'0021')
0000000108301190: 00000000 00000000 00000001 08300188 10008000 00000000 00000000 255C7340 |.....h.....*. |

0000000108301200: Allocated storage element, length=0000000000000800.
To display: IP LIST 0000000108301200 LEN(X'0000000000000800') ASID(X'0021')
0000000108301210: 00000000 00000000 00000001 08948FD8 70004000 00000000 00000000 00000000 |.....m.Q.. .....|

0000000108301A00: Allocated storage element, length=0000000000000080.
To display: IP LIST 0000000108301A00 LEN(X'0000000000000080') ASID(X'0021')
0000000108301A10: 00000000 00000000 00000001 08949088 30004000 00000000 00000000 00000000 |.....m.h.. .....|

0000000108301A80: Allocated storage element, length=0000000000000080.
To display: IP LIST 0000000108301A80 LEN(X'0000000000000080') ASID(X'0021')
0000000108301A90: 00000000 00000000 00000001 089490A0 30004000 00000000 00000000 00000000 |.....m.... .....|

0000000108301B00: Allocated storage element, length=0000000000000080.
To display: IP LIST 0000000108301B00 LEN(X'0000000000000080') ASID(X'0021')
0000000108301B10: 00000000 00000000 00000001 00003F98 30004000 00000000 00000000 00000000 |.....q.. .....|

0000000108301B80: Allocated storage element, length=0000000000000020.
To display: IP LIST 0000000108301B80 LEN(X'0000000000000020') ASID(X'0021')
0000000108301B90: 00000001 08301BB0 00000000 00000000 |.....|

0000000108301BA0: Allocated storage element, length=0000000000000080.

```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 2 of 10)

```

To display: IP LIST 0000000108301BA0 LEN(X'0000000000000080') ASID(X'0021')
0000000108301BB0: 00000000 00000000 00000001 08301B90 70004000 00000000 00000000 00000000 |.....|

0000000108301C20: Allocated storage element, length=0000000000000020.
To display: IP LIST 0000000108301C20 LEN(X'0000000000000080') ASID(X'0021')
0000000108301C30: 00000001 08301C50 00000000 00000000 |.....&.....|

0000000108301C40: Allocated storage element, length=0000000000000080.
To display: IP LIST 0000000108301C40 LEN(X'0000000000000080') ASID(X'0021')
0000000108301C50: 00000000 00000000 00000001 08301C30 70004000 00000000 00000000 00000000 |.....|

0000000108301CC0: Free storage element, length=0000000000FE340.
To display: IP LIST 0000000108301CC0 LEN(X'0000000000FE340') ASID(X'0021')

Summary of analysis for Heap Segment 0000000108300000:
Amounts of identified storage: Free:000FE340 Allocated:00001C80 Total:000FFFC0
Number of identified areas : Free: 1 Allocated: 15 Total: 16
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
This is the last heap segment in the current heap.

Library Heap64 Control Blocks

HPCQ: 0000000100100168
+000000 EYE_CATCHER:HPCQ FIRST:00000001 001005D0
+000010 LAST:00000001 00100600 INITSIZE:00000000 00000001
+000020 INCRSIZE:00000000 00000001 OPTIONS:90000000

HPSQ: 0000000100003878
+000000 BYTES_ALLOC:00000000 0016D500
+000008 CURR_ALLOC:00000000 0016D500 GET_REQ:00000000 00000024
+000018 FREE_REQ:00000000 00000002 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000

THANQ: 00000001001005D0
+000000 EYE_CATCHER:THNQ FLAGS:80000000 NEXT:00000001 00100600
+000010 PREV:00000001 00100168 HEAPID:00000001 00100168
+000020 SEGMENT:00000001 08400000 SEG_LEN:00000000 00100000

HANQ: 0000000108400000
+000000 EYE_CATCHER:HANQ FLAGS:80000000 HEAPID:00000001 00100168
+000020 SEGMENT:00000001 08400000 ROOT:00000001 08400040
+000030 SEG_LEN:00000000 00100000 ROOT_LEN:00000000 000FFFC0

[1]Free Storage Tree for Heap Segment 00000001

Depth      Node      Node      Parent      Left      Right      Left      Right
Address    Length    Node      Node      Node      Node      Length    Length
0 0000000108400040 00000000000FFFC0 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000108400000 LEN(X'000000000100000') ASID(X'0021')

0000000108400040: Free storage element, length=0000000000FFFC0.
To display: IP LIST 0000000108400040 LEN(X'0000000000FFFC0') ASID(X'0021')

Summary of analysis for Heap Segment 0000000108400000:
Amounts of identified storage: Free:000FFFC0 Allocated:00000000 Total:000FFFC0
Number of identified areas : Free: 1 Allocated: 0 Total: 1
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.

THANQ: 0000000100100600
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001 00100168
+000010 PREV:00000001 001005D0 HEAPID:00000001 00100168
+000020 SEGMENT:00000001 08800000 SEG_LEN:00000000 00200000

```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 3 of 10)

```

HANQ: 0000000108800000
+000000 EYE_CATCHER: HANQ  FLAGS:00000000  HEAPID:00000001 00100168
+000020 SEGMENT:00000001 08800000  ROOT:00000001 0896D540
+000030 SEG_LEN:00000000 00200000  ROOT_LEN:00000000 00092AC0

```

This is the last heap segment in the current heap

[1]Free Storage Tree for Heap Segment 00000001

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	000000010896D540	00000000000092AC0	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

[2]Map of Heap Segment 00000001

```

To display entire segment: IP LIST 0000000108800000 LEN(X'000000000200000') ASID(X'0021')

0000000108800040: Allocated storage element, length=0000000000100040.
To display: IP LIST 0000000108800040 LEN(X'0000000000100040') ASID(X'0021')
0000000108800050: BB0A5B83 4FB9BD92 25481460 00000001 03000000 00000005 94818995 40404040 |..$c|..k....-.....main |

0000000108900080: Allocated storage element, length=00000000000132A0.
To display: IP LIST 0000000108900080 LEN(X'00000000000132A0') ASID(X'0021')
0000000108900090: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

0000000108913320: Allocated storage element, length=0000000000010060.
To display: IP LIST 0000000108913320 LEN(X'0000000000010060') ASID(X'0021')
0000000108913330: E2C7D3C7 00010000 00010040 00000000 00000001 08913370 00000001 08923370 |SGLG..... .....j.....k..|

0000000108923380: Allocated storage element, length=0000000000020080.
To display: IP LIST 0000000108923380 LEN(X'0000000000020080') ASID(X'0021')
0000000108923390: C8C3D6D7 00000048 00000001 00000000 00000000 00000000 00000000 00000000 |HCOP.....|

0000000108943400: Allocated storage element, length=00000000000000C0.
To display: IP LIST 0000000108943400 LEN(X'00000000000000C0') ASID(X'0021')
0000000108943410: 00000001 08948E50 00000000 00000000 00000000 00000000 00000000 25101000 |.....m.&.....|

00000001089434C0: Allocated storage element, length=0000000000003EC0.
To display: IP LIST 00000001089434C0 LEN(X'0000000000003EC0') ASID(X'0021')
00000001089434D0: C8C3C5D3 00000000 00000000 00000000 00000000 00000000 00000001 00100138 |HCEL.....|

0000000108947380: Allocated storage element, length=0000000000000E40.
To display: IP LIST 0000000108947380 LEN(X'0000000000000E40') ASID(X'0021')
0000000108947390: C3D3D3E3 0E280001 00000000 00000000 00000000 00000000 00000000 00000000 |CLLT.....|

00000001089481C0: Allocated storage element, length=0000000000000040.
To display: IP LIST 00000001089481C0 LEN(X'0000000000000040') ASID(X'0021')
00000001089481D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

0000000108948200: Allocated storage element, length=0000000000000BE0.
To display: IP LIST 0000000108948200 LEN(X'0000000000000BE0') ASID(X'0021')
0000000108948210: 0103D3C3 40000000 00000000 00000000 00000017 00000051 00000011 00000052 |..LC .....|

0000000108948DE0: Allocated storage element, length=0000000000000020.
To display: IP LIST 0000000108948DE0 LEN(X'0000000000000020') ASID(X'0021')
0000000108948DF0: 8489A56D A9859996 00000000 00000000 |div_zero.....|

0000000108948E00: Allocated storage element, length=0000000000000020.
To display: IP LIST 0000000108948E00 LEN(X'0000000000000020') ASID(X'0021')
0000000108948E10: 00000000 00000000 00000001 0000EE72 |.....|

0000000108948E20: Allocated storage element, length=0000000000000020.
To display: IP LIST 0000000108948E20 LEN(X'0000000000000020') ASID(X'0021')
0000000108948E30: 83859398 A2819497 00000000 00000000 |celqsamp.....|

0000000108948E40: Allocated storage element, length=00000000000000C0.

```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 4 of 10)

```

To display: IP LIST 0000000108948E40 LEN(X'0000000000000C0') ASID(X'0021')
0000000108948E50: 00000000 00000000 00000001 08943410 00000000 255CE000 00000000 255CE000 |.....m.....*.....*...|

0000000108948F00: Allocated storage element, length=0000000000000040.
To display: IP LIST 0000000108948F00 LEN(X'000000000000040') ASID(X'0021')
0000000108948F10: C403D3C8 C1C4D3D3 00000001 08948E50 00000001 089491F8 00000000 00000000 |DLLHADLL.....m.&.....mj8.....|

0000000108948F40: Allocated storage element, length=0000000000000080.
To display: IP LIST 0000000108948F40 LEN(X'000000000000080') ASID(X'0021')
0000000108948F50: C3C4D3D3 00000000 00000001 083010B0 80000000 00000000 00000000 00000000 |CDLL.....|

0000000108948FC0: Allocated storage element, length=0000000000000100.
To display: IP LIST 0000000108948FC0 LEN(X'0000000000000100') ASID(X'0021')
0000000108948FD0: 00000001 089490D0 00000001 08301210 00000001 08301270 00000001 083012D0 |....m.....|

00000001089490C0: Allocated storage element, length=0000000000000040.
To display: IP LIST 00000001089490C0 LEN(X'0000000000000040') ASID(X'0021')
00000001089490D0: 00000000 00000000 60004000 00000000 00000001 08948FD0 00000000 00000000 |.....-.....m.....|

0000000108949100: Allocated storage element, length=0000000000000040.
To display: IP LIST 0000000108949100 LEN(X'0000000000000040') ASID(X'0021')
0000000108949110: 00000001 089490D0 20004000 00000000 00000001 08949080 00000000 00000000 |....m....m.....|

0000000108949140: Allocated storage element, length=0000000000000040.
To display: IP LIST 0000000108949140 LEN(X'0000000000000040') ASID(X'0021')
0000000108949150: 00000000 00000000 20004000 00000000 00000001 08949090 00000000 00000000 |.....m.....|

0000000108949180: Allocated storage element, length=0000000000000040.
To display: IP LIST 0000000108949180 LEN(X'0000000000000040') ASID(X'0021')
0000000108949190: 00000001 00000020 00000001 089491A0 00000001 08948E30 00000000 00000000 |.....mj.....m.....|

00000001089491C0: Allocated storage element, length=0000000000000E40.
To display: IP LIST 00000001089491C0 LEN(X'0000000000000E40') ASID(X'0021')
00000001089491D0: D3D3E340 0E280001 00000000 00000000 00000000 00000000 00000000 00000000 |LLT .....|

000000010894A000: Allocated storage element, length=0000000000000040.
To display: IP LIST 000000010894A000 LEN(X'0000000000000040') ASID(X'0021')
000000010894A010: 00000000 00000000 00000001 08949098 20004000 00000000 00000000 00000000 |.....m.q.....|

000000010894A040: Allocated storage element, length=0000000000000040.
To display: IP LIST 000000010894A040 LEN(X'0000000000000040') ASID(X'0021')
000000010894A050: 00000000 00000000 00000001 089490A8 20004000 00000000 00000000 00000000 |.....m.y.....|

000000010894A080: Allocated storage element, length=0000000000000040.
To display: IP LIST 000000010894A080 LEN(X'0000000000000040') ASID(X'0021')
000000010894A090: 00000001 08949110 20004000 00000000 00000001 00003FAB 00000000 00000000 |....mj...y.....|

000000010894A0C0: Allocated storage element, length=0000000000000040.
To display: IP LIST 000000010894A0C0 LEN(X'0000000000000040') ASID(X'0021')
000000010894A0D0: 00000001 08949150 20004000 00000000 00000001 00003FC8 00000000 00000000 |....mj&..H.....|

000000010894A100: Allocated storage element, length=0000000000000040.
To display: IP LIST 000000010894A100 LEN(X'0000000000000040') ASID(X'0021')
000000010894A110: 00000000 00000000 00000001 00003FB8 20004000 00000000 00000000 00000000 |.....|

000000010894A140: Allocated storage element, length=000000000000014E0.
To display: IP LIST 000000010894A140 LEN(X'000000000000014E0') ASID(X'0021')
000000010894A150: C3C4D3C7 6DC8C4D9 00000001 0894A17C 00000001 0894A59C 00000001 0894B1FC |CDLG_HDR.....m.@.....mv.....m..|

000000010894B620: Allocated storage element, length=0000000000010060.
To display: IP LIST 000000010894B620 LEN(X'0000000000010060') ASID(X'0021')
000000010894B630: E2C7D3C7 00010020 00010040 00000000 00000001 0894B670 00000001 0895B670 |SGLG.....m.....n..|

000000010895B680: Allocated storage element, length=0000000000000EE0.
To display: IP LIST 000000010895B680 LEN(X'0000000000000EE0') ASID(X'0021')
000000010895B690: 00000000 00000020 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

000000010895C560: Allocated storage element, length=0000000000010060.

```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 5 of 10)

```

To display: IP LIST 000000010895C560 LEN(X'0000000000010060') ASID(X'0021')
000000010895C570: E2C7D3C7 00010020 00010040 00000000 00000001 0895C5B0 00000001 0896C5B0 |SGLG.....nE.....oE.|

000000010896C5C0: Allocated storage element, length=00000000000000EE0.
To display: IP LIST 000000010896C5C0 LEN(X'00000000000000EE0') ASID(X'0021')
000000010896C5D0: 00000000 00000020 00000000 00000000 00000000 00000000 00000000 |.....|

000000010896D4A0: Allocated storage element, length=0000000000000040.
To display: IP LIST 000000010896D4A0 LEN(X'0000000000000040') ASID(X'0021')
000000010896D4B0: 4EF0F0F0 F0F0F0F0 F0F2F5F5 C3C5F6F9 F0000000 00000000 00000000 00000000 |+00000000255CE690.....|

000000010896D4E0: Allocated storage element, length=0000000000000020.
To display: IP LIST 000000010896D4E0 LEN(X'0000000000000020') ASID(X'0021')
000000010896D4F0: F0F0F0F0 F0F0F0F0 F2F5F5C3 C5F6F9F0 |00000000255CE690 |

000000010896D500: Allocated storage element, length=0000000000000040.
To display: IP LIST 000000010896D500 LEN(X'0000000000000040') ASID(X'0021')
000000010896D510: 4EF0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F3 F0000000 00000000 00000000 00000000 |+0000000000000030.....|

000000010896D540: Free storage element, length=00000000000092AC0.
To display: IP LIST 000000010896D540 LEN(X'00000000000092AC0') ASID(X'0021')

Summary of analysis for Heap Segment 0000000108800000:
Amounts of identified storage: Free:00092AC0 Allocated:0016D500 Total:001FFFC0
Number of identified areas : Free: 1 Allocated: 34 Total: 35
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
This is the last heap segment in the current heap.

User Heap31 Control Blocks

HPCQ: 0000000100100198
+000000 EYE_CATCHER:HPCQ FIRST:00000001 00100198
+000010 LAST:00000001 00100198 INITSIZE:00000000 00008000
+000020 INCRSIZE:00000000 00008000 OPTIONS:40000000

HPSQ: 0000000100003818
+000000 BYTES_ALLOC:00000000 00000000
+000008 CURR_ALLOC:00000000 00000000 GET_REQ:00000000 00000000
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000000
+000028 FREEMAINS:00000000 00000000

Library Heap31 Control Blocks

HPCQ: 00000001001001C8
+000000 EYE_CATCHER:HPCQ FIRST:00000001 001001C8
+000010 LAST:00000001 001001C8 INITSIZE:00000000 00004000
+000020 INCRSIZE:00000000 00002000 OPTIONS:50000000

HPSQ: 00000001000038A8
+000000 BYTES_ALLOC:00000000 00000000
+000008 CURR_ALLOC:00000000 00000000 GET_REQ:00000000 00000000
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000000
+000028 FREEMAINS:00000000 00000000

User Heap24 Control Blocks

HPCQ: 00000001001001F8
+000000 EYE_CATCHER:HPCQ FIRST:00000001 001001F8
+000010 LAST:00000001 001001F8 INITSIZE:00000000 00001000
+000020 INCRSIZE:00000000 00001000 OPTIONS:30000000

HPSQ: 0000000100003848
+000000 BYTES_ALLOC:00000000 00000000
+000008 CURR_ALLOC:00000000 00000000 GET_REQ:00000000 00000000
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000000
+000028 FREEMAINS:00000000 00000000

```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 6 of 10)

Library Heap24 Control Blocks

```

HPCQ: 0000000100100228
+000000 EYE_CATCHER:HPCQ FIRST:00000001 00100228
+000010 LAST:00000001 00100228 INITSIZE:00000000 00002000
+000020 INCRSIZE:00000000 00001000 OPTIONS:30000000

HPSQ: 00000001000038D8
+000000 BYTES_ALLOC:00000000 00000000
+000008 CURR_ALLOC:00000000 00000000 GET_REQ:00000000 00000000
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000000
+000028 FREEMAINS:00000000 00000000

```

Library Thread Heap64 Control Blocks

```

HPCQ: 00000001001002A8
+000000 EYE_CATCHER:HPCQ FIRST:00000001 00100720
+000010 LAST:00000001 00100720 INITSIZE:00000000 00000001
+000020 INCRSIZE:00000000 00000001 OPTIONS:90000000

HPSQ: 0000000100003998
+000000 BYTES_ALLOC:00000000 00000340
+000008 CURR_ALLOC:00000000 00000340 GET_REQ:00000000 00000001
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000

THANQ: 0000000100100720
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001 001002A8
+000010 PREV:00000001 001002A8 HEAPID:00000001 001002A8
+000020 SEGMENT:00000001 19500000 SEG_LEN:00000000 00100000
HANQ: 0000000119500000

+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001 001002A8
+000020 SEGMENT:00000001 19500000 ROOT:00000001 19500380
+000030 SEG_LEN:00000000 00100000 ROOT_LEN:00000000 000FFC80

```

This is the last heap segment in the current heap

[1]Free Storage Tree for Heap Segment 00000001

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	0000000119500380	000000000000FFC80	00000000000000000	00000000000000000	00000000000000000	00000000000000000	00000000000000000

[2]Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000119500000 LEN(X'000000000100000') ASID(X'0021')

0000000119500040: Allocated storage element, length=0000000000000340.

To display: IP LIST 0000000119500040 LEN(X'0000000000000340') ASID(X'0021')

0000000119500050: D7C3C9C2 00000000 00000000 00000000 00000000 00010310 00000000 |PCIB.....|

0000000119500380: Free storage element, length=000000000000FFC80.

To display: IP LIST 0000000119500380 LEN(X'000000000000FFC80') ASID(X'0021')

Summary of analysis for Heap Segment 0000000119500000:

Amounts of identified storage: Free:000FFC80 Allocated:00000340 Total:000FFFC0

Number of identified areas : Free: 1 Allocated: 1 Total: 2

00000000 bytes of storage were not accounted for.

No errors were found while processing this heap segment.

This is the last heap segment in the current heap.

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 7 of 10)

I/O Heap64 Control Blocks

```
HPCQ: 00000001001002D8
+000000 EYE_CATCHER:HPCQ FIRST:00000001 001006F0
+000010 LAST:00000001 001006F0 INITSIZE:00000000 00000001
+000020 INCRSIZE:00000000 00000001 OPTIONS:90000000

HPSQ: 0000000100003908
+000000 BYTES_ALLOC:00000000 00010380
+000008 CURR_ALLOC:00000000 00010380 GET_REQ:00000000 00000003
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000

THANQ: 00000001001006F0
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001 001002D8
+000010 PREV:00000001 001002D8 HEAPID:00000001 001002D8
+000020 SEGMENT:00000001 19400000 SEG_LEN:00000000 00100000

HANQ: 0000000119400000
+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001 001002D8
+000020 SEGMENT:00000001 19400000 ROOT:00000001 194103C0
+000030 SEG_LEN:00000000 00100000 ROOT_LEN:00000000 000EFC40
```

This is the last heap segment in the current heap

[1]Free Storage Tree for Heap Segment 00000001

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	00000001194103C0	00000000000EFC40	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

[2]Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000119400000 LEN(X'000000000100000') ASID(X'0021')

```
0000000119400040: Allocated storage element, length=000000000000300.
To display: IP LIST 0000000119400040 LEN(X'000000000000300') ASID(X'0021')
0000000119400050: 00000001 19400120 00000000 00000001 00000000 00000000 00000000 00000000 |.....

0000000119400340: Allocated storage element, length=000000000000060.
To display: IP LIST 0000000119400340 LEN(X'000000000000060') ASID(X'0021')
0000000119400350: 94859496 99A84B84 81A38100 00000000 00000000 00000000 00000000 00000000 |memory.data.....

00000001194003A0: Allocated storage element, length=0000000000010020.
To display: IP LIST 00000001194003A0 LEN(X'0000000000010020') ASID(X'0021')
00000001194003B0: A2969485 408481A3 81A29694 85409496 99854084 81A38185 A5859540 94969985 |some datasome more dataeven more|

00000001194103C0: Free storage element, length=00000000000EFC40.
To display: IP LIST 00000001194103C0 LEN(X'00000000000EFC40') ASID(X'0021')
```

Summary of analysis for Heap Segment 0000000119400000:

Amounts of identified storage:	Free:000EFC40	Allocated:00010380	Total:000FFFC0
Number of identified areas	: Free: 1	Allocated: 3	Total: 4

00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
This is the last heap segment in the current heap.

I/O Heap31 Control Blocks

```
HPCQ: 0000000100100308
+000000 EYE_CATCHER:HPCQ FIRST:00000001 00100630
+000010 LAST:00000001 00100630 INITSIZE:00000000 00003000
+000020 INCRSIZE:00000000 00002000 OPTIONS:50000000

HPSQ: 0000000100003938
+000000 BYTES_ALLOC:00000000 000020F0
+000008 CURR_ALLOC:00000000 000020F0 GET_REQ:00000000 00000006
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000
```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 8 of 10)

```

    THANQ: 0000000100100630
+000000 EYE_CATCHER:THNQ  FLAGS:00000000  NEXT:00000001 00100308
+000010 PREV:00000001 00100308  HEAPID:00000001 00100308
+000020 SEGMENT:00000000 255CB000  SEG_LEN:00000000 00003000

    HANQ: 00000000255CB000
+000000 EYE_CATCHER:HANQ  FLAGS:00000000  HEAPID:00000001 00100308
+000020 SEGMENT:255CB000  ROOT:255CD130  SEG_LEN:00003000
+00002C ROOT_LEN:00000ED0

    This is the last heap segment in the current heap

    Free Storage Tree for Heap Segment 00000000

    Depth   Node      Node      Parent   Left      Right      Left      Right
           Address Length      Node      Node      Node      Length  Length
    0  255CD130  00000ED0  00000000  00000000  00000000  00000000  00000000

    Map of Heap Segment 00000000

    To display entire segment: IP LIST 00000000 LEN(X'00003000') ASID(X'0021')

    255CB040: Allocated storage element, length=00000368. To display: IP LIST 255CB040 LEN(X'00000368') ASID(X'0021')
    255CB048: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

    255CB3A8: Allocated storage element, length=00000370. To display: IP LIST 255CB3A8 LEN(X'00000370') ASID(X'0021')
    255CB3B0: C1C6C3C2 00000000 00000000 255CB3D8  AFCBAFCB 00000000 00000001 08301B90 |AFCB.....*.Q.....|

    255CB718: Allocated storage element, length=00001008. To display: IP LIST 255CB718 LEN(X'00001008') ASID(X'0021')
    255CB720: 99858396 998440F1 15998583 96998440  F2159985 83969984 40F31500 00000000 |record 1.record 2.record 3.....|

    255CC720: Allocated storage element, length=00000370. To display: IP LIST 255CC720 LEN(X'00000370') ASID(X'0021')
    255CC728: C1C6C3C2 00000000 00000000 255CC750  AFCBAFCB 00000000 00000001 08301C30 |AFCB.....*G&.....|

    255CCA90: Allocated storage element, length=00000350. To display: IP LIST 255CCA90 LEN(X'00000350') ASID(X'0021')
    255CCA98: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

    255CCDE0: Allocated storage element, length=00000350. To display: IP LIST 255CCDE0 LEN(X'00000350') ASID(X'0021')
    255CCDE8: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

    255CD130: Free storage element, length=00000ED0. To display: IP LIST 255CD130 LEN(X'00000ED0') ASID(X'0021')

    Summary of analysis for Heap Segment 00000000:
    Amounts of identified storage: Free:00000ED0  Allocated:000020F0  Total:00002FC0
    Number of identified areas : Free:      1  Allocated:      6  Total:      7
    00000000 bytes of storage were not accounted for.
    No errors were found while processing this heap segment.
    This is the last heap segment in the current heap.

    I/O Heap24 Control Blocks

    HPCQ: 0000000100100338
+000000 EYE_CATCHER:HPCQ  FIRST:00000001 00100660
+000010 LAST:00000001 00100660  INITSIZE:00000000 00001000
+000020 INCRSIZE:00000000 00001000  OPTIONS:30000000

    HPSQ: 0000000100003968
+000000 BYTES_ALLOC:00000000 000008F0
+000008 CURR_ALLOC:00000000 000008F0  GET_REQ:00000000 00000009
+000018 FREE_REQ:00000000 00000001  GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000

    THANQ: 0000000100100660
+000000 EYE_CATCHER:THNQ  FLAGS:00000000  NEXT:00000001 00100338
+000010 PREV:00000001 00100338  HEAPID:00000001 00100338
+000020 SEGMENT:00000000 0000D000  SEG_LEN:00000000 00001000

```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 9 of 10)

```

HANQ: 0000000000000000
+000000 EYE_CATCHER: HANQ  FLAGS:00000000  HEAPID:00000001 00100338
+000020 SEGMENT:00000000  ROOT:0000D930  SEG_LEN:00001000
+00002C ROOT_LEN:000006D0

This is the last heap segment in the current heap

[1]Free Storage Tree for Heap Segment 00000000

Depth  Node      Node      Parent   Left      Right   Left      Right
      Address  Length  Node      Node      Node      Length   Length
0  0000D930  000006D0  00000000  00000000  00000000  00000000  00000000

[2]  Map of Heap Segment 00000000

To display entire segment: IP LIST 00000000 LEN(X'00001000') ASID(X'0021')

0000D040: Allocated storage element, length=00000088. To display: IP LIST 0000D040 LEN(X'00000088') ASID(X'0021')
0000D048: E3C5D9D4 00000000 00000000 0000D318 00000000 00000027 00000000 00000000 |TERM.....L.....|

0000D0C8: Allocated storage element, length=00000248. To display: IP LIST 0000D0C8 LEN(X'00000248') ASID(X'0021')
0000D0D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

0000D310: Allocated storage element, length=00000058. To display: IP LIST 0000D310 LEN(X'00000058') ASID(X'0021')
0000D318: 6E6E6E40 E3889985 81846D86 A495837A 40E38899 85818440 F1409396 83928995 |>>> Thread_func: Thread 1 lockin|

0000D368: Allocated storage element, length=00000070. To display: IP LIST 0000D368 LEN(X'00000070') ASID(X'0021')
0000D370: E3C5D9D4 00000000 00000000 0000D8E0 00000000 00000047 00000000 00000000 |TERM.....Q.....|

0000D3D8: Allocated storage element, length=00000070. To display: IP LIST 0000D3D8 LEN(X'00000070') ASID(X'0021')
0000D3E0: E3C5D9D4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |TERM.....|

0000D448: Allocated storage element, length=00000248. To display: IP LIST 0000D448 LEN(X'00000248') ASID(X'0021')
0000D450: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

0000D690: Allocated storage element, length=00000248. To display: IP LIST 0000D690 LEN(X'00000248') ASID(X'0021')
0000D698: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

0000D8D8: Allocated storage element, length=00000058. To display: IP LIST 0000D8D8 LEN(X'00000058') ASID(X'0021')
0000D8E0: 40404040 40404040 4081A340 8595A399 A8409686 86A285A3 404EF0F0 F0F0F0F0 | at entry offset +000000|

0000D930: Free storage element, length=000006D0. To display: IP LIST 0000D930 LEN(X'000006D0') ASID(X'0021')

Summary of analysis for Heap Segment 00000000:
Amounts of identified storage: Free:000006D0 Allocated:000008F0 Total:00000FC0
Number of identified areas : Free: 1 Allocated: 8 Total: 9
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
This is the last heap segment in the current heap.

```

Figure 126. Example formatted detailed heap segment report from LEDATA Verbexit (Part 10 of 10)

Heap report sections of the LEDATA output

The Heap Report sections of the LEDATA output provide information for each heap segment in the dump. The detailed heap segment reports include information on the free storage tree in the heap segments, the allocated storage elements, and the cause of heap management data structure problems.

[1]Free Storage Tree Report

Within each heap segment, Language Environment keeps track of unallocated storage areas by chaining them together into a tree. Each free area represents a node in the tree. Each node contains a header, which points to its left and right child nodes. The header also contains the length of each child.

The LEDATA HEAP option formats the free storage tree within each heap, and validates all node addresses and lengths within each node. Each node address is validated to ensure that it:

- Falls on a doubleword boundary
- Falls within the current heap segment
- Does not point to itself
- Does not point to a node that was previously traversed

Each node length is validated to ensure that it:

- Is a multiple of 8
- Is not larger than the heap segment length
- Does not cause the end of the node to fall outside of the current heap segment
- Does not cause the node to overlap another node

If the formatter finds a problem, then it will place an error message describing the problem directly after the formatted line of the node that failed validation

[2]Heap Segment Map Report

The LEDATA HEAP option produces a report that lists all of the storage areas within each heap segment, and identifies the area as either allocated or freed. For each allocated area the contents of the first X'20' bytes of the area are displayed in order to help identify the reason for the storage allocation.

Each allocated storage element has a prefix used by Language Environment to manage the area. The prefix contains a pointer to the start of the heap segment followed by the length of the allocated storage element. For HEAP64 heaps, the prefix is 16 bytes, with 8-byte pointer and length fields. For HEAP31 and HEAP24 heaps, the pointer is 8 bytes with 4-byte pointer and length field. The formatter validates this header to ensure that its heap segment pointer is valid. The length is also validated to ensure that it:

- Is a multiple of 8
- Is not zero
- Is not larger than the heap segment length
- Does not cause the end of the element to fall outside of the current heap segment
- Does not cause the element to overlap a free storage node

If the heap_free_value of the STORAGE run-time option was specified, then the formatter also checks that the free storage within each free storage element is set to the requested heap_free_value. If a problem is found, then an error message describing the problem is placed after the formatted line of the storage element that failed validation.

Diagnosing heap damage problems

Heap storage errors can occur when an application allocates a heap storage element that is too small for it to use, and therefore, accidentally overlays heap storage. If this situation occurs then some of the typical error messages generated are:

- The node address does not represent a valid node within the heap segment
- The length of the segment is not valid, or
- The heap segment pointer is not valid.

If one of the above error messages is generated by one of the reports, then examine the storage element that immediately precedes the damaged node to determine if this storage element is owned by the application program. Check the size of the storage element and ensure that it is sufficient for the program's use. If the size of the storage element is not sufficient then adjust the allocation size.

If an error occurs indicating that the node's pointers form a circular loop within the free storage tree, then check the Free Storage Tree Report to see if such a loop exists. If a loop exists, then contact the IBM support center for assistance because this may be a problem in the Language Environment heap management routines.

Additional diagnostic information regarding heap damage can be obtained by using the HEAPCHK run-time option. This option provides a more accurate time perspective on when the heap damage actually occurred, which could help to determine the program that caused the damage. For more information on HEAPCHK, see *z/OS Language Environment Programming Reference*.

Diagnosing storage leak problems

A storage leak occurs when a program does not return storage back to the heap after it has finished using it. To determine if this problem exists, do one of the following:

- The *call-level* suboption of the HEAPCHK run-time option causes a report to be produced in the CEEDUMP. Any still-allocated (that is, not freed) storage identified by HEAPCHK is listed in the report, along with the corresponding traceback. This shows any storage that wasn't freed, as well as all the calls that were involved in allocating the storage. For more information about the HEAPCHK run-time option, see *z/OS Language Environment Programming Reference*.
- Examine the Heap Segment Map report to see if any data areas, within the allocated storage elements, appear more frequently than expected. If they do, then check to see if these data areas are still being used by the application program. If the data areas are not being used, then change the program to free the storage element after it is done with it.

Diagnosing heap fragmentation problems

Heap fragmentation occurs when allocated storage is interlaced with many free storage areas that are too small for the application to use. Heap fragmentation could indicate that the application is not making efficient use of its heap storage. Check the Heap Segment Map report for frequent free storage elements that are interspersed with the allocated storage elements.

Understanding the HEAPPOOLS trace LEDATA output

The Language Environment IPCS Verbexit LEDATA generates a detailed HEAPPOOLS trace report when the HPT option is used. The argument *value* is the id of the pool to be formatted in the report.

```
*****
64 BIT LANGUAGE ENVIRONMENT DATA
*****
```

Language Environment Product 04 V01 R06.00

[1]Heap Pool Trace Table

[2]POOLID: 00000005 ASID: 0022 AVAILABLE ENTRIES: 5 OF 5

[3]Timestamp: 2003/10/21 15:02:19.410583

Type: FREE Cell Address: 00000001083004C0 CpuId: 01 Tcb: 008F90E8

[4]CALL NAME	CALL ADDRESS	CALL OFFSET
CELQVFQT	000000002556B5A0	00000000
foo8	00000000255019C8	00000046
foo7	0000000025501A30	0000002E
foo6	0000000025501A98	0000002E
foo5	0000000025501B00	0000002E
foo4	0000000025501B68	0000002E
foo3	0000000025501BD0	0000002E
foo2	0000000025501C38	0000002E
foo1	0000000025501CA0	0000002E
main	0000000025501768	00000000

Timestamp: 2003/10/21 15:02:19.410580

Type: FREE Cell Address: 00000001083008D0 CpuId: 01 Tcb: 008F90E8

CALL NAME	CALL ADDRESS	CALL OFFSET
CELQVFQT	000000002556B5A0	00000000
foo9	0000000025501960	00000046
foo8	00000000255019C8	0000002E
foo7	0000000025501A30	0000002E
foo6	0000000025501A98	0000002E
foo5	0000000025501B00	0000002E
foo4	0000000025501B68	0000002E
foo3	0000000025501BD0	0000002E
foo2	0000000025501C38	0000002E
foo1	0000000025501CA0	00000000

Timestamp: 2003/10/21 15:02:19.410565

Type: GET Cell Address: 00000001083008D0 CpuId: 01 Tcb: 008F90E8

CALL NAME	CALL ADDRESS	CALL OFFSET
CELQVGQT	000000002556FF30	00000000
foo9	0000000025501960	0000001A
foo8	00000000255019C8	0000002E
foo7	0000000025501A30	0000002E
foo6	0000000025501A98	0000002E
foo5	0000000025501B00	0000002E
foo4	0000000025501B68	0000002E
foo3	0000000025501BD0	0000002E
foo2	0000000025501C38	0000002E
foo1	0000000025501CA0	00000000

Figure 127. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 1 of 2)

```

Timestamp: 2003/10/21      15:02:19.410562
Type: GET   Cell Address: 00000001083004C0   CpuId: 01   Tcb: 008F90E8
CALL NAME           CALL ADDRESS           CALL OFFSET
CELQVGQT            000000002556FF30      00000000
foo8                00000000255019C8      0000001A
foo7                0000000025501A30      0000002E
foo6                0000000025501A98      0000002E
foo5                0000000025501B00      0000002E
foo4                0000000025501B68      0000002E
foo3                0000000025501BD0      0000002E
foo2                0000000025501C38      0000002E
foo1                0000000025501CA0      0000002E
main                0000000025501768      00000000

```

```

Timestamp: 2003/10/21      15:02:19.410362
Type: GET   Cell Address: 00000001083000B0   CpuId: 01   Tcb: 008F90E8
CALL NAME           CALL ADDRESS           CALL OFFSET
CELQVGQT            000000002556FF30      00000000
set_locale          000000002565F8C0      0000011C
setlocale           00000000256613A0      0000001A
tzset               00000000258559D0      00000658
qinc_m              00000000255523F8      000011C2
CELQINMN            0000000025558458      00000B88
CELQINIT            0000000025523010      00000000

```

Exiting Language Environment Data

Figure 127. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 2 of 2)

[1] Trace Header

HEAPPOOLS trace header information.

[2] Pool Information

Information includes the number of the pool (POOLID) which is currently being formatted, the ASID, and the number of entries formatted and the total number of entries taken.

Note: The trace wraps for each poolid after 1024 enties have been taken.

[3] Timestamp

The time this trace entry was taken.

Note: The trace entries are formatted in reverse order (most recent trace entry first).

[4] Trace Table Entry contents

The individual trace entry:

- The TYPE - GET or FREE.
- The Cell within the pool being acted upon.
- The CPU and TCB which requested or freed the cell.
- A traceback at the time of the request. The number of entries in this traceback is limited by the HEAPCHK run-time option.

Understanding the C/C++-specific LEDATA output

The Language Environment IPCS Verbexit LEDATA generates formatted output of C/C++-specific control blocks from a system dump when the ALL parameter is specified and C/C++ is active in the dump. Figure 128 on page 342 illustrates the C/C++-specific output produced. The system dump being formatted was obtained by specifying the TERMTHDACT(UADUMP) run-time option when running the program CELQSAMP Figure 5 on page 44. “C/C++-specific sections of the LEDATA output” on page 348 describes the information contained in the formatted output. Ellipses are used to summarize some sections of the dump.

For easy reference, the sections of the dump are numbered to correspond with the description of each section that follows.

```

*****
64 BIT CRTL ENVIRONMENT DATA
*****

[1] CGEN: 0000000100006160
+000310 CGENE:00000001 00009FB0      CREDIT:00000000 00000000
+000320 CPRMS:00000001 00003C28      TRACE:00000001
+000330 CTHD:00000001 000074F8      CURR_FECB:00000001 000098D0
+000340 CGEN_CPCB:00000001 00006CD0  CGEN_CEDB:00000001 00008B30
+000350 CFLG3:00      CIO:00000001 00006EC8

[2] CGENE: 0000000100009FB0
+000000 CGENEYE:.-./      CGENESIZE:00C200C4
+000008 CGENPTR:007C00C1 00C300C5      CERRNO:006000A3
+000100 TEMPLONG:00E000A6 00E200E3      AMRC:00E400E5 00E600E7
+000110 STDINFILE:00E800E9 00F200E3      STDOUTFILE:00E500D9 00E200E4
+000120 STDERRFILE:00F000F1 00F200F3      CTYPE:00F400F5 00F600F7
+000138 LC_CTYPE:00E000E8 00E9001F      LC_CHARMAP:089481D0 00000000
+00052C MIN_FLT:00000000 00000000 00000000 00000000
+00053C MAX_FLT:00000000 00000000 00000000 00000000
+00054C FLT_EPS:00000000      DBL_EPS:00000000 00000000
+00055C LDBL_EPS:00000000 00000000 00000000 C7C5D5C5
+000574 IMSPCBLIST:0000A518 00000000      ADDRtbl:00000000 00000001
+000710 ABND_CODE:00000000      RSN_CODE:00000000

[3] CEDB: 0000000100008B30
+000000 EYE:CEDB      SIZE:0000C08      PTR:00000001 00008B30
+000010 CLLST:00000000 251024B0      CEELANG:0003      CASWITCH:0000
+000020 CLWA:00000001 0000A558      CALTLWA:00000000 00000000
+000030 CCADDR:00000000 251010C8      CFLGS:00000000
+000040 CANCHOR:00000000 25101000      RPLLEN:00000000 00000000
+000050 ACBLEN:00000000 00000000      LC:00000000 00000000
+000060 VALID_HIGH:00000001 00009740      _LOW:00000000 25140858
+000070 HEAD_FECB:00000000 251402F8      ATEXIT_CHAIN:00000000 00000000
+000080 _EMPTY_CHAIN:00000000 00000000      MAINPRMS:00000001 00008C80
+000090 STDINFILE:00000001 08949190      STDOUTFILE:00000001 000087C8
+0000A0 STDERRFILE:00000001 000080F8      CTYPE:00000001 00008460
+0000B0 TZDFTL:00000000 255AA936      CINFO:00000000 00003840
+0000C0 CMS_WRITE_DISK:0000      DISK_SET:00009840
+0000C8 MIN_FLT:40400000 00000000 00100000 00000000
+0000D8 MAX_FLT:00000000 00000000 7FFFFFFF FFFFFFFF
+0000E8 FLT_EPS:71FFFFFF      DBL_EPS:3C100000 00000000
+0000F8 LDBL_EPS:34100000 00000000 26100000 00000000      FLAGS1:18000000
+000110 MTF_MAINTASK_BLK:02480000 00000000      EMSG_SETTING:00
+000119 DEPTH:00000000      SCREEN_WIDTH:00000000      USERID:.....
+00012D HEAP24_ANCHOR:C5C7C140 40400000      TCIC:00000000 00000000
+00013D TKCLI:00000000 00000000
+000145 ATEXIT_FUNCS01:00000000 00000000 00000000 00000100 008CA800 00000000 00000000 00000000 00
+00016D ATEXIT_FUNCS02:00000000 00000000 00000000 00000100 008CD000 00000000 00000000 00000000 00
+000195 ATEXIT_FUNCS03:00000000 00000000 00000000 00000100 008CF800 00000000 00000000 00000000 00
+0001BD ATEXIT_FUNCS04:00000000 00000000 00000000 00000100 008D2000 00000000 00000000 00000000 00
+0001E5 ATEXIT_FUNCS05:00000000 00000000 00000000 00000100 008D4800 00000000 00000000 00000000 00
+00020D ATEXIT_FUNCS06:00000000 00000000 00000000 00000100 008D7000 00000000 00000000 00000000 00
+000235 ATEXIT_FUNCS07:00000000 00000000 00000000 00000100 008D9800 00000000 00000000 00000000 00
+00025D ATEXIT_FUNCS08:00000000 00000000 00000000 00000100 008DC000 00000000 00000000 00000000 00
+000285 ATEXIT_FUNCS09:00000000 00000000 00000000 00000100 008DE800 00000000 00000000 00000000 00
+0002AD ATEXIT_FUNCS10:00000000 00000000 00000000 00000100 008E1000 00000000 00000000 00000000 00
+0002D5 ATEXIT_FUNCS11:00000000 00000000 00000000 00000100 008E3800 00000000 00000000 00000000 00
+0002FD ATEXIT_FUNCS12:00000000 00000000 00000000 00000100 008E6000 00000000 00000000 00000000 00
+000325 ATEXIT_FUNCS13:00000000 00000000 00000000 00000100 008E8800 00000000 00000000 00000000 00
+00034D ATEXIT_FUNCS14:00000000 00000000 00000000 00000100 008EB000 00000000 00000000 00000000 00
+000375 ATEXIT_FUNCS15:00000000 00000000 00000000 00000100 008ED800 00000000 00000000 00000000 00
+00039D ATEXIT_FUNCS16:00000000 00000000 00000000 00000100 008F0000 00000000 00000000 00000000 00
+0003C5 ATEXIT_FUNCS17:00000000 00000000 00000000 00000100 008F2800 00000000 00000000 00000000 00
+0003ED ATEXIT_FUNCS18:00000000 00000000 00000000 00000100 008F5000 00000000 00000000 00000000 00
+000415 ATEXIT_FUNCS19:00000000 00000000 00000000 00000100 008F7800 00000000 00000000 00000000 00
+00043D ATEXIT_FUNCS20:00000000 00000000 00000000 00000100 008FA000 00000000 00000000 00000000 00
+000465 ATEXIT_FUNCS21:00000000 00000000 00000000 00000100 008FC800 00000000 00000000 00000000 00
+00048D ATEXIT_FUNCS22:00000000 00000000 00000000 00000100 008FF000 00000000 00000000 00000000 00

```

Figure 128. Example formatted C/C++ output from LEDATA Verbexit (Part 1 of 7)


```

+0004B5 ATEXT_FUNC23:00000000 00000000 00000000 00000100 00901800 00000000 00000000 00000000 00
+0004D0 ATEXT_FUNC24:00000000 00000000 00000000 00000100 00904000 00000000 00000000 00000000 00
+000505 ATEXT_FUNC25:00000000 00000000 00000000 00000100 00906800 00000000 00000000 00000000 00
+00052D ATEXT_FUNC26:00000000 00000000 00000000 00000100 00909000 00000000 00000000 00000000 00
+000555 ATEXT_FUNC27:00000000 00000000 00000000 00000100 0090B800 00000000 00000000 00000000 00
+00057D ATEXT_FUNC28:00000000 00000000 00000000 00000100 0090E000 00000000 00000000 00000000 00
+0005A5 ATEXT_FUNC29:00000000 00000000 00000000 00000100 00910800 00000000 00000000 00000000 00
+0005CD ATEXT_FUNC30:00000000 00000000 00000000 00000100 00913000 00000000 00000000 00000000 00
+0005F5 ATEXT_FUNC31:00000000 00000000 00000000 00000100 00915800 00000000 00000000 00000000 00
+00061D ATEXT_FUNC32:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00
+000645 HEAD_FOREIGN_FECB:00000000 00000000
+00064D SNAP_DUMP_COUNT:00000000 ENVIRON:00000000 00000000
+00065D GETENV_BUF:00000000 00000000 _BUF_LEN:00000000 00000000
+00066D CDLL:00000000 00000000 INSPECT_GLOBALS:00000000 00000108
+00067D _JMP_BUF:948F5000 00000000 BACK_END:00000000 00000000
+00068D _FLAGS:00000000 _TAB:00000000 00000000
+00069D INTOFFLIST:00000000 00000000
+0006A5 CGEN_CRENT:00000000 00000000 CPRMS:00000000 00000000
+0006B5 _CEDCXV:00000000 00000100 _CEDCOV:003C2800 00000000
+0006C5 _EPCBLIST:00000000 00000000 CAA_ADDR:00000000 00000000
+0006DD USERIDLENGTH:00616000 00000000
+0006E5 TZSHR:00000000 00000000 00000400 00000100
+0006F5 MAXUNGETCOUNT:0099 RWSTATIC:0099F800 04000000
+000705 RWLEN:00000000 00000108 CSGDLLI:3005000 00000000
+000715 DLLISIZE:00014000 00000000 IOGET_ANY:00000000 00000000
+000725 _BELOW:00000000 00000025 IOFREE_ANY:5A86B000 00000025
+000735 _BELOW:5A86A000 00000025 CSGSTINIT:5A869000 00000025
+000745 STINITSIZE:5A868000 00000000
+00074D MTFMAINTASKBLK:00000000 00000000 SIGTABLE:00000000 00000000
+00075D INIT_STDIN:00000000 00000100 _STDOUT:009B3800 00000100
+00076D _STDERR:0087C800 00000100 TABNUM:00846000 00000000
+000785 REDIR:00 OPENMYS_FLAGS:00 MRPSTDR:00000000 00000000
+0007A1 MWPSTDR:00000000 00000000 MRPSTDC:00000025 5A864000
+0007B1 MWPSTDC:00000025 5A863000 OWRP1:00000000 00000000
+0007C1 OWRP3:00000000 00000000 STATIC_EDCOV:00000000 00000000
+0007D1 GETENV_BUF2:00000000 00000000 _BUF2_LEN:00000000 00000000
+0007E1 DLCD_MUTEX:00000000 00000000 _CONDV:00000000 00000000
+0007F1 EDCOV:00000108 9490A000 LCX:00000108 9490A800
+000801 MUTEX_ATTR:00000000 00000000 STOR_INIT_B:00000100
+00080D _INCR_B:0099F000 STOR_INIT:00000108 _INCR:948FD000
+000821 DEMANGLE:00300000 00200000 TEMPR15:00000000 00000000
+000831 TERMINATE:00000000 00000000 CXX_INV:00000000 00000000
+000849 D4_JOIN_MUTEX_ATTR:00000000 00000000
+000851 _MUTEX:00000000 00000000 _CONDV_ATTR:00000108 94908000
+000861 _CONDV:00000108 94908800 DLLANCHOR:00000108 94909000
+000871 DLLLAST:00000108 94909800 MEM24P:00000000 00000000
+000881 RTLMutex_ARRAYPTR:00000000 00000000
+000889 MSGCATLIST:00000000 00000000 SRCHP:00000108 948FD800
+000899 ETOAP:00000000 00000000 ATOEP:00000000 00000000
+0008A9 NDMGMP:00000000 00000000 POPENP:00000000 00000000
+0008B9 RND48P:00000000 00000000 BRK_HEAPID:00000000 00000000
+0008C9 _START:00000000 00000000 CURRENT:00000000 00000000
+0008D9 _END:00000000 00000000 RESTARTTABLE:00000000 00000000
+0008F1 SYSLOGP:00000000 00000000 LOGIN_NAME:.....
+000905 PREV_UMASK_VAL:00000000
+000909 HFP_LDBL_LMS:00000000 00000000 00000000 00000000 10000000 00000000 00000000 0000007F FF
+000939 BFP_LDBL_LMS:10000000 00000018 00000000 00000000 01000000 00000000 00000000 0000007F FE
+0009C9 HFP_DBL_LMS:FF800000 00000000 00000000 00000000 10000000 0000007F
+0009E1 BFP_DBL_LMS:FFFFFFFF FFFFFFFF34 10000000 00000000 10000000 0000007F EFFFFFFF FFFFFFFF3C B0
+000A29 HFP_MHDC:F8000000 000000FF F8000000 00000040 05BF0A8B 1457693F F7154765 2B82FE3F DB
+000A99 BFP_MHDC:F6A09E66 7F3BCD3F E6A09E66 7F3BCD40 05BF0A8B 1457693F F7154765 2B82FE3F DB
+000B09 HFP_FLT_LMS:F6A09E66 7F3BCD3F E6A09E66
+000B15 BFP_FLT_LMS:7F3BCD00 1000007F FFFFFFFF3C 10000000 8000007F 7FFFFFFF34 0000007F 800000FF 80
+000B39 HFP_FHDC:A00000FF A000007F C00000FF C0000000 00001000 06000E00 1C000600 0F0020FF C0
+000B61 BFP_FHDC:B2FFB200 3F003F00 3F004B00 4B004B00 01000200 18003500 71000600 0F0021FF 83
+000B89 ASCII_CCSD:CODEC EBDCID_CCSD:BD00 LOGIN_NAME_A:... ..
+000B99 CINFO_A:33041700 00000000 LC_C:00000000 00000000

```

Figure 128. Example formatted C/C++ output from LEDATA Verbexit (Part 2 of 7)

```

+000BB9 CORRESTABLE:00000100 00A17000
+000BC1 TZSHR_A:00000100 00A2E800 00000000 00000000

[4] CTHD: 00000001000074F8
+000000 CTHDEYE:CTHD SIZE:00000528 CTHDPTR:00000001 000074F8
+000010 STORPTR:00000000 00000000 TOKPTR:00000000 25140A60
+000020 ASCTIME_RESULT:.....
+00003A SNAP_DUMP_FLAG:00 FP_MODE:C4
+000040 GMTIME_BKDN:00000001 00007B60 TIMECALLED:00000000
+00004C DATECALLED:00000000 DTCALLED:00000000
+000054 LOC_CALLED:00000000 DOFMT0_DISCARDS:00000000 00000000
+000060 CERRNO:00000000 AMRC:00000000 255CB278
+000070 AMRC2:00000000 255CB360 GDATE:00000000 00000000
+000080 OPTARGV:00000000 00000000 OPTERRV:00000001
+00008C OPTINDV:00000001 OPTOPTV:00000000
+000094 OPTSIND:00000000 DLGHTV:00000000
+0000A0 TZONEV:00000000 00000000 GTDERRV:00000000
+0000B0 OPTARGP:00000001 00007578 OPTERRP:00000001 00007580
+0000C0 OPTINDP:00000001 00007584 OPTOPTP:00000001 00007588
+0000D0 DLGHTP:00000001 00007590 TZONEP:00000001 00007598
+0000E0 GTDERRP:00000001 000075A0 RNDSTGP:00000000 00000000
+0000F0 LOCNAME:00000000 00000000 ENCRYPTP:00000000 00000000
+000100 CRYPTP:00000000 00000000 RND48P:00000000 00000000
+000110 L64AP:00000000 00000000 WCSTOKP:00000000 00000000
+000120 CUSERP:00000000 00000000 GPASSP:00000000 00000000
+000130 UTMXP:00000000 00000000 NDMGMP:00000000 00000000
+000140 RECOMP:00000000 00000000 STACKPTR:00000000 00000000
+000150 STACKSIZE:00000000 00000000 STACKFLAGS:40 MCVTP:00000000
+000160 H_ERRNO:00000000 SD:FFFFFFFF
+000168 HOSTENT_DATA_P:00000000 00000000 HOSTENT_P:00000000 00000000
+000178 NETENT_DATA_P:00000000 00000000 NETENT_P:00000000 00000000
+000188 PROTOENT_DATA_P:00000000 00000000
+000190 PROTOENT_P:00000000 00000000
+000198 SERVENT_DATA_P:00000000 00000000 SERVENT_P:00000000 00000000
+0001A8 NTOA_BUF:..... _LOC1V:00000000 00000000
+0001C8 _LOCSV:00000000 00000000 HERRNOP:00000001 00007658
+0001D8 _LOC1P:00000000 00000000 REXECP:00000000 00000000
+0001E8 CXCEPTION:00000000 00000000 TEMPDCBE:00000000 255CB068
+0001F8 T_ERRNOV:00000000 T_ERRNOP:00000001 000076F0
+000208 _LOC1P:00000000 00000000 _loc2p:00000000 00000000
+000218 _locsp:00000000 00000000 _loc1v:00000000 00000000
+000228 _loc2v:00000000 00000000 THD_STORAGE:00000000 00000000
+000238 CONTEXT_LINK:00000000 00000000 FLAGS1:00000000
+000248 LABEL_VAR:00000001 00007C90 ABND_CODE:00000000
+000254 RSN_CODE:00000000 STRFTIME_ERADTCALLED:00000000
+00025C STRFTIME_ERADATECALLED:00000000
+000260 STRFTIME_ERATIMECALLED:00000000
+000264 STRFTIME_ERAYEARCALLED:00000000 MBLEN_STATE:0000
+00026A MBRTOWC_STATE:0000 WCRTOB_STATE:0000
+00026E MBSRTOWCS_STATE:0000 WCSRTOB_STATE:0000 MBLEN_STATE:0000
+000274 MBTOWC_STATE:0000 CURR_HEAP_ID:00000000
+000280 CURR_CAA:00000000 00000000 CURR_MOD_HANDLE:00000000 00000000
+000290 CURR_BMR:00000000 00000000 CU_LIST:00000000 00000000
+0002A0 CURR_STATUS:00 RAND_NEXT:00000000 00000001
+0002B0 STRERRORBUF:00000001 00007280 TMPAREA:00000000 00000000
+0002C0 IOWORKAREA:00000000 255CB140 TEMPDCB:00000000 00000000
+0002D0 TEMPJFCB:00000000 0000D130 TEMPDCB:00000000 255CB0A0
+0002E0 NAMEBUF:00000000 00000000 ERRNO_JR:00000000
+0002F0 RET_STRUCT:00000000 00000000 BKDN_IS_LOCALTIME:00000000
+0002FC SWPRINTF_SIZE:00000000 SWPRINTF_BUF:00000000 00000000
+000308 S99P:255CB048 MUTEXCTARRAY:00000001 00007EE8
+000318 STRFTIME_ERANAMECALLED:00000000 DLL_LOADLEVEL:00000000
+000340 _CONSTLIST:00000000 00000000 FCB_MUTEX:00000000 00000000
+000350 HSPABHWA:00000000 00000000 MUTEX_SAVE:00000001 00007F98
+000368 INITIAL_CPU_TIME:40C08158 00000000 FCB_MUTEX_OK:00000001

```

Figure 128. Example formatted C/C++ output from LEDATA Verbexit (Part 3 of 7)

```

+000378 FCB_MUTEX_SAVE:00000000 00000000
+000380 ENTRY_ADDRTABLESIZE:00000000 ADDRESS:00000000
+000388 NUMBEROFNAMES:00000000 NAMES1:.....
+0003A5 NAMES2:.....
+0003BE NAMES3:.....
+0003D7 NAMES4:.....
+0003F0 NAMES5:.....
+000409 NAMES6:.....
+000424 ENTRY_SITETABLESIZE:00000000 KIND:00
+00042C NUM_ADDRS:00000000
+000430 ADDRESSES:00000000 00000000 00000000 00000000 00000000 00000000
+000448 NAME:00000000 00000000 00000000 00000000 00000000 00000000
+000460 T_STRErrorBUF:00000001 000073AC
+000468 CTHD_ourFDSET:00000000 00000000 IEEECWAP:00000000 00000000
+000488 RETVAL_P:00000000 00000000

[5]   CPCB: 0000000100006CD0
+000000 CPCB_EYE:CPCB CPCB_SIZE:00000068
+000008 CPCB_PTR:00000000 00000000 FLAGS1:40000000
+000014 TTKNHDR:00000000 00000000 TTKN:00000000 00000000
+000024 FOOTPRINT:00000000 00000001 CODE370:00000B30 00000000
+000034 CIO:00000000 00000001 Reuse:00006EC8
+000044 _RSAbove:00000000 00000001 _RSAboveLen:00006CD0
+000054 _RSBelow:00000000 00000000 _RSBelowLen:00000000

[6]   CIO: 0000000100006EC8
+000000 EYE:CIO SIZE:00000108 PTR:00000000 00000000 FLG1:01
+000011 FLG2:00 FLG3:00 FLG4:00 DUMMYF:00000001 00006FD0
+000020 EDCZ24:00000000 00000000 FCBSTART:00000001 00008118
+000030 DUMMYFCB:00000001 00006FF8 MFCBSTART:00000001 19400050
+000040 IOANYLIST:00000000 00000000 IOBELOWLIST:00000000 00000000
+000050 FCBDLIST:00000000 00000000 PERRORBUF:00000001 00006D90
+000060 TMPCounter:00000000 00000000 TEMPmem:00000000
+000070 PROMPTBUF:00000000 00000000 IO24:00000000 00000000
+000080 IOEXITS:00000000 00000000 TERMINALCHAIN:00000001 000087E8
+000090 VANCHOR:00000000 00000000 XTI:00000000 00000000
+0000A0 ENOWP24:00000000 00000000 MAXNUMDESCRPS:00000000 00000000
+0000B0 DESCARRAY:00000000 00000000 TEMPFILENUM:00000000
+0000C8 CSS:00000000 00000000 DUMMY_NAME:.....
+0000D8 HOSTNAME_CACHE:00000000 00000000
+0000E0 HOSTADDR_CACHE:00000000 00000000 IO31:00000000 00000000
+0000F0 LAST_FD_CLOSE:00000000 00000000 IOGET64:00000000 255A8660
+000100 IOFREE64:00000000 255A8650

[7]   File name: *

FCB: 0000000100008118
+000000 BUFPTR:00000000 0000D318 COUNTIN:00000000 00000000
+000010 COUNTOUT:00000000 0000004E READFUNC:00000000 255A8C40
+000020 WRITEFUNC:00000000 255A8D20 FLAGS1:8000 DEPTH:0000
+000030 NAME:00000001 00008410 _LENGTH:00000000 00000001
+000040 _BUFSIZE:00000000 00000048 MEMBER:.....
+000050 NEXT:00000000 255CC750 PREV:00000000 00000000
+000060 PARENT:00000001 00008118 CHILD:00000001 000087E8
+000070 DDNAME:..... FD:FFFFFFFF DEVTYPE:01 FCBTYPE:0071
+000080 FSCE:00000000 0000D048 UNGETBUF:00000001 00008300
+000090 REPOS:00000000 255A8D40 GETPOS:00000000 255A9690
+0000A0 CLOSE:00000000 255A9670 FLUSH:00000000 255A8D50
+0000B0 UTILITY:00000000 255A9660 USERBUF:00000000 00000000
+0000C0 LRECL:00000000 0000004E BLKSIZE:00000000 0000004E
+0000D0 REALBUFPTR:00000000 0000D318
+0000E0 UNGETCOUNT:00000000 00000000 BUFSIZE:00000000 0000004F
+0000F0 BUF:00000000 0000D318 CURSOR:00000000 0000D318
+0000F8 ENDOFDATA:00000000 00000000 SAVEDBUF:00000000 00000000
+000108 REALCOUNTIN:00000000 00000000
+000110 REALCOUNTOUT:00000000 00000000 POSMAJOR:00000000 00000000

```

Figure 128. Example formatted C/C++ output from LEDATA Verbexit (Part 4 of 7)

```

+000120 SAVEMAJOR:00000000 00000000 POSMINOR:00000000 00000000
+000130 SAVEMINOR:00000000 00000000 STATE:0000 SAVESTATE:0000
+000140 EXITFTELL:00000000 255A9650 EXITUNGEC:00000000 255A9180
+000150 DBCSTART:00000000 00000000 UTILITYAREA:00000000 00000000
+000160 INTERCEPT:00000000 00000000 FLAGS2:41020000 60000800
+000170 DBCSTATE:0000 FCB_CPCB:00000001 00006C0D
+0001C0 LLPOSMAJOR:00000000 00000000
+0001C8 LLSAVEMAJOR:00000000 00000000
+0001D0 LLPOSMINOR:00000000 00000000
+0001D8 LLSAVEMINOR:00000000 00000000

FSCE: 000000000000D048
+000000 GENERIC1:E3C5D9D4 00000000 00000000 0000D318 00000000 00000027
+000018 GENERIC2:00000000 00000000 00000000 0000D7A8 00000000 0000012C
+000030 GENERIC3:00000015 00000000 00000000 255A9710 00000000 255A9700

File name: *

FCB: 0000000100008118
+000000 BUFPTR:00000000 0000D318 COUNTIN:00000000 00000000
+000010 COUNTOUT:00000000 0000004E READFUNC:00000000 255A8C40
+000020 WRITEFUNC:00000000 255A8D20 FLAGS1:8000 DEPTH:0000
+000030 NAME:00000001 00008410 _LENGTH:00000000 00000001
+000040 _BUFSIZE:00000000 00000048 MEMBER:.....
+000050 NEXT:00000000 255CC750 PREV:00000000 00000000
+000060 PARENT:00000001 00008118 CHILD:00000001 000087E8
+000070 DDNAME:..... FD:FFFFFFFF DEVTYPE:01 FCBTYPE:0071
+000080 FSCE:00000000 0000D048 UNGETBUF:00000001 00008300
+000090 REPOS:00000000 255A8D40 GETPOS:00000000 255A9690
+0000A0 CLOSE:00000000 255A9670 FLUSH:00000000 255A8D50
+0000B0 UTILITY:00000000 255A9660 USERBUF:00000000 00000000
+0000C0 LRECL:00000000 0000004E BLKSIZE:00000000 0000004E
+0000D0 REALBUFPTR:00000000 0000D318
+0000E0 UNGETCOUNT:00000000 00000000 BUFSIZE:00000000 0000004F
+0000F0 BUF:00000000 0000D318 CURSOR:00000000 0000D318
+000100 ENDOFDATA:00000000 00000000 SAVEDBUF:00000000 00000000
+000110 REALCOUNTIN:00000000 00000000
+000120 REALCOUNTOUT:00000000 00000000 POSMAJOR:00000000 00000000
+000130 SAVEMAJOR:00000000 00000000 POSMINOR:00000000 00000000
+000140 SAVEMINOR:00000000 00000000 STATE:0000 SAVESTATE:0000
+000150 EXITFTELL:00000000 255A9650 EXITUNGEC:00000000 255A9180
+000160 DBCSTART:00000000 00000000 UTILITYAREA:00000000 00000000
+000170 INTERCEPT:00000000 00000000 FLAGS2:41020000 60000800
+000180 DBCSTATE:0000 FCB_CPCB:00000001 00006C0D
+0001C0 LLPOSMAJOR:00000000 00000000
+0001C8 LLSAVEMAJOR:00000000 00000000
+0001D0 LLPOSMINOR:00000000 00000000
+0001D8 LLSAVEMINOR:00000000 00000000

FSCE: 000000000000D3E0
+000000 GENERIC1:E3C5D9D4 00000000 00000000 00000000 00000000 00000000
+000018 GENERIC2:00000000 00000000 00000000 00000000 00000000 00000000
+000030 GENERIC3:00000014 00000000 00000000 255A9710 00000000 255A9700

File name: *

FCB: 0000000100008118
+000000 BUFPTR:00000000 0000D318 COUNTIN:00000000 00000000
+000010 COUNTOUT:00000000 0000004E READFUNC:00000000 255A8C40
+000020 WRITEFUNC:00000000 255A8D20 FLAGS1:8000 DEPTH:0000
+000030 NAME:00000001 00008410 _LENGTH:00000000 00000001
+000040 _BUFSIZE:00000000 00000048 MEMBER:.....
+000050 NEXT:00000000 255CC750 PREV:00000000 00000000
+000060 PARENT:00000001 00008118 CHILD:00000001 000087E8
+000070 DDNAME:..... FD:FFFFFFFF DEVTYPE:01 FCBTYPE:0071
+000080 FSCE:00000000 0000D048 UNGETBUF:00000001 00008300

```

Figure 128. Example formatted C/C++ output from LEDATA Verbexit (Part 5 of 7)

```

+000090 REPOS:00000000 255A8D40      GETPOS:00000000 255A9690
+0000A0 CLOSE:00000000 255A9670      FLUSH:00000000 255A8D50
+0000B0 UTILITY:00000000 255A9660     USERBUF:00000000 00000000
+0000C0 LRECL:00000000 0000004E      BLKSIZE:00000000 0000004E
+0000D0 REALBUFPTR:00000000 0000D318
+0000E8 UNGETCOUNT:00000000 00000000      BUFSIZE:00000000 0000004F
+0000EB BUF:00000000 0000D318      CURSOR:00000000 0000D318
+0000F8 ENDOFDATA:00000000 00000000      SAVEDBUF:00000000 00000000
+000108 REALCOUNTIN:00000000 00000000
+000110 REALCOUNTOUT:00000000 00000000      POSMAJOR:00000000 00000000
+000120 SAVEMAJOR:00000000 00000000      POSMINOR:00000000 00000000
+000130 SAVEMINOR:00000000 00000000      STATE:0000  SAVESTATE:0000
+000140 EXITFTELL:00000000 255A9650      EXITUNGETC:00000000 255A91B0
+000150 DBCSTART:00000000 00000000      UTILITYAREA:00000000 00000000
+000160 INTERCEPT:00000000 00000000      FLAGS2:41020000 60000800
+000170 DBCSTATE:0000      FCB_CPCB:00000001 00006CD0
+0001C0 LLPOSMAJOR:00000000 00000000
+0001C8 LLSAVEMAJOR:00000000 00000000
+0001D0 LLPOSMINOR:00000000 00000000
+0001D8 LLSAVEMINOR:00000000 00000000

FSCE: 000000000000D370
+000000 GENERIC1:E3C5D9D4 00000000 00000000 0000D8E0 00000000 00000047
+000018 GENERIC2:00000000 00000000 00000000 0000D1E0 00000000 0000012C
+000030 GENERIC3:00000015 00000000 00000000 255A9710 00000000 255A9700

File name: memory.data

FCB: 00000000255CC750
+000000 BUFPTR:00000001 194003D5      COUNTIN:00000000 00000000
+000010 COUNTOUT:00000000 0000FFDB      READFUNC:00000000 255A8C40
+000020 WRITEFUNC:00000000 255A9610     FLAGS1:0000      DEPTH:0000
+000030 NAME:00000000 255CCA48          _LENGTH:00000000 0000000B
+000040 _BUFSIZE:00000000 00000048      MEMBER:.....
+000050 NEXT:00000000 255CB3D8          PREV:00000001 00008118
+000060 PARENT:00000000 255CC750        CHILD:00000000 00000000
+000070 DDNAME:..... FD:FFFFFFFF      DEVTYPE:08  FCBTYPE:0055
+000080 FSCE:00000000 255CC938          UNGETBUF:00000000 255CC938
+000090 REPOS:00000000 255A9630          GETPOS:00000000 255A95B0
+0000A0 CLOSE:00000000 255A95A0          FLUSH:00000000 255A9620
+0000B0 UTILITY:00000000 255A9590        USERBUF:00000000 00000000
+0000C0 LRECL:00000000 00000400          BLKSIZE:00000000 00010000
+0000D0 REALBUFPTR:00000000 00000000
+0000E8 UNGETCOUNT:00000000 00000000      BUFSIZE:00000000 00010000
+0000EB BUF:00000001 194003B0      CURSOR:00000001 194003B0
+0000F8 ENDOFDATA:00000000 00000000      SAVEDBUF:00000000 00000000
+000108 REALCOUNTIN:00000000 00000000
+000110 REALCOUNTOUT:00000000 00000000      POSMAJOR:00000000 00000000
+000120 SAVEMAJOR:00000000 00000000      POSMINOR:00000000 00000000
+000130 SAVEMINOR:00000000 00000000      STATE:0000  SAVESTATE:0000
+000140 EXITFTELL:00000000 00000000      EXITUNGETC:00000000 255A91B0
+000150 DBCSTART:00000000 00000000      UTILITYAREA:00000000 00000000
+000160 INTERCEPT:00000000 00000000      FLAGS2:43020008 40001100
+000170 DBCSTATE:0000      FCB_CPCB:00000001 00006CD0
+0001C0 LLPOSMAJOR:00000000 00000000
+0001C8 LLSAVEMAJOR:00000000 00000000
+0001D0 LLPOSMINOR:00000000 00000000
+0001D8 LLSAVEMINOR:00000000 00000000

FSCE: 00000000255CC938
+000000 GENERIC1:D4C5D4D6 00000000 00000001 19400050 00000001 19400120
+000018 GENERIC2:00010000 00000000 00000000 00000000 00000000 255A8C40
+000030 GENERIC3:00000000 255A9610 00000000 255A9630 00000000 255A9620

```

Figure 128. Example formatted C/C++ output from LEDATA Verbexit (Part 6 of 7)

```

File name: myfile.data

FCB: 00000000255CB3D8
+000000 BUFPTR:00000000 255CB73B COUNTIN:00000000 00000000
+000010 COUNTOUT:00000000 00000FE5 READFUNC:00000000 255A8C40
+000020 WRITEFUNC:00000000 255A91D0 FLAGS1:0000 DEPTH:0000
+000030 NAME:00000000 255CB6D0 LENGTH:00000000 0000000B
+000040 _BUFSIZE:00000000 00000048 MEMBER:.....
+000050 NEXT:00000001 00006FF8 PREV:00000000 255CC750
+000060 PARENT:00000000 255CB3D8 CHILD:00000000 00000000
+000070 DDNAME:..... FD:00000000 DEVTYPE:09 FCBTYPE:007C
+000080 FSCE:00000000 255CB5C0 UNGETBUF:00000000 255CB5C0
+000090 REPOS:00000000 255A92A0 GETPOS:00000000 255A9290
+0000A0 CLOSE:00000000 255A9270 FLUSH:00000000 255A9280
+0000B0 UTILITY:00000000 255A9240 USERBUF:00000000 00000000
+0000C0 LRECL:00000000 00000000 BLKSIZE:00000000 00000000
+0000D0 REALBUFPTR:00000000 255CB720
+0000E0 UNGETCOUNT:00000000 00000000 BUFSIZE:00000000 00001000
+0000F0 BUF:00000000 255CB720 CURSOR:00000000 00000000
+0000F8 ENDOFDATA:00000000 00000000 SAVEDBUF:00000000 00000000
+000100 REALCOUNTIN:00000000 00000000
+000110 REALCOUNTOUT:00000000 00000000 POSMAJOR:FFFFFFFF FFFFFFFF
+000120 SAVEMAJOR:00000000 00000000 POSMINOR:00000000 00000000
+000130 SAVEMINOR:00000000 00000000 STATE:0000 SAVESTATE:0000
+000140 EXITFTELL:00000000 00000000 EXITUNGETC:00000000 255A91B0
+000150 DBCSTART:00000000 00000000 UTILITYAREA:00000000 00000000
+000160 INTERCEPT:00000000 00000000 FLAGS2:40120040 00001300
+000170 DBCSTATE:0000 FCB_CPCB:00000001 00006CD0
+0001C0 LLPOSMajor:00000000 00000000
+0001C8 LLSAVEMajor:00000000 00000000
+0001D0 LLPOSMINOR:00000000 00000000
+0001D8 LLSAVEMINOR:00000000 00000000

FSCE: 00000000255CB5C0
+000000 GENERIC1:C8C6E2C6 00000000 00000000 255A8C40 00000000 255A91D0
+000018 GENERIC2:00000000 255A92A0 00000000 255A9290 00000000 255A9280
+000030 GENERIC3:00000000 00000000 00000491 00000000 030001A4 00000000

Dummy FCB encountered at location 000000100006FF8

CTRL could not obtain the AMRC address from location 00E400E500E600E7

Exiting CRTL Environment Data
Exiting Language Environment Data

```

Figure 128. Example formatted C/C++ output from LEDATA Verbexit (Part 7 of 7)

C/C++-specific sections of the LEDATA output

For the LEDATA output:

[1] CGEN

This section formats the C/C++-specific portion of the Language Environment common anchor area (CAA).

[2] CGENE

This section formats the extension to the C/C++-specific portion of the Language Environment common anchor area (CAA).

[3] CEDB

This section formats the C/C++-specific portion of the Language Environment enclave data block (EDB).

[4] CTHD

This section formats the C/C++ thread-level control block (CTHD).

[5] CPCB

This section formats the C/C++-specific portion of the Language Environment process control block (PCB).

[6] CIO

This section formats the C/C++ IO control block (CIO).

[7] File Control Blocks

This section formats the C/C++ file control block (FCB). The FCB and its related control blocks represent the information needed by each open stream.

Related Control Blocks

FSCE The file specific category extension control block. The FSCE represents the specific type of IO being performed. The following is a list of FSCEs that may be formatted.

OSNS — OS no seek

OSFS — OS fixed text

OSVF — OS variable text

OSUT — OS undefined format text

Other FSCEs will be displayed using a generic overlay.

OSIO The OS IO interface control block.

DCB The data control block. For more information about the DCB, refer to *z/OS DFSMS Macro Instructions for Data Sets*.

DCBE The data control block extension. For more information about the DCBE, refer to *z/OS DFSMS Macro Instructions for Data Sets*.

JFCB The job file control block (JFCB). For more information about the JFCB, refer to *z/OS MVS Data Areas, Vol 3 (IVT-RCWK)*.

Understanding the AUTH LEDATA output

The Language Environment IPCS Verbexit LEDATA generates formatted output of Preinitialized Environments for Authorized Programs-specific control blocks from a system dump when the AUTH parameter is specified. Figure 129 on page 350 illustrates the output produced when the LEDATA verbexit is invoked with the AUTH parameter. Ellipses are used to summarize some sections of the dump.

For easy reference, the sections of the dump are numbered to correspond with the description of each section that follows.

```

*****
Authorized Language Environment Control Blocks
*****

[1] ALEC: 000000007F744000
+000000 ID:ALEC Ascb:00F99600 Flags1:40000000
+00000C UseCount:00000001 ASATable@1:7F73F414
+000014 ASATable@2:7F73F734 ASATable@3:00000000
+00001C ASATable@4:00000000 MCallRtn:82E21EF8
+000024 UCallRtn:82E28D98 LatchSetTok:7F705D40 00000065
+000030 Alei:00000001 001053A0 Ales:00000001 00107CD0
+000040 StackCPID:7F707F00 AROTCB:008FF300 EnvTypeNum:00000000
+00004C WorkECB:808FF1F0
+000050 AROTTok:00000090 00000001 00000003 008FF300
+000060 WTPE:00000019 020071B0 00000000 00000000 ALELVT:00000001
+000074 SLELVT:00100140 FuncTable@:00000000 00000000
+000080 WorkQueue:00000000 00000000 ALESeqNum:00000000 00000027
+000090 ALESCount:00000000 00000001 SystemRtnCode:00000000
+00009C SystemRsnCode:00000000 SystemRtnCodeJr:00000000
+0000A4 SystemRsnCodeJr:00000000 WorkerTCB:008EBE88
+0000B0 SystemOCB@:00000000 00000000

[2] Load Module Control Blocks

Queue #: 0000000000000000

ALMI: 0000000100100540
+000000 ID:ALMI ModuleSize:000015F4 ModuleName:CEEMENU3
+000010 UseCount:00000000 00000000 LoadPoint:00000000 258BE060
+000020 EntryPoint:00000000 258BE060

Queue #: 0000000000000003

ALMI: 0000000100100340
+000000 ID:ALMI ModuleSize:00000200 ModuleName:ALEM001
+000010 UseCount:00000000 00000003 LoadPoint:00000000 258BC000
+000020 EntryPoint:00000000 258BC001

Queue #: 0000000000000006

ALMI: 0000000100100740
+000000 ID:ALMI ModuleSize:00000200 ModuleName:CDIVZERO
+000010 UseCount:00000000 00000001 LoadPoint:00000000 258CA000
+000020 EntryPoint:00000000 258CA001

[3] User Managed Control Blocks

[4] ALEI: 00000001001053A0
+000000 ID:ALEI Flags1:80000000 InstanceNum:00000000 00000025
+000010 Next:00000000 00000000 Prev:00000000 00000000
+000020 Flags2:40000000 EnclaveSeq#:00000001 LAA:01F8FC98
+00002C SavedLAA:7F75CE80 Alec:00000000 7F744000
+000038 CallerPSWKey:00000000 00000070 ASASack:00000000 7F708A18
+000048 ParmListPtr:00000000 7F751D58
+000050 ParmListPtrK0:00000000 7F708318 RTOPtr:00000001 0050042C
+000060 RTOLen:00000000 00000012 RTBL@:00000001 001055B8
+000070 CallAIri:00000001 00100940 EnvAIris:00000000 00000000
+000080 SystemRtnCode:00000000 SystemRsnCode:00000000
+000118 SystemRtnCodeJr:00000000 SystemRsnCodeJr:00000000

```

Figure 129. Example of formatted AUTH output from LEDATA Verbexit (Part 1 of 6)

[5] Routine Control Blocks

Queue #: 000000000000000D

```
Routine: CDIVZERO
ALRI: 0000000100100940
+000000 ID:ALRI      Flags:80000000      InstanceNum:00000000 00000026
+000010 NEXT:00000000 00000000      ALEC:00000000 7F744000
+000020 AleiAddress:00000001 001053A0
+000028 AleiInstanceNum:00000000 00000025  DllName:.....
+000038 RoutineNamePtr:00000001 00100B20
+000040 RoutineNameLen:00000000 00000008
+000048 RoutineAddr:00000000 258CA0C0      QSTRAddr:00000000 258CA000
+000058 DllKeyPtr:00000000 00000000      DllKeyLen:00000000 00000000
+000068 ParmLen:00000000      EnvFlags:40000000
+000070 FuncEnv:00000000 00000010      FuncEntry:00000000 258CA0C0
+0000E0 MasterAlri:00000000 00000000      Ales:00000000 00000000
+0000F0 LUAAlri:00000000 00000000      EnvType:00000000
+0000FC EnclaveSeq#:00000001      NextEnvAlri:00000000 00000000
```

Queue #: 0000000000000010

```
Routine: CDIVZERO
ALRI: 0000000100100940
+000000 ID:ALRI      Flags:80000000      InstanceNum:00000000 00000026
+000010 NEXT:00000000 00000000      ALEC:00000000 7F744000
+000020 AleiAddress:00000001 001053A0
+000028 AleiInstanceNum:00000000 00000025  DllName:.....
+000038 RoutineNamePtr:00000001 00100B20
+000040 RoutineNameLen:00000000 00000008
+000048 RoutineAddr:00000000 258CA0C0      QSTRAddr:00000000 258CA000
+000058 DllKeyPtr:00000000 00000000      DllKeyLen:00000000 00000000
+000068 ParmLen:00000000      EnvFlags:40000000
+000070 FuncEnv:00000000 00000010      FuncEntry:00000000 258CA0C0
+0000E0 MasterAlri:00000000 00000000      Ales:00000000 00000000
+0000F0 LUAAlri:00000000 00000000      EnvType:00000000
+0000FC EnclaveSeq#:00000001      NextEnvAlri:00000000 00000000
```

[6] System Managed Control Blocks

```
[7] ALES: 0000000100107CD0
+000000 ID:ALES      UseCount:00000000
+000008 InstanceNum:00000000 00000000      Next:00000000 00000000
+000018 ENVID:RTOTCB5A      Flags1:00000000 00000000
+000028 Alec:00000000 7F744000      NumEnvType:00000000 00000003
+000038 CPPtr:00000001 001082C8      AllocSize:00000000 00000B48
+000048 SystemRtnCode:00000000      SystemRsnCode:00000000
+000050 SystemRtnCodeJr:00000000      SystemRsnCodeJr:00000000
```

[8] ETINDEX: 00000001

```
ALESETE: 0000000100107E18
+000000 Flags:00000000      ALEI:00000001 0010E0D8
+000010 WTime:00000000 00000000      InitNum:00000000 0000000A
+000020 IncrNum:00000000 00000005      MaxNum:00000000 00000014
+000030 CurNum:00000000 0000000A      RTOPtr:00000001 00207CD0
+000040 RTOLen:00000000 00000400
```

Figure 129. Example of formatted AUTH output from LEDATA Verbexit (Part 2 of 6)

I

[9] Routine Control Blocks

```
Queue #: 000000000000013
Routine: ALEM001
  ALRI: 0000000100200140
+000000 ID:ALRI      Flags:88000000      InstanceNum:00000000 00000022
+000010 NEXT:00000001 00200340      ALEC:00000000 7F744000
+000020 AleiAddress:00000000 00000000
+000028 AleiInstanceNum:00000000 00000000  DllName:.....
+000038 RoutineNamePtr:00000001 00200320
+000040 RoutineNameLen:00000000 00000008
+000048 RoutineAddr:00000000 258BC0C0      QSTRAddr:00000000 258BC000
+000058 DllKeyPtr:00000000 00000000      DllKeyLen:00000000 00000000
+000068 ParmLen:00000000      EnvFlags:00000000
+000070 FuncEnv:00000000 00000000      FuncEntry:00000000 00000000
+0000E0 MasterAlri:00000000 00000000      Ales:00000001 00107CD0
+0000F0 LUAlri:00000001 00200340      EnvType:00000001
+0000FC EnclaveSeq#:00000000      NextEnvAlri:00000000 00000000
```

Queue #: 000000000000017

```
Routine: ALEM001
  ALRI: 0000000100200140
+000000 ID:ALRI      Flags:88000000      InstanceNum:00000000 00000022
+000010 NEXT:00000001 00200340      ALEC:00000000 7F744000
+000020 AleiAddress:00000000 00000000
+000028 AleiInstanceNum:00000000 00000000  DllName:.....
+000038 RoutineNamePtr:00000001 00200320
+000040 RoutineNameLen:00000000 00000008
+000048 RoutineAddr:00000000 258BC0C0      QSTRAddr:00000000 258BC000
+000058 DllKeyPtr:00000000 00000000      DllKeyLen:00000000 00000000
+000068 ParmLen:00000000      EnvFlags:00000000
+000070 FuncEnv:00000000 00000000      FuncEntry:00000000 00000000
+0000E0 MasterAlri:00000000 00000000      Ales:00000001 00107CD0
+0000F0 LUAlri:00000001 00200340      EnvType:00000001
+0000FC EnclaveSeq#:00000000      NextEnvAlri:00000000 00000000
```

[10] ALEI: 000000010010E0D8

```
+000000 ID:ALEI      Flags1:00000000      InstanceNum:00000000 0000000A
+000010 Next:00000001 0010DEC0      Prev:00000000 00000000
+000020 Flags2:00000000      EnclaveSeq#:00000002  LAA:0233F718
+00002C SavedLAA:7F75CE80      Alec:00000000 7F744000
+000038 CallerPSWKey:00000000 00000070      ASASack:00000000 7F708AB0
+000048 ParmListPtr:00000000 7F750AA8
+000050 ParmListPtrK0:00000000 7F708368      RTOPtr:00000001 0000042C
+000060 RTOLen:00000000 00000401      RTBL0:00000000 00000000
+000070 CallAlri:00000001 00200340      EnvAlris:00000001 00200340
+000080 SystemRtnCode:00000000      SystemRsnCode:00000000
+000118 SystemRtnCodeJr:00000000      SystemRsnCodeJr:00000000
```

Figure 129. Example of formatted AUTH output from LEDATA Verbexit (Part 3 of 6)

|

```

[11] Routine: ALEM001
    ALRI: 0000000100200340
+000000 ID:ALRI      Flags:80000000   InstanceNum:00000000 00000022
+000010 NEXT:00000000 00000000      ALEC:00000000 7F744000
+000020 AleiAddress:00000001 0010E0D8
+000028 AleiInstanceNum:00000000 00000000   DllName:.....
+000038 RoutineNamePtr:00000001 00200320
+000040 RoutineNameLen:00000000 00000008
+000048 RoutineAddr:00000000 258BC0C0      QSTRAddr:00000000 258BC000
+000058 DllKeyPtr:00000000 00000000   DllKeyLen:00000000 00000000
+000068 ParmLen:00000010      EnvFlags:40000000
+000070 FuncEnv:00000000 00000010   FuncEntry:00000000 258BC0C0
+0000E0 MasterAlri:00000001 00200140   Ales:00000001 00107CD0
+0000F0 LUAAlri:00000000 00000000   EnvType:00000001
+0000FC EnclaveSeq#:00000001   NextEnvAlri:00000000 00000000

[10] ALEI: 000000010010DEC0
+000000 ID:ALEI      Flags1:00000000   InstanceNum:00000000 00000009
+000010 Next:00000001 0010DCA8      Prev:00000000 00000000
+000020 Flags2:80000000   EnclaveSeq#:00000000   LAA:0233F598
+00002C SavedLAA:00000000   Alec:00000000 7F744000
+000038 CallerPSWKey:00000000 00000070   ASASStack:00000000 00000000
+000048 ParmListPtr:00000000 00000000
+000050 ParmListPtrK0:00000000 00000000   RTOPtr:00000001 00207CD0
+000060 RTOLen:00000000 00000400   RTBL0:00000000 00000000
+000070 CallAlri:00000000 00000000   EnvAlris:00000000 00000000
+000080 SystemRtnCode:00000000   SystemRsnCode:00000000
+000118 SystemRtnCodeJr:00000000   SystemRsnCodeJr:00000000

[10] ALEI: 000000010010DCA8
+000000 ID:ALEI      Flags1:00000000   InstanceNum:00000000 00000008
+000010 Next:00000001 0010DA90      Prev:00000000 00000000
+000020 Flags2:80000000   EnclaveSeq#:00000000   LAA:0233F418
+00002C SavedLAA:00000000   Alec:00000000 7F744000
+000038 CallerPSWKey:00000000 00000070   ASASStack:00000000 00000000
+000048 ParmListPtr:00000000 00000000
+000050 ParmListPtrK0:00000000 00000000   RTOPtr:00000001 00207CD0
+000060 RTOLen:00000000 00000400   RTBL0:00000000 00000000
+000070 CallAlri:00000000 00000000   EnvAlris:00000000 00000000
+000080 SystemRtnCode:00000000   SystemRsnCode:00000000
+000118 SystemRtnCodeJr:00000000   SystemRsnCodeJr:00000000

.
.
.

[8] ETINDEX: 00000002

ALESETE: 0000000100107FA8
+000000 Flags:00000000   ALEI:00000001 001102F0
+000010 WTime:00000000 00000000   InitNum:00000000 0000000B
+000020 IncrNum:00000000 00000006   MaxNum:00000000 00000017
+000030 CurNum:00000000 0000000B   RTOPtr:00000001 002080D0
+000040 RTOLen:00000000 00000400

```

Figure 129. Example of formatted AUTH output from LEDATA Verbexit (Part 4 of 6)

1

[9] Routine Control Blocks

Queue #: 000000000000013

```
Routine: ALEM001
  ALRI: 0000000100200540
+000000 ID:ALRI      Flags:88000000      InstanceNum:00000000 00000023
+000010 NEXT:00000001 00200740      ALEC:00000000 7F744000
+000020 AleiAddress:00000000 00000000
+000028 AleiInstanceNum:00000000 00000000      DllName:.....
+000038 RoutineNamePtr:00000001 00200720
+000040 RoutineNameLen:00000000 00000008
+000048 RoutineAddr:00000000 258BC0C0      QSTRAddr:00000000 258BC000
+000058 DllKeyPtr:00000000 00000000      DllKeyLen:00000000 00000000
+000068 ParmLen:00000000      EnvFlags:00000000
+000070 FuncEnv:00000000 00000000      FuncEntry:00000000 00000000
+0000E0 MasterAlri:00000000 00000000      Ales:00000001 00107CD0
+0000F0 LUAlri:00000001 00200740      EnvType:00000002
+0000FC EnclaveSeq#:00000000      NextEnvAlri:00000000 00000000
```

Queue #: 000000000000017

```
Routine: ALEM001
  ALRI: 0000000100200540
+000000 ID:ALRI      Flags:88000000      InstanceNum:00000000 00000023
+000010 NEXT:00000001 00200740      ALEC:00000000 7F744000
+000020 AleiAddress:00000000 00000000
+000028 AleiInstanceNum:00000000 00000000      DllName:.....
+000038 RoutineNamePtr:00000001 00200720
+000040 RoutineNameLen:00000000 00000008
+000048 RoutineAddr:00000000 258BC0C0      QSTRAddr:00000000 258BC000
+000058 DllKeyPtr:00000000 00000000      DllKeyLen:00000000 00000000
+000068 ParmLen:00000000      EnvFlags:00000000
+000070 FuncEnv:00000000 00000000      FuncEntry:00000000 00000000
+0000E0 MasterAlri:00000000 00000000      Ales:00000001 00107CD0
+0000F0 LUAlri:00000001 00200740      EnvType:00000002
+0000FC EnclaveSeq#:00000000      NextEnvAlri:00000000 00000000
```

Figure 129. Example of formatted AUTH output from LEDATA Verbexit (Part 5 of 6)

I

```

[10] ALEI: 00000001001102F0
+000000 ID:ALEI      Flags1:00000000      InstanceNum:00000000 00000015
+000010 Next:00000001 001100D8      Prev:00000000 00000000
+000020 Flags2:00000000      EnclaveSeq#:00000002      LAA:01F90818
+00002C SavedLAA:7F75CE80      Alec:00000000 7F744000
+000038 CallerPSWKey:00000000 00000070      ASASStack:00000000 7F708AB0
+000048 ParmListPtr:00000000 7F751AA8
+000050 ParmListPtrK0:00000000 7F708368      RTOPtr:00000001 0030042C
+000060 RTOLen:00000000 00000401      RTBL0:00000000 00000000
+000070 CallAlri:00000001 00200740      EnvAlris:00000001 00200740
+000080 SystemRtnCode:00000000      SystemRsnCode:00000000
+000118 SystemRtnCodeJr:00000000      SystemRsnCodeJr:00000000

```

```

.
.
.

```

```

[8] ETINDEX: 00000003

```

```

ALESETE: 0000000100108138
+000000 Flags:00000000      ALEI:00000001 00112508
+000010 WTime:00000000 00000000      InitNum:00000000 0000000C
+000020 IncrNum:00000000 00000007      MaxNum:00000000 0000001A
+000030 CurNum:00000000 0000000C      RTOPtr:00000001 002084D0
+000040 RTOLen:00000000 00000400

```

```

.
.
.

```

Exiting Language Environment Data

Figure 129. Example of formatted AUTH output from LEDATA Verbexit (Part 6 of 6)

Sections of the AUTH LEDATA Verbexit formatted output

[1]ALEC

The ALEC is the anchor control block for all other Preinitialized Environments for Authorized Programs control blocks within the address space. The ALEC is located from the ASXB (Address Space Extension Block).

[2]Load Module Control Blocks

This section is the formatted representation of a table of ALMI control blocks. Each ALMI represents a module that was loaded by Preinitialized Environments for Authorized Programs.

[3]User Managed Control Blocks

This section contains control blocks for all user managed environments. A user managed environment is initialized when the CELAAUTH macro is invoked with REQUEST=USERINIT.

[4]-[5]Control Blocks for one user managed environment

These sections are repeated for each user managed environment that was initialized.

[4]ALEI

Each ALEI control block represents one environment.

[5]Routine Control Blocks

This section is the formatted representation of a table of ALRI control blocks. Each ALRI in this section represents a routine that was called by the user managed environment. Each ALRI appears in the table twice, once for the routine name and once for the routine address.

[6]System Managed Control Blocks

This section contains control blocks for all system managed environments. A set of system managed environments is initialized when the CELAAUTH macro is invoked with REQUEST=MNGDINIT.

[7]-[11]Control Blocks for one set of system managed environments that was initialized

These sections are repeated for each set of system managed environments that was initialized.

[7]ALES

Each ALES represents a set of system managed environments.

[8]-[11]Control blocks for one environment definition entry

These sections are repeated for every environment definition entry (AEDE) that was specified when the set of system managed environments was initialized.

[8]ETINDEX and ALESETE

The ETINDEX is the environment definition entry index value and the ALESETE represents the environment definition entry.

[9]Routine Control Blocks

This section is the formatted representation of a table of ALRI control blocks. Each ALRI in this section represents a routine that was called in one of the environments associated with the ETINDEX and ALESETE above. Each ALRI appears in the table twice, once for the routine name and once for the routine address.

[10]-[11]Control blocks for one system managed environment

These sections are repeated for every environment associated with the ETINDEX and ALESETE above.

[10]ALEI

Each ALEI control block represents one environment. The ALEIs in this section represent system managed environments.

[10]ALRI

This section contains the ALRI control blocks for each routine that was called in the environment identified by the ALEI above. This section does not appear if the environment has not been used to call a routine.

Formatting individual control blocks

In addition to the full LEDATA output which contains many formatted control blocks, the IPCS Control block formatter can also format individual Language Environment control blocks.

The IPCS cbf command can be invoked from the "IPCS Subcommand Entry" screen, option 6 of the "IPCS PRIMARY OPTION MENU".

Syntax

►► CBF—*address*—STRUCTure—(—*cbname*—)◄◄

address

The address of the control block in the dump. This is determined by browsing the dump or running the LEDATA Verbexit.

cbname

The name of the control block to be formatted. The control blocks that can be individually formatted are listed in Table 30 on page 358. In general, the name of each control block is similar to that used by the LEDATA Verbexit and is generally found in the control block's eyecatcher field. However, all control block names are prefixed with CEE in order to uniquely define the Language Environment control block names to IPCS.

For an example of the display which is the result of the command, see Figure 130 on page 358 :

CBF 15890 struct(CELCAA)

```

CEECAA: 0000000100006160
+0002B2 INVAR:8000 FLAG0:00 TORC:00000000 FLAG2:30 LEVEL:12
+000365 _PM:04 DMC:00000000 00000000 CD:00000000
+000374 RS:00000000 ERR:00000001 082FBDf0
+000380 DDSA:00000001 082FF760 EDB:00000001 000039D0
+000390 PCB:00000001 00002468 EYEPTR:00000001 00006148
+0003A0 CAA:00000001 00006160 SHAB:00000000 00000000
+0003B0 PRGCK:00000000 00000000 URC:00000000
+0003C0 PICICB:00000000 00000000 SIGSCTR:00000000
+0003CC SIGSFLG:08000000 THDID:25481460 00000001
+0003D8 RCB:00000001 00002210 MEMBR:00000001 00006B18
+0003E8 SIGNAL_STATUS:00000000 00000008
+0003F0 HCOM_REG14:00000000 00000000 EDCHPXV:00000000 25175FF0
+000400 THREADHEAPID:00000000 00000000 SYS_RTNCODE:00000000
+00040C SYS_RSNCODE:00000000 SIGNGPTR:00000001 00006578
+000418 SIGNG:00000001 AB_STATUS:00 STACKDIRECTION:00
+000420 AB_GRO:00000000 00000000 AB_ICD1:00000000 00000000
+000430 AB_ABCC:00000000 00000000 AB_CRC:00000000 00000000
+000440 USERRTN:00000000 00000000 QINITDSA:00000001 082FF280
+0004E8 IFLAG:0008 TRMRSN:00 DEVH:00000000 00000000
+0004F8 PstatPtr:00000000 00000000 SQELADDR:00000000 255C7340
+000510 VBA:00000000 00000000 TCS:00000000 00000000
+000520 CONDWAITDSAREG:00000000 00000000 THDSTATUS:00000000 00000000
+000570 FBTOK:00000000 00000000 00000000 00000000
+000580 PTXLPTR:00000000 00000000 TICB_PTR:00000001 000050F8
+000598 FWD_CHAIN:00000001 10E013C8 BKWD_CHAIN:00000001 193013C8
+0007E8 TCB:008DD8D8

```

Figure 130. The CAA formatted by the CBFORMAT IPCS command

For more information on using the IPCS CBF command refer to the "CBFORMAT subcommand" section in *z/OS MVS IPCS Commands*, SA22-7594.

Table 30. Language Environment control blocks which can be individually formatted

Control Block	Description
CELCIB	Condition Information Block
CELCIBH	Condition Information Block Header
CELDSA	Dynamic Storage Area
CELDSATR	XPLINK Transition Area
CELEDB	Enclave Data Block
CELENSQ	Enclave Level Storage Management
CELHNQ31	Heap Anchor Node 31-bit
CELHCOM	CEL Exception Manager Communications Area
CELHPCQ	Thread Level Heap Control Block
CELLAA	Library Anchor Area
CELLCA	Library Communication Area
CELPCB	Process Control Block
CELRCB	Region Control Block
CELSANC	Storage Management Control Block
CELSTSB	Storage Report Statistics Block

Table 31. Preinitialized Environments for Authorized Programs control blocks which can be individually formatted

Control Block	Description
CELALEC	Anchor Block
CELALEI	Environment Information Block
CELALES	System Managed Environment Set Block
CELALMI	Module Information Block
CELALRI	Routine Information Block

Requesting a Language Environment trace for debugging

Language Environment provides an in-storage, wrapping trace facility that can reconstruct the events leading to the point where a dump is taken. Language Environment produces a trace table in its dump report when the TRACE run-time option is set to ON and:

- A thread ends abnormally because of an unhandled condition of severity 2 or greater and the TERMTHDACT run-time option is set to DUMP, UADUMP, TRACE, or UATRACE.
- An application terminates normally and the TRACE run-time option is set to DUMP (the default).

For more information about recording done by the TERMTHDACT run-time option or the TRACE run-time option, see *z/OS Language Environment Programming Reference*.

The TRACE run-time option activates Language Environment run-time library tracing and controls the size of the trace buffer, the type of trace events to record, and it determines whether a dump containing only the trace table should be unconditionally taken when the application (enclave) terminates. The trace table contents can be written out either upon demand or at the termination of an enclave.

The contents of the Language Environment dump depend on the values set in the TERMTHDACT run-time option. Under abnormal termination, the following dump contents are generated:

- TERMTHDACT(QUIET) generates a Language Environment dump containing the trace table only
- TERMTHDACT(MSG) generates a Language Environment dump containing the trace table only
- TERMTHDACT(TRACE) generates a Language Environment dump containing the trace table and the traceback
- TERMTHDACT(DUMP) generates a Language Environment dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block)
- TERMTHDACT(UAONLY) generates a system dump of the user address space
- TERMTHDACT(UATRACE) generates a Language Environment dump that contains traceback information, and a system dump of the user address space
- TERMTHDACT(UADUMP) generates a Language Environment dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block), and a user address space dump

- **TERMTHDACT(UAImm)** generates a system dump of the user address space of the original abend or program interrupt that occurred prior to the Language Environment condition manager processing the condition.

Note: **TRAP(ON,NOSPIE)** must be in effect. When **TRAP(ON,SPIE)** is in effect, UAImm equals UAONLY results. For software raised conditions or signals, UAImm is the same as UAONLY.

Under normal termination, with the **TRACE** run-time option set to **DUMP**, Language Environment generates a dump containing the trace table only, independent of the **TERMTHDACT** setting.

Language Environment quiesces all threads that are currently running except for the thread that issued the call to **cdump()**. When you call **cdump()** in a multithread environment, only the current thread is dumped. Enclave- and process-related storage could have changed from the time the dump request was issued.

Locating the trace dump

If your application is running under TSO or batch, and a **CEEDUMP DD** is not specified, Language Environment writes the **CEEDUMP** to the batch log (**SYSOUT=*** by default). You can change the **SYSOUT** class by specifying a **CEEDUMP DD**, or by setting the environment variable, **_CEE_DMPTARG=SYSOUT(x)**, where **x** is the preferred **SYSOUT** class.

If your application is running under z/OS UNIX and is either running in a child process, or if it is invoked by one of the **exec** family of functions, the dump is written to the z/OS UNIX file system. Language Environment writes the **CEEDUMP** to one of the following directories in the specified order:

1. The directory in environment variable **_CEE_DMPTARG**, if found
2. The current working directory, if the directory is not the root directory (**/**), and the directory is writable
3. The directory found in environment variable **TMPDIR** (an environment variable that indicates the location of a temporary directory if it is not **/tmp**)
4. The **/tmp** directory

The name of this file changes with each dump and uses the following format:

/path/CEEDUMP.Date.Time.Pid

path	The path determined from the above algorithm.
Date	The date the dump is taken, appearing in the format YYYYMMDD (such as 20040918 for September 18, 2004).
Time	The time the dump is taken, appearing in the format HHMMSS (such as 175501 for 05:55:01 p.m.).
Pid	The process ID the application is running in when the dump is taken.

Using the Language Environment trace table format in a dump report

The Language Environment trace table is established unconditionally at enclave initialization time if the **TRACE** run-time option is set to **ON**. All threads in the enclave share the trace table; there is no thread-specific table, nor can the table be dynamically extended or enlarged.

Understanding the trace table entry (TTE)

Each trace table entry is a fixed-length record consisting of a fixed-format portion (containing such items as the timestamp, thread ID, and member ID) and a member-specific portion. The member-specific portion has a fixed length, of which some (or all) can be unused. For information about how participating products use the trace table entry, refer to the product-specific documentation. The format of the trace table entry is as follows:

Time of Day	Thread ID	Member ID and flags	Member entry type	Mbr-specific info up to a maximum of 104 bytes
Char (8)	Char (8)	Char (4)	Char (4)	Char (104)

Figure 131. Format of the trace table entry

Following is a definition of each field:

Time The 64-bit value obtained from a store clock (STCK).

Thread ID The 8-byte thread ID of the thread that is adding the trace table entry.

Member ID and Flags

Contains 2 fields:

Member ID The 1-byte member ID of the member making the trace table entry, as follows:

ID	Name
01	CEL
03	C/C++
08	DCE
12	Sockets

Flags 24 flags reserved for internal use.

Member Entry Type

A number that indicates the type of the member-specific trace information that follows the field.

To uniquely identify the information contained in a specific TTE, you must consider Member ID as well as Member Entry Type.

Member-Specific Information

Based on the member ID and the member entry type, this field contains the specific information for the entry, up to 104 bytes.

For C/C++, the entry type of 1 is a record that records an invocation of a base C run-time library function. The entry consists of the name of the invoking function and the name of the invoked function. Entry type 2 is a record that records the return from the base library function. It contains the returned value and the value of errno.

Member-specific information in the trace table entry

Global tracing is activated by using the LE=n suboption of the TRACE run-time option. This requests all Language Environment members to generate trace records in the trace table.

The settings for the global trace events are:

Level	Description
0	No global trace
1	Trace all run-time library (RTL) function entry and exits
2	Trace all RTL mutex init/destroy and lock/unlock
3	Trace all RTL function entry and exits, and all mutex init/destroy and lock/unlock
8	Trace all RTL storage allocation/deallocation

When LE=1 is specified: The following C/C++ records may be generated.

Table 32. LE=1 entry records

Member ID	Record Type	Description
03	00000001	Base C Library function Entry
03	00000002	Base C Library function Exit
03	00000003	Posix C Library function Entry
03	00000004	Posix C Library function Exit
03	00000005	XPLINK Base or Posix C Library function Entry
03	00000006	XPLINK Base or Posix C Library function Exit

For a detailed description of these records, see “C/C++ contents of the Language Environment trace tables” on page 390.

When LE=2 is specified: The following Language Environment records may be generated.

Table 33. LE=2 entry records

Member ID	Record Type	Class	Event	Description
01	00000101	LT	A	Latch Acquire
01	00000102	LT	R	Latch Release
01	00000103	LT	W	Latch Wait
01	00000104	LT	AW	Latch Acquire after Wait
01	00000106	LT	I	Latch Increment (Recursive)
01	00000107	LT	D	Latch Decrement (Recursive)
01	000002FC	LE	EUO	Latch unowned (not released)
01	000002FD	LE	EO	Latch already owned (not acquired)
01	00000301	MX	A	Mutex acquire
01	00000302	MX	R	Mutex release
01	00000303	MX	W	Mutex wait
01	00000304	MX	AW	Mutex acquire after wait
01	00000305	MX	B	Mutex busy (Trylock failed)
01	00000306	MX	I	Mutex increment (recursive)

Table 33. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
01	00000307	MX	D	Mutex decrement (recursive)
01	00000315	MX	IN	Mutex initialize
01	00000316	MX	DS	Mutex destroy
01	0000031D	MX	BI	Shared memory lock init
01	0000031E	MX	BD	Shared memory lock destroy
01	0000031F	MX	BO	shared memory lock obtain
01	00000320	MX	BC	Shared memory lock obtain on condition
01	00000321	MX	BR	Shared memory lock release
01	00000324	MX	CIN	Call to SMC_INIT
01	00000325	MX	CSD	Call to SMC_DESTROY
01	00000326	MX	CSO	Shared resource obtain
01	00000327	MX	CSR	Shared resource release
01	00000328	MX	CST	Call to SMC_SetupToWait
01	00000329	MX	CSP	Call to SMC_POST
01	000004CC	ME	FFR	Error - Forced release (shared mutex)
01	000004CD	ME	FFD	Error - Forced decrement (shared mutex)
01	000004CE	ME	FBD	Error - BPX_SMC(DESTROY) error return
01	000004CF	ME	FBU	Error - BPX_SMC(fail) returns EBUSY
01	000004D0	ME	FIV	Error - BPX_SMC(fail) returns EINVAL
01	000004D4	ME	FDU	Error - Destroy failed (uninitialized) (shared mutex/CV)
01	000004D5	ME	FP	Error - Program check (shared mutex/CV)
01	000004DB	ME	ESC	Error - BPX1SMC error return
01	000004DE	ME	EDL	Shared memory lock returns deadlock
01	000004DF	ME	EIV	Shared memory lock returns invalid
01	000004E0	ME	EPM	Shared memory lock returns eperm
01	000004E1	ME	EAG	Shared memory lock returns eagain
01	000004E2	ME	EBU	Shared memory lock returns ebusy
01	000004E3	ME	ENM	Shared memory lock returns enomem
01	000004E4	ME	EBR	Shared memory lock release error
01	000004E5	ME	EBC	Shared memory lock obtain condition error
01	000004E6	ME	EBO	Shared memory lock obtain error
01	000004E7	ME	EBD	Shared memory lock destroy error
01	000004E8	ME	EBI	Shared memory lock initialize error
01	000004E9	ME	EFR	Mutex forced release
01	000004EA	ME	EFD	Mutex forced decrement
01	000004EB	ME	EDD	Mutex destroy failed (damage)
01	000004EC	ME	EDB	Mutex destroy failed (busy)
01	000004ED	ME	EIA	Mutex initialize failed (attribute)

Table 33. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
01	000004EE	ME	EIS	Mutex initialize failed (storage)
01	000004EF	ME	EF	Mutex release (forced by quiesce)
01	000004F0	ME	EP	Mutex program check
01	000004FA	ME	EDU	Mutex destroy failed (uninitialized)
01	000004FB	ME	EUI	Mutex uninitialized
01	000004FC	ME	EUO	Mutex unowned (not released)
01	000004FD	E	EO	Mutex already owned (not acquired)
01	000004FE	ME	EIN	Mutex initialization failed (duplicate)
01	00000508	CV	MR	CV release mutex
01	00000509	CV	MA	CV reacquire mutex
01	0000050A	CV	MW	CV mutex wait
01	0000050B	CV	MAW	CV reacquire mutex after wait
01	0000050C	CV	CW	CV condition wait
01	0000050D	CV	CTW	CV condition timeout
01	0000050E	CV	CWP	CV wait posted
01	0000050F	CV	CWI	CV wait interrupted
01	00000510	CV	CTO	CV wait timeout
01	00000511	CV	CSS	CV condition signal success
01	00000512	CV	CSM	CV condition signal miss
01	00000513	CV	CBS	CV condition broadcast success
01	00000514	CV	CBM	CV condition broadcast miss
01	00000515	CV	IN	CV initialize
01	00000516	CV	DS	CV destroy
01	00000522	CV	CIN	Call to SMC_INIT
01	00000523	CV	CSD	Call to SMC_DESTROY
01	00000529	CV	CSP	Call to SMC_POST
01	0000052A	CV	CSB	Call to SMC_POSTALL
01	0000052B	CV	CSW	Call to SMC_WAIT
01	0000052C	CV	DBM	Shared condition broadcast - miss
01	0000052D	CV	DBS	Shared condition broadcast - success
01	0000052E	CV	DDS	Destroy (shared mutex/CV)
01	0000052F	CV	DIN	Initialize (shared mutex/CV)
01	00000530	CV	DSM	Condition signal - miss (shared CV)
01	00000531	CV	DSS	Condition signal - success (shared CV)
01	00000532	CV	DWI	Wait interrupted (shared CV)
01	00000533	CV	DTO	Wait timeout (shared CV)
01	00000534	CV	DWP	Wait posted (shared CV)
01	000006CB	CE	FBT	Error - Invalid system TOD (shared)
01	000006D1	CE	FRM	Error - Recursive mutex (shared)

Table 33. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
01	000006D2	CE	FUO	Error - Shared mutex unowned
01	000006D3	CE	FDB	Error - Destroy failed (busy) (shared mutex/CV)
01	000006D4	CE	FDU	Error - Destroy failed (uninitialized) (shared mutex/CV)
01	000006D5	CE	FP	Error - Program check (shared mutex/CV)
01	000006D6	CE	FUI	Error - Shared mutex or CV uninitialized
01	000006D7	CE	ENV	Error - BPX1SMC(fail) returns EINVAL
01	000006D8	CE	EPE	Error - BPX1SMC(fail) returns EPERM
01	000006D9	CE	EAN	Error - BPX1SMC(fail) returns EAGAIN
01	000006DA	CE	EIB	Error - BPX1SMC failed (EBUSY)
01	000006DB	CE	ESC	Error - BPX1SMC failed
01	000006EB	CE	EDD	CV destroy failed (damage)
01	000006EC	CE	EDB	CV destroy failed (busy)
01	000006ED	CE	EIA	CV initialization failed (attribute)
01	000006EE	CE	EIS	CV initialization failed (storage)
01	000006EF	CE	EF	CV forced by quiesce
01	000006F0	CE	EP	CV program check
01	000006F1	CE	EBT	CV invalid system TOD
01	000006F2	CE	EBN	CV invalid timespec (nanoseconds)
01	000006F3	CE	EBS	CV invalid timespec (seconds)
01	000006F4	CE	EPO	CV condition post callable service fail
01	000006F5	CE	ETW	CV condition timed wait callable service fail
01	000006F6	CE	EWA	CV condition wait callable service fail
01	000006F7	CE	ESE	CV condition setup callable service fail
01	000006F8	CE	ERM	CV recursive mutex
01	000006F9	CE	EWM	CV wrong mutex
01	000006FA	CE	EDU	CV destroy failed (uninitialized)
01	000006FB	CE	EUI	CV mutex or CV uninitialized
01	000006FC	CE	EUO	CV mutex unowned
01	000006FE	CE	EIN	CV initialization failed (duplicate)
01	00000702	RW	R	Release
01	00000704	RW	AW	Acquire after wait
01	00000706	RW	I	Increment (recursive)
01	00000707	RW	D	Decrement (recursive)
01	00000715	RW	IN	Initialize
01	00000716	RW	DS	Destroy
01	00000717	RW	RA	Read acquire
01	00000718	RW	WA	Write acquire
01	00000719	RW	RB	Read busy (tryread failed)

Table 33. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
01	0000071A	RW	WB	Write busy (trywrite failed)
01	0000071B	RW	RW	Read wait
01	0000071C	RW	WW	Write wait
01	0000071D	RW	BI	Call to SLK_INIT
01	0000071E	RW	BD	Call to SLK_DESTROY
01	0000071F	RW	BO	Call to SLK_OBTAIN
01	00000720	RW	BC	Call to SLK_OBTAIN_COND
01	00000721	RW	BR	Call to SLK_RELEASE
01	000008DC	RE	EOW	Error - Already owned for write (not acquired)
01	000008DD	RE	EOR	Error - Already owned for read (not acquired)
01	000008DE	RE	EDL	Error - BPX1SLK(fail) returns EDEADLK
01	000008DF	RE	EIV	Error - BPX1SLK(fail) returns EINVAL
01	000008E0	RE	EPM	Error - BPX1SLK(fail) returns EPERM
01	000008E1	RE	EAG	Error - BPX1SLK(fail) returns EAGAIN
01	000008E2	RE	EBS	Error - BPX1SLK(fail) returns EBUSY
01	000008E3	RE	ENM	Error - BPX1SLK(fail) returns ENOMEM
01	000008E4	RE	EBR	Error - BPX1SLK(RELEASE) error return
01	000008E5	RE	EBC	Error - BPX1SLK(OBTAIN_COND) error return
01	000008E6	RE	EBO	Error - BPX1SLK(OBTAIN) error return
01	000008E7	RE	EBD	Error - BPX1SLK(DESTROY) error return
01	000008E8	RE	EBI	Error - BPX1SLK(INIT) error return
01	000008E9	RE	EFR	Error - Forced release
01	000008EA	RE	EFD	Error - Forced decrement
01	000008ED	RE	EIA	Error - Initialization failed (attribute)
01	000008EE	RE	EIS	Error - Initialization failed (storage)
01	000008EF	RE	EF	Error - Forced by quiesce
01	000008F0	RE	EP	Error - Program check
01	000008FB	RE	EUI	Error - Uninitialized
01	000008FC	RE	EUO	Error - Unowned (not released)
01	000008FD	RE	EO	Error - Already owned (not acquired)
01	000008FE	RE	EIN	Error - Initialization failed (duplicate)

The format for the Mutex – Condition Variable – Latch entries in the trace table is:

Table 34. Format of the mutex/CV/latch records

Class	Source		Event		Object Addr	Name1	Name2
unused							

Where each field represents:

Class Two character EBCDIC representation of the trace class.

LT	Latch
LE	Latch Exception
MX	Mutex
ME	Mutex Exception
CV	Condition Variable
CE	Condition Variable Exception

Source

One character EBCDIC representation of the event.

C C/C++

Blank Blank character

Event Two character EBCDIC representation of the event. See Table 33 on page 362.

Object address

Fullword address of the mutex object.

Name 1

Optional eight character field containing the name of the function or object to be recorded.

Name 2

Optional eight character field containing the name of the function or object to be recorded.

When LE=3 is specified: The trace table will include the records generated by both LE=1 and LE=2.

When LE=8 is specified: The trace table will contain only storage allocation records. Currently this is only supported by C/C++.

Table 35. LE=8 entry records

Member ID	Record Type	Description
03	00000005	Storage allocation entry
03	00000006	Storage allocation exit

For a detailed description of these records, see “C/C++ contents of the Language Environment trace tables” on page 390.

Sample dump for the trace table entry

The following is an example of a dump of the trace table when you specify the LE=1 suboption (the library call/return trace):

Language Environment Trace Table:

Most recent trace entry is at displacement: 002080

Displacement	Trace Entry in Hexadecimal	Trace Entry in EBCDIC
+000000	Time 22.02.30.389659 Date 2004.04.08 Thread ID... 2548146000000001	
+000010	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000018	94818995 40404040 40404040 40404040 40404040 40404040 40404040	main
+000038	60606E4D F0F0F6C6 5D409799 8995A386 4D5D4040 40404040 40404040	-->(006F) printf()
+000058	40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000078	40404040 40404040	
+000080	Time 22.02.30.389724 Date 2004.04.08 Thread ID... 2548146000000001	
+000090	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000098	4C60604D F0F0F6C6 5D40D9F1 7EF0F0F0 F0F0F0F0 F0F0F0F0 F040D9F2	<--(006F) R1=0000000000000000 R2
+0000B8	7EF0F0F0 F0F0F0F0 F0F2F5F4 C2F2F8C5 F040D9F3 7EF0F0F0 F0F0F0F0	=00000000254B28E0 R3=000000000000
+0000D8	F0F0F0F0 C340C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	0000C ERRNO=00000000 ERRNO2=0000
+0000F8	F0F0F0F0 00000000	0000....
+000100	Time 22.02.30.389725 Date 2004.04.08 Thread ID... 2548146000000001	
+000110	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000118	94818995 40404040 40404040 40404040 40404040 40404040 40404040	main
+000138	60606E4D F0F1F7F0 5D408493 93939681 844D5D40 40404040 40404040	-->(0170) dllload()
+000158	40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000178	40404040 40404040	
+000180	Time 22.02.30.409904 Date 2004.04.08 Thread ID... 2548146000000001	
+000190	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000198	4C60604D F0F1F7F0 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F8F2 C6C6F4F6 F040D9F2	<--(0170) R1=00000001082FF460 R2
+0001B8	7EF0F0F0 F0F0F0F0 F0F2F5F4 C2F2F8C5 F040D9F3 7EF0F0F0 F0F0F0F0 F1F0F8F9	=00000000254B28E0 R3=00000001089
+0001D8	F4F8C6F1 F040C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	48F10 ERRNO=00000000 ERRNO2=0000
+0001F8	F0F0F0F0 00000000	0000....
+000200	Time 22.02.30.409906 Date 2004.04.08 Thread ID... 2548146000000001	
+000210	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000218	94818995 40404040 40404040 40404040 40404040 40404040 40404040	main
+000238	60606E4D F0F0F6C6 5D409799 8995A386 4D5D4040 40404040 40404040	-->(006F) printf()
+000258	40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000278	40404040 40404040	
+000280	Time 22.02.30.409938 Date 2004.04.08 Thread ID... 2548146000000001	
+000290	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000298	4C60604D F0F0F6C6 5D40D9F1 7EF0F0F0 F0F0F0F0 F0F0F0F0 F040D9F2	<--(006F) R1=0000000000000000 R2
+0002B8	7EF0F0F0 F0F0F0F0 F0F2F5F4 C2F2F8C5 F040D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0	=00000000254B28E0 R3=000000000000
+0002D8	F0F0F0F0 C440C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	0000D ERRNO=00000000 ERRNO2=0000
+0002F8	F0F0F0F0 00000000	0000....
+000300	Time 22.02.30.409939 Date 2004.04.08 Thread ID... 2548146000000001	
+000310	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000318	94818995 40404040 40404040 40404040 40404040 40404040 40404040	main
+000338	60606E4D F0F1F6C4 5D408493 9398A485 99A88695 4D5D4040 40404040	-->(016D) dllqueryfn()
+000358	40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000378	40404040 40404040	

Figure 132. Trace table in dump output

Chapter 12. Debugging AMODE 64 C/C++ routines

This chapter provides specific information to help you debug AMODE 64 applications that contain one or more C/C++ routines. It includes the following topics:

- Debugging C/C++ I/O routines
- Using XL C/C++ compiler listings
- Generating a Language Environment dump of a C/C++ routine
- Finding C/C++ information in a Language Environment dump
- Debugging example of C/C++ routines

There are several debugging features that are unique to C/C++ routines. Before examining the C/C++ techniques to find errors, you might want to consider the following areas of potential problems:

- To prevent errors which may result from differences in LP64 default argument types, you should include function prototypes for all C/C++ function calls. For C/C++ run-time library functions, see *z/OS XL C/C++ Run-Time Library Reference*.

Note: `malloc()` is an example of a RTL function which needs this prototype to work correctly in LP64 applications.

- If you are using the `fetch()` function, refer to *z/OS XL C/C++ Programming Guide* to ensure that you are creating the fetchable module correctly.
- If you are using DLLs, refer to *z/OS XL C/C++ Programming Guide* to ensure that you are using the DLL correctly.
- Ensure that the entry point of the load module is `CELQSTRT`.
- If you suspect that you are using uninitialized storage, you may want to use the `STORAGE` run-time option.
- You should avoid:
 - Incorrect casting
 - Referencing an array element with a subscript outside the declared bounds
 - Copying a string to a target with a shorter length than the source string
 - Declaring but not initializing a pointer variable, or using a pointer to allocated storage that has already been freed

If a routine exception occurred and you need more information than the condition handler provided, run your routine with the following run-time options, `TRAP(ON, NOSPIE)` and `TERMTHDACT(UAIMM)`. Setting these run-time options generates a system dump of the user address space of the original abend or program interrupt prior to the Language Environment condition manager processing the condition. After the system dump is taken by the operating system the Language Environment condition manager continues processing.

Debugging C/C++ input/output programs

You can use C/C++ conventions such as `__amrc` and `perror()` when you debug I/O operations.

Using the `__amrc` and `__amrc2` structures

`__amrc`, a structure defined in `stdio.h`, can help you determine the cause of errors resulting from an I/O operation, because it contains diagnostic information (for example, the return code from a failed VSAM operation).

There are two structures:

- `__amrc` (defined by type `__amrc_type`)
- `__amrc2` (defined by type `__amrc2_type`)

The `__amrc2_type` structure contains secondary information that C can provide.

Because any I/O function calls, such as `printf()`, can change the value of `__amrc` or `__amrc2`, make sure you save the contents into temporary structures of `__amrc_type` and `__amrc2_type` respectively, before dumping them.

Figure 133 shows the structure as it appears in **stdio.h**.

```

typedef struct __amrc_type {
[1]   union {
[2]       int      __error;
        struct {
            unsigned short __syscode,
[3]                __rc;
        } __abend;
        struct {
            unsigned char __fdbk_fill,
[4]                __rc,
                        __ftncd,
                        __fdbk;
        } __feedback;
        struct {
            unsigned short __svc99_info,
[5]                __svc99_error;
        } __alloc;
[1]   } __code;
[6]   unsigned int    __RBA;
[7]   unsigned int    __last_op;
        struct {
            unsigned int    __len_fill; /* __len + 4      */
            unsigned int    __len;
            char             __str[120];
            unsigned int    __parmr0;
            unsigned int    __parmr1;
            unsigned int    __fill2[2];
            char             __str2[64];
[8]   } __msg;
        unsigned char    __rplfdbwd[4];
    } __amrc_type;

```

Figure 133. `__amrc` structure

Figure 134 shows the `__amrc2` structure as it appears in **stdio.h**.

```

    struct {
[9]       int      __error2;
        char      __pad__error2[4];
[10]      FILE     *__fileptr;
[11]      int      __reserved{6};
    }

```

Figure 134. `__amrc2` structure

- [1] **union { ... } __code**
The error or warning value from an I/O operation is in `__error`, `__abend`, `__feedback`, or `__alloc`. Look at `__last_op` to determine how to interpret the `__code` union.
- [2] **__error** A structure that contains error codes for certain macros or services your application uses. Look at `__last_op` to determine the error codes. `__syscode` is the system abend code.
- [3] **__abend** A structure that contains the abend code when `errno` is set to indicate a recoverable I/O abend. `__rc` is the return code. For more information on abend codes, see *z/OS MVS System Codes*.
- [4] **__feedback** A structure that is used for VSAM only. The `__rc` stores the VSAM register 15, `__fdbk` stores the VSAM error code or reason code, and `__RBA` stores the RBA after some operations.
- [5] **__alloc** A structure that contains errors during `fopen` or `freopen` calls when defining files to the system using SVC 99.
- [6] **__RBA** The RBA value returned by VSAM after an ESDS or KSDS record is written out. For an RRDS, it is the calculated value from the record number. In AMODE 64 applications, you can no longer use the address of `_amrc._RBA` as the first argument to `flocate()`. Instead, `_amrc._RBA` must be placed into an unsigned long in order to make it 8 bytes wide, since `flocate()` is updated to indicate that size of (unsigned long) must be specified as the key length (second argument).
- [7] **__last_op** A field containing a value that indicates the last I/O operation being performed by C/C++ at the time the error occurred. These values are shown in Table 36 on page 372.
- [8] **__msg** May contain the system error messages from read or write operations emitted from the DFSMS/MVS SYNADAF macro instruction. Because the message can start with a hexadecimal address followed by a short integer, it is advisable to start printing at `MSG+6` or greater so the message can be printed as a string. Because the message is not null-terminated, a maximum of 114 characters should be printed. This can be accomplished by specifying a `printf` format specifier as `%.114s`.
- [9] **__error2** A secondary error code. For example, an unsuccessful rename or remove operation places its reason code here.
- [10] **__fileptr** A pointer to the file that caused a SIGIOERR to be raised. Use an `fldata()` call to get the actual name of the file.
- [11] **__reserved**
Reserved for future use.

__last_op values

The `__last_op` field is the most important of the `__amrc` fields. It defines the last I/O operation C/C++ was performing at the time of the I/O error. You should note that the structure is neither cleared nor set by non-I/O operations, so querying this field outside of a SIGIOERR handler should only be done immediately after I/O operations. Table 36 lists `__last_op` values you could receive and where to look for further information.

Table 36. *__last_op* values and diagnosis information

Value	Further Information
<code>__IO_INIT</code>	Will never be seen by SIGIOERR exit value given at initialization.
<code>__BSAM_OPEN</code>	Sets <code>__error</code> with return code from OS OPEN macro.
<code>__BSAM_CLOSE</code>	Sets <code>__error</code> with return code from OS CLOSE macro.
<code>__BSAM_READ</code>	No return code (either <code>__abend</code> (<code>errno == 92</code>) or <code>__msg</code> (<code>errno == 66</code>) filled in).
<code>__BSAM_NOTE</code>	NOTE returned 0 unexpectedly, no return code.
<code>__BSAM_POINT</code>	This will not appear as an error lastop.
<code>__BSAM_WRITE</code>	No return code (either <code>__abend</code> (<code>errno == 92</code>) or <code>__msg</code> (<code>errno == 65</code>) filled in).
<code>__BSAM_CLOSE_T</code>	Sets <code>__error</code> with return code from OS CLOSE TYPE=T.
<code>__BSAM_BLDL</code>	Sets <code>__error</code> with return code from OS BLDL macro.
<code>__BSAM_STOW</code>	Sets <code>__error</code> with return code from OS STOW macro.
<code>__TGET_READ</code>	Sets <code>__error</code> with return code from TSO TGET macro.
<code>__TPUT_WRITE</code>	Sets <code>__error</code> with return code from TSO TPUT macro.
<code>__IO_DEVTYPE</code>	Sets <code>__error</code> with return code from I/O DEVTYPE macro.
<code>__IO_RDJFCB</code>	Sets <code>__error</code> with return code from I/O RDJFCB macro.
<code>__IO_TRKCALC</code>	Sets <code>__error</code> with return code from I/O TRKCALC macro.
<code>__IO_OBTAIN</code>	Sets <code>__error</code> with return code from I/O CAMLST OBTAIN.
<code>__IO_LOCATE</code>	Sets <code>__error</code> with return code from I/O CAMLST LOCATE.
<code>__IO_CATALOG</code>	Sets <code>__error</code> with return code from I/O CAMLST CAT. The associated macro is CATALOG.
<code>__IO_UNCATALOG</code>	Sets <code>__error</code> with return code from I/O CAMLST UNCAT. The associated macro is CATALOG.
<code>__IO_RENAME</code>	Sets <code>__error</code> with return code from I/O CAMLST RENAME.
<code>__SVC99_ALLOC</code>	Sets <code>__alloc</code> structure with info and error codes from SVC 99 allocation.
<code>__SVC99_ALLOC_NEW</code>	Sets <code>__alloc</code> structure with info and error codes from SVC 99 allocation of NEW file.
<code>__SVC99_UNALLOC</code>	Sets <code>__unalloc</code> structure with info and error codes from SVC 99 unallocation.
<code>__C_TRUNCATE</code>	Set when C or C++ truncates output data. Usually this is data written to a text file with no newline such that the record fills up to capacity and subsequent characters cannot be written. For a record I/O file this refers to an <code>fwrite()</code> writing more data than the record can hold. Truncation is always rightmost data. There is no return code.
<code>__C_FCBCHECK</code>	Set when C or C++ FCB is corrupted. This is due to a pointer corruption somewhere. File cannot be used after this.
<code>__C_DBCS_TRUNCATE</code>	This occurs when writing DBCS data to a text file and there is no room left in a physical record for anymore double byte characters. A new-line is not acceptable at this point. Truncation will continue to occur until an SI is written or the file position is moved. Cannot happen if <code>MB_CUR_MAX</code> is 1.
<code>__C_DBCS_SO_TRUNCATE</code>	This occurs when there is not enough room in a record to start any DBCS string or else when a redundant SO is written to the file before an SI. Cannot happen if <code>MB_CUR_MAX</code> is 1.

Table 36. `__last_op` values and diagnosis information (continued)

Value	Further Information
<code>__C_DBCS_SI_TRUNCATE</code>	This occurs only when there was not enough room to start a DBCS string and data was written anyways, with an SI to end it. Cannot happen if <code>MB_CUR_MAX</code> is 1.
<code>__C_DBCS_UNEVEN</code>	This occurs when an SI is written before the last double byte character is completed, thereby forcing C or C++ to fill in the last byte of the DBCS string with a padding byte <code>X'FE'</code> . Cannot happen if <code>MB_CUR_MAX</code> is 1.
<code>__C_CANNOT_EXTEND</code>	This occurs when an attempt is made to extend a file that allows writing, but cannot be extended. Typically this is a member of a partitioned data set being opened for update.
<code>__VSAM_OPEN_FAIL</code>	Set when a low level VSAM OPEN fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_OPEN_ESDS</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_OPEN_RRDS</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_OPEN_KSDS</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_OPEN_ESDS_PATH</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_OPEN_KSDS_PATH</code>	Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS.
<code>__VSAM_MODCB</code>	Set when a low level VSAM MODCB macro fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_TESTCB</code>	Set when a low level VSAM TESTCB macro fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_SHOWCB</code>	Set when a low level VSAM SHOWCB macro fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_GENCB</code>	Set when a low level VSAM GENCB macro fails, sets <code>__rc</code> and <code>__fdbk</code> fields in the <code>__amrc</code> struct.
<code>__VSAM_GET</code>	Set when the last op was a low level VSAM GET; if the GET fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_PUT</code>	Set when the last op was a low level VSAM PUT; if the PUT fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_POINT</code>	Set when the last op was a low level VSAM POINT; if the POINT fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_ERASE</code>	Set when the last op was a low level VSAM ERASE; if the ERASE fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_ENDREQ</code>	Set when the last op was a low level VSAM ENDREQ; if the ENDREQ fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__VSAM_CLOSE</code>	Set when the last op was a low level VSAM CLOSE; if the CLOSE fails, sets <code>__rc</code> and <code>__fdbk</code> in the <code>__amrc</code> struct.
<code>__QSAM_GET</code>	<code>__error</code> is not set (if <code>abend (errno == 92)</code> , <code>__abend</code> is set, otherwise if read error (<code>errno == 66</code>), look at <code>__msg</code> .
<code>__QSAM_PUT</code>	<code>__error</code> is not set (if <code>abend (errno == 92)</code> , <code>__abend</code> is set, otherwise if write error (<code>errno == 65</code>), look at <code>__msg</code> .
<code>__QSAM_TRUNC</code>	This is an intermediate operation. You will only see this if an I/O <code>abend</code> occurred.

Table 36. *__last_op* values and diagnosis information (continued)

Value	Further Information
<code>__QSAM_FREEPOOL</code>	This is an intermediate operation. You will only see this if an I/O abend occurred.
<code>__QSAM_CLOSE</code>	Sets <code>__error</code> to result of OS CLOSE macro.
<code>__QSAM_OPEN</code>	Sets <code>__error</code> to result of OS OPEN macro.
<code>__CMS_OPEN</code>	Sets <code>__error</code> to result of FSOPEN.
<code>__CMS_CLOSE</code>	Sets <code>__error</code> to result of FSCLOSE.
<code>__CMS_READ</code>	Sets <code>__error</code> to result of FSREAD.
<code>__CMS_WRITE</code>	Sets <code>__error</code> to result of FSWRITE.
<code>__CMS_STATE</code>	Sets <code>__error</code> to result of FSSTATE.
<code>__CMS_ERASE</code>	Sets <code>__error</code> to result of FSERASE.
<code>__CMS_RENAME</code>	Sets <code>__error</code> to result of CMS RENAME command.
<code>__CMS_EXTRACT</code>	Sets <code>__error</code> to result of DMS EXTRACT call.
<code>__CMS_LINERD</code>	Sets <code>__error</code> to result of LINERD macro.
<code>__CMS_LINEWRT</code>	Sets <code>__error</code> to result of LINEWRT macro.
<code>__CMS_QUERY</code>	<code>__error</code> is not set.
<code>__HSP_CREATE</code>	Indicates last op was a DSPSERV CREATE to create a hiperspace for a hiperspace memory file. If CREATE fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__HSP_DELETE</code>	Indicates last op was a DSPSERV DELETE to delete a hiperspace for a hiperspace memory file during termination. If DELETE fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__HSP_READ</code>	Indicates last op was a HSPSERV READ from a hiperspace. If READ fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__HSP_WRITE</code>	Indicates last op was a HSPSERV WRITE to a hiperspace. If WRITE fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__HSP_EXTEND</code>	Indicates last op was a HSPSERV EXTEND during a write to a hiperspace. If EXTEND fails, stores abend code in <code>__amrc_code_abend_syscode</code> , reason code in <code>__amrc_code_abend_rc</code> .
<code>__LFS_OPEN</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
<code>__LFS_CLOSE</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
<code>__LFS_READ</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .

Table 36. `__last_op` values and diagnosis information (continued)

Value	Further Information
<code>__LFS_WRITE</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
<code>__LFS_LSEEK</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
<code>__LFS_FSTAT</code>	Sets <code>__error</code> with reason code from HFS services. Reason code from HFS services must be broken up. The low order 2 bytes can be looked up in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .

Displaying an error message with the `perror()` function

To find a failing routine, check the return code of all function calls. After you have found the failing routine, use the `perror()` function after the routine to display the error message. `perror()` displays the string that you pass to it and an error message corresponding to the value of `errno`. `perror()` writes to the standard error stream (`stderr`).

If you need additional diagnostic information set the environment variable, `_EDC_ADD_ERRNO2` to 1, and that will append the current `errno2` value to the end of the `perror()` string.

Figure 135 is an example of a routine using `perror()`.

```
#include <stdio.h>
int main(void){
    FILE *fp;

    fp = fopen("myfile.dat", "w");
    if (fp == NULL)
        perror("fopen error");
}
```

Figure 135. Example of a routine using `perror()`

Using `__errno2()` to diagnose application problems

Use `__errno2()` when diagnosing problems in a z/OS UNIX or an OpenExtensions application. This function enables C/C++ application programs to access diagnostic information returned to the C/C++ run-time library from an underlying kernel callable service. `__errno2()` returns the reason code of the last failing kernel callable service called by the C/C++ run-time library. The returned value is intended for diagnostic display purposes only. The function call is always successful.

Note: Since the `__errno2()` function returns the reason code of the kernel callable service that last failed, and not all function calls invoke the kernel, the value returned by `__errno2()` may be misleading.

Figure 136 on page 376 is an example of a routine using `__errno2()`.

```

#include <stdio.h>
#include <errno.h>
FILE *myfopen(const char *fn, const char *mode) {
    FILE *f;
    f = fopen(fn,mode);
    if (f==NULL) {
        perror("fopen() failed");
        printf("__errno2 = %08x\n", __errno2());
    }
    return(f);
}

```

Figure 136. Example of a routine using __errno2()

Figure 137 is an example of a routine using the environment variable `_EDC_ADD_ERRNO2`, and Figure 138 shows the sample output from that routine.

```

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int main(void) {
    FILE *fp;

    /* add errno2 to perror message */
    setenv("_EDC_ADD_ERRNO2","1",1);

    fp = fopen("testfile.dat", "r");
    if (fp == NULL)
        perror("fopen error");

}

```

Figure 137. Example of a routine using _EDC_ADD_ERRNO2

```

fopen error: EDC5129I No such file or directory.
(errno2=0x05620062)

```

Figure 138. Sample output of a routine using _EDC_ADD_ERRNO2

Using C/C++ listings

The following sections discuss C/C++ listings generated when the executable program is created. They also explain how to use these listings to locate information, such as variable values and the timestamp, in the dump.

Generating C/C++ listings and maps

The listings and maps created by the compile and bind steps provide many pieces of information necessary for performing problem analysis tasks. The map of the Writable Static Area (WSA) is provided by the binder in the output listing in the `C_WSA64` section.

In addition, the `@STATIC` is replaced by the binder with `$PRIVnnnnnn` and to find the source listing use the cross reference to associate `$PRIVnnnnnn` with the defining section name and use the section name to find the source in the module

map. So, the output listing provided by the binder should be used when locating variables in executable programs created without using the prelink step.

When you are debugging, you can use various options depending upon which compiler you are using. The following section provides an overview of each listing and specifies the compiler option to use. For a detailed description of available listings, see *z/OS XL C/C++ User's Guide*.

Table 37. Contents of listing and associated compiler options

Name	Compiler Option	Function
Pseudo-assembler listing	LIST	Generates a pseudo-assembler listing, which shows the source listing for the current routine.
Storage Offset Listing	XREF	Produces a storage offset listing, which includes in the source listing a cross reference table of names used in the routine and the line numbers on which they were declared or referenced, and a static map.
Structure Map	AGGREGATE (C only)	Causes a structure map to be included in the source listing. The structure map shows the layout of variables for the type struct or union.
Inline Report	INLRPT and INLINE(,REPORT,,)	Generates an inline report that summarizes all functions inlined and provides a detailed call structure of all the functions.
Binder Output Listing	MAP, LIST, XREF (binder option)	These options control the listing output from the link-edit process.
Source Listing	SOURCE	Generates the source listing, which contains the original source input statements and any compiler diagnostic messages issued for the source.
Cross-Reference Listing	XREF	Cross-reference table containing a list of the identifiers from the source program and the line numbers in which they appear.
External Symbol Cross Reference Listing	ATTR (C only) or XREF	Shows the original name and corresponding mangled name for each symbol.
Object File Map	IPA(MAP)	Displays the names of the object files that were used as input to the IPA Link step.
Source File Map	IPA(MAP)	Identifies the source files included in the object files.
Compiler Options Map	IPA(MAP)	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.
Global Symbols Map	IPA(MAP)	Shows how global symbols are mapped into members of global data structures by the global variable coalescing optimization process. This section will only be output to the listing if the IPA link phase coalesces global variables.
Inline Report for IPA Inliner	INLINE(,REPORT,,) or INLRPT	Describes the actions performed by the IPA Inliner.

C, C++, and C/C++ IPA listings

The options for each listing vary depending upon which Compiler is used. The following section illustrates which options are available for each listing.

XL C compiler listings: The following table specifies which listings are available for the C compiler, and which option(s) must be specified to obtain it.

Table 38. XL C compiler listings

Name	Compiler Option
Source Program	SOURCE
Cross-Reference Listing	XREF
Structure and Union Maps	AGGREGATE
Inline Report	INLINE(,REPORT,,) or INLRPT
Pseudo Assembly Listing	LIST
Storage Offset Listing	XREF

XL C++ compiler listings: The following table specifies which listings are available for the C++ compiler, and which option(s) must be specified to obtain it.

Table 39. XL C++ compiler listings

Name	Compiler Option
Source Program	SOURCE
Cross-Reference Listing	ATTR and XREF
Inline Report	OPTIMIZE and INLINE(,REPORT,,) or INLRPT
Pseudo Assembly Listing	LIST
External Symbol Cross Reference Listing	ATTR or XREF

XL C/C++ IPA link step listings: The following table specifies which listings are available for the C/C++ IPA Link Step, and which option(s) must be specified to obtain it. The syntax for passing IPA(MAP) to the IPA Link phase is either -WI,IPA(MAP) or -WI,I,IPA(MAP). All of the options should be passed to the IPA Link phase using the syntax -WI,I,option. For example, -WI,I,INLRPT.

Table 40. XL C/C++ IPA link step listings

Name	Compiler Option
Object File Map	IPA (MAP)
Source File Map	IPA (MAP)
Compiler Options Map	IPA (MAP)
Global Symbols Map	IPA (MAP)
Inline Report for IPA Inliner	INLINE(,REPORT,,) or INLRPT
Partition Map	IPA (MAP)
Cross Reference Listing	XREF
Pseudo Assembly Listing	LIST

Finding variables

You can determine the value of a variable in the routine at the point of interrupt by using the compiled code listing as a guide to its address, then finding this address in the Language Environment dump or system dump. The method you use depends on the storage class of variable.

It is possible for the routine to be interrupted before the value of the variable is placed in the location provided for it. This can explain unexpected values in the dump.

Steps for finding automatic variables

Perform the following steps to find automatic variables in the Language Environment dump or system dump:

1. Determine the name of the automatic variable and the function it is defined in. As an example, we will find the variable **aa** in the function **main** from the program **cdivzero** shown in Figure 53 on page 172.
2. From the compiler listing, locate the variable in the storage offset listing:

```
aa          5823-0:10      Class = automatic,      Location = 2248(r4),      Length = 4
```

The location is specified as decimal offset (base register). So variable **aa** is located at register 4 + 2248 (X'8C8').

3. From the Traceback (in the Language Environment dump or in the formatted output from the IPCS VERBEXIT LEDATA CEEDUMP subcommand for a system dump) locate the function:

DSA	Entry	E Offset	Load Mod	Program Unit	Service	Status
00000004	main	+00000016	CDIVZERO			Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Attributes
00000004	00000001082FF180	00000000251000C0	0000000000000000	*****	20040408	XPLINK EBCDIC IEEE

If the base register is r4, the register 4 value is always the DSA address for the function.

If the base register is not r4, the register value must be located from saved registers.

If the Status field indicates Exception, use the saved registers from when the condition occurred. In the Language Environment dump, the saved registers can be found in the Condition information associated with the DSA address in the Condition Information for Active Routines section. In the formatted output from the IPCS VERBEXIT LEDATA CM subcommand for a system dump, the saved registers can be found in the CIBH that has the DSA address as the value for the SV1 field.

If the Status field indicates Call, use the saved registers from the DSA address that appears on the line above the function in the Traceback. In the Language Environment dump, the DSAs can be found in the "Control Blocks for Active Routines" section. In the formatted output from the IPCS VERBEXIT LEDATA 'STACK' subcommand for a system dump, the DSAs can be found in the "DSA backchain" section.

Note: Some functions do not save all registers.

4. Add the register value to the offset of the variable to obtain the address of the variable. In the Language Environment dump, the contents of the variable can be read in the DSA Frame section corresponding to the function the variable is contained in. For a system dump, use the IPCS LIST subcommand to display the storage where the variable is located.

The address for variable **aa** is X'1082FF080' + X'980' = X'1082FFA00'.

Restriction: The parameter value might never be stored, since the first few parameters might be passed in registers and there might be no need to save them.

Steps for finding C/C++ parameters

The C/C++ parameter list is always located in the caller's DSA at offset 2176 (X'880'). Parameters that are passed in registers are not always stored in the

parameter list. The compiler option XPLINK(STOREARGS) can be used to ensure that all parameters are stored in the parameter list.

Perform the following steps to find parameters in the Language Environment dump or system dump:

1. Determine the name of the parameter and the function it is for. As an example, we will find the parameter **pp** for the function **funcb** from the program **cdivzero** shown in Figure 53. C routine with a divide-by-zero error.
2. From the compiler listing, locate the parameter in the storage offset listing:
3. From the Traceback (in the Language Environment dump or in the formatted output from the IPCS VERBEXIT LEDATA 'CEEDUMP' subcommand for a system dump) locate the function:

```
pp          5828-0:15      Class = parameter,      Location = 2432(r4),      Length = 8

DSA      Entry      E Offset Load Mod Program Unit Service Status
00000003 funcb      +0000002E CDIVZERO                                Exception

DSA      DSA Addr      E Addr      PU Addr      PU Offset Comp Date Attributes
00000003 000000001082FF080 0000000025100108 0000000000000000 ***** 20040408 XPLINK EBCDIC IEEE
```

If the base register is r4, the register 4 value is always the DSA address for the function.

If the base register is not r4, the register value must be located from saved registers.

If the Status field indicates Exception, use the saved registers from when the condition occurred. In the Language Environment dump, the saved registers can be found in the Condition information associated with the DSA address in the "Condition Information for Active Routines" section. In the formatted output from the IPCS VERBEXIT LEDATA 'CM' subcommand for a system dump, the saved registers can be found in the CIBH that has the DSA address as the value for the SV1 field.

If the Status field indicates Call, use the saved registers from the DSA address that appears on the line above the function in the Traceback. In the Language Environment dump, the DSAs can be found in the "Control Blocks for Active Routines" section. In the formatted output from the IPCS VERBEXIT LEDATA 'STACK' subcommand for a system dump, the DSAs can be found in the "DSA backchain" section.

Note: Some functions do not save all registers.

4. Add the register value to the offset of the parameter to obtain the address of the parameter. In the Language Environment dump, the contents of the parameter can be read in the DSA Frame section corresponding to the function that passed the parameter. For a system dump, use the IPCS LIST subcommand to display the storage where the parameter is located.

The address for parameter **pp** is X'1082FF080' + X'980' = X'1082FFA00'.

Steps for finding members of aggregates

You can define aggregates in any of the storage classes or pass them as parameters to a called function. The first step is to find the start of the aggregate. You can compute the start of the aggregate as described in previous sections, depending on the type of aggregate used.

The aggregate map provided for each declaration in a routine can further assist in finding the offset of a specific variable within an aggregate. Structure maps are generated using the AGGREGATE compiler option. Figure 139 on page 381 shows an example of an aggregate.

```

typedef struct {
    int asid;
    void *addr;
    asfAmodeType amode;
} asfTargetRef;
asfTargetRef tempTargetRef;

```

Figure 139. Example code for structure variables

Figure 140 shows an example of aggregate map.

Aggregate map for: struct with no tag #68			Total size: 24 bytes
asfTargetRef			
Offset Bytes(Bits)	Length Bytes(Bits)	Member Name	
0	4	asid	
4	4	***PADDING***	
8	8	addr	
16	1	amode	
17	7	***PADDING***	

Figure 140. Example of aggregate map

To find the value of variable **tempTargetRef.addr**:

1. Locate the automatic variable **tempTargetRef** in the storage offset listing:

```

tempTargetRef      209-0:209      Class = automatic,      Location = 2264(r4),      Length = 24

```

The variable **tempTargetRef** is located at register 4 + 2264 (X'8D8'). For this example, assume that the register 4 value is X'1082FD3E0'. The result is X'1082FDCB8'(X'1082FD3E0' + X'8D8'). This is the address of the value of the automatic variable **tempTargetRef** in the dump

2. Find the offset of **addr** in the Aggregate Map, shown in Figure Figure 140. The offset is 8. Add the offset from the Aggregate Map to the address of the **tempTargetRef** variable.

The result is X'1082FDCC0' (X'1082FDCB8' + X'8'). This is the address of the value of **tempTargetRef.addr** in the dump

Generating a Language Environment dump of a C/C++ routine

You can use the `cdump()`, `csnap()`, and `ctrace()` C/C++ functions to generate a Language Environment dump of C/C++ routines.

cdump()

If your routine is running under z/OS, you can generate useful diagnostic information by using the `cdump()` function. `cdump()` produces a main storage dump with the activation stack.

When `cdump()` is invoked from a user routine, the C/C++ library issues an OS IEATDUMP macro to obtain a dump of virtual storage. You can use the Interactive Problem Control System (IPCS) to format and analyze IEATDUMP dumps.

The DD definition for CEESNAP must include the desired data set name and DCB information:

```
LRECL=4160, BLKSIZE=4160, and RECFM=FBS
```

If the data set is not defined, or is not usable for any reason, `cdump()` returns a failure code of 1. This occurs even if the call to CEE3DMP is successful.

Because `cdump()` returns a code of 0 only if the IEATDUMP was successful or 1 if it was unsuccessful, you cannot distinguish whether a failure of `cdump()` occurred in the call to CEE3DMP or IEATDUMP. A return code of 0 is issued only if both IEATDUMP and CEE3DMP are successful.

Support for IEATDUMP dumps using the `_cdump` function is provided only under z/OS. In addition to a IEATDUMP dump, a Language Environment formatted dump is also taken.

csnap()

The `csnap()` function produces a condensed storage dump.

To use these functions, you must add `#include <ctest.h>` to your C/C++ code. The dump is directed to output *dumpname*, which is specified in a `//CEEDUMP DD` statement in JCL.

Refer to the *z/OS XL C/C++ Run-Time Library Reference* for more details about the syntax of these functions.

ctrace()

The `ctrace()` function produces a traceback and includes the offset addresses from which the calls were made.

Sample C routine that calls `cdump()`

Figure 141 on page 383 shows a sample C routine that uses the `cdump` function to generate a dump.

Figure 146 on page 387 shows the dump output.

```

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

void hsigfpe(int);
void hsigterm(int);
void atfl(void);

typedef int (*FuncPtr_T)(void);

int st1    = 99;
int st2    = 255;
int xcount = 0;

int main(void) {
    /*
     * 1) Open multiple files
     * 2) Register 2 signals
     * 3) Register 1 atexit function
     * 4) Fetch and execute a module
     */

    FuncPtr_T fetchPtr;
    FILE*     fp1;
    FILE*     fp2;
    int       rc;
    fp1 = fopen("myfile.data", "w");
    if (!fp1) {
        perror("Could not open myfile.data for write");
        exit(101);
    }

    fprintf(fp1, "record 1\n");
    fprintf(fp1, "record 2\n");
    fprintf(fp1, "record 3\n");

    fp2 = fopen("memory.data", "wb,type=memory");
    if (!fp2) {
        perror("Could not open memory.data for write");
        exit(102);
    }
    fprintf(fp2, "some data");
    fprintf(fp2, "some more data");
    fprintf(fp2, "even more data");

    signal(SIGFPE, hsigfpe);
    signal(SIGTERM, hsigterm);

    rc = atexit(atfl);
    if (rc) {
        fprintf(stderr, "Failed on registration of atexit function atfl\n");
        exit(103);
    }
}

```

Figure 141. Example C routine using `cdump()` to generate a dump (Part 1 of 2)

```

    fetchPtr = (FuncPtr_T) fetch("MODULE1");
    if (!fetchPtr) {
        fprintf(stderr, "Failed to fetch MODULE1\n");
        exit(104);
    }
    fetchPtr();
    return(0);
}

void hsigfpe(int sig) {
    ++st1;
    return;
}

void hsigterm(int sig) {
    ++st2;
    return;
}

void atf1() {
    ++xcount;
}

```

Figure 141. Example C routine using cdump() to generate a dump (Part 2 of 2)

Figure 142 shows a fetched C module:

```

#include <ctest.h>

#pragma linkage(func1, fetchable)
int func1(void) {
    cdump("This is a sample dump");
    return(0);
}

```

Figure 142. Fetched module for C routine

Sample C++ routine that generates a Language Environment dump

Figure 143 on page 385 shows a sample C++ routine that uses a protection exception to generate a dump.

```

#include <iostream.h>
#include <ctest.h>
#include "stack.h"

int main() {
    cout << "Program starting:\n";
    cerr << "Error report:\n";

    Stack<int> x;
    x.push(1);
    cout << "Top value on stack : " << x.pop() << '\n';
    cout << "Next value on stack: " << x.pop() << '\n';
    return(0);
}

```

Figure 143. Example C++ routine with protection exception generating a dump

Figure 144 shows the template file `stack.c`

```

#ifndef __STACK__
#include "stack.h"
#endif

template <class T> T Stack<T>::pop() {
    T value = head->value;
    head = head->next;

    return(value);
}

template <class T> void Stack<T>::push(T value) {
    Node* newNode = new Node;
    newNode->value = value;
    newNode->next = head;
    head = newNode;
}

```

Figure 144. Template file STACK.C

Figure 145 on page 386 shows the header file **stack.h**.

```

#ifndef __STACK
#define __STACK__
template <class T> class Stack {
public:
    Stack() {
        char* badPtr = 0; badPtr -= (0x01010101);
        head = (Node*) badPtr; /* head initialized to 0xFEFEFEFF */
    }
    T pop();
    void push(T);
private:
    struct Node {
        T value;
        struct Node* next;
    }* head;
};
#endif

```

Figure 145. Header file STACK.H

Sample Language Environment dump with C/C++-specific information

This sample dump was produced by compiling the routine in Figure 141 on page 383, then running it. Notice the sequence of calls in the traceback section - CELQINIT is the Language Environment module that invokes main and @@FECBMODULE1 fetches the user-defined function func1, which in turn calls the library routine __cdump.

Information for enclave main

Information for thread 2547D8200000000

Traceback:

DSA	Entry	E	Offset	Load Mod	Program Unit	Service	Status
00000001	_cdump	+00000000	CELQLIB			HLE7709	Call
00000002	func1	+00000016	MODULE1				Call
00000003	@@FECBMODULE1						
		-0044FB34			** NoName **		Call
00000004	main	+000001AE	CSAMPLE				Call
00000005	CELQINIT	+00001146	CELQLIB	CELQINIT		HLE7709	Call

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Attributes
00000001	00000001082FEDC0	00000000	254ECD78	0000000000000000	*****	20040312	XPLINK EBCDIC POSIX IEEE
00000002	00000001082FEF00	00000000	255A90C0	0000000000000000	*****	20040408	XPLINK EBCDIC POSIX IEEE
00000003	00000001082FF000	00000000	255AB058	00000000255AB048	0044FB24		XPLINK EBCDIC POSIX Floating Point
00000004	00000001082FF180	00000000	251000C0	0000000000000000	*****	20040408	XPLINK EBCDIC POSIX IEEE
00000005	00000001082FF280	00000000	25102010	0000000025102010	00001146	20040312	XPLINK EBCDIC POSIX Floating Point

Storage for Active Routines:

DSA frame(00000001082FEDC0)

+0800	00000001082FF5C0	00000001	082FEF00	00000000	2559DBE0	
+0810	00000001082FF5D0	00000000	254ECD78	00000000	255A90D8	!..Q
+0820	00000001082FF5E0	00000000	2515B428	00000001	082FF800	8..
+0830	00000001082FF5F0	00000001	082FF880	00000000	00000000	8..
+0840	00000001082FF600	00000001	00006160	00000001	082FF180	/-...1.
+0850	00000001082FF610	00000001	000098D0	00000000	255AB048	q.....!..
+0860	00000001082FF620	00000000	253D2E42	00000000	00000000	
+0870	00000001082FF630	00000000	25598CE0	00000001	00006160	/-
+0880	00000001082FF640	00000001	082FF6A0	00000000	254ED42B	6.....+M.
+0890	00000001082FF650	00000001	082FF690	00000000	00000000	6..
+08A0	00000001082FF660	00000000	00000000	00000000	00000000	
+08B0	00000001082FF670	00000000	00000000	00000001	082FF75C	7*
+08C0	00000001082FF680	00000000	255AB0E0	00000000	255AB0C8	!.....!..H
+08D0	00000001082FF690	00010C7B	49C3C5C5	00000000	00000000	...#.CEE.....
+08E0	00000001082FF6A0	E3889A2	4089A240	8140A281	94979385	This is a sample
+08F0	00000001082FF6B0	4084A494	97404040	40404040	40404040	dump
+0900	00000001082FF6C0	40404040	40404040	40404040	40404040	
+0910	00000001082FF6D0	- +00092F	00000001082FF6EF			same as above
+0930	00000001082FF6F0	00000001	083010D0	00000000	00000020	

DSA frame(00000001082FEF00)

+0800	00000001082FF700	00000001	082FF000	00000000	25598CE0	0.....
+0810	00000001082FF710	00000000	255A90C0	00000000	2515B526	!.....
+0820	00000001082FF720	00000000	2515B428	00000001	082FF800	8..
+0830	00000001082FF730	00000001	082FF880	00000000	00000000	8..
+0840	00000001082FF740	00000001	00006160	00000001	082FF180	/-...1.
+0850	00000001082FF750	00000001	000098D0	00000000	255AB048	q.....!..
+0860	00000001082FF760	00000000	25100300	00000001	00006160	/-
+0870	00000001082FF770	00000000	25591120	00000001	00008C80	
+0880	00000001082FF780	00000000	255A90F0	00000001	00006160	!..0...../-
+0890	00000001082FF790	00000001	00006160	00000001	082FF85C	/-...8*
+08A0	00000001082FF7A0	00000001	082FF8E0	00000001	082FF8E8	8.....8Y
+08B0	00000001082FF7B0	00000001	082FF8F0	00000001	084151D0	80.....
+08C0	00000001082FF7C0	00000000	00000000	00000000	255A7750	!..&
+08D0	00000001082FF7D0	00000000	00000009	00000000	25100391	j
+08E0	00000001082FF7E0	40404040	40404040	00000000	00000001	
+08F0	00000001082FF7F0	00000000	00000000	00000001	000074F8	8

Figure 146. Example dump from sample C routine (Part 1 of 4)

```

DSA frame(00000001082FF000)
+0800 00000001 082FF800 00000001 082FF180 C0F0FFFF EFA407FF |.....1..0...u..|
+0810 00000001 082FF810 00000000 255AB058 00000000 25100270 |.....!.....|
+0820 00000001 082FF820 00000000 25100300 00000000 255AB178 |.....!.....|
+0830 00000001 082FF830 00000001 08300090 00000001 08415100 |.....|
+0840 00000001 082FF840 00000001 000039D0 00000000 00006F48 |.....?..|
+0850 00000001 082FF850 00000000 7F7864E8 00000000 0000001F |.....".Y.....|
+0860 00000001 082FF860 00000007 25100300 00000000 255A7730 |.....!.....|
+0870 00000001 082FF870 00000001 00001000 00000001 08415100 |.....|
+0880 00000001 082FF880 00000001 000039D0 00000000 00006F48 |.....?..|
+0890 00000001 082FF890 00000000 7F7864E8 00000000 00000019 |.....".Y.....|
+08A0 00000001 082FF8A0 00000000 2511F158 00000000 00000000 |.....1.....|
+08B0 00000001 082FF8B0 00000000 00000000 00000000 255A9000 |.....!.....|
+08C0 00000001 082FF8C0 00000001 08415238 00000000 255A9000 |.....!.....|
+08D0 00000001 082FF8D0 00000000 255A9000 00000001 082FF89C |.....!.....8..|
+08E0 00000001 082FF8E0 00000001 083010D0 00000000 255AB048 |.....!.....|
+08F0 00000001 082FF8F0 00000000 00000000 00000000 00000000 |.....|
+0900 00000001 082FF900 00000000 00000000 180F58FF 001007FF |.....|
+0910 00000001 082FF910 58F0C210 58F0F694 58F0C2B8 58F0FFA4 |..0B..06m.0B..0.u|
+0920 00000001 082FF920 00000001 08300080 00000000 00000000 |.....|
+0930 00000001 082FF930 D4D6C4E4 D3C5F140 00000000 00001000 |MODULE1 ..|
+0940 00000001 082FF940 00000000 00000000 00000000 00000000 |.....|
+0950 00000001 082FF950 00000000 00000008 00000000 00000000 |.....|
+0960 00000001 082FF960 00000000 00000000 00000000 00000000 |.....|
+0970 00000001 082FF970 00000000 00000000 00000001 00000000 |.....|

```

```

DSA frame(00000001082FF180)
+0800 00000001 082FF980 00000001 082FF280 00000000 00000000 |.....2.....|
+0810 00000001 082FF990 00000000 251000C0 00000000 25103158 |.....|
+0820 00000001 082FF9A0 00000000 00006FE0 00000000 25100870 |.....?.....|
+0830 00000001 082FF9B0 00000000 251004C8 00000001 08415100 |......H.....|
+0840 00000001 082FF9C0 00000001 000039D0 00000000 00006F48 |.....?..|
+0850 00000001 082FF9D0 00000000 7F7864E8 00000000 0000001F |.....".Y.....|
+0860 00000001 082FF9E0 00000000 00000000 00000000 00000000 |.....|
+0870 00000001 082FF9F0 - +00087F 00000001 082FF9FF |.....| same as above
+0880 00000001 082FFA00 00000000 255AB170 00000000 00000020 |.....!.....|
+0890 00000001 082FFA10 00000000 255AB178 00000000 00000000 |.....!.....|
+08A0 00000001 082FFA20 00000000 00000000 00000000 00000000 |.....|
+08B0 00000001 082FFA30 - +0008FF 00000001 082FFA7F |.....| same as above

```

```

DSA frame(00000001082FF280)
+0800 00000001 082FFA80 00000001 082FF760 00000000 00000000 |.....7-.....|
+0810 00000001 082FFA90 00000000 25102010 00000000 251032FE |.....|
+0820 00000001 082FFAA0 00000000 00000000 00000000 00000000 |.....|
+0830 00000001 082FFAB0 - +00087F 00000001 082FFAFF |.....| same as above
+0880 00000001 082FFB00 00000001 082FFD04 00000001 00006160 |......./-|
+0890 00000001 082FFB10 00000001 082FF760 00000000 00000000 |.....7-.....|
+08A0 00000001 082FFB20 00000000 00000000 00000000 00000000 |.....|
+08B0 00000001 082FFB30 - +0009FF 00000001 082FFCF7 |.....| same as above
+08A0 00000001 082FFC80 00000000 00000000 00000001 082FFD04 |.....|
+08A10 00000001 082FFC90 00000000 00000000 00000000 00000000 |.....|
+08A20 00000001 082FFCA0 - +000A7F 00000001 082FFCFF |.....| same as above
+08A80 00000001 082FFD00 00000000 00000001 00000000 00000000 |.....|
+08A90 00000001 082FFD10 00000000 00000000 00000000 00000000 |.....|
+08AA0 00000001 082FFD20 - +000ADF 00000001 082FFD5F |.....| same as above
+08AE0 00000001 082FFD60 00000000 00000000 00000001 00002468 |.....|
+08AF0 00000001 082FFD70 00000001 000039D0 00000001 00006160 |......./-|
+08B00 00000001 082FFD80 00000000 00000000 00000000 00000000 |.....|
+08B10 00000001 082FFD90 - +000B4F 00000001 082FFDCf |.....| same as above
+08B50 00000001 082FFDD0 00000000 00000000 00000001 08300090 |.....|
+08B60 00000001 082FFDE0 00000000 00000000 00000000 00000000 |.....|
+08B70 00000001 082FFDF0 - +000BAF 00000001 082FFE2F |.....| same as above
+08B80 00000001 082FFE30 00000001 08300090 00000000 00000003 |.....|
+08BC0 00000001 082FFE40 00000001 084151E0 00000001 084132F0 |.....0|
+08BD0 00000001 082FFE50 00000001 082FF280 00000000 25103240 |.....2.....|
+08BE0 00000001 082FFE60 00000000 251000C0 00000000 2511F158 |.....1..|
+08BF0 00000001 082FFE70 00000000 00000000 00000000 00000000 |.....|
+08C00 00000001 082FFE80 - +000CDF 00000001 082FFF5F |.....| same as above

```

CEE3DMP called by program unit (entry point __cdump) at offset +000000000.

Figure 146. Example dump from sample C routine (Part 2 of 4)

Registers on Entry to CEE3DMP:

```

PM..... 0100
GPR0..... ***** GPR1..... ***** GPR2..... ***** GPR3..... *****
GPR4..... 00000001082FEDC0 GPR5..... 0000000025195798 GPR6..... 0000000025191A20 GPR7..... 00000000254ECEC4
GPR8..... 000000002559DBE0 GPR9..... 00000001082FF6A0 GPR10..... 00000000255A90F0 GPR11..... 0000000000000001
GPR12..... 0000000100006160 GPR13..... 00000001082FF180 GPR14..... 00000001000098D0 GPR15..... 00000000255AB048

GPREG STORAGE:
Storage around GPR0 is invalid.
Storage around GPR1 is invalid.
Storage around GPR2 is invalid.
Storage around GPR3 is invalid.
Storage around GPR4 (00000001082FEDC0)
+0800 00000001082FF5C0 00000001 082FEF00 00000000 2559DBE0 |.....|
+0810 00000001082FF5D0 00000000 254ECD78 00000000 255A90D8 |.....+......!.Q|
+0820 00000001082FF5E0 00000000 2515B428 00000001 082FF800 |......8.|
+0830 00000001082FF5F0 00000001 082FF880 00000000 00000000 |.....8.....|
+0840 00000001082FF600 00000001 00006160 00000001 082FF180 |...../.....1.|
+0850 00000001082FF610 00000001 000098D0 00000000 255AB048 |.....q.....!.|
Storage around GPR5 (0000000025195798)
-0020 0000000025195778 00000000 00000000 00000000 00000000 |.....|
-0010 0000000025195788 - +FFFFFFF 0000000025195797 |.....|
+0000 0000000025195798 00000FC8 00000FC8 02260000 00000000 |...H...H.....|
+0010 00000000251957A8 00000000 25195850 00000000 25195844 |.....&.....|
+0020 00000000251957B8 00000000 25195820 00000000 25195820 |.....|
+0030 00000000251957C8 - +00003F 00000000251957D7 |.....|
Storage around GPR6 (0000000025191A20)
-0020 0000000025191A00 F0F6F0F0 0007C8D3 C5F7F7F0 F9000000 |0600..HLE7709...|
-0010 0000000025191A10 00C300C5 00C500F1 00003F20 000007E0 |.C.E.E.1.....|
+0000 0000000025191A20 EB134880 0024B904 0004A748 F820EB5F |.....x.8..^|
+0010 0000000025191A30 48080024 E3040800 00244190 4FFFE813 |.....T.....|...|
+0020 0000000025191A40 49800024 E38004B8 0017E380 80580004 |.....T.....T.....|
+0030 0000000025191A50 E3C08008 0004A788 00504080 4CFC1F88 |T.....xh.& .<..h|
Storage around GPR7 (00000000254ECEC4)
-0020 00000000254ECEA4 C0600000 024E4470 6060C070 00000249 |.-...+.-.....|
-0010 00000000254ECEB4 EB568000 00044130 48D04120 70EB0D76 |.....|
+0000 00000000254ECEC4 07004800 48D0B914 0000A70E 0003A784 |.....x...xd|
+0010 00000000254ECED4 FF71A70E 0005A784 FF6DB902 00BBA774 |..x...xd...x.|
+0020 00000000254ECEE4 FF694130 0000EB48 48000004 47F07002 |......0...|
+0030 00000000254ECF4 00000000 00C300C5 00C500F1 00000600 |.....C.E.E.1....|
Storage around GPR8 (000000002559DBE0)
-0020 000000002559DBC0 00000000 252173D4 00000000 25215398 |......M.....q|
-0010 000000002559DBD0 00000000 252255B8 00000000 25225458 |.....|
+0000 000000002559DBE0 00000000 25195798 00000000 25191A20 |.....q.....|
+0010 000000002559DBF0 00000000 254B4BC8 00000000 254B4BC8 |......H.....H|
+0020 000000002559DC00 00000000 2519B7A4 00000000 2519B350 |......u.....&|
+0030 000000002559DC10 0001018F 49C3C5C5 00000000 00000000 |.....CEE.....|
Storage around GPR9 (00000001082FF6A0)
-0020 00000001082FF680 00000000 255AB0E0 00000000 255AB0C8 |.....!......!.H|
-0010 00000001082FF690 00010C7B 49C3C5C5 00000000 00000000 |...#.CEE.....|
+0000 00000001082FF6A0 E3889A2 4089A240 8140A281 94979385 |This is a sample|
+0010 00000001082FF6B0 4084A494 97404040 40404040 40404040 |dump|
+0020 00000001082FF6C0 40404040 40404040 40404040 40404040 |.....|
+0030 00000001082FF6D0 - +00003F 00000001082FF6DF |.....|
Storage around GPR10(00000000255A90F0)
-0020 00000000255A90D0 C0100000 00100D76 0700A739 0000E370 |.....x...T..|
-0010 00000000255A90E0 48180004 41404100 47F07002 00000000 |......0.....|
+0000 00000000255A90F0 E3889A2 4089A240 8140A281 94979385 |This is a sample|
+0010 00000000255A9100 4084A494 97000000 02CE0300 00000028 |dump.....|
+0020 00000000255A9110 80800281 00000503 0000002C 01000000 |...a.....|
+0030 00000000255A9120 000586A4 9583F140 FFFFFFA8 00000000 |..func1 ...y....|
Storage around GPR11(0000000000000001)
-0001 0000000000000000 Inaccessible storage.
+000F 0000000000000010 Inaccessible storage.
+001F 0000000000000020 Inaccessible storage.
+002F 0000000000000030 Inaccessible storage.
+003F 0000000000000040 Inaccessible storage.
+004F 0000000000000050 Inaccessible storage.
Storage around GPR12(0000000100006160)
-0020 0000000100006140 00000000 00000000 C3C5C5C3 C1C14040 |.....CEECAA|
-0010 0000000100006150 00000000 00000000 00000000 00000000 |.....|
+0000 0000000100006160 - +00003F 000000010000619F |.....|
same as above

```

Figure 146. Example dump from sample C routine (Part 3 of 4)

```

Storage around GPR13(00000001082FF180)
-0020 00000001082FF160 00000001 082FF61C 00000000 254F8268 |.....6.....|b.
-0010 00000001082FF170 00000001 082FF8C8 00000001 082FF630 |.....8H.....6.
+0000 00000001082FF180 00000001 04000000 00000001 082FEFF0 |.....0
+0010 00000001082FF190 00000001 00000000 00000001 08416050 |.....&
+0020 00000001082FF1A0 00000001 082FE5E0 00000000 25195798 |.....V.....q
+0030 00000001082FF1B0 00000000 25103430 00000000 00000000 |.....

Storage around GPR14(00000001000098D0)
-0020 00000001000098B0 00000000 00000000 00000000 00000000 |.....|
-0010 00000001000098C0 - +00003F 000000010000990F |.....|
Storage around GPR15(00000000255AB048)
-0020 00000000255AB028 00008000 00007E78 00000000 00000000 |.....=.
-0010 00000000255AB038 00000000 00000000 255AB000 00000128 |.....!.
+0000 00000000255AB048 00C300C5 00C500F1 00000038 00000184 |.C.E.1.....d
+0010 00000000255AB058 EB134880 0024B904 0014EB4F 46800024 |.....|.
+0020 00000000255AB068 A74BFE80 41940800 B90400F6 A7FBFF00 |x....m....6x..0
+0030 00000000255AB078 E3806100 001707F8 02CE00FF 00000024 |T./....8.....

```

Figure 146. Example dump from sample C routine (Part 4 of 4)

C/C++ contents of the Language Environment trace tables

Language Environment provides four C/C++ trace table entry types that contain character data:

- Trace entry 5 occurs when a C library function is called.
- Trace entry 6 occurs when a C library function returns.

The format for trace table entry 5 is:

```

NameOfCallingFunction
-->(xxxx) NameOfCalledFunction<(input_parameters)>

```

or, for called functions calloc, free, malloc, and realloc:

```

NameOfCallingFunction
—>(xxx) NameOfCalledFunction<(input_parameters)>

```

In addition, when the call is due to one of these C++ operators:

```

-new,
-new[],
-delete,
-delete[]

```

then the C++ operator will appear and the format becomes:

```

NameOfCallingFunction
—>(xxx) NameOfCalledFunction<(input_parameters)>

NameOfC++Operator

```

The input_parameters and NameOfC++Operator only appear for the appropriate functions. The angle brackets (<>) indicate that this information does not always appear.

The format for trace table entry 6 is:

```

<--(xxxx)
R1=xxxxxxxxxxxxxxxx R2=xxxxxxxxxxxxxxxx R3=xxxxxxxxxxxxxxxx
ERRN0=xxxxxxx ERRN02=xxxxxxx

```

In the entry types, (xxx) and (xxxx) are numbers associated with the called library function and are used to associate a specific entry record with its corresponding return record.

For entry types 5 and 6, the number will be the same as the number of the function as seen in the C run-time library definition side-deck, SCEELIB dataset member CELQS003, on the IMPORT statement for that function.

Figure 147 shows an XPLINK trace which has examples of the trace entries 5 and 6.

Language Environment Trace Table:

Most recent trace entry is at displacement: 000680

Displacement	Trace Entry in Hexadecimal	Trace Entry in EBCDIC
+000000	Time 21.58.20.255215 Date 2004.04.20 Thread ID... 8000000000000000	
+000010	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000018	86899385 82A4867A 7A96A585 99869396 A64D8995 A35D4040 40404040 40404040	filebuf::overflow(int)
+000038	60606E4D F0F1F5F6 5D406D6D 85999995 964D5D40 40404040 40404040 40404040	-->(0156) __errno()
+000058	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000078	40404040 40404040	
+000080	Time 21.58.20.255216 Date 2004.04.20 Thread ID... 8000000000000000	
+000090	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000098	4C60604D F0F1F5F6 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F8F7 F1C1C6F8 F040D9F2	<--(0156) R1=000000010871AF80 R2
+0000B8	7EF0F0F0 F0F0F0F0 F0F2F5F5 F2C2F8C2 F840D9F3 7EF0F0F0 F0F0F0F0 F1F0F0F0	=000000002552B8B8 R3=00000001000
+0000D8	F0F7F5F5 F840C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	07558 ERRNO=00000000 ERRNO2=0000
+0000F8	F0F0F0F0 00000000	0000....
+000100	Time 21.58.20.255216 Date 2004.04.20 Thread ID... 8000000000000000	
+000110	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000118	86899385 82A4867A 7A96A585 99869396 A64D8995 A35D4040 40404040 40404040	filebuf::overflow(int)
+000138	60606E4D F0F1F5F6 5D406D6D 85999995 964D5D40 40404040 40404040 40404040	-->(0156) __errno()
+000158	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000178	40404040 40404040	
+000180	Time 21.58.20.255217 Date 2004.04.20 Thread ID... 8000000000000000	
+000190	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000198	4C60604D F0F1F5F6 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F8F7 F1C1C6F8 F040D9F2	<--(0156) R1=000000010871AF80 R2
+0001B8	7EF0F0F0 F0F0F0F0 F0F2F5F5 F2C2F8C2 F840D9F3 7EF0F0F0 F0F0F0F0 F1F0F0F0	=000000002552B8B8 R3=00000001000
+0001D8	F0F7F5F5 F840C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	07558 ERRNO=00000000 ERRNO2=0000
+0001F8	F0F0F0F0 00000000	0000....
+000200	Time 21.58.20.255218 Date 2004.04.20 Thread ID... 8000000000000000	
+000210	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000218	86899385 82A4867A 7A96A585 99869396 A64D8995 A35D4040 40404040 40404040	filebuf::overflow(int)
+000238	60606E4D F0F0F7C5 5D4086A6 9989A385 4D5D4040 40404040 40404040 40404040	-->(007E) fwrite()
+000258	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000278	40404040 40404040	
+000280	Time 21.58.20.255242 Date 2004.04.20 Thread ID... 8000000000000000	
+000290	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000298	4C60604D F0F0F7C5 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F0F0 F0F8C2F3 F040D9F2	<--(007E) R1=0000000100008B30 R2
+0002B8	7EF0F0F0 F0F0F0F0 F0F2F5F5 F2C2F8C2 F840D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0	=000000002552B8B8 R3=00000000000
+0002D8	F0F0F0F0 C540C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	0000E ERRNO=00000000 ERRNO2=0000
+0002F8	F0F0F0F0 00000000	0000....
+000300	Time 21.58.20.255243 Date 2004.04.20 Thread ID... 8000000000000000	
+000310	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000318	86899385 82A4867A 7A96A585 99869396 A64D8995 A35D4040 40404040 40404040	filebuf::overflow(int)
+000338	60606E4D F0F0F6F8 5D408686 93A4A288 4D5D4040 40404040 40404040 40404040	-->(0068) fflush()
+000358	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000378	40404040 40404040	
+000380	Time 21.58.20.255244 Date 2004.04.20 Thread ID... 8000000000000000	
+000390	Member ID.... 03 Flags..... 000000 Entry Type..... 00000006	
+000398	4C60604D F0F0F6F8 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F0F0 F0F8F4F8 F040D9F2	<--(0068) R1=0000000100008480 R2
+0003B8	7EF0F0F0 F0F0F0F0 F0F2F5F5 F2C2F8C2 F840D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0	=000000002552B8B8 R3=00000000000
+0003D8	F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0	00000 ERRNO=00000000 ERRNO2=0000
+0003F8	F0F0F0F0 00000000	0000....
+000400	Time 21.58.20.255245 Date 2004.04.20 Thread ID... 8000000000000000	
+000410	Member ID.... 03 Flags..... 000000 Entry Type..... 00000005	
+000418	86899385 82A4867A 7A96A585 99869396 A64D8995 A35D4040 40404040 40404040	filebuf::overflow(int)
+000438	60606E4D F0F1F5F6 5D406D6D 85999995 964D5D40 40404040 40404040 40404040	-->(0156) __errno()
+000458	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000478	40404040 40404040	

Figure 147. Trace table with XPLINK trace table entries 5 and 6. (Part 1 of 2)

```

+000480 Time 21.58.20.255245 Date 2004.04.20 Thread ID... 8000000000000000
+000490 Member ID.... 03 Flags..... 000000 Entry Type..... 00000006
+000498 4C60604D F0F1F5F6 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F8F7 F1C1C6F8 F040D9F2 |<--(0156) R1=000000010871AF80 R2
+0004B8 7EF0F0F0 F0F0F0F0 F0F2F5F5 F2C2F8C2 F840D9F3 7EF0F0F0 F0F0F0F0 F1F0F0F0 | =000000002552B8B8 R3=00000001000
+0004D8 F0F7F5F5 F840C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0 | 07558 ERRNO=00000000 ERRNO2=0000
+0004F8 F0F0F0F0 00000000 | 0000....

+000500 Time 21.58.20.255246 Date 2004.04.20 Thread ID... 8000000000000000
+000510 Member ID.... 03 Flags..... 000000 Entry Type..... 00000005
+000518 86899385 82A4867A 7A96A585 99869396 A64D8995 A35D4040 40404040 40404040 |filebuf::overflow(int)
+000538 60606E4D F0F1F5F6 5D40D6D0 85999995 964D5D40 40404040 40404040 40404040 |-->(0156) __errno()
+000558 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000578 40404040 40404040

+000580 Time 21.58.20.255247 Date 2004.04.20 Thread ID... 8000000000000000
+000590 Member ID.... 03 Flags..... 000000 Entry Type..... 00000006
+000598 4C60604D F0F1F5F6 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F8F7 F1C1C6F8 F040D9F2 |<--(0156) R1=000000010871AF80 R2
+0005B8 7EF0F0F0 F0F0F0F0 F0F2F5F5 F2C2F8C2 F840D9F3 7EF0F0F0 F0F0F0F0 F1F0F0F0 | =000000002552B8B8 R3=00000001000
+0005D8 F0F7F5F5 F840C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0 | 07558 ERRNO=00000000 ERRNO2=0000
+0005F8 F0F0F0F0 00000000 | 0000....

+000600 Time 21.58.20.255252 Date 2004.04.20 Thread ID... 8000000000000000
+000610 Member ID.... 03 Flags..... 000000 Entry Type..... 00000005
+000618 E2A38183 924C8995 A36E7A7A 97A4A288 4D8995A3 5D404040 40404040 40404040 |Stack<int>::push(int)
+000638 60606E4D F0F0F5F6 5D409481 93939683 4DF1F65D 40404040 40404040 40404040 |-->(0056) malloc(16)
+000658 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
+000678 9585A640 40404040 |new

>>> +000680 Time 21.58.20.255253 Date 2004.04.20 Thread ID... 8000000000000000
+000690 Member ID.... 03 Flags..... 000000 Entry Type..... 00000006
+000698 4C60604D F0F0F5F6 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F8F3 F0C5C1F5 F040D9F2 |<--(0056) R1=000000010830EA50 R2
+0006B8 7EF0F0F0 F0F0F0F0 F0F2F5F5 F2C2F8C2 F840D9F3 7EF0F0F0 F0F0F0F0 F1F0F8F3 | =000000002552B8B8 R3=00000001083
+0006D8 F0C5C1F5 F040C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0 | 0EA50 ERRNO=00000000 ERRNO2=0000
+0006F8 F0F0F0F0 00000000 | 0000....

```

Figure 147. Trace table with XPLINK trace table entries 5 and 6. (Part 2 of 2)

For more information about the Language Environment trace table format, see “Understanding the trace table entry (TTE)” on page 361.

Debugging examples of C/C++ routines

This section contains examples that demonstrate the debugging process for C/C++ routines. Important areas of the output are highlighted. Data unnecessary to the debugging examples has been replaced by ellipses.

Divide-by-zero error

Figure 148 on page 393 illustrates a C program that contains a divide-by-zero error. The code was compiled with RENT so static and external variables need to be calculated from the WSA field. The code was compiled with LP64 and XREF, LIST and OFFSET to generate a listing, which is used to calculate addresses of functions and data. The code was processed by the binder with MAP to generate a binder map, which is used to calculate the addresses of static and external variables. The program was created with the option TERMTHDACT(UADUMP) which produced both a Language environment dump and a system dump.

```

/* C Routine with a Divide-by-Zero Error */
#pragma options(noinline)
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
int statint = 73;
int fa;
int funcb(int *pp);
int main(void) {
    int aa, bb=1;
    aa = bb;
    aa = funcb(&aa);
    return(aa);
}
int funcb(int *pp) {
    int result;
    fa = *pp;
    result = fa/(statint-73);
    printf("Result = %d\n",result);
    return result;
}

```

Figure 148. C routine with a divide-by-zero error

To debug this routine, use the following steps:

1. Locate the Current Condition message in the Condition Information for Active Routines section of the dump. In this example, the message is CEE3209S. The system detected a fixed-point divide exception. This message indicates the error was caused by an attempt to divide by zero. For additional information about CEE3209S, see *z/OS Language Environment Run-Time Messages*.

The traceback section of the dump indicates that the exception occurred at offset X'2E' within function funcb. This information is used along with the compiler-generated Pseudo Assembly Listing to determine where the problem occurred.

If the GONUMBER compiler option is specified, statement number information is in the dump. Figure 149 on page 394 shows the generated traceback from the dump.

Information for enclave main

Information for thread 8000000000000000

Traceback:

DSA	Entry	E	Offset	Load Mod	Program Unit	Service	Status
00000001	CEEHDS	+00000000	CELQLIB	CEEHDS	HLE7709	Call	
00000002	CELQHRD	+00000256	CELQLIB	CELQHRD	HLE7709	Call	
00000003	funcb	+0000002E	CDIVZERO			Exception	
00000004	main	+00000016	CDIVZERO			Call	
00000005	CELQINIT	+00001146	CELQLIB	CELQINIT	HLE7709	Call	

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Attributes
00000001	00000001082FC520	0000000025178AA0	0000000025178AA0	00000000	20040312	XPLINK EBCDIC	Floating Point
00000002	00000001082FEE40	00000000251166C8	00000000251166C8	00000256	20040312	XPLINK EBCDIC	Floating Point
00000003	00000001082FF080	0000000025100108	0000000000000000	*****	20040408	XPLINK EBCDIC	IEEE
00000004	00000001082FF180	00000000251000C0	0000000000000000	*****	20040408	XPLINK EBCDIC	IEEE
00000005	00000001082FF280	0000000025102010	0000000025102010	00001146	20040312	XPLINK EBCDIC	Floating Point

Condition Information for Active Routines

Condition Information for (DSA address 00000001082FF080)

CIB Address: 00000001082FD850

Current Condition:

CEE0198S The termination of a thread was signaled due to an unhandled condition.

Original Condition:

CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

Location:

Program Unit: Entry: funcb Statement: Offset: +0000002E

Machine State:

ILC..... 0002 Interruption Code..... 0009

PSW..... 0785240180000000 000000025100138

GPR0..... 0000000000000000 GPR1..... 0000000100000001 GPR2..... 0000000108401F20 GPR3..... 0000000108400050

GPR4..... 00000001082FF080 GPR5..... 0000000108300060 GPR6..... 0000000000000000 GPR7..... 0000000108300084

GPR8..... 0000000000000FE0 GPR9..... 0000000025100590 GPR10..... 00000000251001E8 GPR11..... 0000000108401F10

GPR12..... 00000001000039D0 GPR13..... 0000000000006F48 GPR14..... 000000007F7864E8 GPR15..... 000000000000001F

FPC..... 00000000

FPR0..... 26100000 00000000 FPR1..... 3FE6A09E 667F3BCD

FPR2..... 18000000 00000000 FPR3..... 34100000 00000000

FPR4..... 3FF6A09E 667F3BCD FPR5..... 00000000 00000000

FPR6..... 3FD45F30 6DC9C883 FPR7..... 00000000 00000000

FPR8..... 00000000 00000000 FPR9..... 00000000 00000000

FPR10..... 00000000 00000000 FPR11..... 00000000 00000000

FPR12..... 00000000 00000000 FPR13..... 00000000 00000000

FPR14..... 00000000 00000000 FPR15..... 00000000 00000000

Storage dump near condition, beginning at location(0000000025100126)

+0000	0000000025100126	60000014	50007000	A76BFFB7	8E000020	-...&...x,.....
+0010	0000000025100136	1D06B914	0081EB56	50100004	C0100000	a.&.....

Parameters, Registers, and Variables for Active Routines:

funcb (DSA address 00000001082FF080):

DOWNSTACK DSA

Saved Registers:

GPR0..... ***** GPR1..... ***** GPR2..... ***** GPR3..... *****

GPR4..... 00000001082FF080 GPR5..... BBBB BBBB BBBB BBBB GPR6..... 00000000251166C8 GPR7..... 0000000108300084

GPR8..... 0000000000000FE0 GPR9..... 0000000025100590 GPR10..... 00000000251001E8 GPR11..... 0000000108401F10

GPR12..... 4040404040404040 GPR13..... 4040404040404040 GPR14..... 4040404040404040 GPR15..... 4040404040404040

GPREG STORAGE:

Storage around GPR0 is invalid.

Storage around GPR1 is invalid.

Storage around GPR2 is invalid.

Storage around GPR3 is invalid.

Storage around GPR4 (00000001082FF080)

+0800 00000001082FF880 00000001 082FF180 00000001 082FEC60 |.....1.....-|

+0810 00000001082FF890 00000000 251000C0 00000000 251000DA |.....|

+0820 00000001082FF8A0 00000000 00006FE0 00000000 25100590 |.....?.....|

+0830 00000001082FF8B0 00000000 251001E8 00000001 08401F10 |.....Y.....|

+0840 00000001082FF8C0 00000001 000039D0 00000000 00006F48 |.....?.....|

+0850 00000001082FF8D0 00000000 7F7864E8 00000000 0000001F |....".Y.....|

:

Figure 149. Sections of the dump from example C/C++ routine

2. Locate the instruction with the divide-by-zero error in the Pseudo Assembly Listing in Figure 150 on page 395.

The offset (within funcb) of the exception from the traceback (X'2E') reveals the divide instruction: DR r0,r6 at that location. Instructions X'1C' through X'30' refer to the result = fa/(statint-73); line of the C/C++ routine.

OFFSET	OBJECT CODE	LINE#	FILE#	PSEUDO	ASSEMBLY	LISTING
Timestamp and Version Information						
000000	F2F0 F0F4			=C'2004'		Compiled Year
000004	F0F5 F1F9			=C'0519'		Compiled Date MMDD
000008	F1F7 F5F0 F5F2			=C'175052'		Compiled Time HHMMSS
00000E	F0F1 F0F6 F0F0			=C'010600'		Compiler Version
Timestamp and Version End						
000001				/* C Routine with a Divide-by-Zero Error from LE Debugging Guide */		
000002				#pragma options(noinline)		
000003				#include <stdio.h>		
000004				#include <stdlib.h>		
000005				#include <errno.h>		
000006				int statint = 73;		
000007				int fa;		
000008				int funcb(int *pp);		
000009				int main(void) {		
000018				@1L0	DS	00
000018	00C300C5					=F'12779717'
00001C	00C500F1					=F'12910833'
000020	000000C8					=F'200'
000024	00000100					=F'256'
000000				main	DS	00
000000	EB67 4710	0024		STMG	r6,r7,1808(r4)	
000006	A74B FF00			AGHI	r4,H'-256'	
00000A				End of Prolog		
000010				int aa, bb=1;		
000011				aa = bb;		
000011	A709 0001			LGHI	r0,H'1'	
000011	5000 48C8			ST	r0,aa(r4,2248)	
000012				aa = funcb(a);		
000012	4110 48C8			LA	r1,aa(r4,2248)	
000016	A775 0019			BRAS	r7,funcb	
00001A	0701			NOPR	1	
00001C	5030 48C8			ST	r3,aa(r4,2248)	
000013				return(aa);		
000013	B914 0033			LGFR	r3,r3	
000014				}		
000014				@1L2	DS	0H
000024				Start of Epilog		
000024	E370 4818	0004		LG	r7,2072(r4)	
00002A	4140 4100			LA	r4,256(r4)	
00002E	47F0 7002			B	2(r7)	
*** General purpose registers used: 1111111100000000						
*** Floating point registers used: 1111111100000000						
*** Size of register spill area: 128(max) 0(used)						
*** Size of dynamic storage: 0						
*** Size of executable code: 50						
000015				int funcb(int *pp) {		
000060				@2L0	DS	00
000060	00C300C5					=F'12779717'
000064	00C500F1					=F'12910833'
000068	000000A8					=F'168'
00006C	00000100					=F'256'
000000				funcb	DS	00
000000	EB68 4710	0024		STMG	r6,r8,1808(r4)	
000006	A74B FF00			AGHI	r4,H'-256'	
00000A				End of Prolog		

Figure 150. Pseudo assembly listing (Part 1 of 2)

```

00000A E360 5008 0004 000018 | LG r6,=A(statint)(,r5,8)
                                000016 | * int result;
                                000017 | * fa = *pp;
000010 E370 5000 0004 000017 | LG r7,=A(fa)(,r5,0)
000016 E300 1000 0014 000017 | LGF r0,(*)int(,r1,0)
                                000018 | * result = fa/(statint-73);
00001C E360 6000 0014 000018 | LGF r6,statint(,r6,0)
000022 5000 7000 000017 | ST r0,fa(,r7,0)
000026 A76B FFB7 000018 | AGHI r6,H'-73'
00002A 8E00 0020 000018 | SRDA r0,32
00002E 1D06 000018 | DR r0,r6
                                000019 | * printf("Result = %d\n",result);
000030 B914 0081 000019 | LGFR r8,r1
000034 EB56 5010 0004 000019 | LMG r5,r6,=A(printf)(r5,16)
00003A C010 0000 0013 000019 | LARL r1,F'19'
000040 B904 0028 000019 | LGR r2,r8
000044 0D76 000019 | BASR r7,r6
000046 0700 000019 | NOPR 0
                                000020 | * return result;
000048 B904 0038 000020 | LGR r3,r8
                                000021 | * }
00004C 000021 | @2L3 DS 0H

00004C Start of Epilog
00004C EB78 4818 0004 000021 | LMG r7,r8,2072(r4)
000052 4140 4100 000021 | LA r4,256(,r4)
000056 47F0 7002 000021 | B 2(,r7)

*** General purpose registers used: 1111111100000000
*** Floating point registers used: 1111111100000000
*** Size of register spill area: 128(max) 0(used)
*** Size of dynamic storage: 0
*** Size of executable code: 90

0000CA 0000
0000CC 0000 0000

Constant Area
000000 D985A2A4 93A3407E 406C8415 00 |Result = %d.. |

PPA1: Entry Point Constants
000000 02 =AL1(2) Version
000001 CE =AL1(206) CEL signature
000002 0300 =H'768' GPR save mask
000004 00000050 =A(PPA2-PPA1)
000008 80800281 =F'-2139094399' Flags
00000C 0000 =H'0' Parm length/4
00000E 0503 =H'1283' Prol len/2; alloca reg; R4 change offset/2
000010 00000032 =F'50' Code length
000014 01000000 =F'16777216' Interface mapping flags
000018 0004 **** AL2(4),C'main'
000020 FFFFFFF38 =F'-200' Offset to Entry Point Marker

PPA1 End

PPA1: Entry Point Constants
000000 02 =AL1(2) Version
000001 CE =AL1(206) CEL signature
000002 0380 =H'896' GPR save mask
000004 00000028 =A(PPA2-PPA1)
000008 80800281 =F'-2139094399' Flags
00000C 0002 =H'2' Parm length/4
00000E 0503 =H'1283' Prol len/2; alloca reg; R4 change offset/2
000010 0000005A =F'90' Code length
000014 01000000 =F'16777216' Interface mapping flags
000018 0005 **** AL2(5),C'funcb'
000020 FFFFFFF58 =F'-168' Offset to Entry Point Marker

PPA1 End

PPA2: Compile Unit Block
000000 0300 2204 =F'50340356' Flags
000004 FFFF FED0 =A(CELQSTRT-PPA2)
000008 0000 0000 =F'0' No PPA4
00000C FFFF FED0 =A(TIMESTAMP-PPA2)
000010 0000 0000 =F'0' No primary
000014 8100 0000 =F'-2130706432' Flags

PPA2 End

```

Figure 150. Pseudo assembly listing (Part 2 of 2)

- Verify the value of the divisor `statint`. The procedure specified below is to be used for determining the value of static variables only. If the divisor is an automatic variable, there is a different procedure for finding the value of the variable.

Because this routine was compiled with the RENT option, find the WSA address in the Enclave Control Blocks section of the dump. In this example, this address is X'108300050'. Figure 151 shows the WSA address.

Enclave Control Blocks:				
.				
.				
.				
DLL Information:				
WSA Addr	Module Addr	Thread ID	Use Count	Name
0000000108300050			00000001	main

Figure 151. C/C++ CAA information in dump

- Routines compiled with the RENT option must also be processed by the binder. The binder produces the Writable Static Map. Find the offset of `statint` in the Writable Static Map in Figure 152. In this example, the offset is X'4'.

CLASS C_WSA64		LENGTH = 38	ATTRIBUTES = MRG, DEFER, RMODE= 64	
		OFFSET =	0 IN SEGMENT 002 ALIGN = QDWORD	

CLASS	OFFSET	NAME	TYPE	LENGTH	SECTION
	0	\$PRIV000011	PART	10	
	10	EXIST#S	PART	20	EXIST#C
	30	func_ptr	PART	8	func_ptr

Figure 152. Writable static map

- Add the WSA address of X'108300050' to the offset of `statint`. The result is X'108300080'. This is the address of the variable `statint`, which is in the writable static area.
- Use IPCS to display the writeable static area in the system dump. The value at location X'108300050' is X'49' (that is, `statint` is 73), and hence the fixed-point divide exception.

LIST 0000000108300050 LEN(X'00000100')	
LIST 01 08300050. ASID(X'0015') LENGTH(X'0100') AREA	
8300050. C36DE6E2 C1F6F440 40404040 40404040	C_WSA64
8300060. 00000001 08300084 00000001 08300080	d.....
8300070. 00000000 000000C0 00000000 2548D200	{.....K.
8300080. 00000049 00000001 00000000 00000000	
8300090 LENGTH(X'10')==>All bytes contain X'00'	
83000A0. 00000001 08300000 00000000 00000220	
83000B0. 00000001 083002D0 00000001 083004B8	}
83000C0. 00000001 083004F5 00000001 08300532	5.....
83000D0. 00000001 0830056F 00000001 083005AC	?.....
83000E0. 00000001 083005E9 00000001 08300626	Z.....
83000F0. 00000001 08300663 00000001 08300A70	
8300100. 00000001 08300AAD 00000000 00000000	
8300110 LENGTH(X'40')==>All bytes contain X'00'	

Figure 153. IPCS storage display of the writeable static area

Calling a nonexistent function

Figure 154 demonstrates the error of calling a nonexistent function. This routine was compiled with the compiler options LP64, LIST and RENT and was run with the option TERMTHDACT(UADUMP).

```
/* C/C++ Example of Calling a Nonexistent Subroutine    */
/*      from LE Debugging Guide                        */
#pragma options(noinline)
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
void funca(int* aa);
int (*func_ptr)(void)=0;
int main(void) {
    int aa;
    funca(&aa);
    printf("result of funca = %d\n",aa);
    return;
}
void funca(int* aa) {
    *aa = func_ptr();
    return;
}
```

Figure 154. C/C++ example of calling a nonexistent subroutine

To debug this routine, use the following steps:

1. Locate the Current Condition message in the Condition Information for Active Routines section of the dump, shown in Figure 155 on page 399. In this example, the message is CEE3206S The system detected a specification exception (System Completion Code=0C6). This message suggests that the error was caused by an attempt to branch to an unknown address. For additional information about CEE3206S, see *z/OS Language Environment Run-Time Messages*.

The traceback section of the dump indicates that the exception occurred at offset X'-25B553DE7' within function funca. The negative offset indicates that the offset cannot be used to locate the instruction that caused the error. Another indication of bad data is the value of X'00000001' in the instruction address of the PSW. This address indicates that an instruction in the routine branched outside the bounds of the routine.

Information for enclave main

Information for thread 8000000000000000

Traceback:

DSA	Entry	E	Offset	Load Mod	Program Unit	Service	Status
00000001	CEEHDS	+00000000	CELQLIB	CEEHDS	HLE7709	Call	
00000002	CELQHRD	+00000256	CELQLIB	CELQHRD	HLE7709	Call	
00000003	funca	-25100111	EXIST			Exception	
00000004	main	+00000012	EXIST			Call	
00000005	CELQINIT	+00001146	CELQLIB	CELQINIT	HLE7709	Call	

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Attributes
00000001	00000001082FC520	0000000025178AA0	0000000025178AA0	00000000	20040312	XPLINK EBCDIC	Floating Point
00000002	00000001082FEE40	00000000251166C8	00000000251166C8	00000256	20040312	XPLINK EBCDIC	Floating Point
00000003	00000001082FF080	0000000025100110	0000000000000000	*****	20040413	XPLINK EBCDIC	IEEE
00000004	00000001082FF180	00000000251000C0	0000000000000000	*****	20040413	XPLINK EBCDIC	IEEE
00000005	00000001082FF280	0000000025102010	0000000025102010	00001146	20040312	XPLINK EBCDIC	Floating Point

Condition Information for Active Routines

Condition Information for (DSA address 00000001082FF080)

CIB Address: 00000001082FD850

Current Condition:

CEE0198S The termination of a thread was signaled due to an unhandled condition.

Original Condition:

CEE3206S The system detected a specification exception (System Completion Code=0C6).

Location:

Program Unit: Entry: funca Statement: Offset: -25100111

Machine State:

ILC..... 0002 Interruption Code..... 0006

PSW..... 0785240180000000 0000000000000001

GPR0..... 0000000108300060 GPR1..... 00000001082FFA48 GPR2..... 0000000108401F20 GPR3..... 0000000108400050

GPR4..... 00000001082FF080 GPR5..... 000A00000000130E1 GPR6..... FFFFFFFF FFFFFFFF GPR7..... 0000000025100132

GPR8..... 00000001082FFA48 GPR9..... 0000000025100568 GPR10..... 00000000251001C0 GPR11..... 0000000108401F10

GPR12..... 00000001000039D0 GPR13..... 0000000000006F48 GPR14..... 000000007F7864E8 GPR15..... 000000000000001F

FPC..... 00000000

FPR0..... 26100000 00000000 FPR1..... 3FE6A09E 667F3BCD

FPR2..... 18000000 00000000 FPR3..... 34100000 00000000

FPR4..... 3FF6A09E 667F3BCD FPR5..... 00000000 00000000

FPR6..... 3FD45F30 6DC9C883 FPR7..... 00000000 00000000

FPR8..... 00000000 00000000 FPR9..... 00000000 00000000

FPR10..... 00000000 00000000 FPR11..... 00000000 00000000

FPR12..... 00000000 00000000 FPR13..... 00000000 00000000

FPR14..... 00000000 00000000 FPR15..... 00000000 00000000

Storage dump near condition, beginning at location(FFFFFFFFFFFFFFEF)

+0000 FFFFFFFF FFFFFFFF Inaccessible storage.

Parameters, Registers, and Variables for Active Routines:

funca (DSA address 00000001082FF080):

DOWNSTACK DSA

Saved Registers:

GPR0..... ***** GPR1..... ***** GPR2..... ***** GPR3..... *****

GPR4..... 00000001082FF080 GPR5..... BBBBBBBBBBBBBBBB GPR6..... 00000000251166C8 GPR7..... 0000000025100132

GPR8..... 00000001082FFA48 GPR9..... 0000000025100568 GPR10..... 00000000251001C0 GPR11..... 0000000108401F10

GPR12..... 4040404040404040 GPR13..... 4040404040404040 GPR14..... 4040404040404040 GPR15..... 4040404040404040

GPREG STORAGE:

Storage around GPR0 is invalid.

Storage around GPR1 is invalid.

Storage around GPR2 is invalid.

Storage around GPR3 is invalid.

Storage around GPR4 (00000001082FF080)

+0800 00000001082FF880 00000001 082FF180 00000001 082FEC60 1.....-

+0810 00000001082FF890 00000000 251000C0 00000000 251000D6 0

+0820 00000001082FF8A0 00000001 08300060 00000000 25100568~.....

+0830 00000001082FF8B0 00000000 251001C0 00000001 08401F10

+0840 00000001082FF8C0 00000001 000039D0 00000000 00006F48?..

+0850 00000001082FF8D0 00000000 7F7864E8 00000000 0000001F".Y.....

Storage around GPR5 (BBBBBBBBBBBBBBBB)

-0020 BBBBBBBBBBBBBBB9B Inaccessible storage.

-0010 BBBBBBBBBBBBBBBAB Inaccessible storage.

+0000 BBBBBBBBBBBBBBBB Inaccessible storage.

+0010 BBBBBBBBBBBBBBBBCB Inaccessible storage.

+0020 BBBBBBBBBBBBBBBDB Inaccessible storage.

+0030 BBBBBBBBBBBBBBBEB Inaccessible storage.

Figure 155. Sections of the dump from example C routine (Part 1 of 2)

```

Enclave Control Blocks:
EDB(00000001000039D0)
+0000 00000001000039D0 C3C5C5C5 C4C24040 00000000 00000000 |CEEEDB .....|
+0010 00000001000039E0 00000000 00000000 00000000 00000000 |.....|
+0020 00000001000039F0 - +0000FF 0000000100003ACF same as above
+0100 0000000100003AD0 90000100 00000000 00000001 00004F40 |.....|
+0110 0000000100003AE0 00000001 00004458 00000000 00000000 |.....|
+0120 0000000100003AF0 00000001 00002468 00000000 00000000 |.....|
+0130 0000000100003B00 00000001 00003C28 00000001 00002FA0 |.....|
+0140 0000000100003B10 00000001 00003B08 00000000 00000000 |.....|
+0150 0000000100003B20 00000001 00006160 02000000 00000000 |...../-.....|
+0160 0000000100003B30 00000000 00000001 00000000 00000000 |.....|
+0170 0000000100003B40 00000000 00006FE0 00000000 00000000 |.....?.....|
+0180 0000000100003B50 00000000 00000200 00000001 000031A8 |.....y.....|
+0190 0000000100003B60 00000001 00003B58 00000000 00000000 |.....|
+01A0 0000000100003B70 00000000 00000000 00000000 00000000 |.....|
+01B0 0000000100003B80 - +0001FF 0000000100003BCF same as above

```

```

DLL Information:
WSA Addr      Module Addr      Thread ID      Use Count      Name
0000000108300050      00000001      00000001      00000001      main

```

Run-Time Options Report:

LAST WHERE SET	OPTION
Installation default	ENVAR("")
Installation default	FILETAG(NOAUTOCVT,NOAUTOTAG)
Installation default	HEAPCHK(OFF,1,0,0,0)
Installation default	HEAPPOLLS64(OFF,8,4000,32,2000,128,700,256,350,1024,100,2048,50,3072,50,4096,50,8192,25,16384,10,32768,5,65536,5)
Installation default	HEAP64(1M,1M,KEEP,32768,32768,KEEP,4096,4096,FREE)
Installation default	INFOMSGFILTER(OFF,,,))
Installation default	IOHEAP64(1M,1M,FREE,12288,8192,FREE,4096,4096,FREE)
Installation default	LIBHEAP64(1M,1M,FREE,16384,8192,FREE,8192,4096,FREE)
Installation default	NATLANG(ENU)
Installation default	POSIX(OFF)
Installation default	PROFILE(OFF,"")
Installation default	RPTOPTS(OFF)
Installation default	RPTSTG(OFF)
Installation default	STACK64(1M,1M,128M)
Installation default	STORAGE(NONE,NONE,NONE,)
Invocation command	TERMTHDACT(UADUMP,,96)
Installation default	NOTEST(ALL,"*","PROMPT","INSPREF")
Installation default	THREADTACK64(OFF,1M,1M,128M)
Installation default	TRACE(OFF,4096,DUMP,LE=0)
Installation default	TRAP(ON,SPIE)

Process Control Blocks:

```

PCB(0000000100002468)
+0000 0000000100002468 C3C5C5D7 C3C24040 00000000 00000000 |CEEPCB .....|
+0010 0000000100002478 00000000 00000000 00000000 00000000 |.....|
+0020 0000000100002488 - +0000FF 0000000100002567 same as above
+0100 0000000100002568 03030208 00000000 00000000 00000000 |.....|
+0110 0000000100002578 00000001 00002810 00000000 00000000 |.....|
+0120 0000000100002588 00000000 00000000 00000000 00000000 |.....|
+0130 0000000100002598 00000000 00000000 00000001 00002210 |.....|
+0140 00000001000025A8 7F800000 00000000 00000000 00000000 |".....|
+0150 00000001000025B8 00000000 00000000 00000000 00000000 |.....|
+0160 00000001000025C8 - +0001BF 0000000100002627 same as above
MEML(0000000100002810)
+0000 0000000100002810 00000000 00000000 00000000 00000000 |.....|
+0010 0000000100002820 - +00005F 000000010000286F same as above
+0060 0000000100002870 00000001 00006CD0 00000000 00000000 |.....%.....|
+0070 0000000100002880 00000000 00000000 00000000 00000000 |.....|
+0080 0000000100002890 - +0001AF 00000001000029BF same as above

```

Figure 155. Sections of the dump from example C routine (Part 2 of 2)

- Find the branch instructions for funca in the listing in Figure 49 on page 163. Notice the BASR r7,r6 instruction at offset X'0020'. This branch is part of the instruction.

OFFSET	OBJECT CODE	LINE#	FILE#	PSEUDO	ASSEMBLY	LISTING
Timestamp and Version Information						
000000	F2F0 F0F4			=C'2004'	Compiled Year	
000004	F0F5 F1F9			=C'0519'	Compiled Date MMDD	
000008	F1F9 F0F1 F2F7			=C'190127'	Compiled Time HHMMSS	
00000E	F0F1 F0F6 F0F0			=C'010600'	Compiler Version	
Timestamp and Version End						
000001				* /* C/C++ Example of Calling a Nonexistent Subroutine */		
000002				* /* from LE Debugging Guide */		
000003				* #pragma options(noinline)		
000004				* #include <stdio.h>		
000005				* #include <stdlib.h>		
000006				* #include <errno.h>		
000007				* #include <signal.h>		
000008				* void funca(int* aa);		
000009				* int (*func_ptr)(void)=0;		
000010				* int main(void) {		
000018				@1L0 DS 0D		
000018	00C300C5			=F'12779717'	XPLink entrypoint marker	
00001C	00C500F1			=F'12910833'		
000020	00000080			=F'176'		
000024	00000100			=F'256'		
000000				main DS 0D		
000000	EB68 4710 0024			STMG r6,r8,1808(r4)		
000006	A74B FF00			AGHI r4,H'-256'		
00000A				End of Prolog		
000011				* int aa;		
000012				* funca(a);		
00000A	B904 0085			LGR r8,r5		
00000E	4110 48C8			LA r1,aa(,r4,2248)		
000012	A775 001F			BRAS r7,funca		
000016	0701			NOPR 1		
000018	E320 48C8 0014			* printf("result of funca = %d\n",aa);		
00001E	EB56 8010 0004			LGF r2,aa(,r4,2248)		
000024	C010 0000 0032			LMG r5,r6,=A(printf)(r8,16)		
00002A	0D76			LARL r1,F'50'		
00002C	0700			BASR r7,r6		
				NOPR 0		
				* return;		
				* }		
00002E				@1L2 DS 0H		
00002E				Start of Epilog		
00002E	EB78 4818 0004			LMG r7,r8,2072(r4)		
000034	4140 4100			LA r4,256(,r4)		
000038	B909 0033			SGR r3,r3		
00003C	47F0 7002			B 2(r7)		
*** General purpose registers used: 1111111100000000						
*** Floating point registers used: 1111111100000000						
*** Size of register spill area: 128(max) 0(used)						
*** Size of dynamic storage: 0						
*** Size of executable code: 64						
000016				* void funca(int* aa) {		
000068				@2L0 DS 0D		
000068	00C300C5			=F'12779717'	XPLink entrypoint marker	
00006C	00C500F1			=F'12910833'		
000070	00000080			=F'128'		
000074	00000100			=F'256'		
000000				funca DS 0D		
000000	EB68 4710 0024			STMG r6,r8,1808(r4)		
000006	A74B FF00			AGHI r4,H'-256'		
00000A				End of Prolog		

Figure 156. Pseudo assembly listing (Part 1 of 2)

```

000017      *      aa = func_ptr();
00000A E360 5000 0004 000017      LG      r6,A(func_ptr)(,r5,0)
000010 E360 6000 0004 000017      LG      r6,func_ptr(,r6,0)
000016 B904 0081      000016      LGR      r8,r1
00001A EB56 6000 0004 000017      LMG      r5,r6,%ADA_%EPA(r6,0)
000020 0D76      000017      BASR      r7,r6
000022 0700      000017      NOPR      0
000024 5030 8000      000017      ST       r3,(*)int(,r8,0)
000018      *      return;
000019      *      }
000028      @2L3      DS      0H

000028      Start of Epilog
000028 EB78 4818 0004 000019      LMG      r7,r8,2072(r4)
00002E 4140 4100      000019      LA       r4,256(,r4)
000032 47F0 7002      000019      B        2(,r7)

*** General purpose registers used: 1111111100000000
*** Floating point registers used: 1111111100000000
*** Size of register spill area: 128(max) 0(used)
*** Size of dynamic storage: 0
*** Size of executable code: 54

0000AE 0000

Constant Area
000000 9985A2A4 93A34096 864086A4 95838140 result of funca
000010 7E406C84 1500      = %d..

PPA1: Entry Point Constants
000000 02      =AL1(2)      Version
000001 CE      =AL1(206)    CEL signature
000002 0380      =H'896'      GPR save mask
000004 00000040      =A(PPA2-PPA1)
000008 80800081      =F'-2139094911'    Flags
00000C 0000      =H'0'      Parm length/4
00000E 0503      =H'1283'    Prol len/2; alloca reg; R4 change offset/2
000010 00000040      =F'64'      Code length
000014 0004 ****      AL2(4),C'main'
00001C FFFFFFF50      =F'-176'    Offset to Entry Point Marker

PPA1 End

PPA1: Entry Point Constants
000000 02      =AL1(2)      Version
000001 CE      =AL1(206)    CEL signature
000002 0380      =H'896'      GPR save mask
000004 00000020      =A(PPA2-PPA1)
000008 80800081      =F'-2139094911'    Flags
00000C 0002      =H'2'      Parm length/4
00000E 0503      =H'1283'    Prol len/2; alloca reg; R4 change offset/2
000010 00000036      =F'54'      Code length
000014 0005 ****      AL2(5),C'funca'
00001C FFFFFFF80      =F'-128'    Offset to Entry Point Marker

PPA1 End

PPA2: Compile Unit Block
000000 0300 2204      =F'50340356'    Flags
000004 FFFF FEF8      =A(CELQSTRT-PPA2)
000008 0000 0000      =F'0'      No PPA4
00000C FFFF FEF8      =A(TIMESTAMP-PPA2)
000010 0000 0000      =F'0'      No primary
000014 8100 0000      =F'-2130706432'    Flags

PPA2 End

```

Figure 156. Pseudo assembly listing (Part 2 of 2)

- Find the offset of func_ptr in the Writable Static Map, shown in Figure 157 on page 403.

```

-----
CLASS  C_WSA64          LENGTH =      38  ATTRIBUTES = MRG, DEFER , RMODE= 64
                        OFFSET =      0  IN SEGMENT 002      ALIGN = QDWORD
-----

```

CLASS OFFSET	NAME	TYPE	LENGTH	SECTION
0	\$PRIV000011	PART	10	
10	CDIVZERO#S	PART	20	CDIVZERO#C
30	statint	PART	4	statint
34	fa	PART	4	fa

Figure 157. Writable static map

4. Add the offset of func_ptr (X'30') to the address of WSA (X'108300050'). The result (X'108300080') is the address of the function pointer func_ptr in the writable static storage area. This value is 0, indicating the variable is uninitialized.

Figure 158 shows the sections of the dump.

```

LIST 000000108300050 LEN(X'00000100')
LIST 01 08300050. ASID(X'0015') LENGTH(X'0100') AREA
_8300050. C36DE6E2 C1F6F440 40404040 40404040 C_WSA64
_8300060. 00000001 08300084 00000001 08300080 .....d.....
_8300070. 00000000 000000C0 00000000 25480200 .....{.....K.
_8300080. 00000049 00000001 00000000 00000000 .....
_8300090 LENGTH(X'10')=>All bytes contain X'00'
_83000A0. 00000001 08300000 00000000 00000220 .....
_83000B0. 00000001 083002D0 00000001 08300488 .....}.....
_83000C0. 00000001 083004F5 00000001 08300532 .....5.....
_83000D0. 00000001 0830056F 00000001 083005AC .....?.....
_83000E0. 00000001 083005E9 00000001 08300626 .....Z.....
_83000F0. 00000001 08300663 00000001 08300A70 .....
_8300100. 00000001 08300AAD 00000000 00000000 .....
_8300110 LENGTH(X'40')=>All bytes contain X'00'

```

Figure 158. IPCS storage display of the writeable static area

Handling dumps written to the z/OS UNIX file system

When a z/OS UNIX C/C++ application program is running in an address space created as a result of a call to `spawnp()`, `vfork()`, or one of the `exec` family of functions, the `SYSDUMP` DD allocation information is not inherited. Even though the `SYSDUMP` allocation is not inherited, a `SYSDUMP` allocation must exist in the parent in order to obtain a HFS storage dump. If the program terminates abnormally while running in this new address space, the kernel causes an unformatted storage dump to be written to an HFS file in the user's working directory. The file is placed in the current working directory or into `/tmp` if the current working directory is not defined. The file name has the following format:

`/directory/coredump.pid`

where `directory` is the current working directory or `tmp`, and `pid` is the hexadecimal process ID (PID) for the process that terminated. For details on how to generate the system dump, see "Steps for Generating a system dump in a z/OS UNIX shell" on page 307.

To debug the dump, use the Interactive Problem Control System (IPCS). If the dump was written to an HFS file, you must allocate a data set that is large enough and has the correct attributes for receiving a copy of the HFS file. For example, from the ISPF DATA SET UTILITY panel you can specify a volume serial and data set name to allocate. Doing so brings up the DATA SET INFORMATION panel for

specifying characteristics of the data set to be allocated. The following filled-in panel shows the characteristics defined for the URCOMP.JRUSL.COREDUMP dump data set:

```

----- DATA SET INFORMATION -----
Command ==>

Data Set Name . . . : URCOMP.JRUSL.COREDUMP

General Data
Management class . . : STANDARD
Storage class . . . : OS390
Volume serial . . . : DPXDU1
Device type . . . : 3380
Data class . . . . . :
Organization . . . : PS
Record format . . . : FB
Record length . . . : 4160
Block size . . . : 4160
1st extent cylinders: 30
Secondary cylinders : 10
Data set name type :

Current Allocation
Allocated cylinders : 30
Allocated extents . : 1

Current Utilization
Used cylinders. . . : 0
Used extents . . . : 0

Creation date . . . : 2001/08/30
Expiration date . . : ***None***

F1=Help    F2=Split  F3=End    F4=Return  F5=Rfind   F6=Rchange
F7=Up      F8=Down   F9=Swap  F10=Left  F11=Right  F12=Cancel

```

Figure 159. IPCS panel for entering data set information

Fill in the information for your data set as shown, and estimate the number of cylinders required for the dump file you are going to copy.

Use the TSO/E OGET or OCOPY command with the BINARY keyword to copy the file into the data set. For example, to copy the HFS memory dump file coredump.00060007 into the data set URCOMP.JRUSL.COREDUMP just allocated, a user with the user ID URCOMP enters the following command:

```
OGET '/u/urcomp/coredump.00060007' 'urcomp.jrusl.coredump' BINARY
```

For more information on using the copy commands, see *z/OS UNIX System Services User's Guide*.

After you have copied the memory dump file to the data set, you can use IPCS to analyze the dump. Refer to “Formatting and analyzing system dumps” on page 308 for information about formatting Language Environment control blocks.

Multithreading consideration

Certain control blocks are locked while a dump is in progress. For example, a `csnap()` of the file control block would prevent another thread from using or dumping the same information. An attempt to do so causes the second thread to wait until the first one completes before it can continue.

Understanding C/C++ heap information in storage reports

Storage reports that contain specific C/C++ heap information can be generated in two ways:

- By setting the Language Environment RPTSTG(ON) run-time option for Language Environment created heaps

- By issuing a stand-alone call to the C function `__uheapreport()` for user-created heaps.

Details on how to request and interpret the reports are provided in the following sections.

Language Environment storage report with HeapPools statistics

To request a Language Environment storage report set `RPTSTG(ON)`. If the C/C++ application specified the `HEAPPOOLS64(ON)` run-time option, then the storage report displays HeapPools statistics. For a sample storage report showing HeapPools statistics for a multithreaded C/C++ application, see *z/OS Language Environment Programming Reference*.

The following describes the C/C++ specific heap pool information.

HeapPools storage statistics

The `HEAPPOOLS64` run-time option controls usage of the heap pools storage algorithm at the enclave level. The heap pools algorithm allows for the definition of one to twelve heap pools, each consisting of a number of storage cells of a specified length.

Usage Note: The use of an alternative Vendor Heap Manager (VHM) overrides the use of the `HEAPPOOLS64` run-time option.

HeapPools statistics:

- Pool *p* size: *ssss*
 - *p* — the number of the pool
 - *ssss* — the cell size specified for the pool.
- Successful Get Heap requests: *xxxx-yyy n*
 - *xxxx* — the low side of the 8 byte range
 - *yyy* — the high side of the 8 byte range
 - *n* — the number of requests in the 8 byte range.
- Requests greater than the largest cell size — the number of storage requests that are not satisfied by heap pools.

Note: Values displayed in the HeapPools Statistics report are not serialized when collected, therefore the values are not necessarily exact.

HeapPools summary: The HeapPools Summary displays a report of the HeapPool Statistics and provides suggested percentages for current cell sizes as well as suggested cell sizes.

- Cell Size — the size of the cell specified in the `HEAPPOOLS64` run-time option
- Cells Per Extent — the cell pool count specified by the `HEAPPOOLS64` run-time option
- Extents Allocated — the number of times that each pool allocated an extent in order to optimize storage usage. The extents allocated should be either one or two. If the number of extents allocated is too high, increase the cell count for the pool.
- Maximum Cells Used — the maximum number of cells used for each pool.
- Cells In Use — the number of cells that were never freed.

Note: A large number in this field could indicate a storage leak.

- Suggested Cell Sizes — sizes that are calculated to optimally use storage (assuming that the application will malloc/free with the same frequency).

Note: The suggested cell sizes are given with no cell counts because the usage of each new cell pool size is not known. If there are less than 12 cell sizes calculated, then the last pool size is set at 65536.

For more information about stack and heap storage for AMODE64 applications, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

C function `__uheapreport()` storage report

To generate a user-created heap storage report use the C function, `__uheapreport()`. Use the information in the report to assist with tuning your application's use of the user-created heap. For a description of the information contained in the report, see "HeapPools storage statistics" on page 405.

For more information on the `__uheapreport()` function, see *z/OS XL C/C++ Run-Time Library Reference*. For tuning tips, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

A sample storage report generated by `__uheapreport()` is shown in Figure 160.

```
Storage Report for Enclave Tue Apr 20 20:29:08 2004
Language Environment V01 R06.00

HeapPools Statistics:
Pool 1 size: 32
  Successful Get Heap requests: - 15
Pool 2 size: 128
  Successful Get Heap requests: - 15
Pool 3 size: 512
  Successful Get Heap requests: - 15
Pool 4 size: 2048
  Successful Get Heap requests: - 15
Pool 5 size: 8192
  Successful Get Heap requests: - 15
Pool 6 size: 16384
  Successful Get Heap requests: - 15
Requests greater than the largest cell size: 0
HeapPools Summary:
Cell Size Cells Per Extents Maximum Cells In
      Size Extent Allocated Cells Used Use
-----
      32      15         1         1     15
     128      15         1         1     15
     512      15         1         1     15
    2048      15         1         1     15
    8192      15         1         1     15
   16384      15         1         1     15
-----
Suggested Cell Sizes:
,32,,128,,512,,2048,,8192,,16384,,0)
End of Storage Report
```

Figure 160. Storage report generated by `__uheapreport()`

Appendix A. Diagnosing problems with Language Environment

This appendix provides information for diagnosing problems in the Language Environment product. It helps you determine if a correction for a product failure similar to yours has been previously documented. If the problem has not been previously reported, it tells you how to open a problem management record (PMR) to report the problem to IBM, and if the problem is with an IBM product, what documentation you need for an Authorized Program Analysis Report (APAR).

Diagnosis checklist

Step through each of the items in the diagnosis checklist below to see if they apply to your problem. The checklist is designed to either solve your problem or help you gather the diagnostic information required for determining the source of the error. It can also help you confirm that the suspected failure is not a user error; that is, it was not caused by incorrect usage of the Language Environment product or by an error in the logic of the routine.

1. If your failing application contains programs that were changed since they last ran successfully, review the output of the compile or assembly (listings) for any unresolved errors.
2. If there have not been any changes in your applications, check the output (job or console logs, CICS transient (CESE) queues) for any messages from the failing run.
3. Check the message prefix to 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 respective origins.

EDC	The prefix for C/C++ messages. The following series of messages are from the C/C++ run-time component of Language Environment: 5000 (except for 5500, which are from the DSECT utility), 6000, and 7000.
IGZ	The prefix for messages from the COBOL run-time component of Language Environment.
FOR	The prefix for messages from the Fortran run-time component of Language Environment.
IBM	The prefix for messages from the PL/I run-time component of Language Environment.
CEE	The prefix for messages from the common run-time component of Language Environment.

4. For any messages received, check for recommendations in the "Programmer Response" sections of the messages in this manual.
5. Verify that abends are caused by product failures and not by program errors. See the appropriate chapters in this manual for a list of Language Environment-related abend codes.
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 most current maintenance level.

7. The preventive service planning (PSP) bucket, an online database available to IBM customers through IBM service channels, gives information about product installation problems and other problems. Check to see whether it contains information related to your problem.
8. Narrow the source of the error.
 - If a Language Environment dump is available, locate the traceback in the Language Environment dump for the source of the problem.
 - For AMODE 64 applications, IBM recommends that you use the IPCS Verbexit IEDATA with the CEEDUMP option to format the traceback. Check the traceback for the source of the problem. For information on how to generate and use a Language Environment or system dump to isolate the cause of the error, see Chapter 3, “Using Language Environment debugging facilities,” on page 35 or Chapter 11, “Using Language Environment AMODE 64 debugging facilities,” on page 287.
 - Alternatively, in a non-XPLINK environment, you can follow the save area chain to find out the name of the failing module and whether IBM owns it. For information on finding the routine name, see “Locating the name of the failing routine for a non-XPLINK application.”
9. After you identify the failure, consider writing a small test case that re-creates the problem. The test case could help you determine whether the error is in a user routine or in the Language Environment product. Do not make the test case larger than 75 lines of code. The test case is not required, but it could expedite the process of finding the problem.
If the error is not a Language Environment failure, refer to the diagnosis procedures for the product that failed.
10. Record the conditions and options in effect at the time the problem occurred. Compile your program with the appropriate options to obtain an assembler listing and data map. If possible, obtain the binder or linkage editor output listing. Note any changes from the previous successful compilation or run. For an explanation of compiler options, refer to the compiler-specific programming guide.
11. If you are experiencing a no-response problem, try to force a dump. For example, CANCEL the program with the dump option.
12. Record the sequence of events that led to the error condition and any related programs or files. It is also helpful to record the service level of the compiler associated with the failing program.

Locating the name of the failing routine for a non-XPLINK application

If a system dump is taken, follow the save area chain to find out the name of the failing routine and whether IBM owns it. Following are the procedures for locating the name of the failing routine, which is the primary entry point name.

1. Find the entry point associated with the current save area. The entry point address (EPA), located in the previous save area at displacement X'10', decimal 16, points to it.
2. Determine the entry point type, of which there are four:

Entry point type is...	If...
Language Environment conforming	The entry point plus 4 is X'00C3C5C5'.
Language Environment conforming OPLINK	The entry point plus 4 is X'01C3C5C5'. OPLINK linkage conventions are used.
C/C++	The entry point plus 5 is X'CE'.

Entry point type is...	If...
Nonconforming	The entry point is none of the above. Nonconforming entry points are for routines that follow the linking convention in which the name is at the beginning of the routine. X'47F0Fxxx' is the instruction to branch around the routine name.

For routines with Language Environment-conforming and C/C++ entry points, Language Environment provides program prolog areas (PPAs). PPA1 contains the entry point name and the address of the PPA2; PPA2 contains pointers to the timestamp, where release level keyword information is found, and to the PPA1 associated with the primary entry point of the routine.

If the entry point type of the failing routine is Language Environment-conforming, go to step 3.

If the entry point type is C/C++, go to step 5.

If the entry point type is nonconforming, go to step 6 on page 410.

3. If the entry point type is Language Environment-conforming, find the entry point name for the Language Environment or COBOL program.
 - a. Use an offset of X'C' from the entry point to locate the address of the PPA1.
 - b. In the PPA1, locate the offset to the length of the name. If OPLINK, then multiply the offset by 2 to locate the actual offset to the length of the name.
 - c. Add this offset to the PPA1 address to find the halfword containing the length of the name, followed by the entry point name.
The entry point name appears in EBCDIC, with the translated version in the right-hand margin of the system dump.
4. Find the Language Environment or COBOL program name.
 - a. Find the address of the PPA2 at X'04' from the start of the PPA1.
 - b. Find the address of the compilation unit's primary entry point at X'10' in the PPA2.
 - c. Find the entry point name associated with the primary entry point as described above. The primary entry point name is the routine name.

See *z/OS Language Environment Vendor Interfaces* for illustrations and details of:

 - the non-XPLINK Language Environment-conforming PPA1 and PPA2.
 - the XPLINK Language Environment-conforming PPA1, and the XPLINK PPA1 optional area fields.
 - the non-XPLINK Language Environment PPA2.
 - the Language Environment PPA2: Compile Unit Block for XPLINK.
 - the PPA2 timestamp and version information.
5. If the entry point type is C/C++, find the C/C++ routine name.
 - a. Use the entry point plus 4 to locate the offset to the entry point name in the PPA1.
 - b. Use this offset to find the length-of-name byte followed by the routine name.
The routine name appears in EBCDIC, with the translated version in the right-hand margin.

Figure 161 on page 410 illustrates the C PPA1.

C Routine Layout Entry and PPA1

00	B xxx(0,15) Branch around prolog data			
04	X'14' Offset to the name	X'CE' (Language Environment signature)	Language Environment Flags	Member flags
08	A(PPA2)			
0C	A (Block Debugging Information (BDI)) or zero			
10	Stack frame size			
	:			
	:			
	:			
yy				
	Length of name		Untruncated entry/label name	

Figure 161. C PPA1

6. If the entry point type is nonconforming, find the PL/I routine name.
 - a. Find the one byte length immediately preceding the entry point. This is the length of the routine name.
 - b. Go back the number of bytes specified in the name length. This is the beginning of the routine name.
7. If the entry point type is nonconforming, find the name of the routine other than PL/I.
 - a. Use the entry point plus 4 as the location of the entry point name.
 - b. Use the next byte as the length of the name. The name directly follows the length of name byte. The entry point name appears in EBCDIC with the translated version in the right-hand margin.

Figure 162 illustrates a nonconforming entry point type.

Nonconforming entry points that can appear do not necessarily follow this linking convention. The location of data in these save areas can be unpredictable.

020000	=	47F0F00C	06D3C9E2	E3C9E300	90ECD00C	E0B	.00..LISTIT.....
020010	=	18CF41B0	C29850BD	000850DB	000418DB		Bq&...&.....
020020	=	4510C052	E3E8D7D3	C9D54040	01020034		TYPLIN
020030	=	C200001E	C5D5E3C5	D940D5E4	D4C2C5D9		B...ENTER NUMBER
020040	=	40D6C640	D9C5C3D6	D9C4E240	D6D940C1		OF RECORDS OR A
020050	=	D3D30ACA	00020058	4510C06C	E6C1C9E3		LL.....%WAIT
020060	=	D9C44040	010202F0	E4000000	0ACA0002		RD ...0U.....

Figure 162. Nonconforming entry point type with sample dump

Searching the IBM Software Support Database

Failures in the Language Environment product can be described through the use of keywords. A keyword is a descriptive word or abbreviation assigned to describe one aspect of a product failure. A set of keywords, called a keyword string, describes the failure in detail. You can use a keyword or keyword string as a search argument against an IBM software support database, such as the Service Information Search (SIS). The database contains keyword and text information describing all current problems reported through APARs and associated PTFs. IBM Support Center personnel have access to the software support database and are responsible for storing and retrieving the information. Using keywords or a keyword string, they will search the database to retrieve records that describe similar known problems.

If you have IBMLink™ or some other connection to the IBM databases, you can do your own search for previously recorded product failures before calling the IBM Support Center.

If your keyword or keyword string matches an entry in the software support database, the search may yield a more complete description of the problem and possibly identify a correction or circumvention. Such a search may yield several matches to previously reported problems. Review each error description carefully to determine if the problem description in the database matches the failure.

If a match is not found, go to “Preparing documentation for an Authorized Program Analysis Report (APAR).”

Preparing documentation for an Authorized Program Analysis Report (APAR)

Prepare documentation for an APAR only after you have done the following:

- Eliminated user errors as a possible cause of the problem.
- Followed the diagnostic procedures.
- You or your local IBM Support Center has been unsuccessful with the keyword search.

Having met these criteria, follow the instructions below.

1. Report the problem to IBM.

If you have not already done so, report the problem to IBM by opening a problem management record (PMR).

If you have IBMLink or some other connection to IBM databases, you can open a PMR yourself. Or, the IBM Software Support Center can open the PMR after consulting with you on the phone. The PMR is used to document your problem and to record the work that the Support Center does on the problem. Be prepared to supply the following information:

- Customer number
- PMR number
- Operating system
- Operating system release level
- Your current Language Environment maintenance level (PTF list and list of APAR fixes applied)
- Keyword strings you used to search the IBM software support database
- Processor number (model and serial)

- A description of how reproducible the error is. Can it be reproduced each time? Can it be reproduced only sometimes? Have you been unable to reproduce it? Supply source files, test cases, macros, subroutines, and input files required to re-create the problem. Test cases are not required, but can often speed the response time for your problem.

If the IBM Support Center concludes that the problem described in the PMR is a problem with the Language Environment product, they will work with you to open an APAR, so the problem can be fixed.

2. Provide APAR documentation. When you submit an APAR, you will need to supply information that describes the failure. Table 41 describes how to produce documentation required for submission with the APAR.

Table 41. Problem resolution documentation requirements

Item	Materials Required	How to Obtain Materials
1	Machine-readable source program, including macros, subroutines, input files, and any other data that might help to reproduce the problem.	IBM-supplied system utility program
2	Compiler listings: • Source listing • Object listing • Storage map • Traceback • Cross-reference listing • JCL listing and linkage editor listing • Assembler-language expansion	Use appropriate compiler options
3	Dumps • Language Environment dump • System dump	See instructions in Chapter 3, "Using Language Environment debugging facilities," on page 35 (as directed by IBM support personnel).
4	Partition/region size/virtual storage size	
5	List of applied PTFs	System programmer
6	Operating instructions or console log	Application programmer
7	JCL statements used to invoke and run the routine, including all run-time options, in machine-readable form	Application programmer
8	System output associated with the MSGFILE run-time option.	Specify MSGFILE(SYSOUT)
9	Contents of the applicable catalog	
10	A hardcopy log of the events leading up to the failure.	Print out each display.

3. Submit the APAR documentation.

When submitting material for an APAR to IBM, carefully pack and clearly identify any media containing source programs, job stream data, interactive environment information, data sets, or libraries.

All magnetic media submitted must have the following information attached and visible:

- The APAR number assigned by IBM.
- A list of data sets on the tape (such as source program, JCL, data).

- A description of how the tape was made, including:
 - The exact JCL listing or the list of commands used to produce the machine-readable source. Include the block size, LRECL, and format of each file. If the file was unloaded from a partitioned data set, include the block size, LRECL, and number of directory blocks in the original data set.
 - Labeling information used for the volume and its data sets.
 - The recording mode and density.
 - The name of the utility program that created each data set.
 - The record format and block size used for each data set.

Any printed materials must show the corresponding APAR number.

The IBM service personnel will inform you of the mailing address of the service center nearest you.

If an electronic link with IBM Service is available, use this link to send diagnostic information to IBM Service.

After the APAR is opened and the fix is produced, the description of the problem and the fix will be in the software support database in SIS, accessible through ServiceLink.

Appendix B. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

www.ibm.com/servers/eserver/zseries/zos/bkserv/

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This publication documents information NOT intended to be used as a Programming Interface of Language Environment in z/OS.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AD/Cycle	IBMLink	SAA
AFP	IMS	SXM
C/370	IMS/ESA	System/370
CICS	Language Environment	TCS
COBOL/370	MVS	VisualAge
DB2	MVS/ESA	z/OS
DFSMS/MVS	Open Class	z/OS.e
DFSORT	OS/390	zSeries
IBM	Resource Link	z/VM

IEEE is a trademark of the Institute of Electrical and Electronics Engineers, Inc. in the United States and other countries.

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

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

Bibliography

This section lists the books in the Language Environment library and other publications that may be helpful when using Language Environment.

Language Products Publications

z/OS Language Environment

- *z/OS Language Environment Concepts Guide*, SA22-7567
- *z/OS Language Environment Programming Guide*, SA22-7561
- *z/OS Language Environment Programming Reference*, SA22-7562
- *z/OS Language Environment Customization*, SA22-7564
- *z/OS Language Environment Debugging Guide*, GA22-7560
- *z/OS Language Environment Writing Interlanguage Communication Applications*, SA22-7563
- *z/OS Language Environment Run-Time Messages*, SA22-7566
- *z/OS Language Environment Vendor Interfaces*, SA22-7568
- *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*, SA22-7569

z/OS XL C/C++

- *z/OS XL C/C++ Language Reference*, SC09-4815
- *z/OS XL C/C++ Compiler and Run-Time Migration Guide for the Application Programmer*, GC09-4913
- *z/OS XL C/C++ Programming Guide*, SC09-4765
- *z/OS XL C/C++ User's Guide*, SC09-4767
- *z/OS XL C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS XL C/C++ Messages*, GC09-4819
- *Standard C++ Library Reference*, SC09-4949
- *C/C++ Legacy Class Libraries Reference*, SC09-7652
- *IBM Open Class Library Transition Guide*, SC09-4948

Enterprise COBOL for z/OS

- *Enterprise COBOL for z/OS Licensed Program Specifications*, GC27-1411
- *Enterprise COBOL for z/OS Customization*, GC27-1410
- *Enterprise COBOL for z/OS Language Reference*, SC27-1408
- *Enterprise COBOL for z/OS Programming Guide*, SC27-1412
- *Enterprise COBOL for z/OS Migration Guide*, GC27-1409

COBOL for OS/390 & VM

- *COBOL for OS/390 & VM Licensed Program Specifications*, GC26-9044
- *COBOL for OS/390 & VM Customization under OS/390*, GC26-9045
- *COBOL for OS/390 & VM Language Reference*, SC26-9046
- *COBOL for OS/390 & VM Programming Guide*, SC26-9049
- *COBOL for OS/390 & VM Compiler and Run-Time Migration Guide*, GC26-4764

COBOL for MVS & VM (Release 2)

- *Licensed Program Specifications*, GC26-4761
- *Programming Guide*, SC26-4767
- *Language Reference*, SC26-4769
- *Compiler and Run-Time Migration Guide*, GC26-4764
- *Installation and Customization under MVS*, SC26-4766
- *Reference Summary*, SX26-3788
- *Diagnosis Guide*, SC26-3138

VS COBOL II

VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045

Debug Tool

- Debug Tool documentation is available at:
<http://www.ibm.com/software/ad/debugtool/library/>

VS FORTRAN Version 2

- *Language Environment Fortran Run-Time Migration Guide, SC26-8499*
- *Language and Library Reference, SC26-4221*
- *Programming Guide for CMS and MVS, SC26-4222*

Enterprise PL/I for z/OS

- *Enterprise PL/I for z/OS Licensed Program Specifications, GC27-1456*
- *Enterprise PL/I for z/OS Programming Guide, SC27-1457*
- *Enterprise PL/I for z/OS Language Reference, SC27-1460*
- *Enterprise PL/I for z/OS Migration Guide, GC27-1458*
- *Enterprise PL/I for z/OS Messages and Codes, SC27-1461*

PL/I for MVS & VM

- *PL/I for MVS & VM Licensed Program Specifications, GC26-3116*
- *PL/I for MVS & VM Programming Guide, SC26-3113*
- *PL/I for MVS & VM Language Reference, SC26-3114*
- *PL/I for MVS & VM Reference Summary, SX26-3821*
- *PL/I for MVS & VM Compiler and Run-Time Migration Guide, SC26-3118*
- *PL/I for MVS & VM Installation and Customization under MVS, SC26-3119*
- *PL/I for MVS & VM Compile-Time Messages and Codes, SC26-3229*
- *PL/I for MVS & VM Diagnosis Guide, SC26-3149*

High Level Assembler for MVS & VM & VSE

- *Programmer's Guide, MVS & VM Edition, SC26-4941*

Related Publications

CICS

- *CICS Transaction Server for z/OS Installation Guide, GC34-6224*
- *CICS Operations and Utilities Guide, SC34-6229*
- *CICS Problem Determination Guide, SC34-6239*
- *CICS Resource Definition Guide, SC34-6228*
- *CICS Data Areas, LY33-6103*
- *CICS Application Programming Guide, SC34-6231*
- *CICS Application Programming Reference, SC34-6232*
- *CICS System Definition Guide, SC34-6226*

DB2

Database 2 Application Programming and SQL Guide, SC26-4377

DFSMS/MVS

z/OS MVS Program Management: User's Guide and Reference, SA22-7643
z/OS DFSMS DFM Guide and Reference, SC26-7395

IPCS

- *z/OS MVS IPCS User's Guide, SA22-7596*
- *z/OS MVS IPCS Commands, SA22-7594*
- *z/OS MVS IPCS Customization, SA22-7595*

DFSORT

z/OS DFSORT Application Programming Guide, SC26-7523

IMS/ESA

IMS Version 8: Application Programming: Design Guide, SC27-1287

IMS Version 8: Application Programming: Database Manager, SC27-1286

IMS Version 8: Application Programming: Transaction Manager, SC27-1289

IMS Version 8: Application Programming: EXEC DLI Commands for CICS and

IMS Version 8:, SC27-1288

msys for Setup

- *z/OS Managed System Infrastructure for Setup User's Guide, SC33-7985*

z/OS

- *z/OS Introduction and Release Guide, GA22-7502*
- *z/OS Program Directory, GI10-0670*
- *z/OS and z/OS.e Planning for Installation, GA22-7504*
- *z/OS Information Roadmap, SA22-7500*
- *z/OS Hot Topics Newsletter, GA22-7501*
- *z/OS Licensed Program Specifications, GA22-7503*
-
- *z/OS ISPF Dialog Tag Language Guide and Reference, SC34-4824*
- *z/OS ISPF Planning and Customizing, GC34-4814*
- *z/OS ISPF Dialog Developer's Guide and Reference, SC34-4821*
-
- *z/OS UNIX System Services User's Guide, SA22-7801*
- *z/OS UNIX System Services Command Reference, SA22-7802*
- *z/OS UNIX System Services Programming: Assembler Callable Services Reference, SA22-7803*
- *z/OS UNIX System Services Planning, GA22-7800*
-
- *z/OS TSO/E Customization, SA22-7783*
- *z/OS TSO/E Programming Services, SA22-7789*
- *z/OS TSO/E System Programming Command Reference, SA22-7793*

z/OS.e

- *z/OS.e Overview, GA22-7869*
- *z/OS.e Licensed Program Specifications, GA22-7868*

Softcopy Publications

z/OS Collection, SK3T-4269

Index

Special characters

- `__abend` 128, 370
- `__alloc` 128, 370
- `__amrc` 128, 370
- `__cabend()` function 278, 286, 307
- `__code` 128, 370
- `__error` 128, 370
- `__feedback` 128, 370
- `__last_op` 128, 370
- `__le_cib_get()` function 278
- `__le_message_get_and_and_write()` function 273, 279
- `__le_message_get()` function 279
- `__le_msg_write()` function 273, 279
- `__msg` 129, 371
- `__reset_exception_handler()` function 278
- `__set_exception_handler()` function 278, 279
- `_BPXK_MDUMP` 77, 307

Numerics

64-bit applications 273, 283, 287, 369

A

abend codes

- `>= 4000` 32, 284
- `< 4000` 32, 284
- 4093 283
- passing to operating system 25
- system, example of 34, 285
- user-specified, example of 34, 285
- user, example of 34, 285, 286
- using 34, 285

abends

- internal, table of output 268
- Language Environment 34, 265, 285
- requested by assembler user exit 25
- system 34, 286
- under CICS 265
- user 34, 286

ABPERC run-time option

- function 9
- generating a system dump and 75
- modifying condition handling behavior and 23

ABTERMENC run-time option 9, 26

using 26

accessibility 415

AGGREGATE compiler option 4, 8

AMODE 64 applications

- classifying errors 283
- debugging C/C++ 369
- preparing for debugging 273
- using debugging 287

anywhere heap

statistics 20

APAR (Authorized Program Analysis Report) 411

documentation 412

application program interfaces (API) 278, 279

application programs

debugging

- handling a storage dump written to a BFS
- file 183, 403

- handling a storage dump written to an HFS
- file 183, 403

argument

in dump 60

arguments, registers, and variables for active

routines 58, 305

assembler language

user exit 25, 26

for CICS 268

generating a system dump with 75, 307

modifying condition handling behavior and 25

using 25, 26

atexit

information in dump 157

Authorized Program Analysis Report (APAR) 411

automatic variables

locating in dump 250

B

base locator

for working storage 204

in dump 204

below heap

statistics 20

binder

module map 377

BLOCKS option of CEE3DMP callable service 37

C

C library function

trace table entries for 390

C return codes to CICS 265

C-CAA (C-specific common anchor area)

See C/C++, C-specific common anchor area (C-CAA)

C/C++

`__amrc`

example of structure 128, 370

information in dump 159

`__msg` 129, 371

atexit

information in dump 157

C-specific common anchor area (C-CAA) 157

`cdump()` function 145, 381

compiler listings 135, 377

IPA link step listing 136, 378

compiler options 3

debugging examples 172, 179, 392

dump

information in 157

parameter in 141

C/C++ (continued)

- signal information in 156
- structure variables, locating in 143
- system, structures in 143
- file
 - control block information 158
 - status and attributes in dump 158
- functions
 - calling dump, example 145, 381
 - cdump() 43, 145, 289, 381
 - csnap() 43, 145, 146, 289, 381, 382
 - ctrace() 43, 145, 146, 289, 381, 382
 - fetch() 127, 369
 - fopen() 129, 371
 - perror() 127, 133, 369, 375
 - printf() 128, 370
 - to produce dump output 43, 289
- memory file control block 158
- stdio.h 128, 370
- timestamp 145

CAA (common anchor area) 62, 306, 309, 310, 311, 325, 348

call chain 60

CALL statement

- CDUMP/CPDUMP 223
- DUMP/PDUMP 222
- SDUMP 224

callable services 22

- CEE3ABD—terminate enclave with an abend
 - See CEE3ABD—terminate enclave with an abend
- CEE3DMP—generate dump
 - See CEE3DMP—generate dump
- CEE3GRO—returns location offset
 - See CEE3GRO—returns location offset
- CEE3SRP—set resume point
 - See CEE3SRP—set resume point
- CEEDCOD—decompose a condition token
 - See CEEDCOD—decompose a condition token
- CEEHDLR—register user condition handler
 - See CEEHDLR—register user condition handler
- CEEMGET—get a message
 - See CEEMGET—get a message
- CEEMOUT—dispatch a message
 - See CEEMOUT—dispatch a message
- CEEMRCE—move resume cursor to a designated label
 - See CEEMRCE—move resume cursor to designated label
- CEEMRCR—move resume cursor relative to handle cursor
 - See CEEMRCR—move resume cursor relative to handle cursor
- CEEMSG—get, format, and dispatch a message
 - See CEEMSG—get, format, and dispatch a message
- CEESGL—signal a condition
 - See CEESGL—signal a condition

case 1 condition token 28, 280

case 2 condition token 28, 280

cdump() function 145, 284, 381

CEE prefix 31, 33, 283, 285

CEE3ABD—terminate enclave with an abend 22, 34, 75

- generating a dump and 75
- handling user abends and 22, 34
- modifying condition handling behavior and 22

CEE3DMP—generate dump 35, 60

- See also Language Environment dump
- generating a Language Environment dump with 35
- options 36
- relationship to PLIDUMP 245
- syntax 36

CEE3GRO—returns location offset 22

CEE3SRP—set resume point 22

CEEBXITA assembler user exit 25

CEECXITA assembler user exit 268

CEEDCOD—decompose a condition token 28

CEEDUMP—Language Environment Dump Service

- See Language Environment dump
- control blocks 94, 322
- locating 115

CEEHDLR—register user condition handler 24

CEEHDLR—register user exception handler 279

CEEMGET—get a message 28

CEEMOUT—dispatch a message 26

CEEMRCE—move resume cursor to designated label 22

CEEMRCR—move resume cursor relative to handle cursor 22

CEEMSG—get, format, and dispatch a message 28

CEESGL—signal a condition 28

CEESTART 127

CEL prefix 283

CELQSTRT 369

character

- data dump 223

CHECK run-time option

- function 9
- modifying condition handling behavior and 23

CHECKOUT compiler option 4

CICS

- abends 265
 - application, from an EXEC CICS command 269
- debugging for 263
- debugging information, table of locations 263
- destination control table (DCT) 263
- example traceback in CESE transient data queue 263
- examples of output 263
- nonzero reason code returned, table of output 268
- reason codes 265
- register and program status word contents 265
- return codes
 - Language Environment 265
- run-time messages 263
- transaction
 - dump 264
 - rollback 268

class test 196

classifying errors table 31, 283

CLLE (COBOL load list entry) 204

COBCOM control block 206

- COBOL
 - base locator for working storage 204
 - compiler options 6
 - debugging examples 207, 219
 - dump
 - external data in 204
 - file information in 204
 - linkage section in 204
 - local variables in 201
 - routine information in 201
 - run unit storage in 205
 - stack frames for active routines in 201
 - working storage in 204
 - errors 195
 - listings 199
 - memory file control block 157
 - program class storage 204
 - return codes to CICS 265
 - routine
 - calling Language Environment dump service 199
- COBVEC control block 206
- command
 - syntax diagrams xvii
- COMMAREA (Communication Area) 264
- compiler options
 - C 3
 - COBOL 6
 - FORTTRAN 7
 - LP64 273, 369
 - PL/I 8
- Compiler options map 136, 377
- condition
 - information
 - for active routines 57, 305
 - in dump 57, 305
 - POSIX 28
 - unhandled 29, 280
- condition handling
 - behavior, modifying 22
 - user-written condition handler 22, 24
- condition information block (CIB) 60, 69
- condition manager 29, 280
- CONDITION option of CEE3DMP callable service 38, 60
- condition token 27, 28, 279
 - case 1 28, 280
 - case 2 28, 280
 - example of 29, 280
- conditions, nested 29, 280
- control block
 - for active routines 58, 305
- Cross-Reference listing 135, 377
- csnap() function 146, 382
- ctrace() function 146, 284, 382

D

- data
 - map listing 199
 - values 60
- DCB (data control block) 204

- DCT (destination control table) 263
- DEBUG run-time option 9
- debugging
 - C, examples 172, 179, 392
 - COBOL, examples 207, 219
 - for CICS 263
 - FORTTRAN, examples 229, 235
 - PL/I, examples 253, 258
 - tool 35, 287
- DEPTHCONDLMT run-time option
 - function 9
 - modifying condition handling behavior and 23
 - wait/loop error and 32
- diagnosis checklist 407
- disability 415
- DISPLAY statement 27, 195
- divide-by-zero error 172, 213, 231, 258, 261, 392
- DSA (dynamic save area) 60
 - See stack, frame
- dummy DSA 60
- dump
 - an area of storage 223
 - date in 56, 303
 - dynamically allocated storage in 59
 - storage
 - written to an HFS file 183, 403
 - symbolic 224
- DUMP suboption of TERMTHDACT run-time option 39, 287
- DUMP/PDUMP routine 222
 - format specifications 222
 - output 222
 - usage considerations 222

E

- ECB (enclave control block) 59, 306
- EDB (enclave data block) 59, 306
- EDC prefix 31, 33, 283, 285
- EIB (exec interface block) 264
- enclave
 - identifier in dump 56, 303
 - member list 59, 306
 - storage 59
 - termination
 - behavior, establishing 26
- entry information 55, 303
- ENTRY option of CEE3DMP callable service 38
- entry point
 - name of active routines in dump 57, 304
- ERRCOUNT run-time option
 - function 9
 - modifying condition handling behavior and 23
 - wait/loop error and 32
- errno 157
- error
 - determining source of 407
 - message while Language Environment was handling
 - another error 29, 280
 - unanticipated 31, 283
- ESD compiler option 8

- examples
 - application abends from 269
 - C routines 172, 179, 392
 - calling a nonexistent subroutine 176, 210, 256
 - COBOL routines 207, 219
 - divide-by-zero error 172, 213, 231, 258, 261, 392
 - FORTRAN routines 229, 235
 - output under CICS 263
 - PL/I routines 253, 258
 - SUBSCRIPTRANGE error 207, 253
- exception handling
 - behavior, modifying 277
 - user-written exception handler 277
- EXEC CICS DUMP statements 264
- external data
 - for COBOL programs in dump 204
- External symbol cross reference listing 135, 377

F

- fetch
 - fetch information in dump 157
- fetch() function 127, 369
- fetchable module 127, 369
- file
 - for COBOL, in dump 204
 - status key 195
- file control block (FCB) 158
- FILES option of CEE3DMP callable service 37
- FLAG compiler option 4
- floating point registers
 - in dump 56
- fopen() function 129, 371
- FOR prefix 31, 33
- FORTRAN
 - compiler options 7
 - debugging examples 229, 235
 - dump services 221
 - errors, determining the source of 219
 - listings 221

G

- general purpose registers 56
- Global symbols map 136, 377
- GMAREA 204
- GONUMBER compiler option 4

H

- HANDLE ABEND EXEC CICS command 263
- header files, C
 - ctest.h 145, 381
 - errno.h 172, 392
 - stdio.h 128, 370
 - stdlib.h 172, 392
- heap storage
 - created by CEECRHP callable service 21
 - in LEDATA Output 95, 326
 - reports 99, 336
 - storage in dump 59

- heap storage *(continued)*
 - user 18
- HEAP64 run-time option 275
- HEAPCHK run-time option
 - function 9, 100, 273, 338
- HEAPOOLS
 - storage statistics 185, 405
 - user-created, __uheapreport() 406
 - user-created, _uheapreport 186
 - trace report 80, 101, 310, 338
- HEAPOOLS64 run-time option 275

I

- I/O
 - conventions 127, 369
- IBM prefix 31, 33
- IGZ prefix 31, 33
- INFMSGFILTER run-time option
 - function 10, 273
- INITIALIZE statement 196
- inline
 - report 135, 377
- Inline report for IPA 136, 377
- instance specific information (ISI) 280
- instruction length counter (ILC) in dump 58, 305
- Inter-procedural Analysis (IPA) 273
- interactive problem control system (IPCS)
 - analyzing a storage dump 183, 403
 - cbf command 357
 - Verbexit 308, 309, 311, 341
- INTERRUPT compiler option
 - function 8
- INTERRUPT run-time option 9, 10
- interruption code in dump 58, 305
- IOHEAP64 run-time option 275
- ITBLK in dump 206

K

- keyboard 415

L

- LAA (library anchor area) 358
- language constructs 195
- Language Environment
 - return codes to CICS 265
 - symbolic feedback code 27, 279
- Language Environment dump 287, 289
 - C information in 156
 - CEEDUMP 35
 - COBOL information in 203
 - default options 38
 - example traceback in 59, 306
 - FORTRAN information in 227
 - multiple enclaves and 74
 - options
 - BLOCKS 37
 - CONDITION 38, 60
 - ENCLAVE 36

Language Environment dump (continued)

options (continued)

- ENTRY 38
- FILES 37
- FNAME 38
- NOBLOCKS 37
- NOCONDITION 38
- NOENTRY 38
- NOFILES 37
- NOSTORAGE 37
- NOTRACEBACK 37
- NOVARIABLES 37
- PAGESIZE(n) 38
- STACKFRAME 37
- STORAGE 37
- THREAD 37
- TRACEBACK 37, 60
- VARIABLES 37, 60

output

- for C routines 150
- for COBOL program 199
- for FORTRAN routines 227
- for PL/I routines 245
- information for multiple enclaves 43, 290

PL/I information in 247

section descriptions 55, 303

TERMTHDACT suboptions 40, 288

title 56, 303

traceback with condition information

- C routine 150, 386
- COBOL program 201
- FORTTRAN routine 219, 227, 228
- Language Environment routine 55, 303
- PL/I routine 245

using C functions 43, 289

using CDUMP/CPDUMP subroutine 221

using CEE3DMP callable service 35, 55, 303

using DUMP/PDUMP subroutine 221

using PLIDUMP subroutine 43, 245

using SDUMP subroutine 221

using TERMTHDACT run-time option 39, 287

LCA (library communication area) 358

LEDATA

- IPCS Verbexit 78, 308
- C/C++ Output 103, 341
- COBOL Output 109
- Format 79, 309
- Parameters 79, 309
- Understanding Output 81, 311

LIBHEAP64 run-time option 275

linkage editor

module map 135

linkage section

for COBOL programs in dump 204

LIST compiler option 4, 6, 8

listings generated by compiler

- C 135, 377
- COBOL 199
- FORTTRAN 221
- PL/I 238

LMESSAGE compiler option 8

local

variables 58

LookAt message retrieval tool xix

LP64

compiler option 273, 369

M

machine state information

in dump 58, 305

MAP compiler option 6, 8

MEMLIMIT storage parameter 283

memory file control block (MFCB) 157, 158

message

classifying errors and 32, 284

run-time, CICS 263

user-created 26

using in your routine 26

message retrieval tool, LookAt xix

module

fetchable 127, 369

module name prefixes, Language Environment 31, 283

MSG suboption

of TERMTHDACT 39, 287

MSGFILE run-time option

function 10

run-time messages and 32, 284

MSGQ run-time option 10

N

nested condition 29, 280

no response (wait/loop) 32, 284

NOBLOCKS option of CEE3DMP callable service 37

NOCONDITION option of CEE3DMP callable service 38

NOENTRY option of CEE3DMP callable service 38

NOFILES option of CEE3DMP callable service 37

NOSTORAGE option of CEE3DMP callable service 37

Notices 417

NOTRACEBACK option of CEE3DMP callable service 37

NOVARIABLES option of CEE3DMP callable service 37

O

Object file map 136, 377

OFFSET compiler option 4, 6, 8

optimizing

C 3, 60

COBOL 6

FORTTRAN 224

PL/I 8

options

C compiler 3

COBOL compiler 6

defaults for dump 40, 288

determining run-time in effect 10, 12, 274

FORTTRAN compiler 7

- options (*continued*)
 - Language Environment run-time 9, 273
 - PL/I compiler 8, 9
- out-of-storage condition
 - virtual storage 283
- OUTDD compiler option 6
- output
 - incorrect 32, 284
 - missing 32, 284

P

- page number
 - in dump 56, 303
- PAGESIZE(n) option of CEE3DMP callable service 38
- parameter
 - checking value of 25
- perror() function 133, 375
- PL/I
 - address of interrupt, finding in dump 249
 - CAA address, finding in dump 252
 - common anchor area (CAA) 252
 - compiler listings
 - object code listing 242
 - static storage map 241
 - variable storage map 242
 - compiler options
 - generating listings with 238
 - list of 8
 - CSECT 241
 - debugging examples 253, 258
 - dump
 - error type, finding in 249
 - parameter list, finding contents in 251
 - PL/I information, finding in 247, 251
 - PLIDUMP subroutine and 245
 - statement number, finding in 249
 - timestamp, finding in 251
 - variables, finding in 250
 - ERROR ON-unit 236, 249
 - errors 235, 238
 - floating-point register 236
 - object code listing 242
 - ON statement control block 242
 - static storage listing 241
 - SUBSCRIPTRANGE condition 236, 237, 253, 255, 256
 - PLIDUMP subroutine 245
 - PMR (problem management record) 411
 - pointer
 - variable 127, 369
 - PPA 409
 - Prelinker map 135
 - preventive service planning (PSP) bucket 408
 - printf() function 27, 128, 370
 - problem management record (PMR) 411
 - procedure division listings 199
 - process
 - control block 59, 306
 - member list 59, 306
 - process control block (PCB) 59, 306

- PROFILE run-time option
 - function 10, 273
- program
 - class storage 204
- program prolog area 409
- program status word (PSW) 58, 305
- pseudo-assembler listing 135, 377
- PSP (preventive service planning) bucket 408

Q

- QUIET suboption of TERMTHDACT run-time option 39, 287

R

- reason code
 - nonzero returned to CICS 268
 - under CICS 265
- registers 0–15
 - in dump 56
- release number
 - in dump 56, 303
- request parameter list (RPL) 158
- return code
 - bad or nonzero 32, 284
- RPTOPTS run-time option 10
- RPTSTG run-time option 12, 275
- run unit
 - COBOL 205
 - level control block 205
 - storage in dump 59, 205
- run-time
 - messages
 - under CICS 263
- run-time options 22, 278
 - determining those in effect 10, 12, 274
 - sample options report 10
 - specifying 25

S

- scope
 - terminator 195
- SDUMP routine
 - description 224
 - format specifications 224
 - output 224
 - usage considerations 224
- service routines
 - CDUMP/CPDUMP 223
 - DUMP/PDUMP 222
 - SDUMP 224
- SET statement 196
- shortcut keys 415
- signal information in dump 156
- sorted cross-reference listing 199
- SOURCE compiler option 4, 6, 8
- Source file map 136, 377
- source listing 135, 377

- stack
 - frame 60
 - frame format 61, 62
- STACK64 run-time option 275
- STACKFRAME option of CEE3DMP callable service 37
- statement numbers
 - in dump 56, 303
- static
 - variables in dump 250
 - writable map 139, 140, 144
- status
 - of routines in dump 57, 304
- stderr 27, 284, 287, 288
- stdio.h 128, 370
- stdout 27
- storage
 - evaluating use of 12, 275
 - for active routines 58, 305
 - leak detecting 59, 100, 306, 338
 - offset listing 135, 377
 - report 12, 275
 - statistics 18, 20
- STORAGE compiler option 8
- storage dump
 - written to an HFS file 183, 403
- STORAGE option of CEE3DMP callable service 37
- STORAGE run-time option 10, 274
- structure
 - map 135, 143, 377
 - variable example code 143
- symbolic
 - feedback code 27, 279
- symbolic dumps 224
 - how to call under FORTRAN 224
- syntax diagrams
 - how to read xvii
- system abend
 - with TRAP(OFF) 32, 284
 - with TRAP(ON) 32, 284
- system dump
 - generating 75, 306
 - in z/OS UNIX shell 77, 307

T

- task global table (TGT) 204
- TERMINAL compiler option 4, 8
- TERMTHDACT run-time option
 - function 9, 39, 256, 273, 287
 - generating a dump and 23, 278
 - modifying condition handling behavior and 10, 274
 - suboptions 39, 287
- TEST compiler option 4, 6, 8
- TEST run-time option 10, 274
- text file name prefixes, Language Environment 31, 283
- THDCOM in dump 206
- THREAD option of CEE3DMP callable service 37
- THREADSTACK64 run-time option 275
- time
 - in dump 56, 303

- TRACE run-time option
 - function 10, 274
 - trace table 116, 360
- TRACE suboption of TERMTHDACT run-time option 39, 287
- TRACEBACK option of CEE3DMP callable service 37, 60
- transaction
 - dump 264
 - rollback 268
 - rollback effects of assembler user exit on 268
 - work area 264
- TRAP run-time option
 - function 10, 274
 - Language Environment condition handling and 24, 75, 307
 - Language Environment exception handling and 278

U

- UADUMP suboption of TERMTHDACT run-time option 23, 40, 278, 288
- UAIMM suboption of TERMTHDACT run-time option 23, 40, 278, 288
- UAONLY suboption of TERMTHDACT run-time option 23, 39, 278, 288
- UATRACE suboption of TERMTHDACT run-time option 23, 39, 278, 288
- unhandled conditions 26, 29, 280
 - establishing enclave termination behavior for 26
- USE EXCEPTION/ERROR declaratives 196
- USE FOR DEBUGGING declarative 196, 197
- user
 - abend 34, 285, 286
 - code 32, 284
 - exit 25, 26
 - heap
 - statistics 20
 - stack
 - statistics 18
- user-specified abends 34, 285
- USRHDLR run-time option 9, 24, 273
- utility and service subroutines
 - CDUMP/CPDUMP 223
 - DUMP/PDUMP 222
 - SDUMP 224

V

- variables
 - in Language Environment dump 60
 - structure example code 143
- VARIABLES option of CEE3DMP callable service 37, 60
- VBREF compiler option 7
- verb cross-reference 199
- Verbexit
 - LEDATA 78, 308
- version number
 - in dump 56, 303

W

working storage
 in dump 59, 204, 306

X

XPLINK

- downward-growing stack 61
- finding XPLINK information in a dump 166
- stack frame format 61, 62
- storage statistics 19
- trace table entries for 167

XREF compiler option 7, 8

XUFLOW run-time option

- function 10
- modifying condition handling behavior and 24

Z

z/OS UNIX System Services

- C application program and 183, 403
- generating a system dump 77, 307

Readers' Comments — We'd Like to Hear from You

z/OS
Language Environment
Debugging Guide

Publication No. GA22-7560-06

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

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



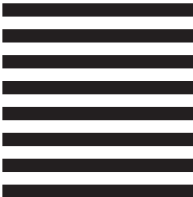
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Program Number: 5637-A01, 5655-G52

Printed in USA

GA22-7560-06

