

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 517.

Tenth Edition, September 2008

This is a major revision of GA22-7560-08.

This edition applies to Language Environment in z/OS Version 1 Release 10 (5694-A01) 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
MHVRCFS, Mail Station P181
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: mhvrdfs@us.ibm.com

World Wide Web: <http://www.ibm.com/systems/z/os/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, 2008. All rights reserved.

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

Contents

Figures	xi
Tables	xv
About this document	xvii
Using your documentation	xvii
How to read syntax diagrams	xix
Symbols	xix
Syntax items	xix
Syntax examples	xx
This debugging guide	xxi
Where to find more information	xxi
Information updates on the web	xxi
Summary of Changes	xxiii

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	5
Fortran compiler options	5
PL/I compiler options	6
Enterprise PL/I for z/OS compiler options	7
Using Language Environment run-time options	8
Determining run-time options in effect	9
Using the CLER CICS transaction to display and set run-time options	11
Controlling storage allocation	11
Stack storage statistics	17
Heap storage statistics	18
HEAPOOLS storage statistics	19
Modifying condition handling behavior	20
Language Environment callable services	20
Language Environment run-time options	20
Customizing condition handlers	22
Invoking the assembler user exit	23
Establishing enclave termination behavior for unhandled conditions	24
Using messages in your routine	24
C/C++	25
COBOL	25
Fortran	25
PL/I	25
Using condition information	25
Using the feedback code parameter	25
Using the symbolic feedback code	27
Chapter 2. Classifying errors	29
Identifying problems in routines	29
Language Environment module names	29
Common errors in routines	29
Interpreting run-time messages	30
Message prefix	31

Message number	31
Severity code	31
Message text	32
Understanding abend codes	32
User abends	32
System abends	32
Using edcmtext to obtain information about errno2 values	33
Format	33
Description	33
Usage notes	33
Message returns	33
Examples	34
Exit Values	34
Chapter 3. Using Language Environment debugging facilities	35
Debug tools	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	62
Using the DLL failure control block	78
Multiple enclave dumps	78
Generating a system dump	80
Steps for generating a system dump in a batch run-time environment	81
Steps for generating a system dump in an IMS run-time environment	81
Steps for generating a system dump in a CICS run-time environment	81
Steps for generating a Language Environment U4039 abend	82
Steps for generating a system dump in a z/OS UNIX shell	82
Formatting and analyzing system dumps	83
Preparing to use the Language Environment support for IPCS	84
Language Environment IPCS Verbexit – LEDATA	84
Format	85
Parameters	85
Understanding the Language Environment IPCS Verbexit LEDATA output	88
Understanding the HEAP LEDATA output	106
Diagnosing heap fragmentation problems	112
Understanding the HEAPPOOLS trace LEDATA output	112
Understanding the C/C++-specific LEDATA output	116
C/C++-specific sections of the LEDATA output	138
Understanding the COBOL-specific LEDATA output	140
COBOL-specific sections of the LEDATA Output	142
Formatting individual control blocks	143
Requesting a Language Environment trace for debugging	145
Locating the trace dump	146
Using the Language Environment trace table format in a dump report	147
Understanding the trace table entry (TTE)	147
Sample dump for the trace table entry	154

Part 2. Debugging language-specific routines 157

Chapter 4. Debugging C/C++ routines	159
Debugging C/C++ programs	159
Using the __amrc and __amrc2 structures to debug input/output	159
Displaying an error message with the perror() function	166

Using <code>__errno2()</code> to diagnose application problems	166
Diagnosing DLL problems	168
Using C/C++ listings	168
Finding variables.	169
Generating a Language Environment dump of a C/C++ routine.	177
<code>cdump()</code>	177
<code>csnap()</code>	178
<code>ctrace()</code>	178
Sample C routine that calls <code>cdump()</code>	178
Sample C++ routine that generates a Language Environment dump	180
Sample Language Environment dump with C/C++-specific information	182
Finding C/C++ information in a Language Environment dump	188
Sample Language Environment dump with XPLINK-specific information	193
Finding XPLINK information in a Language Environment dump.	198
C/C++ contents of the Language Environment trace tables	199
Debugging examples of C/C++ routines	206
Divide-by-zero error.	206
Calling a nonexistent non-XPLINK function	210
Calling a nonexistent XPLINK function.	213
Handling dumps written to the z/OS UNIX file system	216
Multithreading consideration	217
Understanding C/C++ heap information in storage reports	218
Language Environment storage report with HEAPPOOLS statistics	218
C function <code>__uheapreport()</code> storage report	219
MEMCHECK VHM memory leak analysis tool	221
Invoking MEMCHECK VHM.	221
MEMCHECK VHM environment variables	222
MEMCHECK VHM report sample scenario	223
MEMCHECK VHM report examples.	224
Chapter 5. Debugging COBOL programs	229
Determining the source of error	229
Tracing program logic	229
Finding input/output errors	230
Handling input/output errors.	230
Validating data (class test)	230
Assessing switch problems	230
Generating information about procedures.	231
Using COBOL listings	233
Generating a Language Environment dump of a COBOL program.	233
COBOL program that calls another COBOL program	233
COBOL program that calls the Language Environment CEE3DMP callable service	234
Sample Language Environment dump with COBOL-specific information	235
Finding COBOL information in a dump.	237
Debugging example COBOL programs.	242
Subscript range error	242
Calling a nonexistent subroutine	245
Divide-by-zero error.	248
Chapter 6. Debugging Fortran routines	255
Determining the source of errors in Fortran routines.	255
Identifying run-time errors	255
Using Fortran compiler listings.	257
Generating a Language Environment dump of a Fortran routine	257
DUMP/PDUMP subroutines.	258

CDUMP/CPDUMP subroutines	259
SDUMP subroutine	260
Finding Fortran information in a Language Environment dump	263
Understanding the Language Environment traceback table	264
Examples of debugging Fortran routines	265
Calling a nonexistent routine	265
Divide-by-zero error.	267
Chapter 7. Debugging PL/I for MVS & VM routines	271
Determining the source of errors in PL/I for MVS & VM routines	271
Logic errors in the source routine.	271
Invalid use of PL/I for MVS & VM	271
Unforeseen errors	272
Invalid input data.	272
Compiler or run-time routine malfunction	272
System malfunction.	272
Unidentified routine malfunction	272
Storage overlay problems	273
Using PL/I for MVS & VM compiler listings	274
Generating PL/I for MVS & VM listings and maps.	274
Finding information in PL/I for MVS & VM listings.	275
Generating a Language Environment dump of a PL/I for MVS & VM routine	281
PLIDUMP syntax and options	281
PLIDUMP usage notes	282
Finding PL/I for MVS & VM information in a dump	283
Traceback	283
Control blocks for active routines.	286
Control blocks associated with the thread.	290
PL/I for MVS & VM contents of the Language Environment trace table	292
Debugging example of PL/I for MVS & VM routines	293
Subscript range error	293
Calling a nonexistent subroutine	297
Divide-by-zero error.	299
Chapter 8. Debugging Enterprise PL/I routines	307
Determining the source of errors in Enterprise PL/I routines	307
Logic errors in the source routine.	307
Invalid use of Enterprise PL/I	307
Unforeseen errors	308
Invalid input data.	308
Compiler or run-time routine malfunction	308
System malfunction.	308
Unidentified routine malfunction	308
Storage overlay problems	309
Using Enterprise PL/I compiler listings	310
Generating Enterprise PL/I listings and maps	310
Finding information in Enterprise PL/I listings	311
Generating a Language Environment dump of an Enterprise PL/I routine	316
PLIDUMP syntax and options	317
PLIDUMP usage notes	318
Finding Enterprise PL/I information in a dump	319
Traceback	319
Control blocks for active routines.	321
Control blocks associated with the thread.	323
Enterprise PL/I contents of the Language Environment trace table	326
Debugging example of Enterprise PL/I routines	326

Subscript range error	326
Calling a nonexistent subroutine	330
Divide-by-zero error.	334
Chapter 9. Debugging under CICS	339
Accessing debugging information.	339
Locating Language Environment run-time messages	339
Locating the Language Environment traceback.	339
Locating the Language Environment dump	340
Using CICS transaction dump	341
Using CICS register and program status word contents	341
Using Language Environment abend and reason codes	341
Using Language Environment return codes to CICS.	342
Activating Language Environment feature trace records under CICS.	342
Ensuring transaction rollback	345
Finding data when Language Environment returns a nonzero return code	345
Finding data when Language Environment abends internally	345
Finding data when Language Environment abends from an EXEC CICS command	346
Displaying and modifying run-time options with the CLER transaction	346

Part 3. Debugging Language Environment AMODE 64 applications 349

Chapter 10. Preparing your AMODE 64 application for debugging	351
Setting compiler options	351
XL C and XL C++ compiler options for AMODE 64 applications	351
Using Language Environment run-time options.	351
Determining run-time options in effect	352
Controlling storage allocation	353
Storage statistics for AMODE 64 applications	357
Modifying exception handling behavior.	359
Language Environment application program interfaces (API).	359
Language Environment run-time options	359
Customizing exception handlers	360
Using condition information	360
Using the feedback code parameter.	360
Using the symbolic feedback code	362
Chapter 11. Classifying AMODE 64 application errors	363
Identifying problems in routines	363
Language Environment module names	363
Common errors in routines	363
Interpreting run-time messages	364
Message prefix	365
Message number	365
Severity code	365
Message text	365
Understanding abend codes	365
User abends	366
System abends	366
Chapter 12. Using Language Environment AMODE 64 debugging facilities	367
Debugging tools	367
Language Environment dumps	367
Generating a Language Environment dump with TERMTHDACT	367
Generating a Language Environment dump with language-specific functions	369

Understanding the Language Environment dump	369
Generating a system dump	386
Steps for generating a system dump in a batch run-time environment	387
Steps for generating a system dump in a z/OS UNIX shell	388
Formatting and analyzing system dumps	389
Preparing to use the Language Environment support for IPCS	389
Language Environment IPCS Verbexit – LEDATA	389
Format	390
Parameters	390
Understanding the Language Environment IPCS Verbexit LEDATA output	393
Understanding the HEAP LEDATA output	409
Diagnosing heap fragmentation problems	420
Understanding the heap pools trace LEDATA output	420
Understanding the C/C++-specific LEDATA output	426
C/C++-specific sections of the LEDATA output	438
Understanding the AUTH LEDATA output	440
Sections of the AUTH LEDATA Verbexit formatted output	445
Formatting individual control blocks	446
Requesting a Language Environment trace for debugging	448
Locating the trace dump	449
Using the Language Environment trace table format in a dump report	450
Understanding the trace table entry (TTE)	450
Sample dump for the trace table entry	456
Chapter 13. Debugging AMODE 64 C/C++ routines	459
Debugging C/C++ programs	459
Using the __amrc and __amrc2 structures to debug input/output	459
Displaying an error message with the perror() function	466
Using __errno2() to diagnose application problems	467
Using C/C++ listings	469
Finding variables	469
Generating a Language Environment dump of a C/C++ routine	472
cdump()	472
csnap()	473
ctrace()	473
Sample C routine that calls cdump()	473
Sample C++ routine that generates a Language Environment dump	475
Sample Language Environment dump with C/C++-specific information	477
C/C++ contents of the Language Environment trace tables	483
Debugging examples of C/C++ routines	485
Divide-by-zero error	485
Calling a nonexistent function	492
Handling dumps written to the z/OS UNIX file system	499
Multithreading consideration	500
Understanding C/C++ heap information in storage reports	500
Language Environment storage report with heap pools statistics	501
C function __uheapreport() storage report	503
Appendix A. Diagnosing problems with Language Environment	507
Diagnosis checklist	507
Locating the name of the failing routine for a non-XPLINK application	508
Searching the IBM Software Support Database	511
Preparing documentation for an Authorized Program Analysis Report (APAR)	511
Appendix B. Accessibility	515
Using assistive technologies	515

Keyboard navigation of the user interface.	515
z/OS information	515
Notices	517
Policy for unsupported hardware	519
Programming Interface Information	519
Trademarks.	519
Bibliography	521
Language Products Publications	521
Related Publications	522
Index	525

Figures

1. Options report example produced by run-time option RPTOPTS(ON).	10
2. Storage report produced by run-time option RPTSTG(ON)	12
3. Storage report produced by RPTSTG(ON) with XPLINK	14
4. Language Environment condition token	27
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	64
9. Downward-growing (XPLINK) stack frame format	65
10. Common anchor area	66
11. Condition information block	75
12. Machine state information block	77
13. Language Environment dump of multiple enclaves	79
14. Example of formatted output from LEDATA Verbexit	89
15. Example of formatted PTBL output from LEDATA Verbexit	105
16. Example formatted detailed heap segment report from LEDATA Verbexit	107
17. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit	112
18. Example formatted C/C++ output from LEDATA Verbexit	117
19. Example formatted COBOL output from LEDATA Verbexit	141
20. The CAA formatted by the CBFORMAT IPCS command	144
21. Format of the trace table entry	147
22. Trace table in dump output.	155
23. __amrc structure	160
24. __amrc2 structure	161
25. Example of a routine using perror()	166
26. Example of a routine using __errno2()	167
27. Sample output of a routine using __errno2()	167
28. Example of a routine using _EDC_ADD_ERRNO2	167
29. Sample output of a routine using _EDC_ADD_ERRNO2	167
30. Example of a routine using __err2ad() in combination with __errno2()	168
31. Sample output of routine using __err2ad() in combination with __errno2()	168
32. Writable static map produced by prelinker	171
33. Location of RENT static variable in storage.	172
34. Writable static map produced by prelinker	172
35. Location of NORENT static variable in storage	173
36. Example code for parameter variable	174
37. Example code for parameter variable	174
38. Partial storage offset listing	175
39. Example code for structure variable	175
40. Example of aggregate map	176
41. Writable static map produced by prelinker	176
42. Example C routine using cdump() to generate a dump	179
43. Fetched module for C routine.	180
44. Example C++ routine with protection exception generating a dump	181
45. Template file STACK.C	181
46. Header file STACK.H.	182
47. Example dump from sample C routine	183
48. Memory file control block	191
49. Registers on entry to CEE3DMP	192
50. Parameters, registers, and variables for active routines	192
51. Condition information for active routines	193
52. Sample XPLINK-compiled program (tranmain) which calls a NOXPLINK-compiled program	194
53. Sample NOXPLINK-compiled program (trandll) which calls an XPLINK-compiled program	195

54. Example dump of calling between XPLINK and non-XPLINK programs	196
55. Trace table with C/C++ trace table entry types 1 thru 4	201
56. Trace table with XPLINK trace table entries 5 and 6.	203
57. Trace table with trace table entry types 7 and 8	204
58. C routine with a divide-by-zero error	206
59. Sections of the dump from example C/C++ routine	207
60. Pseudo assembly listing.	208
61. C/C++ CAA information in dump.	209
62. Writable static map	209
63. Enclave storage section of dump	210
64. C/C++ example of calling a nonexistent subroutine	210
65. Sections of the dump from example C routine	211
66. Pseudo assembly listing.	212
67. Writable static map	212
68. Enclave control blocks and storage sections in dump	213
69. C/C++ example of calling a nonexistent subroutine	213
70. Sections of the dump from example C routine.	214
71. Pseudo assembly listing.	215
72. Writable static map	216
73. Enclave control blocks and storage sections in dump	216
74. IPCS panel for entering data set information	217
75. Storage report generated by __uheapreport()	221
76. Trace report generated by MEMCHECK VHM.	225
77. Heap Leak Report generated by MEMCHECK VHM	227
78. Example of using the WITH DEBUGGING MODE clause	232
79. COBOL program COBDUMP1 calling COBDUMP2.	234
80. COBOL program COBDUMP2 calling the Language Environment dump service CEE3DMP	235
81. Sections of the Language Environment dump called from COBDUMP2	236
82. Control block information for active COBOL routines	238
83. Storage for active COBOL programs	239
84. Enclave-level data for COBOL programs.	241
85. Process-level control blocks for COBOL programs	242
86. COBOL example of moving a value outside an array range.	243
87. Sections of Language Environment dump for COBOLX	244
88. COBOL listing for COBOLX	245
89. COBOL example of calling a nonexistent subroutine	245
90. Sections of Language Environment dump for COBOLY	246
91. COBOL Listing for COBOLY	247
92. Parameters, registers, and variables for active routines section of dump for COBOLY	248
93. Main COBOL program, COBOL subroutine, and assembler routine	249
94. Sections of Language Environment dump for program COBOLZ1	250
95. COBOL listing for COBOLZ2	251
96. Listing for ASSEMZ3	252
97. Variables section of Language Environment dump for COBOLZ2.	252
98. Listing for COBOLZ2	253
99. Variables section of Language Environment dump for COBOLZ1.	253
100. Example program that calls SDUMP	262
101. Language Environment dump generated using SDUMP	263
102. Sections of the Language Environment dump	264
103. Example of calling a nonexistent routine.	265
104. Sections of the Language Environment dump resulting from a call to a nonexistent routine	266
105. Fortran routine with a divide-by-zero error	267
106. Language Environment dump from divide-by-zero Fortran example	268
107. PL/I for MVS & VM routine compiled with LIST and MAP	275
108. Compiler-generated listings from example PL/I for MVS & VM routine	276
109. Traceback section of dump	284

110. Task traceback section	285
111. Control blocks for active routines section of the dump	287
112. Control blocks associated with the thread section of the dump	291
113. Example of moving a value outside an array range	294
114. Sections of the Language Environment dump	295
115. Example of calling a nonexistent subroutine	297
116. Sections of the Language Environment dump	298
117. PL/I for MVS & VM routine with a divide-by-zero error.	300
118. Variables from routine SAMPLE	300
119. Object code listing from example PL/I for MVS & VM routine	301
120. Language Environment dump from example PL/I for MVS & VM routine	302
121. Enterprise PL/I routine compiled with LIST, MAP, and SOURCE	311
122. Compiler-generated listings from example Enterprise PL/I routine	312
123. Traceback section of dump	320
124. Control blocks for active routines section of the dump	322
125. Control blocks associated with the thread section of the dump	324
126. Example of moving a value outside an array range	327
127. Sections of the Language Environment dump	328
128. Example of calling a nonexistent subroutine	331
129. Sections of the Language Environment dump	332
130. Enterprise PL/I routine with a divide-by-zero error	334
131. Variables from routine SAMPLE	334
132. Object code listing from example Enterprise PL/I routine	335
133. Language Environment dump from example Enterprise PL/I routine.	336
134. Language Environment traceback written to the transient data queue	340
135. CICS trace output in the ABBREV format.	343
136. CICS trace output in the FULL format.	344
137. 64-bit options report	353
138. 64-bit storage report	354
139. Language Environment condition token	361
140. The C program CELQSAMP	371
141. The C DLL CELQDLL	373
142. Example dump using CEE3DMP	373
143. Example of formatted output from LEDATA Verbexit	394
144. Example of formatted PTBL output from LEDATA Verbexit	408
145. Example formatted detailed heap segment report from LEDATA Verbexit	410
146. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit	421
147. Example formatted C/C++ output from LEDATA Verbexit	427
148. Example of formatted AUTH output from LEDATA Verbexit	441
149. The CAA formatted by the CBFORMAT IPCS command	447
150. Format of the trace table entry	450
151. Trace table in dump output.	457
152. __amrc structure	461
153. __amrc2 structure	461
154. Example of a routine using perror()	467
155. Example of a routine using __errno2()	467
156. Sample output of routine using __errno2()	468
157. Example of a routine using _EDC_ADD_ERRNO2	468
158. Sample output of a routine using _EDC_ADD_ERRNO2	468
159. Example of a routine using __err2ad() in combination with __errno2()	469
160. Sample output of routine using __err2ad() in combination with __errno2()	469
161. Example code for structure variables	472
162. Example of aggregate map	472
163. Example C routine using cdump() to generate a dump	474
164. Fetched module for C routine.	475
165. Example C++ routine with protection exception generating a dump	476

166. Template file STACK.C	476
167. Header file STACK.H	477
168. Example dump from sample C routine	478
169. Trace table with XPLINK trace table entries 5 and 6.	484
170. C routine with a divide-by-zero error	486
171. Sections of the dump from example C/C++ routine	487
172. Pseudo assembly listing.	489
173. C/C++ CAA information in dump.	491
174. Writable static map	492
175. IPCS storage display of the writeable static area.	492
176. C/C++ example of calling a nonexistent subroutine	493
177. Sections of the dump from example C routine.	494
178. Pseudo assembly listing.	497
179. Writable static map	499
180. IPCS storage display of the writeable static area.	499
181. IPCS panel for entering data set information	500
182. Storage report generated by ___uheapreport()	505
183. C PPA1.	510
184. Nonconforming entry point type with sample dump	510

Tables

1. How to Use z/OS Language Environment Publications	xviii
2. Syntax examples	xx
3. Common error symptoms, possible causes, and programmer responses	30
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	67
8. Language Environment Control blocks that can be individually formatted	144
9. LE=1 entry records	148
10. LE=2 entry records	149
11. Format of the mutex/CV/latch records	153
12. LE=8 entry records	154
13. LE=20 entry records	154
14. __last_op values and diagnosis information	162
15. Finding the WSA base address	169
16. Compiler-generated COBOL listings and their contents	233
17. Compiler-generated Fortran listings and their contents	257
18. Compiler-generated PL/I for MVS & VM listings and their contents	274
19. Typical comments in a PL/I for MVS & VM static storage listing	277
20. Comments in a PL/I for MVS & VM object code listing	279
21. PL/I for MVS & VM mnemonics	280
22. Compiler-generated PL/I listings and their contents	310
23. Finding data when Language Environment returns a nonzero return code	345
24. Finding data when Language Environment abends internally	345
25. Finding data when Language Environment abends from an EXEC CICS command	346
26. Common error symptoms, possible causes, and programmer responses	364
27. TERMTHDACT suboptions, level of information, and destinations	367
28. Language Environment control blocks which can be individually formatted	447
29. Preinitialized Environments for Authorized Programs control blocks which can be individually formatted	448
30. LE=1 entry records	451
31. LE=2 entry records	451
32. Format of the mutex/CV/latch records	455
33. LE=8 entry records	456
34. __last_op values and diagnosis information	463
35. Problem resolution documentation requirements	512

About this document

This document supports z/OS (5694–A01).

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)
- z/OS C/C++
- OS/390 C/C++
- 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
- Enterprise COBOL for z/OS and OS/390
- COBOL for OS/390 & VM
- COBOL for MVS & VM (formerly COBOL/370)
- Enterprise PL/I for z/OS
- Enterprise PL/I for z/OS and OS/390
- VisualAge PL/I
- PL/I for MVS & VM (formerly PL/I MVS™ & VM)
- VS FORTRAN and FORTRAN IV (in compatibility mode)

Although not all compilers listed are currently supported, Language Environment supports the compiled objects that they created.

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 and PL/I compiler products.

Debug Tool is also available as a standalone product. Debug Tool Utilities and Advanced Functions is also available. For more information, see <http://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 521.

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</i> and your language programming guide
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.	<pre> >>—KEYWORD—required_item—————>>> </pre>
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.	<pre> >>—KEYWORD— required_choice1 required_choice2 —————>>> </pre>
Optional item.	
Optional items appear below the main path of the horizontal line.	<pre> >>—KEYWORD— optional_item —————>>> </pre>
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.	<pre> >>—KEYWORD— optional_choice1 optional_choice2 —————>>> </pre>
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.	<pre> >>—KEYWORD— default_choice1 optional_choice2 optional_choice3 —————>>> </pre>
Variable.	
Variables appear in lowercase italics. They represent names or values.	<pre> >>—KEYWORD—variable—————>>> </pre>
Repeatable item.	
An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.	<pre> >>—KEYWORD—repeatable_item—————>>> </pre>
A character within the arrow means you must separate repeated items with that character.	<pre> >>—KEYWORD—repeatable_item—————>>> </pre>
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.	<pre> >>—KEYWORD— fragment —————>>> </pre> <p>fragment:</p> <pre> ,required_choice1 ,required_choice2 ,default_choice ,optional_choice ————— </pre>

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:

- Language Environment
- 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.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS™, see the online document at:
http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR

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

Summary of Changes

Summary of Changes for GA22-7560-09 z/OS Version 1 Release 10

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-08, which supported z/OS Version 1 Release 9.

The following summarizes the changes to that information.

New Information

- “HEAPPOOLS storage statistics” on page 502 has been added to “Language Environment storage report with heap pools statistics” on page 501.

Changed Information

- A usage note has been added to “Language Environment IPCS Verbexit – LEDATA” on page 84 and “Language Environment IPCS Verbexit – LEDATA” on page 389
- In “Understanding the C/C++-specific LEDATA output” on page 116, Figure 18 on page 117 has been updated with changed and new sections.
- In “Understanding the C/C++-specific LEDATA output” on page 426, Figure 147 on page 427 has been updated with changed and new sections.
- A new parameter has been added to “Language Environment IPCS Verbexit – LEDATA” on page 84.
- In “Determining run-time options in effect” on page 352, the figure of 64-bit options report has been updated.
- In “Controlling storage allocation” on page 11, Figure 3 on page 14 has been replaced.
- In “HEAPPOOLS storage statistics” on page 218, HEAPPOOLS statistics has been updated.
- In “HEAPPOOLS64 storage statistics” on page 501, HEAPPOOLS64 statistics and HEAPPOOLS64 summary has been updated.
- In “Understanding the HEAPPOOLS trace LEDATA output” on page 112, Figure 17 on page 112 has been updated.
- In “Understanding the heap pools trace LEDATA output” on page 420, Figure 146 on page 421 has been updated.
- “Storage statistics for AMODE 64 applications” on page 357 has been moved from Appendix C in *z/OS Language Environment Programming Reference* to “Controlling storage allocation” on page 353.

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.

This document contains 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 GA22-7560-08 z/OS Version 1 Release 9

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-07, which supported z/OS Version 1 Release 8.

The following summarizes the changes to that information.

New Information

- A new chapter, Chapter 8, “Debugging Enterprise PL/I routines,” on page 307, has been added to provide information about debugging Enterprise PL/I routines.
- Language Environment® improves DLL error diagnostics so that you can more easily debug DLL application problems. For more information, see “Diagnosing DLL problems” on page 168 and “Using the DLL failure control block” on page 78.
- A new section, “Using edcmtext to obtain information about errno2 values” on page 33, has been added to provide the edcmtext utility (similar to bpxmtext), which allows faster error resolution when an errno2 is encountered in Language Environment.

Changed Information

- In Chapter 3, “Using Language Environment debugging facilities,” on page 35, the following sections have been updated:
 - “Debug tools” on page 35
 - “Understanding the Language Environment dump” on page 43
 - “The Common Anchor Area” on page 65
 - “Formatting and analyzing system dumps” on page 83
- In Chapter 4, “Debugging C/C++ routines,” on page 159, the following sections have been updated:
 - “Using __errno2() to diagnose application problems” on page 166
 - “Sample Language Environment dump with C/C++-specific information” on page 182
 - “Sample Language Environment dump with XPLINK-specific information” on page 193
 - “Debugging examples of C/C++ routines” on page 206
 - “Understanding C/C++ heap information in storage reports” on page 218, example storage reports and the description of the HeapPools storage statistics section of the storage report are updated.
- In Chapter 5, “Debugging COBOL programs,” on page 229, figures have been updated in the following sections:
 - “Sample Language Environment dump with COBOL-specific information” on page 235
 - “Finding COBOL information in a dump” on page 237
 - “Debugging example COBOL programs” on page 242
- In Chapter 7, “Debugging PL/I for MVS & VM routines,” on page 271, the following sections have been updated:
 - “Finding PL/I for MVS & VM information in a dump” on page 283
 - “Debugging example of PL/I for MVS & VM routines” on page 293

- In Chapter 9, “Debugging under CICS,” on page 339, Figure 134 on page 340, Figure 134 on page 340 has been updated; enhancements are made in the CICS CLER transaction for displaying and modifying Language Environment run-time options. For more information, see “Displaying and modifying run-time options with the CLER transaction” on page 346.
- In Chapter 10, “Preparing your AMODE 64 application for debugging,” on page 351, Figure 137 on page 353 and Figure 138 on page 354 have been updated.
- In Chapter 12, “Using Language Environment AMODE 64 debugging facilities,” on page 367, the following sections have been updated:
 - “Understanding the Language Environment dump” on page 369
 - “Language Environment IPCS Verbexit – LEDATA” on page 389
 - “Understanding the Language Environment IPCS Verbexit LEDATA output” on page 393
 - “Understanding the HEAP LEDATA output” on page 409
 - “Formatting individual control blocks” on page 446
- In Chapter 13, “Debugging AMODE 64 C/C++ routines,” on page 459, the following sections have been updated:
 - “Using __errno2() to diagnose application problems” on page 467
 - “Calling a nonexistent function” on page 492
 - “Understanding C/C++ heap information in storage reports” on page 500

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

Summary of Changes for GA22-7560-07 z/OS Version 1 Release 8

This document contains information previously presented in *z/OS Language Environment Debugging Guide*, GA22-7560-06, which supported z/OS Version 1 Release 7.

The following summarizes the changes to that information.

New Information

- In Chapter 3, “Using Language Environment debugging facilities,” on page 35, the Language Environment trace table entry and sample dump are updated for XPLINK transitions support.
- In Chapter 4, “Debugging C/C++ routines,” on page 159, the C/C++ contents of the trace tables are updated to include extended VSAM support.
- In Chapter 4, “Debugging C/C++ routines,” on page 159, the C/C++ __armc and __armc2 structures are updated for C run-time support for large data sets (more than 64KB tracks per volume).
- In Chapter 3, “Using Language Environment debugging facilities,” on page 35 and Chapter 12, “Using Language Environment AMODE 64 debugging facilities,” on page 367, the sections titled, “Generating a system dump,” have been updated to include steps for capturing an IPCS-readable dump when no SYSMDUMP DD statement is available. For more information, see: “Generating a system dump” on page 80 and “Generating a system dump” on page 386.

Changed Information

- In Chapter 3, “Using Language Environment debugging facilities,” on page 35 and Chapter 12, “Using Language Environment AMODE 64 debugging facilities,” on page 367, the examples of formatted output from the Language Environment IPCS verbexit LEDATA are updated.
- In Chapter 4, “Debugging C/C++ routines,” on page 159, the steps for finding RENT static variables are updated.
- In Chapter 12, “Using Language Environment AMODE 64 debugging facilities,” on page 367, the CEEDUMP dump example using CEE3DMP is updated to show that source statement numbers are displayed in the traceback section for AMODE 64 applications. For more information, see, “Understanding the Language Environment dump” on page 369.

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, see 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*.

See 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 the following documents:

- *Enterprise COBOL for z/OS Programming Guide, SC23-8529*
- *Enterprise COBOL for z/OS Programming Guide, SC27-1412*
- *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*.

Enterprise PL/I for z/OS compiler options

The following Enterprise PL/I for z/OS compiler options can be specified when preparing your Enterprise PL/I for z/OS 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 Enterprise PL/I for z/OS compiler options, see *Enterprise PL/I for z/OS 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, DYNDUMP, 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.
CEEDUMP	Specifies options to control the processing of the Language Environment dump report.
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.)
DYNDUMP	Provides a way to obtain IPCS-readable dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement
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 11.
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 10 shows a sample options report.

Options Report for Enclave main 12/07/06 6:25:56 PM
 Language Environment V01 R09.00

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	CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC)
Installation default	CHECK(ON)
Installation default	COUNTRY(US)
Installation default	NODEBUG
Installation default	DEPTHCONDLMT(10)
Installation default	DYNDUMP(*USERID,NODYNAMIC,TDUMP)
Installation default	ENVAR("")
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)
Installation default	HEAPOOLS(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	INFOMSGFILTER(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)
Installation default	POSIX(OFF)
Installation default	PROFILE(OFF,"")
Installation default	PRTUNIT(6)
Installation default	PUNUNIT(7)
Installation default	RDRUNIT(5)
Installation default	RECPAD(OFF)
Invocation command	RPTOPTS(ON)
SETCEE command	RPTSTG(ON)
Installation default	NORTEREUS
Installation default	NOSIMVRD
Installation default	STACK(131072,131072,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	VCTRSVAVE(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.

For more information about the CICS CLER transaction, see “Displaying and modifying run-time options with the CLER transaction” on page 346.

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 12 and Figure 3 on page 14 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 01/08/08 4:27:49 PM
Language Environment V01 R10.00

STACK statistics:
Initial size: 131072
Increment size: 131072
Maximum used by all concurrent threads: 5416
Largest used by any thread: 5416
Number of segments allocated: 1
Number of segments freed: 0

THREADSTACK statistics:
Initial size: 4096
Increment size: 4096
Maximum used by all concurrent threads: 45536
Largest used by any thread: 6552
Number of segments allocated: 60
Number of segments freed: 0

XPLINK STACK statistics:
Initial size: 524288
Increment size: 131072
Largest used by any thread: 20400
Number of segments allocated: 1
Number of segments freed: 0

XPLINK THREADSTACK statistics:
Initial size: 131072
Increment size: 131072
Largest used by any thread: 22160
Number of segments allocated: 30
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): 286576
Successful Get Heap requests: 71
Successful Free Heap requests: 1
Number of segments allocated: 10
Number of segments freed: 0

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

```

|
|
|   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): 1139712
|     Successful Get Heap requests:             487
|     Successful Free Heap requests:            431
|     Number of segments allocated:             50
|     Number of segments freed:                 36
|
|   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:          0
|     Successful Discard Heap requests:         0
|     Total heap storage used:                  0
|     Successful Get Heap requests:              0
|     Successful Free Heap requests:            0
|     Number of segments allocated:             0
|     Number of segments freed:                 0
|
|   HEAPOOLS Statistics:
|     Pool 1  size: 8 Get Requests:            3
|       Successful Get Heap requests: 1- 8      3
|     Pool 2  size: 32 Get Requests:          268
|       Successful Get Heap requests: 9- 16      36
|       Successful Get Heap requests: 17- 24     3
|       Successful Get Heap requests: 25- 32    229
|     Pool 3  size: 128 Get Requests:        186
|       Successful Get Heap requests: 33- 40     3
|       Successful Get Heap requests: 41- 48     8
|       Successful Get Heap requests: 49- 56    111
|       Successful Get Heap requests: 57- 64     4
|       Successful Get Heap requests: 65- 72     2
|       Successful Get Heap requests: 73- 80     4
|       Successful Get Heap requests: 81- 88     6
|       Successful Get Heap requests: 89- 96     2
|       Successful Get Heap requests: 97- 104    1
|       Successful Get Heap requests: 105- 112   5
|       Successful Get Heap requests: 113- 120  31
|       Successful Get Heap requests: 121- 128   9
|

```

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

```

Pool 4 size: 256 Get Requests: 38
  Successful Get Heap requests: 137- 144 2
  Successful Get Heap requests: 145- 152 2
  Successful Get Heap requests: 153- 160 2
  Successful Get Heap requests: 161- 168 1
  Successful Get Heap requests: 169- 176 4
  Successful Get Heap requests: 177- 184 4
  Successful Get Heap requests: 185- 192 2
  Successful Get Heap requests: 193- 200 2
  Successful Get Heap requests: 201- 208 3
  Successful Get Heap requests: 209- 216 2
  Successful Get Heap requests: 217- 224 3
  Successful Get Heap requests: 225- 232 3
  Successful Get Heap requests: 233- 240 1
  Successful Get Heap requests: 241- 248 3
  Successful Get Heap requests: 249- 256 4
Pool 5.1 size: 1024 Get Requests: 230
Pool 5.2 size: 1024 Get Requests: 3
Pool 5.3 size: 1024 Get Requests: 5
  Successful Get Heap requests: 257- 264 225
  Successful Get Heap requests: 273- 280 2
  Successful Get Heap requests: 281- 288 10
  Successful Get Heap requests: 841- 848 1
Pool 6 size: 2048 Get Requests: 2
  Successful Get Heap requests: 1113- 1120 1
  Successful Get Heap requests: 1137- 1144 1
Requests greater than the largest cell size: 2
HEAPPOOLS Summary:
Specified Element Extent Cells Per Extents Maximum Cells In
Cell Size Size Percent Extent Allocated Cells Used Use
-----
8 16 10 204 1 1 0
32 40 10 81 3 226 225
128 136 10 24 4 88 77
256 264 10 12 1 1 0
1024 1032 10 4 57 228 227
1024 1032 10 4 1 2 0
1024 1032 10 4 1 3 0
2048 2056 10 4 1 2 2
-----
Suggested Percentages for current Cell Sizes:
HEAPP(ON,8,1,32,28,128,37,256,1,(1024,3),90,2048,13,0)
Suggested Cell Sizes:
HEAPP(ON,
32,,56,,88,,120,,128,,168,,
208,,248,,288,,848,,1144,,2080,)
Largest number of threads concurrently active: 11
End of Storage Report

```

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

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. See *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 218.

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.
CEE3AB2	Terminate enclave with an abend and reason code.
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: <ul style="list-style-type: none">• 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 for MVS & VM 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	<p>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.</p> <p>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.</p> <p>Language Environment ignores ABPERC(0Cx). No abend is percolated and Language Environment condition handling semantics are in effect.</p>
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	<p>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:</p> <ul style="list-style-type: none"> • 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)	<p>Fully enables the Language Environment condition handler. This causes the Language Environment condition handler to intercept error conditions and routine interrupts.</p> <p>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.</p> <p>During normal operation, you should use TRAP(ON) when running your applications.</p>
TRAP(OFF)	<p>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.</p> <p>Specify TRAP(OFF) when you do not want Language Environment to issue an ESTAE or an ESPIE.</p> <p>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.</p> <p>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>.</p>
USRHDLR	<p>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.</p> <p>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.</p>
XUFLOW	<p>Specifies whether an exponent underflow causes a routine interrupt.</p>

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 23 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 30.)

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 27).

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 Severity Number	16 - 31 Message Number
---------------------------	---------------------------

For Case 2 condition tokens,
Condition_ID is:

0 - 15 Class Code	16 - 31 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)
- CEL (Language Environment and C/C++ run-time library)
- 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 11 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 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. See the messages and codes book of the operating system.
System abend with TRAP(ON)	System-detected error	See 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	See 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* and in *z/OS MVS Diagnosis: Reference*.

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 24, 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. If the DYNDUMP run-time option is used in combination with the TERMTHDACT run-time option, the system dump can be written without the ddname specified. System dumps show the memory state at the time of the condition. See “Generating a system dump” on page 80 for more information about system dumps.

Using edcmtext to obtain information about errno2 values

Language Environment provides the **edcmtext** utility (similar to **bpxmtext**), which allows faster error resolution when an **errno2** is encountered in Language Environment. Use the **edcmtext** utility to display **errno2** reason code text. This utility produces a description and action for the **errno2** value.

The **bpxmtext** utility calls **edcmtext** when the **errno2** value is in the range reserved for the C run-time library or **edcmtext** can be invoked directly with the **errno2** value as input.

Format

z/OS UNIX[®] environment:

edcmtext *errno2_value*

TSO/E environment:

EDCMTEXT *errno2_value*

Description

edcmtext displays the description and action text for C/C++ run-time library **errno2** values, no other values are supported by this command. This command is intended as an aid for problem determination.

errno2_value is specified as 8 hexadecimal characters.

The user can specify one of the following in place of a **errno2** value to view a help dialog: *-h*, *help*, *?*.

The user can specify the **-U** option to display the output in uppercase.

Usage notes

errno2_values are also accepted in mixed case and with hex digits prefixed with the "0x".

The range of values for the C/C++ run-time library is 0X'C0000000' through 0X'FFFFFFFF'.

The utility **bpxmtext** displays the description and action text for reason codes returned from the kernel, in addition to *errno2_values* returned from the C/C++ run-time library. The user should use **bpxmtext** when the source of the *errno2_values* is unknown. For further information about **bpxmtext**, see *z/OS UNIX System Services Command Reference*.

Message returns

If the user specifies a *-h*, *help* or *?* in place of the *errno2_value*, the following message is displayed:

Usage: edcmtext *errno2_value*

If no text is available for the *errno2_value*, the following message is displayed:

errno2_value: No information is currently available for this *errno2_value*.

If the *errno2_value* is not comprised of 1-8 hex digits, the following message is displayed:

Usage: edcmtext errno2_value

If the *errno2_value* is not in the C/C++ run-time library range, the following message is displayed:

Notice: The *errno2_value* is not in the C/C++ run-time library range.

If the environment that *edcmtext* is being run in is not TSO/E or z/OS UNIX the following message is displayed:

Error: The environment is not TSO/E or z/OS UNIX.

Examples

The command:

```
edcmtext C00B0021
```

produces data displayed in the following format:

```
JrEdclopsEInval01: The mode argument passed to fopen() or freopen() did not begin with r, w, or a.
```

Action: Correct the mode argument. The first keyword of the mode argument must be the open mode. Ensure the open mode is specified first and begins with r, w, or a.

Source: edclopst.c

Exit Values

- | | |
|---------------|--|
| 0 | Successful completion |
| 2 | Failure due to an argument that is not 1–8 hex digits |
| 8 | Bad Input due to an <i>errno2_value</i> out of the C/C++ run-time range. |
| 14 | Environment not TSO/E or z/OS UNIX |
| >20 | Contact IBM due to Internal Error |

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 tools

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++ for MVS/ESA, Enterprise COBOL for z/OS, COBOL for OS/390 & VM, COBOL for MVS & VM, PL/I for MVS & VM, Enterprise PL/I for z/OS, and VisualAge for Java compiler products. IBM Debug Tool for z/OS is also available as a standalone product for debugging XL C/C++ applications. For more information on Debug Tool, see <http://www.ibm.com/software/awdtools/debugtool/>.

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. See the Debug Tool publications for a detailed explanation of how to invoke and run Debug Tool. For more information, see <http://www.ibm.com/software/awdtools/debugtool/>.

You can use IBM WebSphere® Developer Debugger for System z™ to get a workstation graphical interface to IBM Debug Tool for z/OS. For more information, see <http://www-306.ibm.com/software/awdtools/debugtool/tools/wddsz/>.

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 CEEDUMP 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 (`/`), the directory is writable, and the CEEDUMP pathname does not exceed 1024 characters.
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.

Table 4. CEE3DMP options (continued)

Dump Options	Abbreviation	Action Taken
ENCLAVE(CURRENT)	ENCL(CUR)	Dumps the current enclave.
ENCLAVE(n)	ENCL(n)	Dumps a fixed number of enclaves, indicated by <i>n</i> .
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.

Table 4. CEE3DMP options (continued)

Dump Options	Abbreviation	Action Taken
STACKFRAME(ALL)	SF(ALL)	Dumps all stack frames from the call chain. This is the default setting for STACKFRAME.
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:

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

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

- BLOCKS
- CONDITION
- ENCLAVE(ALL)
- FILES
- FNAME(CEEDUMP)
- GENOPTS
- NOENTRY

- STACKFRAME(ALL)
- STORAGE
- THREAD(CURRENT)
- TRACEBACK
- VARIABLES

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 OCx abends

Options	TERMTHDACT(X,CESE,)	TERMTHDACT(X,CICSDDS,)
QUIET	<ul style="list-style-type: none"> No output. 	<ul style="list-style-type: none"> No output.
MSG	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. 	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE.
TRACE	<ul style="list-style-type: none"> The traceback is written to the CESE queue, followed by U4038 abend with nodump option. 	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> CEEDUMP to CESE queue followed by U4038 abend with nodump option. 	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> U4039 abend with traceback to CESE queue followed by U4038 abend with nodump option. 	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> U4039 abend with CEEDUMP to CESE queue followed by U4038 abend with nodump option. 	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> U4039 abend followed by U4038 abend with nodump option. 	<ul style="list-style-type: none"> U4039 abend followed by U4038 abend with nodump option. No CEEDUMP information is generated. Same as CESE.
UAIMM	<ul style="list-style-type: none"> U4039 abend followed by U4038 abend with nodump option. 	<ul style="list-style-type: none"> 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 177.

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 257.

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 for MVS & VM routine” on page 281.

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 56.

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.d"),NOOPT,TEST(SYM))
#pragma runopts(TERMTHDACT(UADUMP),POSIX(ON),DYNDUMP(,DYNAMIC,))
#pragma runopts(TRACE(ON,1M,NODUMP,LE=1),HEAPCHK(ON,1,0,10,10))
#pragma runopts(RPTSTG(ON),HEAPPOOLS(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)

```

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;
    CEEHDR (&pgmptr, &token, &fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf( "CEEHDR 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 with incorrect function name...\n");
    pgmptr.address = (_POINTER)dllqueryfn(handle, "name_not_in_dll");
    if (pgmptr.address != NULL) {
        perror("Found incorrect function name in DLL");
        exit(111);
    }

    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.b"),TEST(SYM),NOOPT)
#pragma export(dump_n_perc)
#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 GENOPTS"
#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 56.

ASID: 003F Job ID: JOB14342 Job name: CELSAMP Step name: STEP1 PID: 33620309 Parent PID: 1 User name: HEALY

[2] CEE3845I CEEDUMP Processing started.

[3] CEE3DMP called by program unit //'POSIX.CRTL.C(CELDLL)' (entry point dump_n_perc) at statement 34 (offset +0000012E).

[4] Registers on Entry to CEE3DMP:

```
PM..... 0100
GPR0..... 20FC8078 GPR1..... 20FC7F18 GPR2..... 20FC7F78 GPR3..... 213563EA
GPR4..... 20FC7F28 GPR5..... 21356620 GPR6..... 00000002 GPR7..... 2090E2A8
GPR8..... A09C6C34 GPR9..... 20FC0C00 GPR10..... 20FC5CA7 GPR11..... A0A82198
GPR12..... 209159B0 GPR13..... 20FC7E80 GPR14..... A13564E0 GPR15..... A09F0ED0
FPR0..... 4DBFF67D 46DBE848 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
GPREG STORAGE:
Storage around GPR0 (20FC8078)
-0020 20FC8058 40404040 40404040 40404040 40404040 40404040 40404040 40404000 | .....S....."
+0000 20FC8078 00000000 00000000 00000000 00000000 2090E2D0 00000000 00000000 20FC7F28 | .....f-...."
+0020 20FC8098 00000000 00000000 00000000 00000000 00000000 20FC7E80 20FC8660 A09F1D0E | .....f-...."
Storage around GPR1 (20FC7F18)
-0020 20FC7EF8 20FC5180 209CF4D8 209CF4D4 20FC7F68 20FC8018 20FC7F54 00000000 20FC7F68 | .....4Q..4M.."....."
+0000 20FC7F18 20FC7F28 20FC7F78 20FC8078 00000000 E2819497 93854084 A4949740 97999684 | ..".....Sample dump prod
+0020 20FC7F38 A4838584 4082A840 83819393 89958740 C3C5C5F3 C4D4D740 40404040 40404040 | uced by calling CEE3DMP
:
Storage around GPR15(209F0ED0)
-0020 209F0EB0 209EFD0A F2F0F0F6 F1F2F1F5 F1F0F2F5 F0F0F0F1 F0F9F0F0 0005C4F1 F9F0F800 | ...20061215102500010900..D1908.
+0000 209F0ED0 A7F4000A 00C3C5C5 000005B8 00003328 47F0F001 90ECD00C A7B50004 209F4048 | x4...CEE.....00.....x.....
+0020 209F0EF0 58BB0000 0D8041F0 00005800 B0085810 D04C1E01 5500C00C A7D40006 58F0C2BC | .....0.....<.....xM...0B.
```

[5] Information for enclave main

[6] Information for thread 20DA9FB000000000

Registers on Entry to CEE3DMP:

```
PM..... 0100
GPR0..... 20FC8078 GPR1..... 20FC7F18 GPR2..... 20FC7F78 GPR3..... 213563EA
GPR4..... 20FC7F28 GPR5..... 21356620 GPR6..... 00000002 GPR7..... 2090E2A8
GPR8..... A09C6C34 GPR9..... 20FC0C00 GPR10..... 20FC5CA7 GPR11..... A0A82198
GPR12..... 209159B0 GPR13..... 20FC7E80 GPR14..... A13564E0 GPR15..... A09F0ED0
FPR0..... 4DBFF67D 46DBE848 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
GPREG STORAGE:
Storage around GPR0 (20FC8078)
-0020 20FC8058 40404040 40404040 40404040 40404040 40404040 40404040 40404000 | .....S....."
+0000 20FC8078 00000000 00000000 00000000 00000000 2090E2D0 00000000 00000000 20FC7F28 | .....f-...."
+0020 20FC8098 00000000 00000000 00000000 00000000 00000000 20FC7E80 20FC8660 A09F1D0E | .....f-...."
```

[7] Traceback:

DSA	Entry	E	Offset	Statement	Load Mod	Program Unit	Service	Status
1	dump_n_perc		+0000012E	34	CELDLL	CELDLL	1.3.B	Call
2	CEEPGTFN		+0000005A		CEEPLPKA			Call
3	CEEHDSP		+000024BC		CEEPLPKA	CEEHDSP	D1908	Call
4	main		+000009BA	150	CELSAMP	CELSAMP	1.1.D	Exception
5	EDCZMINV		+000000C2		CEEEV003			Call
6	CEEBEXT		+000001B6		CEEPLPKA	CEEBEXT	D1908	Call

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	20FC7E80		213563B0	213563B0	+0000012E	20070105	C/C++ POSIX EBCDIC HFP
2	20FC7DC8		20A82198	20A822F8	-00000106	20061215	LIBRARY POSIX
3	20FC4CA8		209C4870	209C4870	+000024BC	20061215	CEL POSIX
4	20FC4208		20900478	20900478	+000009BA	20070105	C/C++ POSIX EBCDIC HFP
5	20FC40F0		20E629EE	20E629EE	+000000C2	20061215	LIBRARY POSIX
6	20FC4030		20992840	20992840	+000001B6	20061215	CEL POSIX

DSA	Entry	Program Unit	Load Module
1	dump_n_perc	//'POSIX.CRTL.C(CELDLL)'	CELDLL
4	main	//'POSIX.CRTL.C(CELSAMP)'	CELSAMP

Figure 7. Example dump using CEE3DMP (Part 1 of 9)

[8]Condition Information for Active Routines

```
Condition Information for //'POSIX.CRTL.C(CELSAMP)' (DSA address 20FC4208)
CIB Address: 20FC55C8
Current Condition:
CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).
Location:
Program Unit: //'POSIX.CRTL.C(CELSAMP)'
Entry:      main Statement: 150 Offset: +000009BA
Machine State:
ILC..... 0002      Interruption Code..... 0009
PSW..... 078D2400 A0900E34
GPR0..... 00000000 GPR1..... A0D9AD0C GPR2..... 20FC42C5 GPR3..... 209004B2
GPR4..... 20FC42C2 GPR5..... 20901288 GPR6..... 00000000 GPR7..... 00000001
GPR8..... 00000030 GPR9..... 80000000 GPR10.... A0E629E2 GPR11.... A0992840
GPR12.... 209159B0 GPR13.... 20FC4208 GPR14.... A0900E1E GPR15.... 00000012

Storage dump near condition, beginning at location: 20900E22
+000000 20900E22 4400C1AC 5800D0AC 41600001 8E600020 1D601807 5000D0AC 4400C1AC 58F039E2 |..A.....-...-...&.....A..0.S|
GPREG STORAGE:
Storage around GPR0 (00000000)
+0000 00000000 Inaccessible storage.
+0020 00000020 Inaccessible storage.
+0040 00000040 Inaccessible storage.
```

[9]Parameters, Registers, and Variables for Active Routines:

```
dump_n_perc (DSA address 20FC7E80):
UPSTACK DSA
Parameters:
new_cond      struct _FEEDBACK *
              0x2090E8F4
result        signed long int *
              0x20FC56B4
input_token   signed long int *
              0x20FC56A8
cond          struct _FEEDBACK *
              0x20FC55E0

Saved Registers:
GPR0..... 20FC8078 GPR1..... 20FC7F18 GPR2..... 20FC7F78 GPR3..... 213563EA
GPR4..... 20FC7F28 GPR5..... 21356620 GPR6..... 00000002 GPR7..... 2090E2A8
GPR8..... A09C6C34 GPR9..... 20FC0C00 GPR10.... 20FC5CA7 GPR11.... A0A82198
GPR12.... 209159B0 GPR13.... 20FC7E80 GPR14.... A13564E0 GPR15.... A09F0ED0

GPREG STORAGE:
Storage around GPR0 (20FC8078)
-0020 20FC8058 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404000 |.

Local Variables:
title[0..6]    unsigned char    'S'  'a'  'm'  'p'  'l'  'e'  ' '
title[7..13]   unsigned char    'd'  'u'  'm'  'p'  ' '  'p'  'r'
title[14..20]  unsigned char    'o'  'd'  'u'  'c'  'e'  'd'  ' '
title[21..27]  unsigned char    'b'  'y'  ' '  'c'  'a'  'l'  'l'
title[28..34]  unsigned char    'i'  'n'  'g'  ' '  'C'  'E'  'E'
title[35..41]  unsigned char    '3'  'D'  'M'  'P'  ' '  ' '  ' '
title[42..48]  unsigned char    ' '  ' '  ' '  ' '  ' '  ' '  ' '
title[49..55] to title[70..76] elements same as above.
title[77..79]  unsigned char    'T'  'H'  'R'  'E'  'A'  'D'  '('
options[0..6]  unsigned char    'A'  'L'  'L'  ')'  ' '  'B'  'L'
options[7..13] unsigned char    'O'  'C'  'K'  'S'  ' '  'S'  'T'
options[14..20] unsigned char    'O'  'R'  'A'  'G'  'E'  ' '  'G'
options[21..27] unsigned char    'E'  'N'  'O'  'P'  'T'  'S'  ' '
options[28..34] unsigned char    ' '  ' '  ' '  ' '  ' '  ' '  ' '
options[35..41] unsigned char    ' '  ' '  ' '  ' '  ' '  ' '  ' '
options[42..48] to options[245..251] elements same as above.
options[252..254]
fc          struct _FEEDBACK
tok_sev     signed short int    0
tok_msgno   signed short int    0
tok_case    unsigned:2          0
tok_sever   unsigned:3          0
tok_ctrl    unsigned:3          0
tok_facid[0..2] unsigned char    '\0'  '\0'  '\0'
tok_isi     signed int          0
__func__[0..6] static unsigned char    'd'  'u'  'm'  'p'  ' '  'n'  ' '
__func__[7..11] static unsigned char    'p'  'e'  'r'  'c'  '\0'  ' '  ' '

:
```

Figure 7. Example dump using CEE3DMP (Part 2 of 9)

```

main (DSA address 20FC4208):
UPSTACK DSA
Saved Registers:
  GPR0..... 00000000  GPR1..... A0D9AD0C  GPR2..... 20FC42C5  GPR3..... 209004B2
  GPR4..... 20FC42C2  GPR5..... 20901288  GPR6..... 00000000  GPR7..... 00000001
  GPR8..... 00000030  GPR9..... 80000000  GPR10..... A0E629E2  GPR11..... A0992840
  GPR12.... 209159B0  GPR13.... 20FC4208  GPR14.... A0900E1E  GPR15.... 00000012
GPREG STORAGE:
Storage around GPR0 (00000000)
+0000 00000000   Inaccessible storage.
:
:
[10]Control Blocks for Active Routines:
DSA for dump_n_perc: 20FC7E80
+000000  FLAGS.... 1099   member... 2840   BKC..... 20FC7DC8  FWC..... 20FC80A8  R14..... A13564E0
+000010  R15..... A09F0ED0  R0..... 20FC8078  R1..... 20FC7F18  R2..... 20FC7F78  R3..... 213563EA
+000024  R4..... 20FC7F28  R5..... 21356620  R6..... 00000002  R7..... 2090E2A8  R8..... A09C6C34
+000038  R9..... 20FC0C00  R10..... 20FC5CA7  R11..... A0A82198  R12..... 209159B0  reserved. 00000000
+00004C  NAB..... 20FC80A8  PNAB.... 209CF57F  reserved. 209CE580 20FC7F68
+000064  reserved. 20FC7F44  reserved. 2090E4BC  MODE.... 20FC8018  reserved. 2090E4C0
+000078  reserved. 20FC5180  reserved. 209CF4D8
DSA for CEEPGETFN: 20FC7DC8
+000000  FLAGS.... 1000   member... 0000   BKC..... 20FC4CA8  FWC..... 20FC8128  R14..... A0A821F4
+000010  R15..... 213563B0  R0..... 20FE5B40  R1..... 2090E2D0  R2..... 20FC55C8  R3..... 21366CE0
+000024  R4..... 20FC7DC8  R5..... 209C95C8  R6..... 00000000  R7..... 20FC7E8C  R8..... 00000000
+000038  R9..... A0992B28  R10..... 00000000  R11..... 209C3A88  R12..... 209159B0  reserved. 00000000
+00004C  NAB..... 20FC7E80  PNAB.... 00000000  reserved. 00000000 00000000
+000064  reserved. 00000000  reserved. 00000000  MODE.... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000
DSA for CEEHDSP: 20FC4CA8
+000000  FLAGS.... 0808   member... CEE1   BKC..... 20FC4208  FWC..... 20FC7DC8  R14..... A09C6D2E
+000010  R15..... A0A82198  R0..... 21366CE0  R1..... 2090E2D0  R2..... 20FC55C8  R3..... 2090CDB0
+000024  R4..... 209C949C  R5..... 209C95C8  R6..... 00000002  R7..... 2090E2A8  R8..... A09C6C34
+000038  R9..... 20FC6CA6  R10..... 20FC5CA7  R11..... A09C4870  R12..... 209159B0  reserved. 00000000
+00004C  NAB..... 20FC7DC8  PNAB.... 20FE663A  reserved. A0A512DA 20A5BE6C
+000064  reserved. 210F72AC  reserved. 2090EFF0  MODE.... 0000010B  reserved. 20A584C0
+000078  reserved. 20FE6630  reserved. 00000004
DSA for main: 20FC4208
+000000  FLAGS.... 1000   member... 0000   BKC..... 20FC40F0  FWC..... 20FC42E8  R14..... A0900E1E
+000010  R15..... 20DA8624  R0..... 2090155C  R1..... 20FC42A0  R2..... 20FC42C5  R3..... 209004B2
+000024  R4..... 20FC42C2  R5..... 20901288  R6..... 20FC42CC  R7..... 20FC42D0  R8..... 00000030
+000038  R9..... 80000000  R10..... A0E629E2  R11..... A0992840  R12..... 209159B0  reserved. 00000000
+00004C  NAB..... 20FC42E8  PNAB.... 00000000  reserved. 00000000 00000000
+000064  reserved. 00000000  reserved. 00000000  MODE.... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000
CIB for main: 20FC55C8
+000000  20FC55C8  C3C9C240 00000000 00000000 010C0004 00C3C5C5 F2F0F0F6 00030C89 59C3C5C5 | CIB .....CEE2006...i.CEE
+000020  20FC55E8 00000002 20FC56D8 00030C89 59C3C5C5 00000002 00000000 20FC4208 A0B09168 | .....Q...i.CEE.....j.
+000040  20FC5608 00000000 20FC4208 20900E34 2090E6F0 00000003 00000000 00000000 00000000 | .....W0.....
+000060  20FC5628 00000000 00000000 40404040 40404040 40404040 40404040 40404040 40404040 | .....
+000080  20FC5648 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 | .....
+0000A0  20FC5668 40404040 40404040 40404040 00404040 40250000 940C9000 00000009 00000000 | .....m.....
+0000C0  20FC5688 00000000 20901698 20FC4208 20FC4208 20900E32 40404040 40404040 00000001 | .....q.....
+0000E0  20FC56A8 00000002 00000003 00000002 00000014 00000003 40404040 40404040 00000000 | .....
+000100  20FC56C8 00000008 2090E908 40404040 40404040 E9D4C3C8 02000001 00000000 A0D9AD0C | .....Z. ZMCH.....R..
:
:

```

Figure 7. Example dump using CEE3DMP (Part 3 of 9)

[11]Storage for Active Routines:

```

DSA frame: 20FC7E80
+000000 20FC7E80 10992840 20FC7DC8 20FC80A8 A13564E0 A09F0ED0 20FC8078 20FC7F18 20FC7F78 |.r. .'.H...y.....".....|
+000020 20FC7EA0 213563EA 20FC7F28 21356620 00000002 2090E2A8 A09C6C34 20FC0C00 20FC5CA7 |.....".....Sy.%.....*x|
+000040 20FC7EC0 A0A82198 209159B0 00000000 20FC80A8 209CF57F 209CE580 209159B0 20FC7F68 |.y.q.j.....y..5"V..j.....|
+000060 20FC7EE0 20FC7F54 20FC7F44 2090E4BC 20FC8018 2090E4C0 20FC7F78 20FC5180 209CF4D8 |.....".....U.....U.....".....4Q|
+000080 20FC7F00 209CF4D4 20FC7F68 20FC8018 20FC7F54 00000000 20FC7F68 20FC7F28 20FC7F78 |..4M.."....."....."....."|
+0000A0 20FC7F20 20FC8078 00000000 E2819497 93854084 A4949740 97999684 A4838584 4082A840 |.....Sample dump produced by|
+0000C0 20FC7F40 83819393 89958740 C3C5C5F3 C4D4D740 40404040 40404040 40404040 40404040 |calling CEE3DMP|
+0000E0 20FC7F60 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |.....THREAD(A|
+000100 20FC7F80 D3D3D540 C2D3D6C3 D2E240E2 E3D6D9C1 C7C540C7 C5D5D6D7 E3E24040 40404040 |LL) BLOCKS STORAGE GENOPTS|
+000120 20FC7FA0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |.....S.....".....|
+000140 20FC7FC0 - +0001DF 20FC805F same as above |.....f.....|
+0001E0 20FC8060 40404040 40404040 40404040 40404040 40404040 40404000 00000000 00000000 |.....|
+000200 20FC8080 00000000 00000000 2090E2D0 00000000 00000000 20FC7F28 00000000 00000000 |.....S.....".....|
+000220 20FC80A0 00000000 00000000 00000000 20FC7E80 20FC8660 A09F1D0E A09E2838 04000000 |.....=.....f.....|
DSA frame: 20FC7DC8
+000000 20FC7DC8 10000000 20FC4CA8 20FC8128 A0A821F4 213563B0 20FE5B40 2090E2D0 20FC55C8 |.....<y.a.y.4.....$ ..S.....H|
+000020 20FC7DE8 21366CE0 20FC7DC8 209C95C8 00000000 20FC7E8C 00000000 A0992B28 00000000 |.%. 'H..nH.....=. ....r.....|
+000040 20FC7E08 209C3A88 209159B0 00000000 20FC7E80 00000000 00000000 00000000 00000000 |.h.j.....=. ....|
+000060 20FC7E28 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 20FC7E48 209C95C0 20FC7E80 20FC7E93 20FC7E8C 20FC7E7C 20FC7E78 20FC7E93 A09CE6BE |.n...=. =!...=. =@...=. =\..W..|
+0000A0 20FC7E68 209C9580 00000073 20FC5100 2090F110 20916350 20FC4030 10992840 20FC7DC8 |.V.....1..j.&.. ..r.. .'H|
DSA frame: 20FC4208
+000000 20FC4208 10000000 20FC40F0 20FC42E8 A0900E1E 20DA8624 2090155C 20FC42A0 20FC42C5 |..... 0...Y.....f.....*.....E|
+000020 20FC4228 209004B2 20FC42C2 20901288 20FC42CC 20FC42D0 00000030 80000000 A0E629E2 |.....B...h.....W.S|
+000040 20FC4248 A0992840 209159B0 00000000 20FC42E8 00000000 00000000 00000000 00000000 |.r. .j.....Y.....|
+000060 20FC4268 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 20FC4288 00000000 00000000 00000000 00000000 20FE5898 00000000 2090155C 2090154D |.....q.....*.....(|
+0000A0 20FC42A8 00000003 20901294 213667F0 00000000 2130420C 21305434 00000000 00000000 |.....m..0.....|
+0000C0 20FC42C8 00000000 00000002 21366CE0 00000000 00000000 00000000 20FC0C00 00000000 |.....%.....|
DSA frame: 20FC40F0
+000000 20FC40F0 10AF92CC 20FC4030 20FC4294 A1366D50 A0900478 20FC4208 20FC0CA0 A0E62AB0 |..k... ..m.._&.....W..|
+000020 20FC4110 00000002 A0992924 20914648 20903DA4 209046E8 00000030 80000000 A0E629E2 |.....r..j.....u...Y.....W.S|
+000040 20FC4130 A0992840 209159B0 00000000 20FC4208 20914648 00000000 00000000 00000000 |.r. .j.....j.....|
+000060 20FC4150 00000000 00000000 00100000 00000001 00000001 00000200 00000000 00000000 |.....|
+000080 20FC4170 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000A0 20FC4190 - +0000FF 20FC41EF same as above |.....|
+000100 20FC41F0 00000000 00000000 00000000 00000000 00000000 00000000 10000000 20FC40F0 |..... 0|

```

[12]Control Blocks Associated with the Thread:

```

CAA: 209159B0
+000000 209159B0 00000800 00000000 20FC4018 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 209159D0 00000000 00000000 20912A58 00000000 00000000 00000000 00000000 00000000 |.....j.....|
+000040 209159F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 20915A10 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
:
+0003C0 20915D70 00000000 00000000 A7F4FEE8 A7F401A0 A09B77A0 A0AAEC90 A0A079F8 A0A79F20 |.....x4.Yx4......8.x..|
+0003E0 20915D90 20BC5B4C 21366C20 00000000 00000000 00000000 00000000 20FC7B60 20916118 |.$.<..%.....#..-j/..|
Thread Synchronization Queue Element (SQEL): 2090EFF0
+000000 2090EFF0 00000000 00000000 00000000 00000000 20FE5920 00000008 210F79E0 00000000 |.....|
+000020 2090F010 209159B0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.j.....|
CEEDLLF: 21366C20
+000000 EYE..... CEEDLLF VERSION.. 01 FLAGS.... 00 SIZE.... 0060 SERVICE.. 04
+000000 REFTYPE... 02 LOADTYPE.. 00 reserved.. 00 padding.. 00000000 PREV.... 21366BC0
+000018 padding.. 00000000 NEXT..... 213668C0 FBTKO A.. 00000DF6 FBTKO B.. 41C3C5C5 FBTKO C.. 00000000
+00002C padding.. 00000000 padding.. 00000000 DLLNAME.. 21366C88 padding.. 00000000 SYMNAME.. 21366C98
+000040 DLLMLEN.. 00000006 SYMMLEN.. 0000000F RETCODE1.. 00000000 RSNCODE1.. 00000000 RETCODE2.. 00000000
+000054 RSNCODE2.. 00000000 reserved.. 00000000 reserved.. 00000000
CEEDLLF_DLL_NAME(21366C88)
+0000 21366C88 C3C5D3C4 D3D30000 21365000 00000018 |CELDLL...&.....|
CEEDLLF_SYMBOL_NAME(21366C98)
+0000 21366C98 95819485 6D9596A3 6D89956D 84939300 |name_not_in_dll..|
DUMMY DSA: 20916350
+000000 FLAGS.... 0000 member... 0000 BKC..... 00006F58 FWC..... 20FC4030 R14..... A09048D6
+000010 R15..... A0992840 R0..... 7D000009 R1..... 20FC0CA0 R2..... 00000000 R3..... 00000000
+000024 R4..... 00000000 R5..... 00000000 R6..... 00000000 R7..... A0917778 R8..... 20903DA0
+000038 R9..... 008DCCD0 R10..... 00000000 R11..... A0904802 R12..... 209159B0 reserved.. 00000000
+00004C NAB..... 20FC4030 PNAB..... 20FC4030 reserved.. 00000000 00000000
+000064 reserved.. 00000000 reserved.. 00000000 MODE..... 00000000 reserved.. 00000000
+000078 reserved.. 00000000 reserved.. 00000000
:

```

Figure 7. Example dump using CEE3DMP (Part 4 of 9)

```

[6]Information for thread 20DAEB000000001
[7]Traceback:
DSA  Entry      E Offset  Statement  Load Mod      Program Unit      Service  Status
  1  CEEOPML2    +00353C2C  CEEPLPKA   CEEPLPKA      CEEOPML2        D1908   Call
  2  EDCOWRP2    +00000F1E  CEEV003    CEEV003        CEEOPML2        D1908   Call
  3  thread_func +000000AE  47         CELSAMP        CELSAMP          1.1.D    Call
  4  CEEOPCMM    +00000908  CEEBINIT   CEEBINIT       CEEOPCMM        D1908   Call

DSA  DSA Addr  E Addr  PU Addr  PU Offset  Comp Date  Compile Attributes
  1  21312F50  20A4EEA0 20A4EEA0 +00353C2C 20061215  CEL    POSIX
  2  21312CC8  20DA1BAC 20DA0908 +000021C2 20061215  LIBRARY  POSIX
  3  21312C20  20901140 20901140 +000000AE 20070105  C/C++    POSIX EBCDIC HFP
  4  21359EE0  0000E460 0000E460 +00000908 20061215  CEL    POSIX

Fully Qualified Names
DSA  Entry      Program Unit      Load Module
  3  thread_func // 'POSIX.CRTL.C(CELSAMP)'  CELSAMP

[9]Parameters, Registers, and Variables for Active Routines:
CEEOPML2 (DSA address 21312F50):
UPSTACK DSA
Saved Registers:
GPR0..... 00000001 GPR1..... 21310D78 GPR2..... 210F70E4 GPR3..... 21310D78
GPR4..... DECEF288 GPR5..... 00000000 GPR6..... 20FC0C58 GPR7..... 20FE5920
:
thread_func (DSA address 21312C20):
UPSTACK DSA
Parameters:
parm          void *          0x20901288
Saved Registers:
GPR0..... 00000001 GPR1..... 21310D78 GPR2..... 210F70E4 GPR3..... 21310D78
GPR4..... DECEF288 GPR5..... 00000000 GPR6..... 20FC0C58 GPR7..... 20FE5920
:

[10]Control Blocks for Active Routines:
DSA for CEEOPML2: 21312F50
+000000  FLAGS.... 0000      member... 0001      BKC..... 21312CC8  FWC..... 213130D0  R14..... A0A4FC58
+000010  R15..... A0A50EE0  R0..... 00000000  R1..... 21312FD4  R2..... 210F70E4  R3..... 21310D78
+000024  R4..... 20FE5944  R5..... 20A5022C  R6..... 20FC0C58  R7..... 20FE5920  R8..... 20901288
:

[11]Storage for Active Routines:
DSA frame: 21312CC8
+000000 21312CC8 10914B58 21312C20 21312F50 A0DA2ACC A0A4EEA0 00000000 21312CB8 20FC0C58 |.j.....&.....u.....|
+000020 21312CE8 2090117A 21312C20 20901288 21312798 20FEE128 21359FEC 00000080 8000F45F |.....h...q.....4^|
:

[12]Control Blocks Associated with the Thread:
CAA: 21311BD8
+000000 21311BD8 00000800 00000000 21312C08 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 21311BF8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
:

[13]Enclave variables:
*.*.C(CELSAMP)':>mut
struct
__m          unsigned long int          553539872
*.*.C(CELSAMP)':>thread[0]
struct
__[0..6]     unsigned char          '\x20' '\xDA' '\xEB' '\0' '\0' '\0' '\0'
__[7]        unsigned char          '\x01'
*.*.C(CELSAMP)':>thread[1]
struct
__[0..6]     unsigned char          '\x20' '\xDB' '\xEB' '\x10' '\0' '\0' '\0'
__[7]        unsigned char          '\x02'
*.*.C(CELSAMP)':>threads_joined
signed int          0
*.*.C(CELSAMP)':>t1
unsigned char *     0x20901288
*.*.C(CELSAMP)':>t2
unsigned char *     0x20901294
*.*.C(CELSAMP)':>main
signed int (void)   0x20900478
*.*.C(CELSAMP)':>thread_func
void * ()           0x20901140
*.*.C(CELSAMP)':>thread_cleanup
void ()            0x20900EE0
*.*.C(CELDLL)':>wsa_array[0..6] unsigned char          'C' 'E' 'L' 'D' 'L' 'L' ' '
*.*.C(CELDLL)':>wsa_array[7..9] unsigned char          'W' 'S' 'A'
*.*.C(CELDLL)':>dump_n_perc
void ()            0x213563B0

```

Figure 7. Example dump using CEE3DMP (Part 5 of 9)

[14] Enclave Control Blocks:

```

EDB: 20914648
+000000 20914648 C3C5C5C5 C4C24040 C7000001 20915870 20914D50 00000000 00000000 00000000 | CEEEDB G...j...j(&.....
+000020 20914668 20914B58 20914B88 A0917778 20914198 00000000 00000000 20914768 00000000 | .j...j.h.j...j.q.....J.....
+000040 20914688 00000000 00000000 00006F58 00000000 00000000 A0A34190 2090C880 20FC0C48 | .....?.....t.....H.....
+000060 209146A8 0000F460 210F7000 20913E58 00000000 20916550 00000000 20AF6660 209159B0 | ..4-...j.....j.&.....-j..
+000080 209146C8 90000000 0000FAF6 00000000 00000000 00000002 00000000 00006FF0 008FF4E8 | .....6.....?0..4Y
+0000A0 209146E8 00000001 00000100 2090C988 209146F0 00000000 00000000 00000000 00000002 | .....Ih.j.0.....

MEML: 20915870
+000000 20915870 00000000 00000000 20994BE8 00000000 00000000 00000000 20994BE8 00000000 | .....r.Y.....r.Y.....
+000020 20915890 00000000 00000000 20994BE8 00000000 209118D8 00000000 A0B09168 00000000 | .....r.Y.....j.Q.....j.....
+000040 209158B0 00000000 00000000 20994BE8 00000000 00000000 00000000 20994BE8 00000000 | .....r.Y.....r.Y.....
+000060 209158D0 - +00011F 2091598F same as above

Mutex and Condition Variable Blocks (MCVB+MHT+CHT): 210F7018
+000000 210F7018 00008F50 210F7044 000003F8 00001FC0 00000000 20FC3088 210F7444 000000F8 | ..&.....8.....h.....8
+000020 210F7038 000007C0 00000000 20FC30A0 00000000 00000000 00000000 00000000 00000000 | .....
+000040 210F7058 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....

:
+000460 210F7478 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000480 210F7498 - +00053F 210F7557 same as above

Thread Synchronization Enclave Latch Table (EPALT): 210F7544
+000000 210F7544 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000020 210F7564 - +00009F 210F75E3 same as above
+0000A0 210F75E4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....?.....0...&

:
+000640 210F7B84 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000660 210F7BA4 - +0009FF 210F7F43 same as above

DLL Information:
WSA Addr Module Addr Thread ID Use Count Name
20FE5B40 213562F0 20DA9FB000000000 00000001 CELDLL
HEAPCHK Option Control Block (HCOP): 20FC00D0
+000000 20FC00D0 C8C3D6D7 00000038 00000001 00000000 00000000 0000000A 0000000A 00000000 | HCOP.....
+000020 20FC00F0 2138B028 20FC0108 21102028 00000000 00000000 00000000 C8C3C6E3 00000200 | .....HCFT.....

HEAPCHK Element Table (HCEL) for Heapid 21384F9C :
Header: 2138B028
+000000 2138B028 C8C3C5D3 21368028 00000000 21384F9C 000001F4 00000006 00000006 00000000 | HCEL.....|.....4.....
Address Seg Addr Length Address Seg Addr Length

Table: 2138B048
+000000 2138B048 2138A020 2138A000 00000050 2136D760 2138A070 2138A000 00000028 2136D888 | .....&..P-.....Qh
+000020 2138B068 2138A098 2138A000 00000108 2136D9B0 2138A1A0 2138A000 00000050 2136DAD8 | ..q.....R.....&..Q
+000040 2138B088 2138A1F0 2138A000 00000028 2136DC00 2138D020 2138D000 00000408 2136DD28 | ..0.....

HEAPCHK Element Table (HCEL) for Heapid 21366FC4 :
Header: 21368028
+000000 21368028 C8C3C5D3 20FC0CE0 2138B028 21366FC4 000001F4 00000001 00000001 00000000 | HCEL.....?D...4.....
Address Seg Addr Length Address Seg Addr Length

Table: 21368048
+000000 21368048 21367020 21367000 000002A8 20FC3310 00000000 00000000 00000000 00000000 | .....y.....

HEAPCHK Element Table (HCEL) for Heapid 00000000 :
Header: 20FC0CE0
+000000 20FC0CE0 C8C3C5D3 00000000 21368028 00000000 000001F4 00000005 00000005 00000000 | HCEL.....4.....
Address Seg Addr Length Address Seg Addr Length

Table: 20FC0D00
+000000 20FC0D00 20FE4038 20FE4018 00001028 20FC2C48 20FE5060 20FE4018 00000828 20FC2D10 | .....&-.....
+000020 20FC0D20 20FE5888 20FE4018 00000CC8 20FC2DE8 20FE6550 20FE4018 00002028 20FC2F18 | ..h.....H...Y...&.....
+000040 20FC0D40 20FE8578 20FE4018 00000CC8 21366EA8 00000000 00000000 00000000 00000000 | ..e.....H...y.....

WSA address.....20FC0C00
Heap Storage Diagnostics
Stg Addr ID Length Entry E Addr E Offset Load Mod
2138A020 21384F9C 00000050
CEEV#GTS 20AB53F8 +00000000 CEEPLPKA
CEEVGTST 20AC65E0 +00000072 CEEPLPKA
identify_cu 20D6F778 +0000055A CEEEV003
SetDumpVars 20D79000 +00000204 CEEEV003
_zdmpd 20E4B9C0 +000002E4 CEEEV003
CEEKMDR 209E2838 +00003D68 CEEPLPKA
CEEKDUMP 209F0ED0 +00000E3C CEEPLPKA
dump_n_perc 213563B0 +0000012E CELDLL
CEEPGTFN 20A82198 +0000005A CEEPLPKA

2138A070 21384F9C 00000028
CEEV#GTS 20AB53F8 +00000000 CEEPLPKA
CEEVGTST 20AC65E0 +00000072 CEEPLPKA
identify_cu 20D6F778 +00000686 CEEEV003
SetDumpVars 20D79000 +00000204 CEEEV003
_zdmpd 20E4B9C0 +000002E4 CEEEV003
CEEKMDR 209E2838 +00003D68 CEEPLPKA
CEEKDUMP 209F0ED0 +00000E3C CEEPLPKA
dump_n_perc 213563B0 +0000012E CELDLL
CEEPGTFN 20A82198 +0000005A CEEPLPKA

```

Figure 7. Example dump using CEE3DMP (Part 6 of 9)

```

:
:
:
20FE8578 00000000 00000CC8          20AB53F8 +00000000 CEEPLPKA
:          CEEV#GTS 20AB53C8 +0001EF84 CEEPLPKA
:          CEEVGQT 20AC3008 +0000026A CEEPLPKA
:          flocks 20D8C398 +00000228 CEEEV003
:          _fopen 20C50CE0 +000001F2 CEEEV003
:          fopen 20C50F50 +0000006C CEEEV003
:          EDCOWRP3 20DA860C +0000121A CEEEV003
:          main 20900478 +000007BC CELSAMP
:          EDCZMINV 20E629EE -FFAB3B58 CEEEV003
:          CEEBBEXT 20992840 +000001B6 CEEPLPKA
:
Language Environment Trace Table:
Most recent trace entry is at displacement: 002780
Displacement          Trace Entry in Hexadecimal          Trace Entry in EBCDIC
-----
+000000 Time 04.22.34.474835 Date 2007.01.06 Thread ID... 20DA9FB000000000
+000010 Member ID... 03 Flags..... 000000 Entry Type..... 00000001
+000018 94818995 40404040 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
+000078 40404040 40404040

+000080 Time 04.22.34.478351 Date 2007.01.06 Thread ID... 20DA9FB000000000
+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

+002700 Time 04.22.34.494829 Date 2007.01.06 Thread ID... 20DA9FB000000000
+002710 Member ID... 03 Flags..... 000000 Entry Type..... 00000002
+002718 4C60604D F0F8F55D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(085) R15=00000000 ERRNO=0000
+002738 F0F0F0F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |0000.....
+002758 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+002778 00000000 00000000

+002780 Time 04.22.34.497209 Date 2007.01.06 Thread ID... 20DBEB1000000002
+002790 Member ID... 01 Flags..... 000000 Entry Type..... 00001910
+002798 20DBEB10 00000002 21335798 21335740 80000000 00000000 00000000 00000000 |.....q.....
+0027B8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0027D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0027F8 00000000 00000000

[15]Enclave Storage:
Initial (User) Heap          : 20FE4018
+000000 20FE4018 C8C1D5C3 20914B28 20914B28 00000000 A0FE4018 20FE9240 00008000 00002DD8 |HANC.j...j.....k.....Q
+000020 20FE4038 20FE4018 00001028 00000005 00000005 20FE5068 20FE5250 20FE528D 20FE52CA |.....&.....&.....
+000040 20FE4058 20FE5307 20FE5344 20FE5381 20FE53BE 20FE53FB 20FE5808 20FE5845 00000000 |.....a.....
+000060 20FE4078 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+000080 20FE4098 00000000 00000000 00000000 00000000 00000000 00000000 00000000 20FE5438 |.....
+0000A0 20FE40B8 20FE5475 20FE54B2 20FE54EF 20FE552C 20FE5569 20FE55A6 20FE55E3 00000000 |.....w...T.....

+004560 20FE8578 20FE4018 00000CC8 00000001 00000001 20FE5BC8 00000000 00000001 00000001 |.....H.....$.H.....
+004580 20FE8598 20FE5C50 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |...*&.....
+0045A0 20FE85B8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0045C0 20FE85D8 - +007FFF 20FEC017          same as above
LE/370 Anywhere Heap          : 20FC0000
+000000 20FC0000 C8C1D5C3 20FED000 20914B58 20914B58 A0FC0000 20FC3500 00004000 00000B00 |HANC.....j...j.....
+000020 20FC0020 20FC0000 00000018 D2C4C240 00000000 00000000 00000000 20FC0000 00000090 |.....KDB.....
+000040 20FC0040 E3E5C240 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |TVB.....

+0034E0 20FC34E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+003500 20FC3500 - +003FFF 20FC3FFF          same as above
LE/370 Anywhere Heap          : 20FED000
+000000 20FED000 C8C1D5C3 210F9000 20FC0000 20914B58 20FED000 00000000 00100028 00000000 |HANC.....j.....
+000020 20FED020 20FED000 00100008 BFF67D46 D755378E 20DA9FB0 00000000 03000000 00000001 |.....6'.P.....
+000040 20FED040 94818995 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |main

Additional Heap, heapid = 21384F9C          : 2138A000
+000000 2138A000 C8C1D5C3 2138D000 21384F9C 21384F9C 2138A000 2138A218 000003E8 000001D0 |HANC.....|...|.....s...Y.....
+000020 2138A020 2138A000 00000050 00000000 21356E38 213563B0 2138A0A0 20912178 46F11000 |.....&.....>.....j...l...
+000040 2138A040 00000003 00000000 00040000 2138A078 00000000 21356390 21356390 00000000 |.....
+000060 2138A060 213567BC 21356390 00000000 00000000 2138A000 00000028 00186161 7DD7D6E2 |.....//'.POS
:
:
:

```

Figure 7. Example dump using CEE3DMP (Part 7 of 9)

[16]File Status and Attributes:

..

[17]Run-Time Options Report:

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	CEEDUMP (60,SYSOUT=*,FREE=END,SPIN=UNALLOC)
Installation default	CHECK (ON)
Installation default	COUNTRY (US)
Installation default	NODEBUG
Installation default	DEPTHCONDLMT (10)
DD:CEEOPPTS	DYNDUMP (POSIX.HEALY.ZOS19,DYNAMIC,TDUMP)
Installation default	ENVAR ("")
Installation default	ERRCOUNT (0)
Installation default	ERRUNIT (6)
Installation default	FILEHIST
Installation default	FILETAG (NOAUTOCVT,NOAUTOTAG)
Default setting	NOFLOW
Installation default	HEAP (32768,32768,ANYWHERE,KEEP,8192,4096)
Programmer default	HEAPCHK (ON,1,0,10,10)
Programmer default	HEAPPOLS (ON,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	INFOMSGFILTER (OFF,,,))
Installation default	INQPCPN
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)
Programmer default	POSIX (ON)
Installation default	PROFILE (OFF, "")
Installation default	PRTUNIT (6)
Installation default	PUNUNIT (7)
Installation default	RDRUNIT (5)
Installation default	RECPAD (OFF)
Installation default	RPTOPTS (OFF)
Programmer default	RPTSTG (ON)
Installation default	NORTEREUS
Installation default	NOSIMVRD
Installation default	STACK (131072,131072,ANYWHERE,KEEP,524288,131072)
Installation default	STORAGE (NONE,NONE,NONE,0)
Programmer default	TERMTHDACT (UADUMP,,96)
Installation default	NOTEST (ALL,"*","PROMPT","INSPREF")
Installation default	THREADHEAP (4096,4096,ANYWHERE,KEEP)
Installation default	THREADSTACK (OFF,4096,4096,ANYWHERE,KEEP,131072,131072)
Programmer default	TRACE (ON,1048576,NODUMP,LE=1)
Installation default	TRAP (ON,SPIE)
Installation default	UPSI (00000000)
Installation default	NOUSRHDLR (,)
Installation default	VCTRSAVE (OFF)
Installation default	XPLINK (OFF)
Installation default	XUFLOW (AUTO)

Figure 7. Example dump using CEE3DMP (Part 8 of 9)

[18]Process Control Blocks:

```
PCB: 20914198
+000000 20914198 C3C5C5D7 C3C24040 03030288 00000000 00000000 00000000 209143D0 A0AF94D0 |CEEPCB ...h.....j....m|
+000020 209141B8 A0AEFB50 A0AF6A08 A0AF6528 2090AAE8 20913F68 00000000 00000000 20914648 |...&.....Y.j.....j..|
+000040 209141D8 A0AF6858 7F800000 00000000 000141D4 00000000 80000000 A0A5C1E0 00000000 |..."......M.....vA....|

MEML: 209143D0
+000000 209143D0 00000000 00000000 20994BE8 00000000 00000000 00000000 20994BE8 00000000 |.....r.Y.....r.Y....|
+000020 209143F0 00000000 00000000 20994BE8 00000000 2090FE68 00000000 A0B09168 20FC0000 |.....r.Y.....j.....|
+000040 20914410 00000000 00000000 20994BE8 00000000 00000000 00000000 20994BE8 00000000 |.....r.Y.....r.Y....|
+000060 20914430 - +00011F 209144EF same as above

Thread Synchronization Process Latch Table (PPALT): 210F7F44
+000000 210F7F44 DF6F1010 2090EFF0 210F77EC 00000000 00000000 00000000 00000000 00000000 |.?.....0.....|
+000020 210F7F64 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 210F7F84 00000000 00000000 00000000 00000000 DF6F1010 2090EFF0 210F7A44 00000000 |.....?.0.....|
+000060 210F7FA4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 210F7FC4 - +0009FF 210F8943 same as above
```

[19]Additional Language Specific Information:

```
errno information :
Thread Id .... 20DA9FB000000000 Errno ..... 0 Errnojr .... 00000000
[20]CEE3846I CEEDUMP Processing completed.
```

Figure 7. Example dump using CEE3DMP (Part 9 of 9)

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 188.
- For COBOL routines, see “Finding COBOL information in a dump” on page 237.
- For Fortran routines, see “Finding Fortran information in a Language Environment dump” on page 263.
- For PL/I routines, see “Finding PL/I for MVS & VM information in a dump” on page 283.

[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

The contents of the second line of the page heading vary depending on the environment in which the CEEDUMP is issued.

For CEEDUMPs produced under a batch environment, the following items are displayed:

- ASID: Describes the address space ID.
- Job ID: Describes the JES Job ID.
- Job name: Describes the job name.
- Step name: Describes the job's step name in which the CEEDUMP was produced.
- UserID: Describes the TSO userid who issued the job.

For jobs running with POSIX(ON), the following additional items are displayed:

- PID: Displays the associated process ID.
- Parent PID: Displays the associated parent PID.

For CEEDUMPs produced under the z/OS UNIX shell, the following items are displayed:

- ASID: Describes the address space ID.
- PID: Displays the associated process ID.
- Parent PID: Displays the associated parent PID.
- User name: Contains the userid associated to the CEEDUMP.

For CEEDUMPs produced under CICS, the following items are displayed:

- Transaction ID and task number.

[2] CEE3845I CEEDUMP Processing started.

Message CEE3845I identifies the start of the Language Environment dump processing. Similarly, message CEE3846I identifies the end of the dump processing. Message number CEE3845I can be used to locate the start of the next CEEDUMP report when scanning forward in a data set that contains several CEEDUMP reports.

[3] Caller Program Unit and Offset

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

[4] 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 through 15

- 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.

[5] - [17] Enclave Information

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

[5] 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 78.

[6] - [12] Thread Information

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

[6] Information for thread

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

[7] Traceback

In a multithread case, the traceback reflects only the current thread. For all active routines, the traceback section shows routine information in three parts. The first part contains:

- DSA number
A number assigned to the information for this active routine by dump processing. The number is used to associate information from the first part of the traceback with information in the second and third parts of the traceback.
- Entry
For COBOL, Fortran, PL/I, and Enterprise PL/I for z/OS 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, the string ‘** NoName **’ will appear.
- Entry point offset
- Statement number
Refers to the line number in the source code (program unit) in which a call was made or an exception took place (see Status column). The statement number appears only if your routine was compiled with the options required to generate statement numbers.
- Load module
The load module name displayed can be a partitioned data set member or an UNIX executable file. The load module name is also displayed in the third part of the traceback.
- Program unit

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

If the program unit name is available to Language Environment (for example, for C/C++, the routine was compiled with TEST(SYM)), the program unit name will appear under this column, according to the following rules:

- If your compiled routine is in a partitioned data set, only the member will be output.
 - If your compiled routine is in a sequential data set, only the last qualifier will be shown.
 - If your compiled routine is in an UNIX filename, only what fits of the filename will be displayed in a line.
- 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 used to associate information from the first part of the traceback with information in the second and third parts 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 could have been called using SVC-assisted linkage. Adding the program unit address to the offset gives the location of the current instruction in the routine. This offset is from the starting address of the routine.
- Compile Date
Contains the year, month and day in which the routine was compiled.
- Attributes
The available compilation attributes of the compile unit including:
 - A label identifying the LE-supported language such as COBOL, ENT PL/I, C/C++, and so on.
 - Compilation attributes such as EBCDIC, ASCII, IEEE or hexadecimal floating point (HFP). The compilation attributes will only be displayed if there is enough information available.
 - If the CEEDUMP was created under a POSIX environment, POSIX will be displayed.

The third part of the traceback, which is also referred to as 'Fully Qualified Names' section, contains the following:

- DSA number
- Entry
- Program unit

Similar to the Program Unit column in part 1 except that the server name and the complete program unit (PU) name will be displayed. A PU name will appear here only if it is available to Language Environment.

- Load Module

The complete pathname of a load module name residing in an UNIX filename will be displayed here if available. The load module's full pathname will be displayed if the PATH environment variable is set such that the pathname of the load module's directory appears before the current directory (.). For load modules found in data sets, the same output shown in the traceback part 1 will also be displayed here.

[8] 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)
- Storage pointed to by General Purpose Registers

These values are the current values at the time the condition was raised.

[9] 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.

[10] 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

[11] 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.

[12] Control Blocks Associated with the Thread

This section lists the contents of the Language Environment common anchor area (CAA), thread synchronization queue element (SQEL), DLL failure data, and dummy stack frame. Other language-specific control blocks can appear in this section. DLL failure data is described in “Using the DLL failure control block” on page 78.

[13] Enclave variables:

This section displays language specific global variables. This section also shows the variable type. Variables are displayed only if the symbol tables are available.

[14] 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.

[15] 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.

[17] Run-Time Options Report

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

[18] 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.

[19] Additional Language Specific Information:

This section displays any additional information not included in other sections. For C/C++, it shows the thread id of the thread that generated the dump and the settings of the errno and errnojr variables for that thread.

[20] CEE3846I CEEDUMP Processing completed.

Message CEE3846I identifies the end of the Language Environment dump processing. Similarly, message CEE3845I identifies the start of the dump processing. Message number CEE3846I can be used to locate the end of the previous CEEDUMP report when scanning backward in a data set that contains several CEEDUMP reports.

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 64 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 65 shows the format of the downward-growing stack frame.

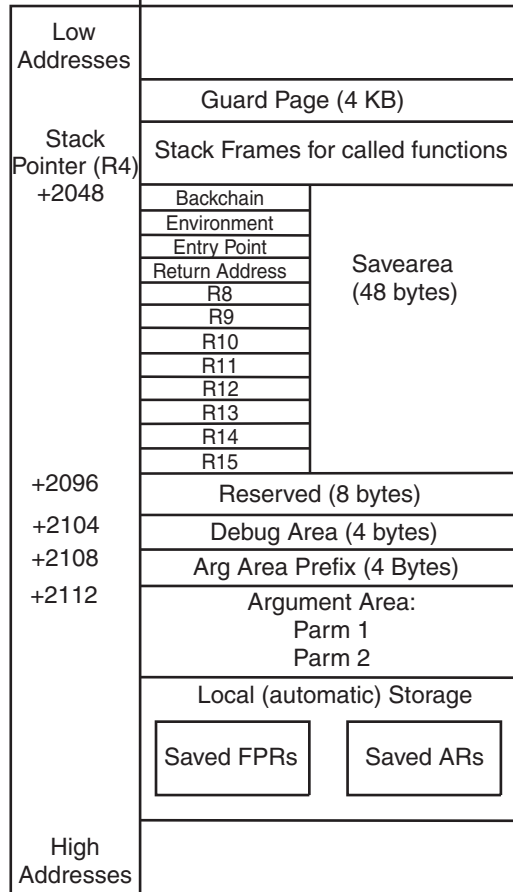


Figure 9. Downward-growing (XPLINK) stack frame format

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 66 shows the format of the Language Environment CAA.

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_DCRENT - 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	
~ ~		
0003E4	CEECAA_CEEDLLF - DLL Failure	
0003E8	CEECAA_SAVSTACK - Zero or saved stack pointer	
0003F4	CEECAA_SAVSTACK_ASYNC - Zero or address of field that is zero or saved stack pointer	

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
CEECAA_CEEDLLF	Address of the newest CEEDLLF control block.

Table 7. List of CAA fields (continued)

CAA Field	Explanation								
CEECAAFLAG0	<p>CAA flag bits. The bits are defined as follows:</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </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								
CEECAALANGP	<p>PL/I language compatibility flags external to Language Environment. The bits are defined as follows:</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </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	<p>Start of the current storage segment.</p> <p>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.</p>								
CEECAAEOS	<p>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.</p>								
CEECAATORC	<p>Thread level return code. The thread level return code set by CEESRC callable service.</p>								
CEECAATOVF	<p>Address of stack overflow routine.</p>								
CEECAAATTN	<p>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.</p>								
CEECAAHLEXIT	<p>Address of the Exit List Control Block set by the HLL user exit CEEBINT.</p>								
CEECAAHOOKS	<p>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.</p>								

Table 7. List of CAA fields (continued)

CAA Field	Explanation
CEECAASYSTM	<p>Underlying operating system. The value indicates the operating system supporting the active environment.</p> <p>Value Operating System</p> <p>0 Undefined. This value should not appear after Language Environment is initialized.</p> <p>1 Unsupported</p> <p>3 z/OS</p>
CEECAHRDWR	<p>Underlying hardware. This value indicates the type of hardware on which the routine is running.</p> <p>Value Hardware</p> <p>0 Undefined. This value should not appear after Language Environment is initialized.</p> <p>1 Unsupported</p> <p>2 System/370™, non-XA</p> <p>3 System/370, XA</p> <p>4 System/370, ESA</p>
CEECAASBSYS	<p>Underlying subsystem. This value indicates the subsystem (if any) on which the routine is running.</p> <p>Value Subsystem</p> <p>0 Undefined. This value should not occur after Language Environment is initialized.</p> <p>1 Unsupported</p> <p>2 None. The routine is not running under a Language Environment-recognized subsystem.</p> <p>3 TSO</p> <p>4 IMS™</p> <p>5 CICS</p>
CEECAAF2	<p>CAA Flag 2.</p> <p>Bit Description</p> <p>0 Bimodal addressing is available.</p> <p>1 Vector hardware is available.</p> <p>2 Thread terminating.</p> <p>3 Initial thread</p> <p>4 Library trace is active. The TRACE run-time option was set.</p> <p>5 Reserved</p> <p>6 CEECAA_ENQ_Wait_Interruptible. Thread is in an enqueue wait.</p> <p>7 Reserved</p>

Table 7. List of CAA fields (continued)

CAA Field	Explanation
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.
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.
CEECAAGETSX	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.

Table 7. List of CAA fields (continued)

CAA Field	Explanation								
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.								
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.								
CEECAAF1AG1	CAA flag bits. The bits are defined as follows: <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Bit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>CEECAASORT. A call to DFSORT™ is active.</td> </tr> <tr> <td>1</td> <td>CEECAA_USE_OLD_STK. Use the old stack.</td> </tr> <tr> <td>2–7</td> <td>Reserved</td> </tr> </tbody> </table>	Bit	Description	0	CEECAASORT. A call to DFSORT™ is active.	1	CEECAA_USE_OLD_STK. Use the old stack.	2–7	Reserved
Bit	Description								
0	CEECAASORT. A call to DFSORT™ is active.								
1	CEECAA_USE_OLD_STK. Use the old stack.								
2–7	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.								

Table 7. List of CAA fields (continued)

CAA Field	Explanation
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.
CEECAAOGETS	Overflow from user stack allocations.
CEECAAOGETLS	Overflow from library stack allocations.
CEECAARSRV1	Reserved.
CEECAAPICIB	Address of the preinitialization compatibility control block.
CEECAAOGETSX	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.
CEECAA_SIGSCTR	SIGSAFE counter.

Table 7. List of CAA fields (continued)

CAA Field	Explanation																										
CEECAA_SIGSFLG	<p>SIGSAFE flags. SIGSAFE flags indicate the signal safety of the library.</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>Description</th> </tr> </thead> <tbody> <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> <tr> <td>5</td> <td>CEECAA_SIGRESYNCH. CEECAA_sigputsynch flag was on last time CEEOSIGR resolicited a signal.</td> </tr> <tr> <td>6</td> <td>CEECAA_FRZ_UNSAFE. This thread is in an unsafe state to be frozen.</td> </tr> <tr> <td>7</td> <td>CEECAA_NOAPPREGS. User application registers may be saved in a nonstandard place.</td> </tr> <tr> <td>8</td> <td>CEECAA_EINTR_RSOL. Secondary Signal resolicitation is in progress, after EINTR errno from inner function.</td> </tr> <tr> <td>9</td> <td>CEECAA_EINTR_PUTB. Secondary resolicited signal has been put back.</td> </tr> <tr> <td>10</td> <td>CEECAA_EINTR_REST. User signal catcher returned after catching secondary resolicited signal with SA_RESTART in effect.</td> </tr> <tr> <td>11</td> <td>CEECAA_EINTR_SIGG. Stray signal interrupted CEEOSIGG while secondary signal resolicitation was in progress.</td> </tr> </tbody> </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.	5	CEECAA_SIGRESYNCH. CEECAA_sigputsynch flag was on last time CEEOSIGR resolicited a signal.	6	CEECAA_FRZ_UNSAFE. This thread is in an unsafe state to be frozen.	7	CEECAA_NOAPPREGS. User application registers may be saved in a nonstandard place.	8	CEECAA_EINTR_RSOL. Secondary Signal resolicitation is in progress, after EINTR errno from inner function.	9	CEECAA_EINTR_PUTB. Secondary resolicited signal has been put back.	10	CEECAA_EINTR_REST. User signal catcher returned after catching secondary resolicited signal with SA_RESTART in effect.	11	CEECAA_EINTR_SIGG. Stray signal interrupted CEEOSIGG while secondary signal resolicitation was in progress.
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.																										
5	CEECAA_SIGRESYNCH. CEECAA_sigputsynch flag was on last time CEEOSIGR resolicited a signal.																										
6	CEECAA_FRZ_UNSAFE. This thread is in an unsafe state to be frozen.																										
7	CEECAA_NOAPPREGS. User application registers may be saved in a nonstandard place.																										
8	CEECAA_EINTR_RSOL. Secondary Signal resolicitation is in progress, after EINTR errno from inner function.																										
9	CEECAA_EINTR_PUTB. Secondary resolicited signal has been put back.																										
10	CEECAA_EINTR_REST. User signal catcher returned after catching secondary resolicited signal with SA_RESTART in effect.																										
11	CEECAA_EINTR_SIGG. Stray signal interrupted CEEOSIGG while secondary signal resolicitation was in progress.																										
CEECAATHDID	Thread id. This field is the thread identifier.																										
CEECAA_DCRENT	DCE's read/write static external anchor.																										
CEECAA_DANCHOR	DCE's per-thread anchor.																										
CEECAA_CTOC	TOC anchor for CRENT.																										
CEECAACICRSN	CICS reason code from member language.																										
CEECAAMEMBR	Address of thread-level member list.																										
CEECAA_SIGNAL_STATUS	Signal status of the terminating thread member list.																										
CEECAA_SAVSTACK	Zero or saved stack pointer. This field can be used to save the stack pointer before calling a routine with OS_NOSTACK linkage. After the call returns, this field must be set back to zero.																										

|
|
|
|

Table 7. List of CAA fields (continued)

CAA Field	Explanation
CEECAA_SAVSTACK_ASYNC	Zero or address of field that is zero or saved stack pointer. An application that has large sections of code that do not require access to the Language Environment stack but could benefit from having an additional register available can use this field.

For information about the DLL failure control block, CEEDLLF, see *z/OS Language Environment Vendor Interfaces*.

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 75 shows the condition information block.

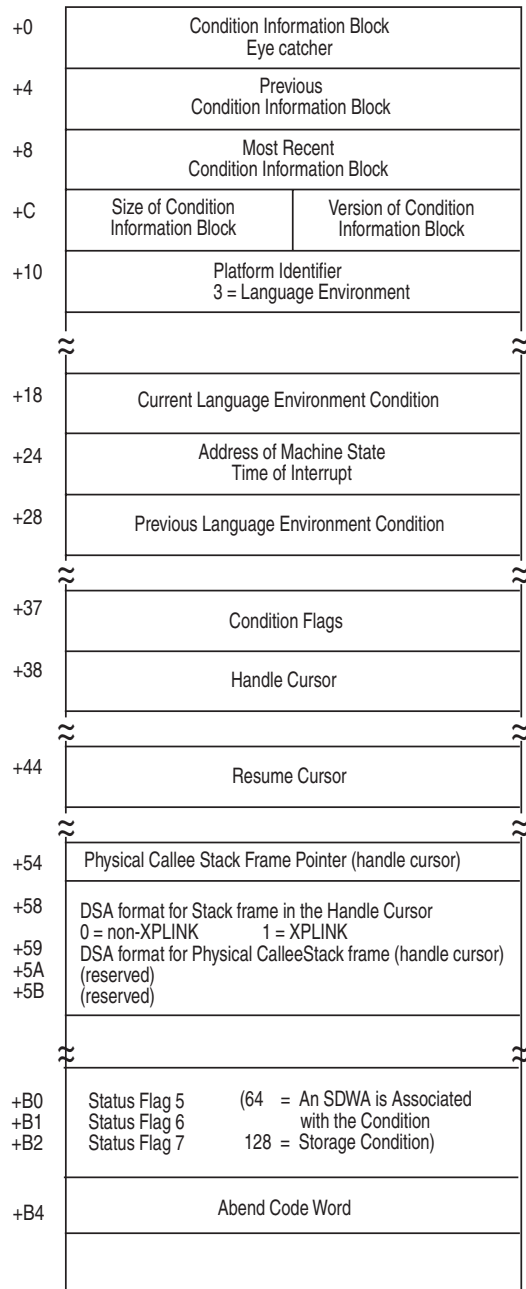


Figure 11. Condition information block (Part 1 of 2)

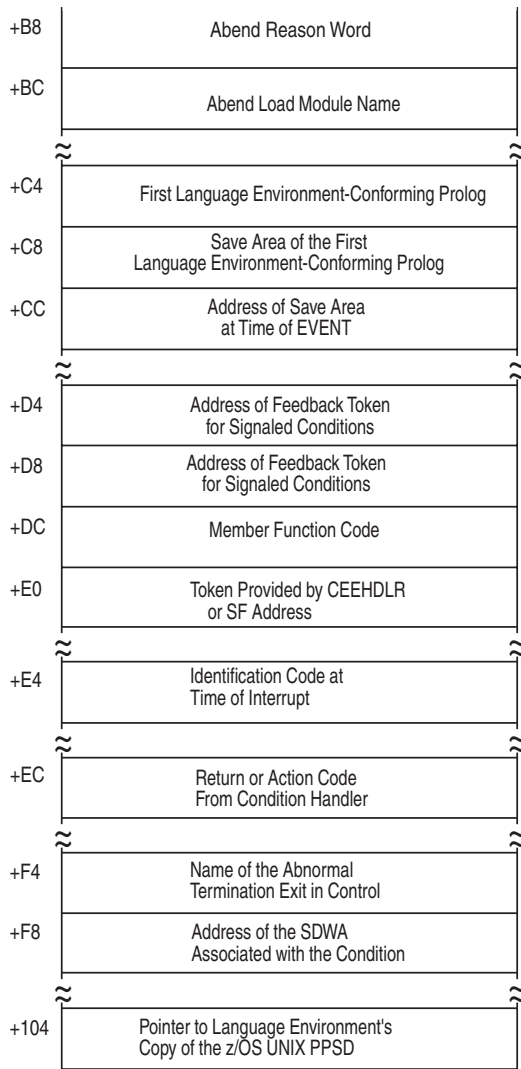


Figure 11. Condition information block (Part 2 of 2)

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.

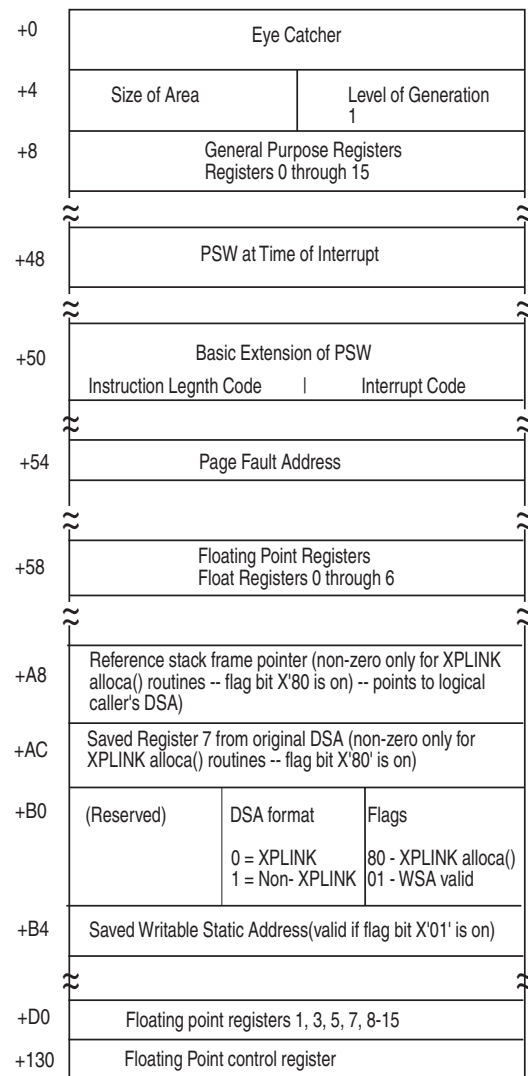


Figure 12. Machine state information block

Using the DLL failure control block

The CEEDLLF control block contains error diagnostics corresponding to an implicit or explicit DLL failure. Diagnostics describing up to 10 of the most recent DLL failures are available in a circular list of CEEDLLF control blocks. When viewing a dump, the in-use CEEDLLF control blocks are displayed from newest to oldest.

| See “Understanding the Language Environment IPCS Verbexit LEDATA output” on
| page 88 for the contents of CEEDLLF fields.

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 79 illustrates the information available in the Language Environment dump and the order of information for multiple enclaves.

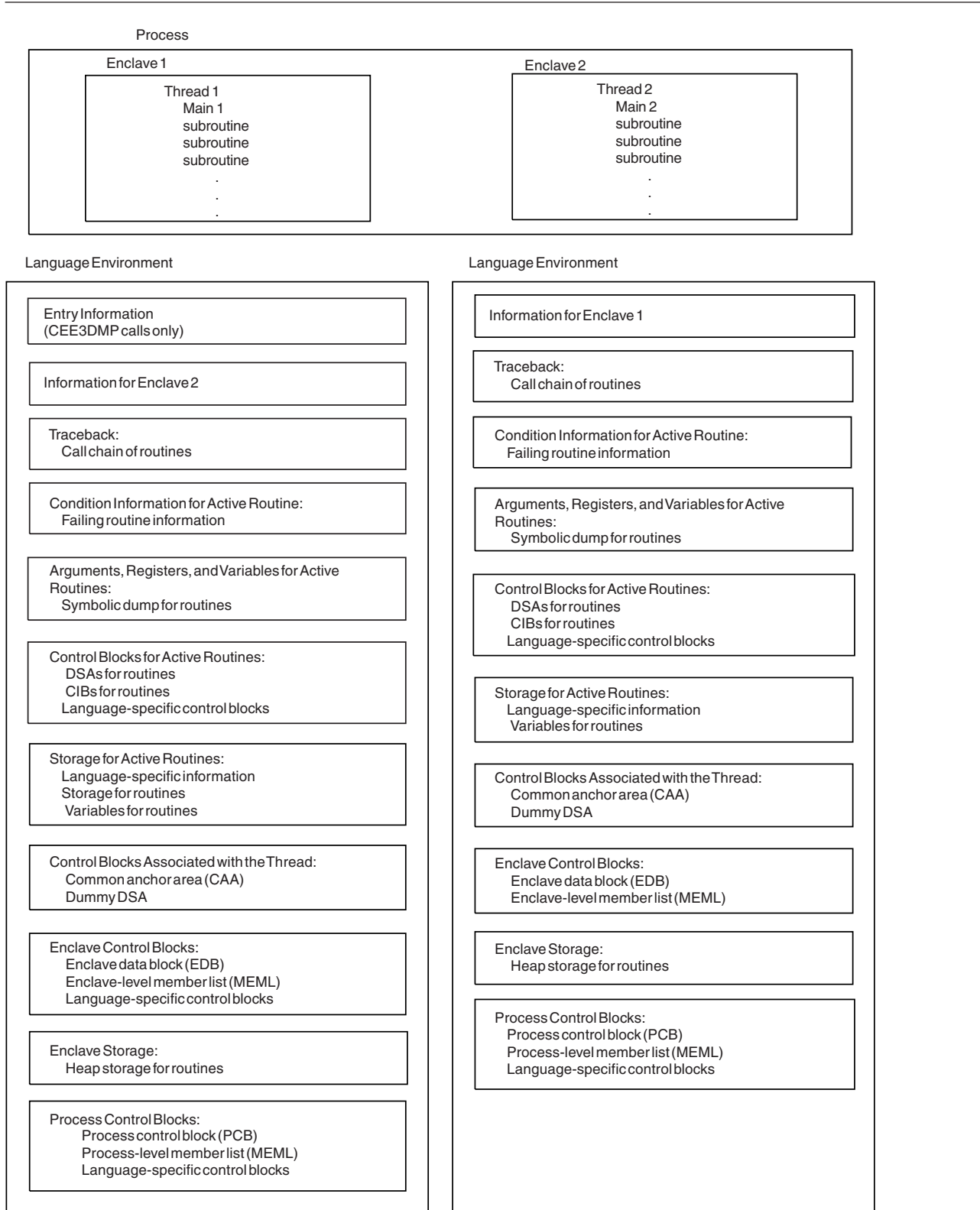


Figure 13. Language Environment dump of multiple enclaves

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:

DYNDUMP(*hlq*,DYNAMIC,TDUMP)

You can use the DYNDUMP run-time option to obtain IPCS-readable dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement.

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.

See 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. Decide whether to include a SYSMDUMP DD card or use the DYNDUMP run-time option.
 - Include a SYSMDUMP DD card with the desired data set name and DCB information:
LRECL=4160, BLKSIZE=4160, and RECFM=FBS.
 - Specify the DYNDUMP run-time option with the following information:
DYNDUMP (*hlq*,DYNAMIC,TDUMP)
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. Decide whether to include a SYSMDUMP DD card or use the DYNDUMP run-time option.
 - Include a SYSMDUMP DD card with the desired data set name and DCB information:
LRECL=4160, BLKSIZE=4160, and RECFM=FBS.
 - Specify the DYNDUMP run-time option with the following information:
DYNDUMP (*hlq*,DYNAMIC,TDUMP)
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.”

Note: DYNDUMP is ignored in a CICS environment.

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:

- Using `_BPXK_MDUMP`

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 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*.

- Using DYNDUMP

1. Specify Language Environment run-time options:

```
export _CEE_RUNOPTS="termthdact(suboption),DYNDUMP(hlq,DYNAMIC,TDUMP)"
```

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 39.
- *hlq* is the high level qualifier for the dump data set to be created.

2. Rerun the program.

When you are done, the system dump is written to the name generated by the DYNDUMP run-time option.

For additional DYNDUMP information see *z/OS Language Environment Programming 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, see *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
- COBOL Control Blocks

Format

Syntax

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

Report Type Parameters:

```
[ SUM ]  
[ HEAP | STACK | SM ]  
[ HPT(value) ]  
[ CM ]  
[ MH ]  
[ CEEDUMP ]  
[ COMP(value) ]  
[ PTBL(value) ]  
[ 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. If only the HPT keyword is specified with no value, the trace behaves similar to when the value is 0.

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.

PTBL(value)

Requests that PreInit tables be formatted according to the following values:

CURRENT

If current is specified, the PreInit table associated with the current or specified TCB is displayed.

address

If an address is specified, the PreInit table at that address is specified.

* All active and dormant PreInit tables within the current address space are displayed; this option is time-consuming.

ACTIVE

The PreInit tables for all TCBs in the address space are displayed.

COMP(value)

Requests component control blocks to be formatted according to the following values:

C Requests a report on C/C++ Run-Time control blocks.

CIO

Requests a report on C/C++ I/O control blocks.

COBOL

Requests a report on COBOL-specific control blocks.

ALL

Requests a report on all the previous control blocks.

If the value specified in COMP is not one of the values (C, CIO, COBOL, or ALL), a message is displayed and it continues executing as if COMP(ALL) was specified.

Note: The ALL parameter for LEDATA also generates a report that includes all the component control blocks.

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 106.

EXCception

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.

Usage Notes: Only hexadecimal characters can be specified as addresses provided in LEDATA parameters. Special characters cause the formatter to fail. Therefore, in order to specify a 64-bit address as a parameter, it must be in the form like 123456789 instead of 1_23456789.

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 89 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 100 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 100.

```

ALL
*****
LANGUAGE ENVIRONMENT DATA
*****
Language Environment Product 04 V01 R09.00

```

```
[1] TCB: 008DCB08          LE Level: 15          ASID: 003F
```

```
[2] Active Members: C/C++
```

```
[3] CEECAA: 209159B0
+000000 FLAG0:00 LANGP:08 BOS:20FC4018 EOS:00000000
+000044 TORC:00000000 TOVF:80014608 ATTN:2090CDF8
+00015C HLLXIT:00000000 HOOK:50C0D064 0DC058C0 C0060DCC
+0001A4 DIMA:00009C64 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:209125F8 CRENT:20FC0C00 CTHD:20910858
+000210 EDCV:A0DA74E4 CEDB:209118D8 EDCOV:20DA094C
+000258 TCASRV_USERWORD:00000000 TCASRV_WORKAREA:2090C5D8
+000260 TCASRV_GETMAIN:00000000 TCASRV_FREEMAIN:00000000
+000268 TCASRV_LOAD:8000FCDB TCASRV_DELETE:8000FBF8
+000270 TCASRV_EXCEPTION:00000000 TCASRV_ATTENTION:00000000
+000278 TCASRV_MESSAGE:00000000 LNS:00000000 SAVR:20C67E8A
+0002AC SYSTEM:03 HRDWR:03 SBSYS:02 FLAG2:B0 LEVEL:15
+0002B1 PM:04 GETLS:80011EE0 CELV:A0917778 GETS:80011FD0
+0002C0 LBOS:00019000 LEOS:00000000 LNAB:00019018
+0002CC DMC:00000000 ABCODE:00000000 RSNCODE:00000000
+0002D8 ERR:20FC55C8 GETSX:800138B0 DDSA:20916350
+0002E4 SECTSI:00000000 PARTSUM:00000000
+0002EC SSEXPN:00000000 EDB:20914648 PCB:20914198
+0002F8 EYEPR:20915998 PTR:209159B0 GETS1:800139A8
+000304 SHAB:00000000 PRGCK:00000004 FLAG1:00 URC:00000000
+000314 ESS:00000000 LESS:00000000 OGETS:800140D0
+000320 OGETLS:00000000 PICICB:00000000 GETSX:00000000 GOSMR:0000
+000330 LEOV:A0A34190 SIGSCTR:00000000 SIGSFLG:00000000
+00033C THDID:20DA9FB0 00000000 DCRENT:00000000
+000348 DANCHOR:00000000 CTCOC:00000000 RCB:20913F68
+000354 CICSRSN:00000000 MEMBR:209163F0
+00035C SIGNAL_STATUS:00000008 HCOM_REG7:00000000
+000364 STACKFLOOR:7FFFFFFF HPGETS:00000000 EDCHPXV:00000000
+000370 FOR1:00000000 FOR2:00000000 THREADHEAPID:2091622C
+00037C SYS_RTNCODE:00000000 SYS_RSNCODE:00000000 GETFN:20A82198
+000390 SIGNGPTR:20FC0C28 SIGNG:00000001 FORDBG:00000000
+00039C AB_STATUS:00 STACKDIRECTION:00 AB_GRP:00000000
+0003A4 AB_ICD1:00000000 AB_ABCC:00000000 AB_CRC:00000000
+0003B0 GTS:80011530 LERN5N1:00000000 HERP:209C9DF0
+0003BC USTKBOS:00000000 USTKEOS:00000000
+0003C4 USERRTN:00000000 UDHOOK:A7F4FEE8 A7F401A0
+0003D0 HPXV_B:A09B77A0 HPXV_M:A0AAEC90 HPXV_L:A0A079F8
+0003DC HPXV_O:A0A79F20 4VEC3:20BC5B4C DLLF:21366C20
+0003FC SMCB:20916118 ERRCM:2090CDB0 MIB_PTR:00000000
+000434 STV:00 A_ISA:00000000 ISA_SIZE:00000000
+000440 PTATPTR:00000000 SIGSSDSA1:00 SIGSSDSA2:00
+000446 STACKUNSTABLE:00 STACK_FLAG:00 SQELADDR:2090EFF0
+00044C VBA:20FC0040 TCS:21366818 THDSTATUS:00000000
+00049C TICB_PTR:2090E2A0 FWD_CHAIN:209159B0
+0004A8 BKWD_CHAIN:209159B0 TCB@:008DCB08 SS_TOP_D:7FFFFFFF
+000700 SS_DSA_U:00000000 DLLFFLAG:00
```

```
[4] CEEDLLF: 21366C20
+000000 EYE:CEEDLLF VERSION:01 FLAGS:00 SIZE:0060 SERVICE:04
+000000 REFERENCE_TYPE:02 LOAD_TYPE:00 PREV:21366BC0
+00001C NEXT:213668C0 FBTK:00000DF6 41C3C5C5 00000000 (CEE35741)
+000034 DLL_NAME:21366C88 SYMBOL_NAME:21366C98
+000040 DLL_NAME_LEN:00000006 SYMBOL_NAME_LEN:0000000F
+000048 RETCODE_1:00000000 RSNCODE_1:00000000
+000050 RETCODE_2:00000000 RSNCODE_2:00000000
```

```
CEEDLLF_DLL_NAME: 21366C88
+000000 21366C88 C3C5D3C4 D3D30000 21365000 00000018 95819485 6D9596A3 6D89956D 84939300 |CEDLLF....&.....name_not_in_dll. |
```

```
CEEDLLF_SYMBOL_NAME: 21366C98
+000000 21366C98 95819485 6D9596A3 6D89956D 84939300 21365000 00000018 95819485 6D9596A3 |name_not_in_dll...&.....name_not |
```

Figure 14. Example of formatted output from LEDATA Verbxix (Part 1 of 12)

```

[5] CEEPCB: 20914198
+000000 PCBEYE:CEEPCB   SYSTM:03   HRDWR:03   SBSYS:02   FLAG2:88
+00000C DBGEGH:00000000   DMEMBR:209143D0   ZLOD:A0AF94D0
+000020 ZDEL:A0AFB50     ZGETST:A0AF6A08   ZFREEST:A0AF6528
+00002C LVTL:2090AAE8     RCB:20913F68     SYSEIB:00000000
+000038 PSL:00000000     PSA:20914648     PSRA:A0AF6858
+000044 OMVS_LEVEL:7F800000   PCB_CHAIN:00000000
+00004C PCB_VSSFE:000141D4   PCB_PRFEH:00000000
+000056 HABD_CLEANUP:0000   LPKA_LODTPY:00000003   IMS:00000000
+00008C ABENDCODE:00000000   REASON:00000000   F3456:000080C2
+000098 MEML:209143B8     MEMBR:209143D0   PCB_EYE:00000000
+0000A4 PCB_BKC:00006F58     PCB_FWC:00000000
+0000AC PCB_R14:A09048D6     PCB_R15:00007910     PCB_R0:7D000009
+0000B8 PCB_R1:00006FF0     PCB_R2:209046E0     PCB_R3:00000000
+0000C4 PCB_R4:00000000     PCB_R5:00000000     PCB_R6:00000000
+0000D0 PCB_R7:00000000     PCB_R8:20903DA0     PCB_R9:008DCCD0
+0000DC PCB_R10:00000000   PCB_R11:A0904802
+0000E4 PCB_R12:00000000   CELV24:00000000   CELV31:A0917778
+0000F0 SLDR:8000FDC8     SECTSIZ:00000000   PARTSUM:00000000
+0000FC SSEXPN:00000000     B MPS:20918868     BMPE:20990FB0
+000108 BLEHL:2090A5C0     BCMXB:20914018     BSTV:02     PM_BYTE:00
+000112 INI_ AMODE:00     FLAGS1:28     ISA:20905000
+000118 ISA_SIZ:00011550   SRV_CNT:00000000
+000120 SRV_UWORD:00000000   WORKAR:00000000   LOAD:00010280
+00012C DELETE:0000FF8     GETSTOR:80011D58     FREESTOR:80011868
+000138 EXCEPT:00000000   ATTN:00000000     MSGS:00000000
+000144 ABEND:00009108     MSGOU:0000C760     GLAT:A0A32168
+000150 RLAT:A0A68F70     ELAT:A0A27E90     IPTQ:A0A7B9F0
+00015C IENV:A0A7AE90     DBG_LODTPY:FFFFFFF   DUMMY_STK:20905010
+000168 DUMMY_LIB:00000000   DUMMY_CAA:20909018
+000170 TST_LVL:FFFFFFF     GETCAA:20914178     SETCAA:20914180
+00017C LLTPTR:20909EA0     AUE:00000000     RC:00000000
+000188 REASON:00000000     RC_MOD:00000000     AUE_UWORD:00000000
+000194 FB_TOKEN:.....     EO V:A0A34190     PPA:210F7F44
+0001A8 PPA_SIZ:0000A000     BELOW:20913E58     BELOW_LEN:000026F8
+0001B4 PICB:2090C430     UTL1:00000008     ZINA:A0AF7C20
+0001C0 ZINB:80014820     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: 209143D0
+000000 MEMLDEF:..... EXIT:20994BE8   LLVTL:00000000

```

```

[6] CEERCB: 20913F68
+000000 EYE:CEERCB   SYSTM:03   HRDWR:03   SBSYS:02   FLAGS:88
+000014 DMEMBR:20909D78   ZLOD:A0AFB140   ZDEL:A0AF1A58
+000020 ZGETST:A0AF6A08   ZFREEST:A0AF6528   VERSION_ID:04010900

```

```

[7] CEEEDB: 20914648
+000000 EYE:CEEEDB   FLAG1:D7   BIPM:00   BPM:00
+00000B CREATOR_ID:01     MEMBR:20915870   OPTCB:20914D50
+000014 URC:00000000     RSNCD:00000000   DBGEGH:00000000
+000020 BANHP:20914B58   BBEHP:20914B88   BCELV:A0917778
+00002C PCB:20914198   ELIST:00000000   PL_ASTRPTR:00000000
+000038 DEFPLPTR:20914768   CXIT_PAGE:00000000
+000040 DEBUG_TERMID:00000000   PARENT:00000000   R13_PARENT:00006F58
+000054 LEOV:A0A34190     ENVAR:2090C880   ENVIRON:20FC0C48
+000060 CEEOSIGR:0000F460   OTRB:210F7000   PSA31:20913E58
+00006C PSL31:00000000     PSA24:20916550   PSL24:00000000
+000078 PSRA:20AF6660     CAACHAIN0:209159B0   FLAG1A:90
+000084 CEEOSGR1:0000FAF6   MEMBERCOMPAT1:00
+000090 THREADSACTIVE:00000001   CURMSGFILEDCBPTR:00015060
+000098 CEEINT_INPUT_R1:00006FF0   LAST_RBADDR:008FF4E8
+0000A0 LAST_RBCNT:00000001

```

```

CEEMEML: 20915870
+000000 MEMLDEF:..... EXIT:20994BE8   LLVTL:00000000

```

```

[8] PMCB: 2090C5A8
+000000 EYE:PCMB   PREV$:00000000   NEXT$:00000000
+000010 LVT_CURR$:A0917778   LLT_CURR$:21366050   FLAGS:20000000

```

[9] 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)

```

Figure 14. Example of formatted output from LEDATA Verberit (Part 2 of 12)

```

INSTALLATION DEFAULT OVR ANYHEAP(00016384,00008192,ANY ,FREE)
INSTALLATION DEFAULT OVR NOAUTOTASK
INSTALLATION DEFAULT OVR BELOWHEAP(00008192,00004096,FREE)
INSTALLATION DEFAULT OVR CBLLOPTS(ON)
INSTALLATION DEFAULT OVR CBLPSHPOP(ON)
INSTALLATION DEFAULT OVR CBLQDA(OFF)
INSTALLATION DEFAULT OVR CEEDUMP(00000060,SYSDUMP=*,
FREE=END,SPIN=UNALLOC)

INSTALLATION DEFAULT OVR CHECK(ON)
INSTALLATION DEFAULT OVR COUNTRY(US)
INSTALLATION DEFAULT OVR NODEBUG
INSTALLATION DEFAULT OVR DEPTHCONDLMT(00000010)
DD:CEEOPST OVR DYNDDUMP(POSIX.HEALY.ZOS19,
DYNAMIC,TDUMP)

INSTALLATION DEFAULT OVR ENVAR("")
INSTALLATION DEFAULT OVR ERRRCOUNT(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,00000001,00000000,00000010,
00000010)
PROGRAMMER DEFAULT OVR HEAPPOLLS(ON,
00000008,00000010,
00000032,00000010,
00000128,00000010,
00000256,00000010,
00001024,00000010,
00002048,00000010,
00000000,00000010,
00000000,00000010,
00000000,00000010,
00000000,00000010,
00000000,00000010,
00000000,00000010,
00000000,00000010)

INSTALLATION DEFAULT OVR INFMSGFILTER(OFF)
INSTALLATION DEFAULT OVR INQPCOPN
INSTALLATION DEFAULT OVR INTERRUPT(OFF)
INSTALLATION DEFAULT OVR LIBSTACK(00004096,00004096,FREE)
INSTALLATION DEFAULT OVR MSGFILE(SYSDUMP ,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 NOSIMVRD
INSTALLATION DEFAULT OVR STACK(00131072,00131072,ANY ,KEEP,
00524288,00131072)

INSTALLATION DEFAULT OVR STORAGE(NONE,NONE,NONE,00000000)
PROGRAMMER DEFAULT OVR TERMTHDACT(UADUMP,CESE,00000096)
INSTALLATION DEFAULT OVR NOTEST(ALL,*,PROMPT,INSPREF)
INSTALLATION DEFAULT OVR THREADHEAP(00004096,00004096,ANY ,KEEP)
INSTALLATION DEFAULT OVR THREADSTACK(OFF,00004096,00004096,ANY ,KEEP,
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 XPLINK(OFF)
INSTALLATION DEFAULT OVR XUFLOW(AUTO)
*****

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 3 of 12)

[10] Heap Storage Control Blocks

Heappools trace available. To display: IP VERBX LEDATA 'HPT(*) CAA(209159B0)'

```

ENSM: 20914B10
+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:20914BEC
+000018 UHEAP:C8D7C3C2 20FE4018 20FE4018 00008000 00008000 00002000 00001000 00000000 00
+000048 AHEAP:C8D7C3C2 20FC0000 2138E000 00004000 00002000 00002000 00001000 00000000 00
+000078 BHEAP:C8D7C3C2 20914B88 20914B88 00002000 00001000 00002000 00001000 80000000 00
+0000A8 ENSM_ADDL_HEAPS:21366FB8

```

```

STSB: 20914BEC
+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: 20914C84
+000000 MAX_ALLOC:00009F28 CURR_ALLOC:00003E50
+000008 LARGEST:00009F28 GETMAINS:00000001
+000010 FREEMAINS:00000000

```

```

SKSB: 20914CAC
+000000 MAX_ALLOC:00001330 CURR_ALLOC:00000000
+000008 LARGEST:00000D98 GETMAINS:00000002
+000010 FREEMAINS:00000000

```

```

SKSB: 20914C98
+000000 MAX_ALLOC:000000D0 CURR_ALLOC:000000D0
+000008 LARGEST:000000D0 GETMAINS:00000001
+000010 FREEMAINS:00000000

```

User Heap Control Blocks

```

HPCB: 20914B28
+000000 EYE_CATCHER:HPCB FIRST:20FE4018 LAST:20FE4018

```

```

HPSB: 20914C0C
+000000 BYTES_ALLOC:00005228 CURR_ALLOC:00005228
+000008 GET_REQ:00000005 FREE_REQ:00000000
+000010 GETMAINS:00000001 FREEMAINS:00000000

```

```

HPSB: 20914CC0
+000000 BYTES_ALLOC:00000000 CURR_ALLOC:00000000
+000008 GET_REQ:00000000 FREE_REQ:00000000
+000010 GETMAINS:00000000 FREEMAINS:00000000

```

```

HANC: 20FE4018
+000000 EYE_CATCHER:HANC NEXT:20914B28 PREV:20914B28
+00000C HEAPID:00000000 SEG_ADDR:A0FE4018 ROOT_ADDR:20FE9240
+000018 SEG_LEN:00008000 ROOT_LEN:00002DD8

```

This is the last heap segment in the current heap.

Free Storage Tree for Heap Segment 20FE4018

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	20FE9240	00002DD8	00000000	00000000	00000000	00000000	00000000

Map of Heap Segment 20FE4018

To display entire segment: IP LIST 20FE4018 LEN(X'00008000') ASID(X'003F')

```

20FE4038: Allocated storage element, length=00001028. To display: IP LIST 20FE4038 LEN(X'00001028') ASID(X'003F')
20FE4040: 00000005 00000005 20FE5068 20FE5250 20FE5280 20FE52CA 20FE5307 20FE5344 |.....&.....&.....|

```

```

20FE5060: Allocated storage element, length=00000828. To display: IP LIST 20FE5060 LEN(X'00000828') ASID(X'003F')
20FE5068: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

```

```

20FE5888: Allocated storage element, length=00000CC8. To display: IP LIST 20FE5888 LEN(X'00000CC8') ASID(X'003F')
20FE5890: 00000003 00000003 C3C4D3D3 00000000 40000000 00000000 209003B8 20903DB0 |.....CDLL....|

```

```

20FE6550: Allocated storage element, length=00002028. To display: IP LIST 20FE6550 LEN(X'00002028') ASID(X'003F')
20FE6558: 00000006 00000006 00000000 20FC2E84 70004000 00000000 20FE6584 00000000 |.....d..|

```

```

20FE8578: Allocated storage element, length=00000CC8. To display: IP LIST 20FE8578 LEN(X'00000CC8') ASID(X'003F')
20FE8580: 00000001 00000001 20FE5BC8 00000000 00000001 00000001 20FE5C50 00000000 |.....$H.....*&....|

```

```

20FE9240: Free storage element, length=00002DD8. To display: IP LIST 20FE9240 LEN(X'00002DD8') ASID(X'003F')

```

Figure 14. Example of formatted output from LEDATA Verboxit (Part 4 of 12)

Summary of analysis for Heap Segment 20FE4018:
 Amounts of identified storage: Free:00002DD8 Allocated:00005208 Total:00007FE0
 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.

Anywhere Heap Control Blocks

```
HPCB: 20914B58
+000000 EYE_CATCHER:HPCB FIRST:20FC0000 LAST:2138E000

HPSB: 20914C3C
+000000 BYTES_ALLOC:00371670 CURR_ALLOC:00319DB0
+000008 GET_REQ:00000B58 FREE_REQ:00000B07
+000010 GETMAINS:00000011 FREEMAINS:00000005

HANC: 20FC0000
+000000 EYE_CATCHER:HANC NEXT:20FED000 PREV:20914B58
+00000C HEAPID:20914B58 SEG_ADDR:A0FC0000 ROOT_ADDR:20FC3500
+000018 SEG_LEN:00004000 ROOT_LEN:00000B00
```

Free Storage Tree for Heap Segment 20FC0000

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	20FC3500	00000B00	00000000	20FC3278	00000000	00000090	00000000
1	20FC3278	00000090	20FC3500	00000000	00000000	00000000	00000000

Map of Heap Segment 20FC0000

To display entire segment: IP LIST 20FC0000 LEN(X'00004000') ASID(X'003F')

```
20FC0020: Allocated storage element, length=00000018. To display: IP LIST 20FC0020 LEN(X'00000018') ASID(X'003F')
20FC0028: D2C4C240 00000000 00000000 00000000 |KDB .....
```

:

Below Heap Control Blocks

```
HPCB: 20914B88
+000000 EYE_CATCHER:HPCB FIRST:20914B88 LAST:20914B88

** NO SEGMENTS ALLOCATED **
```

```
HPSB: 20914C54
+000000 BYTES_ALLOC:00000000 CURR_ALLOC:00000000
+000008 GET_REQ:00000000 FREE_REQ:00000000
+000010 GETMAINS:00000000 FREEMAINS:00000000
```

Additional Heap Control Blocks

```
HPSB: 20914C6C
+000000 BYTES_ALLOC:00000908 CURR_ALLOC:00000908
+000008 GET_REQ:00000007 FREE_REQ:00000000
+000010 GETMAINS:00000003 FREEMAINS:00000002

ADHP: 21366FB8
+000000 EYE_CATCHER:ADHP NEXT:F0F00000 HEAPID:21366FC4

HPCB: 21366FC4
+000000 EYE_CATCHER:HPCB FIRST:21367000 LAST:21367000

HANC: 21367000
+000000 EYE_CATCHER:HANC NEXT:21366FC4 PREV:21366FC4
+00000C HEAPID:21366FC4 SEG_ADDR:21367000 ROOT_ADDR:213672C8
+000018 SEG_LEN:00001000 ROOT_LEN:00000038
```

This is the last heap segment in the current heap.

Free Storage Tree for Heap Segment 21367000

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	213672C8	00000038	00000000	00000000	00000000	00000000	00000000

Map of Heap Segment 21367000

Figure 14. Example of formatted output from LEDATA Verbexit (Part 5 of 12)

To display entire segment: IP LIST 21367000 LEN(X'00001000') ASID(X'003F')

21367020: Allocated storage element, length=000002A8. To display: IP LIST 21367020 LEN(X'000002A8') ASID(X'003F')
21367028: D7C3C9C2 00000000 00000000 000102A0 00000000 00000000 00000000 00000000 |PCIB.....|

213672C8: Free storage element, length=00000D38. To display: IP LIST 213672C8 LEN(X'00000D38') ASID(X'003F')

Summary of analysis for Heap Segment 21367000:

Amounts of identified storage: Free:00000D38 Allocated:000002A8 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.

[11]Stack Storage Control Blocks

```
SMCB: 20916118
+000000 EYE_CATCHER:SMCB US_EYE_CATCHER:USTK USFIRST:20FC4018
+00000C USLAST:20FC4018 USBOS:20FC4018 USEOS:20FE4018
+000018 USNAB:20FC7DC8 USINITSZ:00020000 USINCRSZ:00020000
+000024 USANYBELOW:00000000 USKEEPFREE:00000000 USPOOL:00000000
+000030 USPREALLOC:00000001 US_BYTES_ALLOC:00009F28
+000038 US_CURR_ALLOC:00003E50 US_GETMAINS:00000000
+000040 US_FREEMAINS:00000000 US_OPLINK:00 LS_THIS_IS:LSTK
+00004C LSFIRST:00019000 LSLAST:00019000 LSBOS:00019000
+000058 LSEOS:0001A000 LSNAB:00000000 LSINITSZ:00001000
+000064 LSINCRSZ:00001000 LSANYBELOW:80000000
+00006C LSKEEPFREE:00000001 LSPPOOL:80000001 LSPREALLOC:00000000
+000078 LS_BYTES_ALLOC:000000D0 LS_CURR_ALLOC:000000D0
+000080 LS_GETMAINS:00000000 LS_FREEMAINS:00000000 LS_OPLINK:00
+00008C RSBOS:20FC4000 RSEOS:20FC4018 RSIZE:00000018
+000098 RACTIVE:00000001 SA_REG00:20FC7E68
+0000A0 SA_REG01:20FC7DC8 SA_REG02:20914648
+0000A8 SA_REG03:20914198 SA_REG04:209C949C
+0000B0 SA_REG05:20FC512C SA_REG06:00000000
+0000B8 SA_REG07:2090E2A8 SA_REG08:A09C83C2
+0000C0 SA_REG09:20FC6CA6 SA_REG10:20FC5CA7
+0000C8 SA_REG11:209D4C70 SA_REG12:209159B0
+0000D0 SA_REG13:20FC4CA8 SA_REG14:A09D4CA2
+0000D8 SA_REG15:00000000
+0000DC SAVEREG_XINIT:00000000 00000000 00000000 00000000
+0000EC CEEVGTST:800120C0 ST_DSA_ALLOC_FLAG:00000000
+0000F4 ST_DSA_ALLOC_VAL:00000000 ALLOCSEG:00000000
+0000FC BELOW16M_FLAG:00000000 LOCAL_ALLOC:FFFFFFFF00
+00010C LOCAL_GETMAINS:00000000 LOCAL_FREEMAINS:00000000
+00015C MOREFLAGS:00000000 DS_THIS_IS:.... DSFIRST:20916278
+000168 DSLAST:20916278 DSBOS:20916278 DSINITSZ:00000000
+00017C DSINCRSZ:00000000 DSGUARDSZ:00000000
+000184 DSKEEPFREE: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
```

DSA backchain

```
DSA: 20FC7DC8
+000000 FLAGS:0000 MEMD:0000 BKC:20FC4CA8 FWC:20FC7EC8
+00000C R14:A0AE4FBE R15:A0AE7128 R0:20992840
+000018 R1:20FC7E5C R2:20FC40F0 R3:A0917778
+000024 R4:20FC7E78 R5:20FC4030 R6:00000000
+000030 R7:20FC7E88 R8:00000001 R9:FFFFFFFFC
+00003C R10:00000000 R11:A0AE4DB8 R12:209159B0
+000048 LWS:00000000 NAB:20FC7E68 PNAB:00000000
+000064 RENT:00000000 CILC:00000000 MODE:00000000
+000078 RMR:00000000
```

Contents of DSA at location 20FC7DC8:

```
+00000000 00000000 20FC4CA8 20FC7EC8 A0AE4FBE A0AE7128 20992840 20FC7E5C 20FC40F0 |.....<y..=H..|.....r. ..=*.. 0
+00000020 A0917778 20FC7E78 20FC4030 00000000 20FC7E88 00000001 FFFFFFFC 00000000 |.j....=. . . . . =h.....
+00000040 A0AE4DB8 209159B0 00000000 20FC7E68 00000000 00000000 00000000 00000000 |..(.j.....=. . . . .
+00000060 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+00000080 20FC512C 20FC5244 20FC515C 20FC5184 20FC5188 20FC7E8C 20FC7E7C 20FC7E80 |.....*. . . . . h. . . . =@. . . .
```

```
DSA: 20FC4CA8
+000000 FLAGS:0808 MEMD:CEE1 BKC:20FC4208 FWC:20FC7DC8
+00000C R14:A09C8580 R15:209D4C70 R0:00000073
+000018 R1:20FC5100 R2:20914648 R3:20914198
+000024 R4:209C949C R5:20FC512C R6:00000000
+000030 R7:2090E2A8 R8:A09C83C2 R9:20FC6CA6
+00003C R10:20FC5CA7 R11:A09C4870 R12:209159B0
+000048 LWS:00000000 NAB:20FC7DC8 PNAB:20FE663A
+000064 RENT:210F72AC CILC:2090EFF0 MODE:0000010B
+000078 RMR:20FE6630
```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 6 of 12)

```

Contents of DSA at location 20FC4CA8:
+00000000 0808CEE1 20FC4208 20FC7DC8 A09C8580 209D4C70 00000073 20FC5100 20914648 |.....'H..e...<.....j..
+00000020 20914198 209C949C 20FC512C 00000000 2090E2A8 A09C83C2 20FC6CA6 20FC5CA7 |.j.q..m.....Sy..cB..%w..*x
+00000040 A09C4870 209159B0 00000000 20FC7DC8 20FE663A A0A512DA 20A543BA 20A5BE6C |.....j.....'H.....v...v...v.%
+00000060 20FC4B74 210F72AC 2090EFF0 0000010B 20A584C0 20FC2E94 20FE6630 00000004 |.....0.....vd...m.....
+00000080 20FC5168 20FC5178 20FC5246 209C95C0 209C95C4 20FC517C 20FC5247 209C95C0 |.....n.....nD...@.....n.
+000000A0 209C95C0 209C95C0 209C95C0 A09C4D04 209C5820 00000073 20FC5100 20914648 |.n...n...n... (.....j..
+000000C0 2090CDB0 209C949C 00000000 2090E52C 00000000 20FC6CA6 20FC5CA7 A09C4870 |.....m.....V.....%w..*x...
+000000E0 209159B0 2090E52C A09C529A 209C5D44 00000073 209C95A8 00000001 2090CDB0 |.j....V.....).....ny.....

```

To display entire DSA: IP LIST 20FC4CA8 LEN(X'00003120') ASID(X'003F')

```

DSA: 20FC4208
+00000000 FLAGS:1000 MEMD:0000 BKC:20FC40F0 FWC:20FC42E8
+0000000C R14:A0900E1E R15:20DA8624 R0:2090155C
+00000018 R1:20FC42A0 R2:20FC42C5 R3:209004B2
+00000024 R4:20FC42C2 R5:20901288 R6:20FC42CC
+00000030 R7:20FC42D0 R8:00000030 R9:80000000
+0000003C R10:A0E629E2 R11:A0992840 R12:209159B0
+00000048 LWS:00000000 NAB:20FC42E8 PNAB:00000000
+00000064 RENT:00000000 CILC:00000000 MODE:00000000
+00000078 RMR:00000000

```

Contents of DSA at location 20FC4208:

```

+00000000 10000000 20FC40F0 20FC42E8 A0900E1E 20DA8624 2090155C 20FC42A0 20FC42C5 |.....0...Y.....f.....*.....E
+00000020 209004B2 20FC42C2 20901288 20FC42CC 20FC42D0 00000030 80000000 A0E629E2 |.....B...h.....W.S
+00000040 A0992840 209159B0 00000000 20FC42E8 00000000 00000000 00000000 00000000 |.r..j.....Y.....
+00000060 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+00000080 00000000 00000000 00000000 00000000 20FE5898 00000000 2090155C 2090154D |.....q.....*..(
+000000A0 00000003 20901294 213667F0 00000000 2130420C 21305434 00000000 00000000 |.....m..0.....
+000000C0 00000000 00000002 21366CE0 00000000 00000000 00000000 20FC0C00 00000000 |.....%.....

```

```

DSA: 20FC40F0
+00000000 FLAGS:10AF MEMD:92CC BKC:20FC4030 FWC:20FC4294
+0000000C R14:A1366D50 R15:A0900478 R0:20FC4208
+00000018 R1:20FC0CA0 R2:A0E62AB0 R3:00000002
+00000024 R4:A0992924 R5:20914648 R6:20903DA4
+00000030 R7:209046E8 R8:00000030 R9:80000000
+0000003C R10:A0E629E2 R11:A0992840 R12:209159B0
+00000048 LWS:00000000 NAB:20FC4208 PNAB:20914648
+00000064 RENT:00000000 CILC:00100000 MODE:00000001
+00000078 RMR:00000000

```

Contents of DSA at location 20FC40F0:

```

+00000000 10AF92CC 20FC4030 20FC4294 A1366D50 A0900478 20FC4208 20FC0CA0 A0E62AB0 |..k... ..m.._&.....W..
+00000020 00000002 A0992924 20914648 20903DA4 209046E8 00000030 80000000 A0E629E2 |.....r..j.....u...Y.....W.S
+00000040 A0992840 209159B0 00000000 20FC4208 20914648 00000000 00000000 00000000 |.r..j.....j.....
+00000060 00000000 00000000 00100000 00000001 00000001 00000200 00000000 00000000 |.....
+00000080 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+000000A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+000000C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+000000E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....

```

To display entire DSA: IP LIST 20FC40F0 LEN(X'00000118') ASID(X'003F')

```

DSA: 20FC4030
+00000000 FLAGS:0000 MEMD:0000 BKC:20916350 FWC:20FC44A8
+0000000C R14:A09929F8 R15:20E629EE R0:7D000009
+00000018 R1:20FC40B0 R2:209046E0 R3:00000002
+00000024 R4:A0992924 R5:20914648 R6:20903DA4
+00000030 R7:209046E8 R8:00000030 R9:00000008
+0000003C R10:A0E629E2 R11:A0992840 R12:209159B0
+00000048 LWS:00000000 NAB:20FC40F0 PNAB:00000000
+00000064 RENT:00000000 CILC:00000000 MODE:00000000
+00000078 RMR:00000000

```

Figure 14. Example of formatted output from LEDATA Verboxit (Part 7 of 12)

Contents of DSA at location 20FC4030:

```
+00000000 00000000 20916350 20FC44A8 A09929F8 20E629EE 7D000009 20FC40B0 209046E0 |.....j.&...y.r.8.W.'.....|
+00000020 00000002 A0992924 20914648 20903DA4 209046E8 00000030 00000008 A0E629E2 |.....r...j.....u...Y.....W.S|
+00000040 A0992840 209159B0 00000000 20FC40F0 00000000 00000000 00000000 00000000 |.r. .j.....0.....|
+00000060 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+00000080 20992B08 20903DA4 20FC40E0 20914780 20FC40D0 20FC40D8 20FC40D4 A0A34190 |.r.....u. .j..... Q. .M.t..|
+000000A0 20FC0CA0 00000000 00000000 20AF9CD6 00000001 A0AF7CD8 209159B0 00000000 |.....0.....@Q.j.....|
```

User Stack Control Blocks

```
STKH: 20FC4018
+000000 EYE_CATCHER:STKU NEXT:2091611C PREV:2091611C
+00000C SEGMENT_LEN:00020000
```

Library Stack Control Blocks

```
STKH: 00019000
+000000 EYE_CATCHER:STKL NEXT:20916160 PREV:20916160
+00000C SEGMENT_LEN:00001000
```

[12]Condition Management Control Blocks

```
HCOM: 2090CDB0
+000000 PICA_AREA:00000000 00000000 EYES:HCOM CAA_PTR1:209159B0
+000014 CVTDCB:9B FLAG:60F0C000 EXIT_STK:21366D50
+000020 RSM_PTR:00000000 HDLL_STK:21366D08
+000028 SRP_TOKEN:00000000 CSTK:2136CE48 CIBH:20FC5AD8
+000084 COND_LOG:2136A028 DSA_4083:00000000
+00047C SHUNT_VALIDFLAG:00000001 SHUNT_COUNTER:00000018
+000484 SHUNT_ADDR:209F5940 SHUNT_PSW:078D2000 A09F5948
+000490 HCOM_REG0:00000000 HCOM_REG1:20FC9EFC
+000498 HCOM_REG2:00000000 HCOM_REG3:00000000
+0004A0 HCOM_REG4:5ECEFC28 HCOM_REG5:00000001
+0004A8 HCOM_REG6:00000000 HCOM_REG7:00000000
+0004B0 HCOM_REG8:00001000 HCOM_REG9:5ECEFC28
+0004B8 HCOM_REG10:209F5940 HCOM_REG11:A09F4EE0
+0004C0 HCOM_REG12:209159B0 HCOM_REG13:20FC9DA0
+0004C8 HCOM_REG14:A09F585A HCOM_REG15:00000000
+0004D0 SHUNT_CODE1:00000000 SHUNT_CODE2:00000004

CIBH: 20FC5AD8
+000000 EYE:CIBH BACK:2090E2A8 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
+000070 OLD_MIB:00000000 COND_64:00000000 00000000
+00007C MIB:00000000 PL:00000000 SV2:00000000
+000088 SV1:00000000 INT:00000000 MID:00000000
+000094 HDL_SF:00000000 HDL_EPT:00000000 HDL_RST:00000000
+0000A0 RSM_SF:00000000 RSM_POINT:00000000 RSM_MACHINE:00000000
+0000B0 COND_DEFAULT:00000000 Q_DATA_TOKEN:00000000 FDBK:00000000
+0000BC ABNAME:..... BBRANCH_OFFSET:00000000
+000220 BBRANCH_STMTID:..... BBRANCH_STMTLEN:0000
Machine State
+000248 MCH_EYE:....
+000250 GPR00:00000000 GPR01:00000000
+000258 GPR02:00000000 GPR03:00000000
+000260 GPR04:00000000 GPR05:00000000
+000268 GPR06:00000000 GPR07:00000000
+000270 GPR08:00000000 GPR09:00000000
+000278 GPR10:00000000 GPR11:00000000
+000280 GPR12:00000000 GPR13:00000000
+000288 GPR14:00000000 GPR15:00000000
+000290 PSW:00000000 00000000
+000298 ILC:0000 IC1:00 IC2:00 PFT:00000000
+0002A0 FLT_0:00000000 00000000 FLT_2:00000000 00000000
+0002B0 FLT_4:00000000 00000000 FLT_6:00000000 00000000
+000300 EXT:00000000
+000304 BEA:00000000
```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 8 of 12)

```

+000318 FLT_1:00000000 00000000
+000320 FLT_3:00000000 00000000
+000328 FLT_5:00000000 00000000
+000330 FLT_7:00000000 00000000
+000338 FLT_8:00000000 00000000 FLT_9:00000000 00000000
+000348 FLT_10:00000000 00000000 FLT_11:00000000 00000000
+000358 FLT_12:00000000 00000000 FLT_13:00000000 00000000
+000368 FLT_14:00000000 00000000 FLT_15:00000000 00000000
+000378 FPC:00000000 APF_FLAGS:00
+000388 GPR_H00:00000000 GPR_H01:00000000
+000390 GPR_H02:00000000 GPR_H03:00000000
+000398 GPR_H04:00000000 GPR_H05:00000000
+0003A0 GPR_H06:00000000 GPR_H07:00000000
+0003A8 GPR_H08:00000000 GPR_H09:00000000
+0003B0 GPR_H10:00000000 GPR_H11:00000000
+0003B8 GPR_H12:00000000 GPR_H13:00000000
+0003C0 GPR_H14:00000000 GPR_H15:00000000 +000AE0 ABCC:00000000 HRC:00000000 RSM_SF_FMT:00
+000AE9 RSM_PH_CALLEE_FMT:00 SV1_FMT:00 RSM_PH_CALLEE:00000000
+000AF0 INT_FCN_EP:00000000 HDL_SF_FMT:00 HDL_PH_CALLEE_FMT:00
+000AF6 SV2_FMT:00 HDL_PH_CALLEE:00000000

CIBH: 2090E2A8
+000000 EYE:CIBH BACK:00000000 FRWD:20FC5AD8
+000010 PTR_CIB:20FC55C8 FLAG1:C5 ERROR_LOCATION_FLAGS:3F
+000018 HDLQ:00000000 STATE:00000000 PRM_DESC:00000000
+000024 PRM_PREFIX:00000000
+000028 PRM_LIST:20FC55E0 20FC56A8 20FC56B4 2090E8F4
+000038 PARM_DESC:00000000 PARM_PREFIX:00000000
+000040 PARM_LIST:20FC56A4 20FC55C8 20FC56B4 2090E8F4 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 (CEE3209S)
+000070 OLD_MIB:00000002 COND_64:00030C89 59C3C5C5 (CEE3209S)
+00007C MIB:00000002 PL:20901698 SV2:20FC4208
+000088 SV1:20FC4208 INT:20900E32 MID:00000003
+000094 HDL_SF:20916350 HDL_EPT:20994BE8 HDL_RST:00000000
+0000A0 RSM_SF:20FC4208 RSM_POINT:20900E34 RSM_MACHINE:2090E6F0
+0000B0 COND_DEFAULT:00000003 Q_DATA_TOKEN:2090E3E0 FDBK:00000000
+0000BC ABNAME:..... BBRANCH_OFFS:00000000
+000220 BBRANCH_STMTID:..... BBRANCH_STMTLEN:0000
Machine State
+000248 MCH_EYE:ZMCH
+000250 GPR00:00000000 GPR01:A0D9AD0C
+000258 GPR02:20FC42C5 GPR03:209004B2
+000260 GPR04:20FC42C2 GPR05:20901288
+000268 GPR06:00000000 GPR07:00000001
+000270 GPR08:00000030 GPR09:80000000
+000278 GPR10:A0E629E2 GPR11:A0992840
+000280 GPR12:209159B0 GPR13:20FC4208
+000288 GPR14:A0900E1E GPR15:00000012
+000290 PSW:078D2400 A0900E34
+000298 ILC:0002 IC1:00 IC2:09 PFT:00000000
+0002A0 FLT_0:4DBFF67D 46DBE848 FLT_2:00000000 00000000
+0002B0 FLT_4:00000000 00000000 FLT_6:00000000 00000000
+000300 EXT:00000000
+000304 BEA:00000000
+000318 FLT_1:00000000 00000000
+000320 FLT_3:00000000 00000000
+000328 FLT_5:00000000 00000000
+000330 FLT_7:00000000 00000000
+000338 FLT_8:00000000 00000000 FLT_9:00000000 00000000
+000348 FLT_10:00000000 00000000 FLT_11:00000000 00000000
+000358 FLT_12:00000000 00000000 FLT_13:00000000 00000000
+000368 FLT_14:00000000 00000000 FLT_15:00000000 00000000

```

Figure 14. Example of formatted output from LEDATA Verboxit (Part 9 of 12)

```

+000378 FPC:00000000      APF_FLAGS:00
+000388 GPR_H00:00000000      GPR_H01:00000000
+000390 GPR_H02:00000000      GPR_H03:00000000
+000398 GPR_H04:00000000      GPR_H05:00000000
+0003A0 GPR_H06:00000000      GPR_H07:00000000
+0003A8 GPR_H08:00000000      GPR_H09:00000000
+0003B0 GPR_H10:00000000     GPR_H11:00000000
+0003B8 GPR_H12:00000000     GPR_H13:00000000
+0003C0 GPR_H14:00000000     GPR_H15:00000000

+000AE0 ABCC:00000000      HRC:00000000      RSM_SF_FMT:00
+000AE9 RSM_PH_CALLEE_FMT:00      SV1_FMT:00      RSM_PH_CALLEE:00000000
+000AF0 INT_FCN_EP:00000000     HDL_SF_FMT:00      HDL_PH_CALLEE_FMT:00
+000AF6 SV2_FMT:00      HDL_PH_CALLEE:20FC4030

      CIB: 20FC55C8
+000000 EYE:CIB      BACK:00000000      FRWD:00000000      SIZ:010C
+00000E VER:0004      PLAT_ID:00C3C5C5      COND_64:000300C6 59C3C5C5 (CEE0198S)
+000020 MIB:00000000      MACHINE:20FC56D8      OLD_COND_64:00030C89 59C3C5C5 (CEE3209S)
+000030 OLD_MIB:00000002      FLG_1:00      FLG_2:00      FLG_3:00
+000037 FLG_4:04      HDL_SF:20FC4030      HDL_EPT:20994BE8
+000040 HDL_RST:00000000      RSM_SF:20FC4208      RSM_POINT:20900E34
+00004C RSM_MACHINE:2090E6F0      COND_DEFAULT:00000003
+000054 PH_CALLEE_SF:20FC40F0      HDL_SF_FMT:00      PH_CALLEE_SF_FMT:00
+00005A BBRANCH_STMTLEN:0000      BBRANCH_OFFSET:00000000
+000060 BBRANCH_STMTID:.....      VSR:40404040 40404040
+000098 VSTOR:40404040      VRPSA:40404040      MCB:40404040
+0000A4 MRN:40404040 40404040      MFLAG:00      FLG_5:48      FLG_6:23
+0000B2 FLG_7:00      FLG_8:00      ABCD:940C9000      ABRC:00000009
+0000BC ABNAME:00000000 00000000      PL:20901698      SV2:20FC4208
+0000CC SV1:20FC4208      INT:20900E32      Q_DATA_TOKEN:40404040
+0000D8 FDBK:40404040      FUN:00000067      TOKE:20FC4030
+0000E4 MID:00000003      STATE:00000000      RTCC:FFFFFFFF
+0000F0 PPAV:00000003      AB_TERM_EXIT:40404040 40404040
+0000FC SDWA_PTR:00000000      SIGNO:00000008      PPSD:2090E908

```

[13]Message Processing Control Blocks

```

      CMXB: 20914018
+000000 EYE:CMXB      SIZE:0148      FLAGS:8000      DHEAD1:0001A000
+00000C DHEAD2:00015000

      MDST forward chain from CMXBDHEAD(1)

      MDST: 0001A000
+000000 EYE:MDST      SIZE:0100      CTL:40      CEEDUMPLOC:00
+000008 NEXT:00015000      PREV:00000000      DDNAM:CEEDUMP

      MDST: 00015000
+000000 EYE:MDST      SIZE:0100      CTL:40      CEEDUMPLOC:00
+000008 NEXT:00000000      PREV:0001A000      DDNAM:SYSOUT

      MDST back chain from CMXBDHEAD(2)

      MDST: 00015000
+000000 EYE:MDST      SIZE:0100      CTL:40      CEEDUMPLOC:00
+000008 NEXT:00000000      PREV:0001A000      DDNAM:SYSOUT
      MDST: 0001A000
+000000 EYE:MDST      SIZE:0100      CTL:40      CEEDUMPLOC:00
+000008 NEXT:00015000      PREV:00000000      DDNAM:CEEDUMP

      TMXB: 2090EEC0
+000000 EYE:TMXB      MIB_CHAIN_PTR:21383028

      MGF: 21383028
+000000 EYE:CMIB      PREV:2136D668      NEXT:20FC3410

      MGF: 20FC3410
+000000 EYE:CMIB      PREV:21383028      NEXT:2090EEF8

      MGF: 2090EEF8
+000000 EYE:CMIB      PREV:20FC3410      NEXT:2136D668

      MGF: 2136D668
+000000 EYE:CMIB      PREV:2090EEF8      NEXT:21383028

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 10 of 12)

```

[14] Information for enclave main

[15] Information for thread 20DA9FB000000000
PCB Address: 20914198
TCB Address: 008DCB08

[16] Registers and PSW:
GPR0..... 84000000 GPR1..... 84000FC7 GPR2..... 20FC55C8 GPR3..... 00020009
GPR4..... 209C949C GPR5..... 20FC512C GPR6..... 00000000 GPR7..... 2090E2A8
GPR8..... 20FC55C8 GPR9..... 20FC6CA6 GPR10.... 20FC512C GPR11.... 209D4C70
GPR12.... 209159B0 GPR13.... 20FC7DC8 GPR14.... A09D4CA2 GPR15.... 00000000
PSW..... 078D1400 A09D4D46

[17] Traceback:
DSA      Entry      E Offset  Statement  Load Mod      Program Unit      Service  Status
1        CEEHSDMP    +000000D6  CEEPLPKA   CEEPLPKA      CEEHSDMP          D1908   Call
2        CEEHDSP     +00003D0E  CEEPLPKA   CEEPLPKA      CEEHDSP           D1908   Call
3        main       +000009BA  CELSAMP    CELSAMP        CEEHDSP           D1908   Exception
4        EDCZMINV   +000000C0  CEEEV003   CEEEV003      CEEHDSP           D1908   Call
5        CEEBBEXT   +000001B6  CEEPLPKA   CEEPLPKA      CEEBBEXT          D1908   Call

DSA      DSA Addr  E Addr  PU Addr  PU Offset  Comp Date  Compile Attributes
1        20FC7DC8  209D4C70 209D4C70 +000000D6 20061215 CEL POSIX
2        20FC4CA8  209C4870 209C4870 +00003D0E 20061215 CEL POSIX
3        20FC4208  20900478 20900478 +000009BA 20070105 C/C++ POSIX EBCDIC HFP
4        20FC40F0  20E629EE 20E629EE +000000C0 20061215 LIBRARY POSIX
5        20FC4030  20992840 20992840 +000001B6 20061215 CEL POSIX

[18] Control Blocks Associated with the Thread:
Thread Synchronization Queue Element (SQEL): 2090EFFF
+000000 2090EFFF 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 2090F010 209159B0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.j.....|

[19] Enclave Control Blocks:
Mutex and Condition Variable Blocks (MCVB+MHT+CHT): 210F7018
+000000 210F7018 00008F50 210F7044 000003F8 00001FC0 00000000 20FC3088 210F7444 000000F8 |...&.....8.....h.....8|
+000020 210F7038 000007C0 00000000 20FC30A0 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 210F7058 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
:
Thread Synchronization Enclave Latch Table (EPALT): 210F7544
+000000 210F7544 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 210F7564 - +00055F 210F7AA3 same as above |.....|
+000560 210F7AA4 00000000 00000000 00000000 00000000 A0A83100 00000000 00000000 00000000 |.....y.....|
+000580 210F7AC4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0005A0 210F7AE4 - +00061F 210F7B63 same as above |.....|
+000620 210F7B64 00000000 00000000 00000000 00000000 00000000 00000000 20A83100 00000000 |.....y.....|
+000640 210F7B84 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000660 210F7BA4 - +0009FF 210F7F43 same as above |.....|
Thread Synchronization Trace Block (OTRB): 210F7000
+000000 210F7000 210EF000 00000002 000007FF 210EF000 A0000000 A0000000 00008F50 210F7044 |..0.....0.....&....|
Thread Synchronization Trace Table (OTRTBL): 210EF000
+000000 210EF000 0000C7D3 40D5E640 210F79E0 00000002 21334BD8 21333D78 DECC288 00000000 |..GL NW .....Q.....Bh....|
+000020 210EF020 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 210EF040 - +007FFF 210F6FFF same as above |.....|
HEAPCHK Option Control Block (HCOP): 20FC00D0
+000000 20FC00D0 C8C3D6D7 00000038 00000001 00000000 00000000 0000000A 0000000A 00000000 |HCOP.....|
+000020 20FC00F0 21368028 20FC0108 21102028 00000000 00000000 00000000 C8C3C6E3 00000200 |.....HCFT....|
HEAPCHK Element Table (HCEL) for Heapid 21366FC4 :
Header: 21368028
+000000 21368028 C8C3C5D3 20FC0CE0 00000000 21366FC4 000001F4 00000001 00000001 00000000 |HCEL.....?D...4.....|
Address Seg Addr Length Address Seg Addr Length
Table: 21368048
+000000 21368048 21367020 21367000 000002A8 20FC3310 00000000 00000000 00000000 00000000 |.....y.....|
HEAPCHK Element Table (HCEL) for Heapid 00000000 :
Header: 20FC0CE0
+000000 20FC0CE0 C8C3C5D3 00000000 21368028 00000000 000001F4 00000005 00000005 00000000 |HCEL.....4.....|
Address Seg Addr Length Address Seg Addr Length
Table: 20FC0D00
+000000 20FC0D00 20FE4038 20FE4018 00001028 20FC2C48 20FE5060 20FE4018 00000828 20FC2D10 |...&-.....|
+000020 20FC0D20 20FE5888 20FE4018 00000CC8 20FC2DE8 20FE6550 20FE4018 00002028 20FC2F18 |..h...H...Y...&..|
+000040 20FC0D40 20FE8578 20FE4018 00000CC8 21366EAB 00000000 00000000 00000000 00000000 |..e...H...>y.....|

```

Figure 14. Example of formatted output from LEDATA Verbexit (Part 11 of 12)

[20] Language Environment Trace Table:

Most recent trace entry is at displacement: 004700

Displacement	Trace Entry in Hexadecimal	Trace Entry in EBCDIC
+000000	Time 04.22.34.474835 Date 2007.01.06 Thread ID... 20DA9FB000000000	
+000010	Member ID.... 03 Flags..... 0000CE 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 04.22.34.478351 Date 2007.01.06 Thread ID... 20DA9FB000000000	
+000090	Member ID.... 03 Flags..... 0000CE Entry Type..... 00000002	
+000098	4C60604D F0F8F55D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 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 04.22.34.478354 Date 2007.01.06 Thread ID... 20DA9FB000000000	
+000110	Member ID.... 03 Flags..... 0000CE 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 04.22.34.478361 Date 2007.01.06 Thread ID... 20DA9FB000000000	
+000190	Member ID.... 03 Flags..... 0000CE 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
:		
+004600	Time 04.22.38.116963 Date 2007.01.06 Thread ID... 20DA9FB000000000	
+004610	Member ID.... 03 Flags..... 0000CE Entry Type..... 00000002	
+004618	4C60604D F0F8F55D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0	<--(085) R15=00000000 ERRNO=0000
+004638	F0F0F0F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000	0000.....
+004658	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
+004678	00000000 00000000
+004680	Time 04.22.38.116964 Date 2007.01.06 Thread ID... 20DA9FB000000000	
+004690	Member ID.... 03 Flags..... 0000CE Entry Type..... 00000001	
+004698	A3889985 81846D83 93858195 A4974040 40404040 40404040 40404040 40404040	thread cleanup
+0046B8	60606E4D F0F8F55D 40979989 95A3864D 5D404040 40404040 40404040 40404040	-->(085) printf()
+0046D8	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+0046F8	40404040 40404040	
+004700	Time 04.22.38.116969 Date 2007.01.06 Thread ID... 20DA9FB000000000	
+004710	Member ID.... 03 Flags..... 0000CE Entry Type..... 00000002	
+004718	4C60604D F0F8F55D 40D9F1F5 7EF0F0F0 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0	<--(085) R15=00000000 ERRNO=0000
+004738	F0F0F0F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000	0000.....
+004758	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
+004778	00000000 00000000

[21] Process Control Blocks:

Thread Synchronization Process Latch Table (PPALT): 210F7F44

+000000	210F7F44	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
+000020	210F7F64	- +0009FF	210F8943								same as above

[22] No PIPICB associated with CAA at address : 209159B0

Exiting Language Environment Data

Figure 14. Example of formatted output from LEDATA Verbexit (Part 12 of 12)

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] - [9] 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). See “The Common Anchor Area” on page 65 for a description of the fields in the CAA.

[4] CEEDLLF

| This section formats the contents of all Language Environment CEEDLLF (DLLF)
| control blocks that are in use. See The CEEDLLF DLL Failure Control Block in *z/OS*
| *Language Environment Vendor Interfaces* for more information about the CEEDLLF
| control block chain.

[5] CEEPCB

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

[6] CEERCB

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

[7] CEEEDB

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

[8] PMCB

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

[9] Run-Time Options

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

[10] 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 106.

[11] 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)
 - See “The upward-growing (non-XPLINK) stack frame section” on page 63 or “The downward-growing (XPLINK) stack frame section” on page 64 for a description of the fields in the DSA.
- Chain of stack segment headers (STKH)
 - An STKH immediately precedes each segment of stack storage.

[12] 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. See “The condition information block” on page 74 for a description of fields in these control blocks.

[13] Message Processing Control Blocks

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

[14] - [21] CEEDUMP Formatted Control Blocks

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

[14]-[17] NTHREADS information

One or more instances of these sections are included when the NTHREADS() parameter is specified on the LEDATA invocation. For a description of NTHREADS, see “Report type parameters” on page 85.

[14] Enclave Identifier

This statement names the enclave for which information is provided.

[15] Information for thread

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

[16] 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.

[17] Traceback

For all active routines in a particular thread, the traceback section shows routine information in two parts. The first part contains the following items:

- DSA number
A number assigned to the information for this active routine by dump processing. The number is used to associate information from the first part of the traceback with information in the second part of the traceback.
- 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, the string `*** NoName ***` will appear.
- Entry point offset
- Statement number
This field contains no Language Environment data.
- Load module
- 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.
- 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 the following items:

- DSA number
A number assigned to the information for this active routine by dump processing. The number is 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**
Contains the year, month and day in which the routine was compiled.
- **Attributes**
The available compilation attributes of the compile unit including:
 - A label identifying the LE-supported language such as COBOL, ENT PL/I, C/C++, and so on.
 - Compilation attributes such as EBCDIC, ASCII, IEEE, or hexadecimal floating point (HFP). The compilation attributes will only be displayed if there is enough information available.
 - POSIX, If the CEEDUMP was created under a POSIX environment.

[18] Control Blocks Associated with the Thread

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

[19] 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.

[20] 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.

[21] Process Control Blocks

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

[22] Preinitialization Information

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

This section formats information related to preinitialization. See PTBL LEDATA output for more information. If the preinitialization service CEEPIPI was not used to initialize this environment, the message: No PIPICB associated with CAA is displayed instead.

PTBL LEDATA output

The Language Environment IPCS Verbexit LEDATA command generates formatted output of Preinit tables when the PTBL or ALL parameter are specified. If ALL is specified, PTBL defaults to CURRENT value. Figure 15 on page 105 illustrates the output produced when the Verbexit LEDATA command is invoked with the PTBL

parameter.

```
PTBL(CURRENT)
*****
                LANGUAGE ENVIRONMENT DATA
*****

Language Environment Product 04 V01 R09.00

PreInitialization Programming Interface Trace Data
CEEIPI Environment Table Entry and Trace Entry :
Active CEEIPI Environment ( Address 20905CB0 )
Eyecatcher : CEEXIPTB
TCB address : 008D6E88

CEEIPI Environment :
Non-XPLINK Environment
Environment Type : MAIN
Sequence of Calls not active
Exits not established
Signal Interrupt Routines not registered
Service Routines are not active

CEEIPI Environment Enclave Initialized
Number of CEEIPI Table Entries = 3

CEEIPI Table Entry Information :
CEEIPI Table Index 0 ( Entry 1 )
Routine Name = ISJPPCA3
Routine Type = C/C++
Routine Entry Point = A0910530
Routine Function Pointer = A0910620
Routine Entry is Non-XPLINK
Routine was loaded by Language Environment
Routine Address was resolved
Routine Function Descriptor was valid
Routine Return Code = 0
Routine Reason Code = 0

Entry of routine in CEEIPI Table for Index 0 ( 20905DB8 )

+000000 20905DB8 A0910620 20919B30 80000000 00000000 00000000 00000000 00000000 00000000 |.j...j.....|
+000020 20905DD8 00000000 00000000 00000000 A0910530 00000003 209197D8 00000003 20910530 |.....j.....jpQ.....j..|
+000040 20905DF8 A0910530 00009AD0 C9E2D1D7 D7C3C1F3 00000000 00000000 00000000 00000000 |.j.....ISJPPCA3.....|

CEEIPI Table Index 1 ( Entry 2 ) not in use.
CEEIPI Table Index 2 ( Entry 3 ) not in use.
```

Figure 15. Example of formatted PTBL output from LEDATA Verbexit (Part 1 of 2)

```

CEEPIPI Trace Table Entries :
Call Type = INIT_MAIN
PIPI Driver Address = A090068A
Load Service Return Code = 0
Load Service Reason Code = 0
Most Recent Return Code = 0
Most Recent Reason Code = 0
An ABEND will be issued if storage can not be obtained
PreInit Environment will not allow EXEC CICS commands
Service RC = 0 :A new environment was initialized.

Call Type = ADD_ENTRY
Routine Table Index = 1
Routine Name = ISJPPCA1
Routine Address = A0FCC548
Load Service Return Code = 0
Load Service Reason Code = 3
Service RC = 0 :The routine was added to the PreInit table.

Call Type = IDENTIFY_ENTRY
Routine Table Index = 1
Routine Programming Language = C/C++
Service RC = 0 :The programming language has been returned.

Call Type = CALL_MAIN
Routine Table Index = 1
Enclave Return Code = 0
Enclave Reason Code = 0
Routine Feedback Code = 0000000000000000
Service RC = 0 :The environment was activated and the routine called.

Call Type = DELETE_ENTRY
Routine Table Index = 1
Routine Name = ISJPPCA1
Routine Address = A0FCC548
Service RC = 0 :The routine was deleted from the PreInit table.

Call Type = CALL_MAIN
Routine Table Index = 0
Enclave Return Code = 0
Enclave Reason Code = 0
Routine Feedback Code = 0000000000000000
Service RC = 0 :The environment was activated and the routine called.

Exiting Language Environment Data

```

Figure 15. Example of formatted PTBL output from LEDATA Verbexit (Part 2 of 2)

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 16 on page 107 illustrates the output produced by specifying the HEAP option. “Heap report sections of the LEDATA output” on page 110 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_HEAPS: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      Length  Length
  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 24703F70 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 247006F0 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 16. 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

      Node      Node      Parent      Left      Right      Left      Right
Depth 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 16. 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

      Node      Node      Parent      Left      Right      Left      Right
Depth 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 16. 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

   Node      Node      Parent      Left      Right      Left      Right
Depth 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 16. 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.

```

HPT(3)
*****
                        LANGUAGE ENVIRONMENT DATA
*****

Language Environment Product 04 V01 R10.00

[1] HEAPPOOLS Trace Table

[2] POOLID: 3  ASID: 0024  AVAILABLE ENTRIES: 12 OF 12

[3] Timestamp: 2008/03/14          14:10:22.614088
    Type: FREE  Cell Address: 25E91AC0  Cpuid: 01  Tcb: 008AFCF0
[4] CALL NAME                      CALL ADDRESS      CALL OFFSET
GetStorage::~~GetStorage()         25E53360           00000088
foo8()                             25E53598           000000B6
foo7()                             25E53678           0000005A
foo6()                             25E536F0           0000005A
foo5()                             25E53768           0000005A
foo4()                             25E537E0           0000005A
foo3()                             25E53858           0000005A
foo2()                             25E538D0           0000005A
foo1()                             25E53948           0000005A
thread                             25E53A50           00000000

```

Figure 17. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 1 of 4)

```

Timestamp: 2008/03/14          14:10:22.614087
Type: FREE Cell Address: 25E91B48 CpuId: 01 Tcb: 008AFCF0
CALL NAME          CALL ADDRESS    CALL OFFSET
GetStorage::~~GetStorage()  25E53360    00000088
foo9()             25E53430    000000B6
foo8()             25E53598    0000009A
foo7()             25E53678    0000005A
foo6()             25E536F0    0000005A
foo5()             25E53768    0000005A
foo4()             25E537E0    0000005A
foo3()             25E53858    0000005A
foo2()             25E538D0    0000005A
foo1()             25E53948    00000000

Timestamp: 2008/03/14          14:10:22.614034
Type: FREE Cell Address: 25E91BD0 CpuId: 01 Tcb: 008AFA60
CALL NAME          CALL ADDRESS    CALL OFFSET
GetStorage::~~GetStorage()  25E53360    00000088
foo8()             25E53598    000000B6
foo7()             25E53678    0000005A
foo6()             25E536F0    0000005A
foo5()             25E53768    0000005A
foo4()             25E537E0    0000005A
foo3()             25E53858    0000005A
foo2()             25E538D0    0000005A
foo1()             25E53948    0000005A
thread             25E53A50    00000000

Timestamp: 2008/03/14          14:10:22.614032
Type: FREE Cell Address: 25E91C58 CpuId: 01 Tcb: 008AFA60
CALL NAME          CALL ADDRESS    CALL OFFSET
GetStorage::~~GetStorage()  25E53360    00000088
foo9()             25E53430    000000B6
foo8()             25E53598    0000009A
foo7()             25E53678    0000005A
foo6()             25E536F0    0000005A
foo5()             25E53768    0000005A
foo4()             25E537E0    0000005A
foo3()             25E53858    0000005A
foo2()             25E538D0    0000005A
foo1()             25E53948    00000000

Timestamp: 2008/03/14          14:10:22.614030
Type: GET Cell Address: 25E91C58 CpuId: 01 Tcb: 008AFA60
CALL NAME          CALL ADDRESS    CALL OFFSET
GetStorage::~GetStorage(int) 25E53298    0000008C
foo9()             25E53430    00000086
foo8()             25E53598    0000009A
foo7()             25E53678    0000005A
foo6()             25E536F0    0000005A
foo5()             25E53768    0000005A
foo4()             25E537E0    0000005A
foo3()             25E53858    0000005A
foo2()             25E538D0    0000005A
foo1()             25E53948    00000000

```

Figure 17. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 2 of 4)

```

Timestamp: 2008/03/14      14:10:22.614029
Type: GET Cell Address: 25E91BD0 CpuId: 01 Tcb: 008AFA60
CALL NAME      CALL ADDRESS      CALL OFFSET
GetStorage::GetStorage(int)  25E53298      0000008C
foo8()         25E53598      00000086
foo7()         25E53678      0000005A
foo6()         25E536F0      0000005A
foo5()         25E53768      0000005A
foo4()         25E537E0      0000005A
foo3()         25E53858      0000005A
foo2()         25E538D0      0000005A
foo1()         25E53948      0000005A
thread        25E53A50      00000000

Timestamp: 2008/03/14      14:10:22.612412
Type: GET Cell Address: 25E91B48 CpuId: 01 Tcb: 008AFCF0
CALL NAME      CALL ADDRESS      CALL OFFSET
GetStorage::GetStorage(int)  25E53298      0000008C
foo9()         25E53430      00000086
foo8()         25E53598      0000009A
foo7()         25E53678      0000005A
foo6()         25E536F0      0000005A
foo5()         25E53768      0000005A
foo4()         25E537E0      0000005A
foo3()         25E53858      0000005A
foo2()         25E538D0      0000005A
foo1()         25E53948      00000000

Timestamp: 2008/03/14      14:10:22.612410
Type: GET Cell Address: 25E91AC0 CpuId: 01 Tcb: 008AFCF0
CALL NAME      CALL ADDRESS      CALL OFFSET
GetStorage::GetStorage(int)  25E53298      0000008C
foo8()         25E53598      00000086
foo7()         25E53678      0000005A
foo6()         25E536F0      0000005A
foo5()         25E53768      0000005A
foo4()         25E537E0      0000005A
foo3()         25E53858      0000005A
foo2()         25E538D0      0000005A
foo1()         25E53948      0000005A
thread        25E53A50      00000000

Timestamp: 2008/03/14      14:10:22.593976
Type: GET Cell Address: 25E91A38 CpuId: 01 Tcb: 008AFE88
CALL NAME      CALL ADDRESS      CALL OFFSET
CEEOPMI        0601F218      00000822
CEEOPC         0600C2C8      00000CEE
pthread_create  0658FE40      00000632
main           25E53B00      000000EE
EDCZMINV       064C2106      00000000

```

Figure 17. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 3 of 4)

```

Timestamp: 2008/03/14      14:10:22.557633
Type: GET   Cell Address: 25E919B0 CpuId: 01 Tcb: 008AFE88
CALL NAME   CALL ADDRESS   CALL OFFSET
CEEOPMI     0601F218     00000822
pthread_mutex_init 06428B90     00000094
pthread_create 0658FE40     000002FC
main        25E53B00     000000EE
EDCZMINV    064C2106     00000000

Timestamp: 2008/03/14      14:10:22.551547
Type: GET   Cell Address: 25E91928 CpuId: 01 Tcb: 008AFE88
CALL NAME   CALL ADDRESS   CALL OFFSET
CEEOPMI     0601F218     00000822
pthread_mutex_init 06428B90     00000094
pthread_create 0658FE40     0000026C
main        25E53B00     000000EE
EDCZMINV    064C2106     00000000

Timestamp: 2008/03/14      14:10:22.544328
Type: GET   Cell Address: 25E918A0 CpuId: 01 Tcb: 008AFE88
CALL NAME   CALL ADDRESS   CALL OFFSET
dllinit     0622FBF8     0000009E
CEEZIDT     060C4B08     00000000
Exiting Language Environment Data

```

Figure 17. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 4 of 4)

[1] Trace Header

HEAPPOOLS trace header information.

[2] Pool Information

Information includes the number of the pool (POOLID) that 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 18 on page 117 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 138 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: 20C13A10
+00007C OS_SPCTYPE:00000000 CGENE:20C10B90 CRENT:20C14FF8
+0001F8 CFLTINIT:4E000000 00000000 CPRMS:20C127A8 TRACE:00000000
+000208 CTHD:20C0E8A0 CURR_FECB:20C101D8 CEDCXV:A0DF66D8
+000214 CGEN_CPCB:20C0DEA8 CGEN_CEDB:20C0F938 CFLG3:00
+000220 CIO:20C0E098 FDSETFD:00000000 FCB_MUTEXOK:0000
+00022C T_C16:00000000 T_C17:00000000 CEDCOV:2117A9A8
+000238 CTOSV:00000000 TRTSPACE:20C0EE50

[2]CGENE: 20C10B90
+000000 CGENEYE:GENE CGENESIZE:000006E0 CGENEPTR:20C10B90
+00000D CERRNO:00000000 TEMPLONG:00000000 AMRC:20C0E770
+000104 STDINFILE:20C0F718 STDOUTFILE:20C0F2D8
+00010C STDERRFILE:20C0F4F8 CTYPE:21035412 LC_CTYPE:21035412
+000124 LC_CHARMAP:21035F00
+000500 MIN_FLT:00100000 00000000 00000000 00000000
+000510 MAX_FLT:7FFFFFFF FFFFFFFF 71FFFFFF FFFFFFFF
+000520 FLT_EPS:3C100000 DBL_EPS:34100000 00000000
+000530 LDLB_EPS:26100000 00000000 18000000 00000000
+000544 IMSPCBLIST:20C145A4 ADDRIBL:20C0EC28
+0006D4 ABND_CODE:00000000 RSN_CODE:00000000

[3]CEDB: 20C0F938
+000000 EYE:CEDB SIZE:000007C0 PTR:20C0F938 CLLST:20C00568
+000010 CEELANG:0003 CASWITCH:0000 CLWA:20C11270
+000018 CALTLWA:20C115C0 CCADDR:20C00A10 CFLGS:00000080
+000024 CEESTART:20C00000 CANCHOR:00000000 RPLEN:00000000
+000030 ACBLEN:00000000 LC:20C10100 VALID_HIGH:210306C0
+00003C _LOW:2102FBEB HEAD_FECB:00000000 ATEXIT_COUNT:00000000
+000048 _EMPTY_COUNT:20C0FA10 MAINPRMS:2132D768
+000050 STDINFILE:20C0F718 STDOUTFILE:20C0F2D8
+000058 STDERRFILE:20C0F4F8 CTYPE:21035412 TZDFLT:00003840
+000064 CINFO:20C10190 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 LDLB_EPS:26100000 00000000 18000000 00000000 FLAGS1:02080000
+0000B4 MTF_MAINTASK_BLK:00000000 EMSG_SETTING:00 DEPTH:00000000
+0000C0 SCREEN_WIDTH:00000000 USERID:CHARUM .
+0000CC HEAP24_ANCHOR:00000000 TCIC:00000000 TKCLI:00000000
+0000D8 ATEXIT_FUNCS01:20C0FA24 00000000 00000000 00000000 00000000
+0000EC ATEXIT_FUNCS02:20C0FA38 00000000 00000000 00000000 00000000
+000100 ATEXIT_FUNCS03:20C0FA4C 00000000 00000000 00000000 00000000
+000114 ATEXIT_FUNCS04:20C0FA60 00000000 00000000 00000000 00000000
+000128 ATEXIT_FUNCS05:20C0FA74 00000000 00000000 00000000 00000000
+00013C ATEXIT_FUNCS06:20C0FA88 00000000 00000000 00000000 00000000
+000150 ATEXIT_FUNCS07:20C0FA9C 00000000 00000000 00000000 00000000
+000164 ATEXIT_FUNCS08:20C0FAB0 00000000 00000000 00000000 00000000
+000178 ATEXIT_FUNCS09:20C0FAC4 00000000 00000000 00000000 00000000
+00018C ATEXIT_FUNCS10:20C0FAD8 00000000 00000000 00000000 00000000
+0001A0 ATEXIT_FUNCS11:20C0FAEC 00000000 00000000 00000000 00000000
+0001B4 ATEXIT_FUNCS12:20C0FB00 00000000 00000000 00000000 00000000
+0001C8 ATEXIT_FUNCS13:20C0FB14 00000000 00000000 00000000 00000000
+0001DC ATEXIT_FUNCS14:20C0FB28 00000000 00000000 00000000 00000000
+0001F0 ATEXIT_FUNCS15:20C0FB3C 00000000 00000000 00000000 00000000
+000204 ATEXIT_FUNCS16:20C0FB50 00000000 00000000 00000000 00000000
+000218 ATEXIT_FUNCS17:20C0FB64 00000000 00000000 00000000 00000000
+00022C ATEXIT_FUNCS18:20C0FB78 00000000 00000000 00000000 00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 1 of 22)

```

+000240 ATEXTIT_FUNCS19:20C0FB8C 00000000 00000000 00000000 00000000
+000254 ATEXTIT_FUNCS20:20C0FBA0 00000000 00000000 00000000 00000000
+000268 ATEXTIT_FUNCS21:20C0FBB4 00000000 00000000 00000000 00000000
+00027C ATEXTIT_FUNCS22:20C0FBC8 00000000 00000000 00000000 00000000
+000290 ATEXTIT_FUNCS23:20C0FBD8 00000000 00000000 00000000 00000000
+0002A4 ATEXTIT_FUNCS24:20C0FBF0 00000000 00000000 00000000 00000000
+0002B8 ATEXTIT_FUNCS25:20C0FC04 00000000 00000000 00000000 00000000
+0002CC ATEXTIT_FUNCS26:20C0FC18 00000000 00000000 00000000 00000000
+0002E0 ATEXTIT_FUNCS27:20C0FC2C 00000000 00000000 00000000 00000000
+0002F4 ATEXTIT_FUNCS28:20C0FC40 00000000 00000000 00000000 00000000
+000308 ATEXTIT_FUNCS29:20C0FC54 00000000 00000000 00000000 00000000
+00031C ATEXTIT_FUNCS30:20C0FC68 00000000 00000000 00000000 00000000
+000330 ATEXTIT_FUNCS31:20C0FC7C 00000000 00000000 00000000 00000000
+000344 ATEXTIT_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 CDLL:21351CE0
+000370 INSPECT_GLOBALS:00000000 _JMP_BUFF:00000000
+000378 _BACK_END:00000000 _FLAGS:00000000 _TAB:00000000
+000384 _INTOFFLIST:00000000 _CGEN_CRENT:20C14FF8 _CPRMS:20C127A8
+000390 _CEDCXV:A0DF66D8 _CEDCOV:2117A9A8
+000398 _EPCBLIST:00000000 _CAA_ADDR:20C13A10
+0003A0 CDLL_PENDING:00000000 USERIDLENGTH:00000006
+0003A8 TZSHR_1:20C102B8 TZSHR_2:20C102BC
+0003B0 MAXUNGETCOUNT:0004 DLL_DIAG_FLAG:04 RWSTATIC:20C14FF8
+0003B8 RWLEN:00000008 CSGDLLI:00000000 DLLISIZE:00000000
+0003C4 IOGET_ANY:213244E8 _BELOW:213240B8 IOFREE_ANY:21323D80
+0003D0 _BELOW:21323A48 CSGSTINIT:00000000 STINITSIZE:00000000
+0003DC MTFMAINTASKBLK:00000000 SIGTABLE:20C103B8
+0003E4 INIT_STDIN:20C0F718 _STDOUT:20C0F2D8
+0003EC _STDERR:20C0F4F8 _TABNUM:00000008 FLAGS2:00000000
+000400 OPENMVS_FLAGS:00 MRPSTDR:2100AA80 MWPSTDR:2100A8A8
+00040C MRPSTDC:21009EE0 MWPSTDC:21009D08 OWRP1:00000000
+000418 OWRP3:00000000 STATIC_EDCOV:00000000 GETENV_BUF2:00000000
+000424 _BUF2_LEN:00000000 _DLCB_MUTEX:00000000 _CONDV:00000000
+000430 EDICOV:00000000 LCX:20C102B8 _MUTEX_ATTR:00000000
+00043C STOR_INIT_B:00001000 _INCR_B:00001000
+000444 STOR_INIT:00003000 _INCR:00002000 DEMANGLE:00000000
+000454 TEMPR15:00000000 _TERMINATE:210FC578
+00045C CXX_INV:00000000 D4_JOIN_MUTEX_ATTR:00000000
+000468 _MUTEX:00000000 _CONDV_ATTR:00000000 _CONDV:00000000
+000474 DLLANCHOR:00000000 DLLLAST:00000000 MEM24P:20C145A8
+000480 RTLMUTEX_ARRAYPTR:00000000 MSGCATLIST:00000000
+000488 SRCHP:00000000 ETOAP:00000000 ATOEP:00000000
+000494 NDMGMTP:00000000 POPENP:00000000 RND48P:00000000
+0004A0 BRK_HEAPID:00000000 _START:00000000 _CURRENT:00000000
+0004AC _END:00000000 RESTARTTABLE:00000000 SYSLOGP:00000000
+0004BC LOGIN_NAME:..... PREV_UMASK_VAL:00000000
+0004D0 HFP_LDBL_LMS:00100000 00000000 00000000 00000000 7FFFFFFF FFFFFFFF 71FFFFFF FFFFFFFF 26
+000500 BFP_LDBL_LMS:00010000 00000000 00000000 00000000 7FEFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 3F
+000590 HFP_DBL_LMS:00100000 00000000 7FFFFFFF FFFFFFFF 34100000 00000000
+0005A8 BFP_DBL_LMS:00100000 00000000 7FEFFFFF FFFFFFFF 3CB00000 00000000 7FF00000 00000000 FF
+0005F0 HFP_MHDC:412B7E15 1628AED3 41171547 652B82FE 406F2DEC 549B9439 40B17217 F7D1CF7A 41
+000660 BFP_MHDC:4005BF0A 8B145769 3FF71547 652B82FE 3FDBC7B 1526E50E 3FE62E42 FEFA39EF 40
+0006D0 HFP_FLT_LMS:00100000 7FFFFFFF 3C100000
+0006DC BFP_FLT_LMS:00800000 7F7FFFFF 34000000 7F800000 FF800000 7FA00000 FFA00000 7FC00000 FF
+000700 HFP_FHDC:00000010 006000E 001C0006 000F0020 FFC0FFC0 FFC0FFB2 FFB2FFB2 003F003F 00
+000728 BFP_FHDC:00010002 00180035 00710006 000F0021 FF83FC03 C003FFDB FECDECB0 00800400 40
+000750 ASCII_CC SID:0333 EBCDIC_CC SID:0417 LOGIN_NAME_A:.....
+000760 CINFO_A:00000000 LC C:20C108B0 _A:00000000
+00076C LCX_A:00000000 CORRESTABLE:00000000
+000774 TZSHR_A:00000000 00000000 CGENVEC31:20EA8500
+000780 CEDBVEC31:00000000 FORKQ_HEAD:00000000
+000788 FORKQ_TAIL:00000000 PTHREAD_YIELD1:0001D100
+000790 PTHREAD_YIELD2:00002E80 ICONV_MODE:U _TECH:LMREC....
+00079E _USERSET:D

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 2 of 22)

```

[4]CTHD: 20C0E8A0
+000000 CTHDEYE:CTHD      SIZE:00000380    CTHDPTR:20C0E8A0
+00000C STORPTR:00000000    TOKPTR:2102F210
+000014 ASCTIME_RESULT:.....
+00002E SNAP_DUMP_FLAG:00    GMTIME_BKDN:20C0EE28
+000034 TIMECALLED:00000000    DATECALLED:00000000
+00003C DTCALLED:00000000    LOC_CALLED:00000000
+000044 DOFMTO_DISCARDS:00000000    CERRNO:00000031    AMRC:20C0E770
+000050 AMRC2:20C0E878    GDATE:00000000    OPTARGV:00000000
+00005C OPTERRV:00000001    OPTINDV:00000001
+000064 OPTOPTV:00000000    OPTSIND:00000000    DLGHTV:00000000
+000070 TZONEV:00000000    GTDTERRV:00000000    OPTARGP:20C0E8F8
+00007C OPTERRP:20C0E8FC    OPTINDP:20C0E900
+000084 OPTOPTP:20C0E904    DLGHTP:20C0E90C    TZONEP:20C0E910
+000090 GTDTERRP:20C0E914    RNDSTGP:00000000
+000098 LOCNAME:00000000    ENCRYPTP:00000000    CRYPTP:00000000
+0000A4 RND48P:00000000    L64AP:00000000    WCSTOKP:00000000
+0000B0 CUSERP:00000000    GPASSP:00000000    UTMPXP:00000000
+0000BC NDMGMTP:00000000    RECOMP:00000000    STACKPTR:00000000
+0000C8 STACKSIZE:00000000    STACKFLAGS:40    THREAD_CHARMODE:E
+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
+000114 LOCS_V:00000000    HERRNOP:20C0E9B0    _LOC1P:00000000
+000124 REXECP:00000000    CXXEXCEPTION:20C0E560    TEMPDCBE:00000000
+000130 T_ERRNOV:20C0E9CC    T_ERRNOP:20C0E9E0    _LOC1_P:20C0E9E4
+00013C _LOC2_P:20C0E9B4    LOCS_P:00000000    _LOC1_V:00000000
+000148 _LOC2_V:00000000    THD_STORAGE:00000000    CONTEXT_LINK:00000000
+000154 FLAGS1:20C0EF50    LABEL_VAR:00000000    ABND_CODE:00000000
+000160 RSN_CODE:00000000    STRFTIME_ERADTCALLED:00000000
+000168 STRFTIME_ERADATECALLED:00000000
+00016C STRFTIME_ERATIMECALLED:00000000
+000170 STRFTIME_ERAYEARCALLED:00000000    MBRLEN_STATE:0000
+000176 MBRTOWC_STATE:0000    WCRSTOMB_STATE:0000
+00017A MBSRTOWCS_STATE:0000    WCSRSTOMBS_STATE:0000    MBLLEN_STATE:F0F0
+000180 MBTOWC_STATE:0000    SBSCS:..    CPTYPE:..
+000184 CURR_HEAP_ID:00000000    CURR_CAA:00000000
+00018C CURR_MOD_HANDLE:00000000    CURR_BMR:00000000
+000194 CU_LIST:00024040    CURR_STATUS:00    CURR_CASE:00
+00019C RAND_NEXT:20C0E2D8    STRERRORBUF:00000000
+0001A4 TMPAREA:20C0E638    IOWORKAREA:00012088
+0001AC TEMPDCB:000120E8    TEMPJFCB:20C0E598
+0001B4 TEMPDCSB:00000000    NAMEBUF:C00B0641
+0001BC ERRNO_JR:00000000    RET_STRUCT:00000000
+0001C4 BKDN_IS_LOCALTIME:00000000    SWPRINTF_SIZE:00000000
+0001CC SWPRINTF_BUF:20C0E540    S99P:20C0F1A8    MUTEXCTARRAY:00000000
+0001D8 STRFTIME_ERANAMECALLED:00000000    HPJRTL_1:00000000
+0001E0 HPJRTL_2:00000000    HPJRTL_3:00000000
+0001E8 HPJRTL_4:00000000    HPJRTL_5:00000000
+0001F0 HPJRTL_6:00000000    HPJRTL_7:00000000
+0001F8 HPJRTL_8:00000000    DLL_LOADLEVEL:00000000
+000200 DLL_CONSTLIST:00000000    FCB_MUTEX:20C0E280
+000208 HSPABHWA:20C0F204    MUTEX_SAVE:00012198
+000210 I024WORKAREA:4D000000    INITIAL_CPU_TIME:0000F603 00000001
+00021C FCB_MUTEX_OK:00000000    FCB_MUTEX_SAVE:00000000
+000224 ENTRY_ADDRTABLESIZE:00000000    ADDRESS:00000000
+00022C NUMBEROFNAMES:00000000    NAMES1:.....
+000249 NAMES2:.....
+000262 NAMES3:.....
+00027B NAMES4:.....
+000294 NAMES5:.....
+0002AD NAMES6:.....
+0002C8 ENTRY_SITETABLESIZE:00000000    KIND:00
+0002D0 NUM_ADDRS:00000000    ADDRESSES_1:00000000
+0002D8 ADDRESSES_2:00000000    ADDRESSES_3:00000000
+0002E0 ADDRESSES_4:00000000    ADDRESSES_5:00000000
+0002E8 ADDRESSES_6:00000000    NAME:.....{U.
+000304 T_STRErrorBUF:00000000    THD_ourfdset:00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 3 of 22)

```

+00030C IEEECWAP:00000000      FPC_SAVEAREA_TRAP:00
+000311 FPC_SAVEAREA_XCP:00      FPC_SAVEAREA_DXC:00
+000313 FPC_SAVEAREA_RMODE:00    LAST_YIELD_1:00000000
+000318 LAST_YIELD_2:00000000    YIELD_COUNT:00000000
+000320 RETVAL_P:00000000        CTHD_SETENV_FB:.....
+000338 LIBASCIIWAP:00000000    SETLOCALE_ACALLED:00
+00033E ASCIINAMEBUFL:0000      ASCIINAMEBUF:00000000
+00034C LOCNAME_A:00000000        EBCDICENTRY:00000000
+000354 ASCIIENTRY:00000000      CTHD_NASCIIWAP:00000000
+00035C CTHD_PROCRESP:00000000    CTHD_OPTN_P:00000000
+000364 CTHD_GAI_STRERROR_P:00000000
+000368 CTHD_CKPATHDD_PARAMS:00000000
[5]CPCB: 20C0DEA8
+000000 CPCB_EYE:CPCB      CPCB_SIZE:00000080      CPCB_PTR:00000000
+00000C FLAGS1:40000000    TTKNHDR:00000000    TTKN:00000000
+000018 FOOTPRINT:20C0F938    CODE370:00000000    CIO:20C0E098
+000024 _Reuse:00000000      _RSAbove:20C0DEA8    _RSAboveLen:00003FF0
+000030 _RSBelow:20C145A0      _RSBelowLen:00000010
[6]CIO: 20C0E098
+000000 EYE:CIO      SIZE:00000090      PTR:00000000      FLG1:09
+00000D FLG2:E0      FLG3:00      FLG4:00      DUMMYF:20C0E128
+000014 EDCZ24:00000000    FCBSTART:2135A040    DUMMYFCB:20C0E148
+000020 MFCBSTART:2135A470    IOANYLIST:2135A000
+000028 IOBELOWLIST:00014000    FCBDDLST:00000000
+000030 PERRORBUF:20C0DF60    TMPCOUNTER:00000000
+000038 TEMPMEM:00000000    PROMPTBUF:00000000    I024:000122D0
+000044 IOEXITS:000127C8    TERMINALCHAIN:00000000
+00004C VANCHOR:00000000    XTI:00000000      ENOWP24:20F484A0
+000058 MAXNUMDESCRPS:00000000    DESCARRAY:00000000
+000064 TEMPFILENUM:00000000    CSS:00000000      DUMMY_NAME:.....
+000074 HOSTNAME_CACHE:00000000    HOSTADDR_CACHE:00000000
+00007C IO31:20E62508      LAST_FD_CLOSE:00000000 00000000
+000088 IOGET64:213244E8      IOFREE64:21323D80

```

Exiting CRTL Environment Data

```

*****
                        CRTL I/O CONTROL BLOCKS
*****
      CIO: 20C0E098
+000000 EYE:CIO      SIZE:00000090      PTR:00000000      FLG1:09
+00000D FLG2:E0      FLG3:00      FLG4:00      DUMMYF:20C0E128
+000014 EDCZ24:00000000    FCBSTART:2135A040    DUMMYFCB:20C0E148
+000020 MFCBSTART:2135A470    IOANYLIST:2135A000
+000028 IOBELOWLIST:00014000    FCBDDLST:00000000
+000030 PERRORBUF:20C0DF60    TMPCOUNTER:00000000
+000038 TEMPMEM:00000000    PROMPTBUF:00000000    I024:000122D0
+000044 IOEXITS:000127C8    TERMINALCHAIN:00000000
+00004C VANCHOR:00000000    XTI:00000000      ENOWP24:20F484A0
+000058 MAXNUMDESCRPS:00000000    DESCARRAY:00000000
+000064 TEMPFILENUM:00000000    CSS:00000000      DUMMY_NAME:.....
+000074 HOSTNAME_CACHE:00000000    HOSTADDR_CACHE:00000000
+00007C IO31:20E62508      LAST_FD_CLOSE:00000000 00000000
+000088 IOGET64:213244E8      IOFREE64:21323D80

```

```

      FFIL: 2135A020
+000000 MARKER1:AFCB      FILE:00000000      _FP:2135A040
+00000C MARKER2:AFCBAFCB      FF_FLAGS:01000000
+000014 FCBMUTEX:00000000      THREADID:00000000 00000000

```

File name: /u/charum/b235/in.txt

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 4 of 22)

```

[7]FCB: 2135A040
+000000 BUFPTR:00000000 COUNTIN:00000000 COUNTOUT:00000000
+00000C READFUNC:2135A110 WRITEFUNC:2135A130 FLAGS1:0000
+000016 DEPTH:0000 NAME:2135A1FC _LENGTH:00000015
+000020 _BUFSIZE:00000044 MEMBER:..... NEXT:2135A268
+000030 PREV:00000000 PARENT:2135A040 CHILD:00000000
+00003C DDNAME:..... FD:00000000 DEVTYPE:09 FCBTYPE:007C
+00004C FSCE:2135A174 UNGETBUF:2135A174 REPOS:212FD080
+000058 GETPOS:212FCF50 CLOSE:21318C78 FLUSH:212FCEE8
+000064 UTILITY:21318870 USERBUF:00000000 LRECL:00000000
+000070 BLKSIZE:00000000 REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000 BUFSIZE:00001000 BUF:00000000
+000084 CURSOR:00000000 ENDOFDATA:00000000 SAVEDBUF:00000000
+000090 REALCOUNTIN:00000000 REALCOUNTOUT:00000000
+000098 POSMAJOR:FFFFFFFF SAVEMAJOR:00000000
+0000A0 POSMINOR:00000000 SAVEMINOR:00000000 STATE:0000
+0000AA SAVESTATE:0000 EXITFTELL:00000000 EXITUNGETC:00000000
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:00190040 00001300
+0000C8 DBCSSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:212FD418
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:212FD220
+0000FC WADDR_WSA:00000000 _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 WWSA:00000000

HFSF: 2135A174
+000000 HFSF_EYE:HFSF READ:2131A1F8 WRITE:213197A0
+00000C REPOS:21319198 GETPOS:21318EA8 FLUSH:21318A08
+000018 READBUFLEN:00000000 OPENFLAG:00000403 FLAG1:00000000
+000024 HFSF_ST_MODE:030001C0 HFSF_LAST_FSTAT:4DC12A2E C5C4ADF8

FFIL: 2135A248
+000000 MARKER1:AFCB FILE:00000000 _FP:2135A268
+00000C MARKER2:AFCBAFCB FF_FLAGS:01000000
+000014 FCBMUTEX:00000000 THREADID:00000000 00000000

File name: CHARUM.A.B

FCB: 2135A268
+000000 BUFPTR:00000000 COUNTIN:00000000 COUNTOUT:00000000
+00000C READFUNC:2135A338 WRITEFUNC:2135A358 FLAGS1:0000
+000016 DEPTH:0000 NAME:2135A424 _LENGTH:0000000A
+000020 _BUFSIZE:00000044 MEMBER:..... NEXT:2135A660
+000030 PREV:2135A040 PARENT:2135A268 CHILD:00000000
+00003C DDNAME:..... FD:FFFFFFFF DEVTYPE:08 FCBTYPE:0055
+00004C FSCE:2135A39C UNGETBUF:2135A39C REPOS:20F52B90
+000058 GETPOS:20FDD460 CLOSE:20FDD058 FLUSH:20F52AF8
+000064 UTILITY:20F3C030 USERBUF:00000000 LRECL:00000400
+000070 BLKSIZE:00000400 REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000 BUFSIZE:00000400 BUF:00000000
+000084 CURSOR:00000000 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:20F481E8
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:02020008 40000100
+0000C8 DBCSSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:20F48580
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:20F531A0
+0000FC WADDR_WSA:00000000 _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 WWSA:00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 5 of 22)

```

MEMO: 2135A39C
+000000 MEMO_EYE:MEMO      MFCB:2135A470      NEBULA:2135A4E4
+00000C NEBINDEX:0001      CURRDS:00000000    READ:20F48580
+000018 WRITE:20FDF820      REPOS:20FDD670     FLUSH:20FDCDE0

FFIL: 2135A640
+000000 MARKER1:AFCB      FILE:00000000      FP:2135A660
+00000C MARKER2:AFCBFCB    FF_FLAGS:01000000
+000014 FCBMUTEX:00000000    THREADID:00000000 00000000

File name: CHARUM.B.C

FCB: 2135A660
+000000 BUFPTR:00000000    COUNTIN:00000000    COUNTOUT:00000000
+00000C READFUNC:2135A730    WRITEFUNC:2135A750  FLAGS1:0000
+000016 DEPTH:0000  NAME:2135A81C  _LENGTH:0000000A
+000020 _BUFSIZE:00000044    MEMBER:.....     NEXT:2135AA60
+000030 PREV:2135A268      PARENT:2135A660    CHILD:00000000
+00003C DDNAME:.....     FD:FFFFFFFF        DEVTYPER:0A  FCBTYPE:0056
+00004C FSCE:2135A794      UNGETBUF:2135A794  REPOS:21313E38
+000058 GETPOS:21313AF0     CLOSE:213150A8     FLUSH:21313A70
+000064 UTILITY:20F3C030     USERBUF:00000000   LRECL:00001000
+000070 BLKSIZE:00001000    REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000  BUFSIZE:00001000   BUF:00000000
+000084 CURSOR:00000000    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:20F481E8
+0000B4 DBCSTART:00000000    UTILITYAREA:00000000
+0000BC INTERCEPT:00000000  FLAGS2:02020028 40004100
+0000C8 DBCSTATE:0000      FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000  READ:20F48580
+0000DC RADDR_WSA:00000000    _GETFN:00000000    RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000    RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000  WRITE:21314110
+0000FC WADDR_WSA:00000000    _GETFN:00000000    WDLL_INDEX:00000000
+000108 WCEESG003:00000000    WWSA:00000000

HSPF: 2135A794
+000000 HSPF_EYE:HSPF      MFCB:2135A970      READ:20F48580
+00000C WRITE:21316858     REPOS:21315650     GETPOS:21315470
+000018 FLUSH:21314A30    CURBUFSIZEUSED:00000000
+000020 LASTBLKOFFSET:00000000  HSPFARM:2135A868

FFIL: 2135AA40
+000000 MARKER1:AFCB      FILE:00000000      FP:2135AA60
+00000C MARKER2:AFCBFCB    FF_FLAGS:01000000
+000014 FCBMUTEX:00000000    THREADID:00000000 00000000

File name: CHARUM.C.D

FCB: 2135AA60
+000000 BUFPTR:00000000    COUNTIN:00000000    COUNTOUT:00000000
+00000C READFUNC:2135AB30    WRITEFUNC:2135AB50  FLAGS1:0000
+000016 DEPTH:0000  NAME:2135AC1C  _LENGTH:0000000A
+000020 _BUFSIZE:00000044    MEMBER:.....     NEXT:2135AD30
+000030 PREV:2135A660      PARENT:2135AA60    CHILD:00000000
+00003C DDNAME:SYS00011  FD:FFFFFFFF        DEVTYPER:00  FCBTYPE:0027
+00004C FSCE:2135AB94      UNGETBUF:2135AB94  REPOS:20F5F2A0
+000058 GETPOS:20F48330     CLOSE:20F7E1A0     FLUSH:20F5F3A8
+000064 UTILITY:20F5F160     USERBUF:00000000   LRECL:00000050
+000070 BLKSIZE:00000050    REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000  BUFSIZE:00000051   BUF:00000000
+000084 CURSOR:00000000    ENDOFDATA:00000000  SAVEDBUF:00000000
+000090 REALCOUNTIN:00000000  REALCOUNTOUT:00000000
+000098 POSMAJOR:00000000    SAVEMAJOR:00000000
+0000A0 POSMINOR:00000000    SAVEMINOR:00000000  STATE:0000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 6 of 22)

```

+0000AA SAVESTATE:0000 EXITFTELL:20F482A8 EXITUNGETC:20F481E8
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:02120020 40489100
+0000C8 DBCSSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:20F48580
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:20F7ED60
+0000FC WADDR_WSA:00000000 _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 WWSA:00000000

OSNS: 2135AB94
+000000 OSNS_EYE:OSNS READ:20F48580 WRITE:20F7E640
+00000C REPOS:20F5F588 GETPOS:20F48330 CLOSE:20F7E1A0
+000018 FLUSH:20F7E4C0 UTILITY:213126A0 EXITFTELL:20F482A8
+000024 EXITUNGETC:20F481E8 OSIOBLK:2135AC68
+00002C NEWLINEPTR:00000000 RECLENGTH:00000050 FLAGS:81000000

OSIO: 2135AC68
+000000 OSIO_EYE:OSIO DCBW:00012020 DCBRU:00000000
+00000C JFCB:000127E8 CURRMBUF:00000000 MBUFCOUNT:00000000
+000018 READMAX:00000000 CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF BLKSPERTRK:0000 OSIO_ACCESS_METHOD:02
+000027 OSIO_NOSEEK_TO_SEEK:00 FIRSTPOS:00000000
+00002C LASTPOS:00000000 NEWPOS:00000000 READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005 FCB:2135AA60 PARENT:2135AC68
+000044 FLAGS1:81000000 DCBERU:00000000 DCBEW:2135ACD0
+000050 OSIO_VOLSEQ:0000 OSIO_NEWVOLSEQ:0000 OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000 APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00012020
+000000 DCBRELAD:2135ACD0 DCBFDAD:00000000 D2000300
+000014 DCBBUFNO:05 DCBSRG1:40 DCBEODAD:000000 DCBRECFCM:80
+000025 DCBEXLSA:0127D0 DCBDDNAM:...&..'d DCBMACR1:A0
+000033 DCBMACR2:58 DCBSYNAD:000000 DCBBLKSI:0050 DCBNCP:21
+000052 DCBLRECL:0050

DCBE: 2135ACD0
+000000 DCBEID:DCBE DCBELEN:0038 RESERVED0:0000
+000008 DCBEDCB:00012020 DCBERELA:00000000 DCBEFLG1:C0
+000011 DCBEFLG2:88 DCBENSTR:0000 DCBEFLG3:80
+000024 DCBESIZE:00000000 DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C MULTSDN:00

JFCB: 000127E8
+000000 JFCBDSNM:CHARUM.C.D
+00002C JFCBELNM: JFCBTSDM:80 JFCBDSCB:01291C
+000046 JFCBVLSQ:0000 JFCBIND1:00 JFCBIND2:40
+000058 JFCBUFNO:00 JFCDSRG1:00 JFCDSRG2:00
+000064 JFCRECFCM:00 JFCBLKSI:0000 JFCLRECL:0000 JFCNCP:00
+000075 JFCBNVOL:01 JFCBNVOLS:SL4A00
+000094 JFCBEXTL:00 JFCBEXAD:0009AF JFCFLGS1:00
+0000AE JFCBVLCT:01

FFIL: 2135AD10
+000000 MARKER1:AFCB FILE:00000000 __FP:2135AD30
+00000C MARKER2:AFCBAFCB FF_FLAGS:01000000
+000014 FCBMUTEX:00000000 THREADID:00000000 00000000

File name: CHARUM.D.E

FCB: 2135AD30
+000000 BUFPTR:2135AFE0 COUNTIN:00000007 COUNTOUT:00000000
+00000C READFUNC:2135AE00 WRITEFUNC:2135AE20 FLAGS1:9000
+000016 DEPTH:0000 NAME:2135AECC LENGTH:0000000A
+000020 BUFSIZE:00000044 MEMBER:..... NEXT:2135B0A0
+000030 PREV:2135AA60 PARENT:2135AD30 CHILD:00000000
+00003C DDNAME:SYS00012 FD:FFFFFFFF DEVTYPE:00 FCBTYP:002A
+00004C FSCE:2135AE64 UNGETBUF:2135AE64 REPOS:20F87590

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 7 of 22)

```

+000058 GETPOS:20F89070 CLOSE:20F862E0 FLUSH:20F86D50
+000064 UTILITY:20F85D60 USERBUF:00000000 LRECL:00000050
+000070 BLKSIZE:00000050 REALBUFPTR:2135AFE0
+000078 UNGETCOUNT:00000000 BUFSIZE:00000051 BUF:2135AFE0
+000084 CURSOR:2135AFE0 ENDOFDATA:2135AFE0 SAVEDBUF:00000000
+000090 REALCOUNTIN:00000000 REALCOUNTOUT:00000050
+000098 POSMAJOR:00000000 SAVEMAJOR:00000000
+0000A0 POSMINOR:00000000 SAVEMINOR:00000000 STATE:0000
+0000AA SAVESTATE:0000 EXITFTELL:00000000 EXITUNGETC:00000000
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:021A0000 40440100
+0000C8 DBCSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:20F8C258
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:20F89A10
+0000FC WADDR_WSA:00000000 _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 WWSA:00000000

OSFS: 2135AE64
+000000 OSFS_EYE:OSFS READ:20F8C258 WRITE:20F89A10
+00000C REPOS:20F87590 GETPOS:20F89070 CLOSE:20F862E0
+000018 FLUSH:20F86D50 UTILITY:20F85D60 EXITFTELL:00000000
+000024 EXITUNGETC:00000000 OSIOBLK:2135AF38 SAVECURSOR:21
+00002D NEWLINEPTR:35AFE021 ENDOFBLOCK:35AFE600
+000035 CURRBLKSIZE:00000000 LASTBLKSIZE:00005000
+00003D HIGHMAJOR:00000000 RELRECNM:00000000
+000045 MAXFTELLBLK:00000001 RECBITS:FFFFFF00
+00004D RECSPERBLOCK:00000700 SAVECHAR:00000140 FLAGS1:72000000

OSIO: 2135AF38
+000000 OSIO_EYE:OSIO DCBW:000128A0 DCBRU:000129F8
+00000C JFCB:00012908 CURRMBUF:000129C0 MBUFCOUNT:00000001
+000018 READMAX:00000001 CURBLKNM:00000000
+000020 LASTBLKNM:00000257 BLKSPERTRK:0000 OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00 FIRSTPOS:00000100
+00002C LASTPOS:00073600 NEWPOS:00000000 READFUNCNUM:00000003
+000038 WRITEFUNCNUM:00000005 FCB:2135AD30 PARENT:2135AF38
+000044 FLAGS1:B5020000 DCBERU:2135B040 DCBEW:2135AFA0
+000050 OSIO_VOLSEQ:0000 OSIO_NEWVOLSEQ:0000 OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000 APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 000128A0
+000000 DCBRELAD:2135AFA0 DCBFDAD:02000000 3B000236
+000014 DCBBUFNO:00 DCBSRG1:40 DCBEODAD:000000 DCBREFCM:80
+000025 DCBEXLSA:0127D0 DCBDDNAM:.u....d DCBMACR1:65
+000033 DCBMACR2:B8 DCBSYNAD:000000 DCBBLKSI:0050 DCBNCP:01
+000052 DCBLRECL:0050

DCB: 000129F8
+000000 DCBRELAD:2135B040 DCBFDAD:00000000 1F000501
+000014 DCBBUFNO:00 DCBSRG1:40 DCBEODAD:000000 DCBREFCM:80
+000025 DCBEXLSA:0127D0 DCBDDNAM:.u...=u DCBMACR1:0A
+000033 DCBMACR2:F0 DCBSYNAD:000000 DCBBLKSI:0050 DCBNCP:01
+000052 DCBLRECL:0050

DCBE: 2135AFA0
+000000 DCBEID:DCBE DCBELEN:0038 RESERVED0:0000
+000008 DCBEDCB:000128A0 DCBERELA:00000000 DCBEFLG1:C0
+000011 DCBEFLG2:88 DCBENSTR:0000 DCBEFLG3:00
+000024 DCBESIZE:00000000 DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C MULTSDN:00

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 8 of 22)


```

DCBE: 2135B040
+000000 DCBEID:DCBE          DCBELEN:0038      RESERVED0:0000
+000008 DCBEDCB:000129F8      DCBERELA:00000000    DCBEFLG1:C0
+000011 DCBEFLG2:88          DCBENSTR:0000      DCBEFLG3:00
+000024 DCBESIZE:00000000      DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C      MULTSDN:00

JFCB: 00012908
+000000 JFCBDSNM:CHARUM.D.E
+00002C JFCBELNM:                JFCBTSDM:80          JFCBDSCB:003E15
+000046 JFCBVLSQ:0000          JFCBIND1:00         JFCBIND2:40
+000058 JFCBUFNO:00            JFCDSRG1:00         JFCDSRG2:00
+000064 JFCRECFM:00            JFCBLKSI:0000      JFCLRECL:0000    JFCNCP:00
+000075 JFCBNVOL:01            JFCBNVOLS:SL621D
+000094 JFCBEXTL:00          JFCBEXAD:0009FF    JFCFLGS1:00
+0000AE JFCBVLCT:01

MBUF: 000129C0
+000000 NEXTMBUF:000129C0      BUFFER:2135AFE0     CHECKRESULT:00000000
+00000C BLKSIZE:00000050
+000010 DECB:7F000000 80800000 000129F8 2135AFE0 00006AF8 00000000 00000000 00000000

FFIL: 2135B080
+000000 MARKER1:AFCB          FILE:00000000      FP:2135B0A0
+00000C MARKER2:AFCB         FF_FLAGS:01000000
+000014 FCBMUTEX:00000000      THREADID:00000000 00000000

File name: CHARUM.E.F

FCB: 2135B0A0
+000000 BUFPTR:00000000      COUNTIN:00000000    COUNTOUT:00000000
+00000C READFUNC:2135B170      WRITEFUNC:2135B190  FLAGS1:0000
+000016 DEPTH:0000      NAME:2135B25C      _LENGTH:0000000A
+000020 _BUFSIZE:00000044      MEMBER:.....      NEXT:2135B370
+000030 PREV:2135AD30          PARENT:2135B0A0    CHILD:00000000
+00003C DDNAME:SYS00013      FD:FFFFFFFF         DEVTYPE:00      FCBTYPE:0020
+00004C FSCE:2135B1D4      UNGETBUF:2135B1D4  REPOS:20F7CAFO
+000058 GETPOS:20F7CECO      CLOSE:20F77BE8     FLUSH:20F7CA78
+000064 UTILITY:213126A0        USERBUF:00000000    LRECL:00000404
+000070 BLKSIZE:00000408      REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000      BUFSIZE:00000408    BUF:00000000
+000084 CURSOR:00000000      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:20F481E8
+0000B4 DBCSTART:00000000      UTILITYAREA:00000000
+0000BC INTERCEPT:00000000      FLAGS2:02120020 20441100
+0000C8 DBCSTATE:0000      FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000      READ:20F48580
+0000DC RADDR_WSA:00000000      _GETFN:00000000    RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000      RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000      WRITE:20F7D260
+0000FC WADDR_WSA:00000000      _GETFN:00000000    WDLL_INDEX:00000000
+000108 WCEESG003:00000000      WWSA:00000000

OSVF: 2135B1D4
+000000 OSVF_EYE:OSVF          READ:20F48580      WRITE:20F7ADD8
+00000C REPOS:20F786B0      GETPOS:20F7A5C8    CLOSE:00000000
+000018 FLUSH:20F77F38      UTILITY:213126A0    EXITFTELL:00000000
+000024 EXITUNGETC:20F481E8      OSIOBLK:2135B2A8
+00002C SAVECURSOR:00000000      NEWLINEPTR:00000000
+000034 CURBLKSIZE:00000000      LASTBLKSIZE:00000000
+00003C HIGHMAJOR:00000000      MAXFTELLBLK:001FFFFF
+000044 RECBITS:00000000B      FLAGS:72000000     RECLEN:00000000
+000050 CURRBLKNUM:00000000      LASTBLKNUM:00000000
+000058 OSWRITETOK:00000000      RECCOUNTOUT1:00000000
+000060 BLKCOUNTOUT1:00000000      SAVECOUNTOUT:00000000  SDWCODE:0000
+00006A SAVECHAR:.. LASTTTTBLT:00000000      LASTWRTSIZE:00000000
+000073 LRECLUSED:00000000      OLDLRECLUSED:00000000
+00007B READERCOUNT:00000000      UNWRITTENDATA:00000000
+000083 MINRECLEN:00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 9 of 22)

```

OSIO: 2135B2A8
+000000 OSIO_EYE:OSIO      DCBW:00012A60      DCBRU:00000000
+00000C JFCB:00012AC8      CURRMBUF:00000000      MBUFCOUNT:00000001
+000018 READMAX:00000000      CURBLKNUM:FFFFFFF
+000020 LASTBLKNUM:FFFFFFF      BLKSPERTRK:0000      OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00      FIRSTPOS:00000100
+00002C LASTPOS:00000100      NEWPOS:00000000      READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005      FCB:2135B0A0      PARENT:2135B2A8
+000044 FLAGS1:81020000      DCBERU:00000000      DCBEW:2135B310
+000050 OSIO_VOLSEQ:0000      OSIO_NEWVOLSEQ:0000      OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000      APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00012A60
+000000 DCBRELAD:2135B310      DCBFDAD:00000000      01000D00
+000014 DCBBUFNO:00      DCBSRG1:40      DCBEODAD:000000      DCBRCFM:40
+000025 DCBEXLSA:0127D0      DCBDDNAM:.....      DCBMACR1:65
+000033 DCBMACR2:B8      DCBSYNAD:000000      DCBBLKSI:0408      DCBNCP:01
+000052 DCBLRECL:0404

DCBE: 2135B310
+000000 DCBEID:DCBE      DCBELEN:0038      RESERVED0:0000
+000008 DCBEDCB:00012A60      DCBERELA:00000000      DCBEFLG1:C0
+000011 DCBEFLG2:88      DCBENSTR:0000      DCBEFLG3:00
+000024 DCBESIZE:00000000      DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C      MULTSDN:00

JFCB: 00012AC8
+000000 JFCBDSNM:CHARUM.E.F
+00002C JFCBELNM:      JFCBTSDM:80      JFCBDSCB:004B17
+000046 JFCBVLSQ:0000      JFCBIND1:00      JFCBIND2:40
+000058 JFCBUFNO:00      JFCDSRG1:00      JFCDSRG2:00
+000064 JFCRCFM:00      JFCBLKSI:0000      JFCLRECL:0000      JFCNCP:00
+000075 JFCBNVOL:01      JFCBNVOLS:SL880B
+000094 JFCBEXTL:00      JFCBEXAD:000A4F      JFCFLGS1:00
+0000AE JFCBVLCT:01

FFIL: 2135B350
+000000 MARKER1:AFCB      FILE:00000000      FP:2135B370
+00000C MARKER2:AFCBAFCB      FF_FLAGS:01000000
+000014 FCBMUTEX:00000000      THREADID:00000000      00000000

File name: CHARUM.F.G

FCB: 2135B370
+000000 BUFPTR:00000000      COUNTIN:00000000      COUNTOUT:00000000
+00000C READFUNC:2135B440      WRITEFUNC:2135B460      FLAGS1:0000
+000016 DEPTH:0000      NAME:2135B52C      _LENGTH:0000000A
+000020 _BUFSIZE:00000044      MEMBER:.....      NEXT:2135B640
+000030 PREV:2135B0A0      PARENT:2135B370      CHILD:00000000
+00003C DDNAME:SYS00014      FD:FFFFFFF      DEVTYPE:00      FCBTYPE:0028
+00004C FSCE:2135B4A4      UNGETBUF:2135B4A4      REPOS:20F590C0
+000058 GETPOS:20F59368      CLOSE:20F804A0      FLUSH:20F59048
+000064 UTILITY:20F58D70      USERBUF:00000000      LRECL:00000000
+000070 BLKSIZE:00001800      REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000      BUFSIZE:00001800      BUF:00000000
+000084 CURSOR:00000000      ENDOFDATA:00000000      SAVEDBUF:00000000
+000090 REALCOUNTIN:00000000      REALCOUNTOUT:00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 10 of 22)

```

+000098 POSMAJOR:00000000 SAVEMAJOR:00000000
+0000A0 POSMINOR:00000000 SAVEMINOR:00000000 STATE:0000
+0000AA SAVESTATE:0000 EXITFTELL:00000000 EXITUNGETC:20F481E8
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:02120000 60441100
+0000C8 DBCSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:20F48580
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:20F595D8
+0000FC WADDR_WSA:00000000 _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 WWSA:00000000

OSFS: 2135B4A4
+000000 OSFS_EYE:OSFS READ:20F48580 WRITE:20F83078
+00000C REPOS:20F80CC0 GETPOS:20F826B8 CLOSE:00000000
+000018 FLUSH:20F808B0 UTILITY:213126A0 EXITFTELL:00000000
+000024 EXITUNGETC:20F481E8 OSIOBLK:2135B578 SAVECURSOR:00
+00002D NEWLINEPTR:00000000 ENDOFBLOCK:00000000
+000035 CURRBLKSIZE:00000000 LASTBLKSIZE:00000000
+00003D HIGHMAJOR:00000000 RELRECNUM:00000000
+000045 MAXFTELLBLK:00000000 RECBITS:07FFFF00
+00004D RECSPERBLOCK:00000D00 SAVECHAR:00000000 FLAGS1:72000000

OSIO: 2135B578
+000000 OSIO_EYE:OSIO DCBW:00012B80 DCBRU:00000000
+00000C JFCB:00012BE8 CURRMBUF:00000000 MBUFCOUNT:00000001
+000018 READMAX:00000000 CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF BLKSPERTRK:0000 OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00 FIRSTPOS:00000100
+00002C LASTPOS:00000100 NEWPOS:00000000 READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005 FCB:2135B370 PARENT:2135B578
+000044 FLAGS1:81020000 DCBERU:00000000 DCBEW:2135B5E0
+000050 OSIO_VOLSEQ:0000 OSIO_NEWVOLSEQ:0000 OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000 APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00012B80
+000000 DCBRELAD:2135B5E0 DCBFDAD:00000000 EA000800
+000014 DCBBUFNO:00 DCBSRG1:40 DCBEODAD:000000 DCBRECFCM:C0
+000025 DCBEXLSA:0127D0 DCBDDNAM:.....fd DCBMACR1:65
+000033 DCBMACR2:B8 DCBSYNAD:000000 DCBBLKSI:1800 DCBNCP:01
+000052 DCBLRECL:0000

DCBE: 2135B5E0
+000000 DCBEID:DCBE DCBELEN:0038 RESERVED0:0000
+000008 DCBEDCB:00012B80 DCBERELA:00000000 DCBEFLG1:C0
+000011 DCBEFLG2:88 DCBENSTR:0000 DCBEFLG3:00
+000024 DCBESIZE:00000000 DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C MULTSDN:00

JFCB: 00012BE8
+000000 JFCBDSNM:CHARUM.F.G
+00002C JFCBELNM: JFCBTSDM:80 JFCBDSCB:01402A
+000046 JFCBVLSQ:0000 JFCBIND1:00 JFCBIND2:40
+000058 JFCBUFNO:00 JFCDSRG1:00 JFCDSRG2:00
+000064 JFCRECFM:00 JFCBLKSI:0000 JFCLRECL:0000 JFCNCP:00
+000075 JFCBNVOL:01 JFCBNVOLS:SL4C06
+000094 JFCBEXTL:00 JFCBEXAD:000A9F JFCFLGS1:00
+0000AE JFCBVLCT:01

FFIL: 2135B620
+000000 MARKER1:AFCB FILE:00000000 _FP:2135B640
+00000C MARKER2:AFCBAFCB FF_FLAGS:01000000
+000014 FCBMUTEX:00000000 THREADID:00000000 00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 11 of 22)

File name: CHARUM.G.H

```
FCB: 2135B640
+000000 BUFPTR:00000000 COUNTIN:00000000 COUNTOUT:00000000
+00000C READFUNC:2135B710 WRITEFUNC:2135B730 FLAGS1:8000
+000016 DEPTH:0000 NAME:2135B7FC _LENGTH:0000000A
+000020 _BUFSIZE:00000044 MEMBER:..... NEXT:2135B910
+000030 PREV:2135B370 PARENT:2135B640 CHILD:00000000
+00003C DDNAME:SYS00015 FD:FFFFFFFF DEVTYPE:00 FCBTYPE:0041
+00004C FSCE:2135B774 UNGETBUF:2135B774 REPOS:20F5F2A0
+000058 GETPOS:20F48330 CLOSE:20FB8930 FLUSH:20F5F3A8
+000064 UTILITY:20F5F160 USERBUF:00000000 LRECL:00000050
+000070 BLKSIZE:00000050 REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000 BUFSIZE:00000051 BUF:00000000
+000084 CURSOR:00000000 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:20F482A8 EXITUNGETC:20F481E8
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:02120020 40488100
+0000C8 DBCSSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:20F48580
+0000DC RADDR_WSA:00000000_GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:20FBA568
+0000FC WADDR_WSA:00000000_GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 WWSA:00000000

OSNS: 2135B774
+000000 OSNS_EYE:OSNS READ:20F48580 WRITE:20FB8D88
+00000C REPOS:20F5F588 GETPOS:20F48330 CLOSE:20FB8930
+000018 FLUSH:20FB8CC0 UTILITY:20FB8600 EXITFTELL:20F482A8
+000024 EXITUNGETC:20F481E8 OSIOBLK:2135B848
+00002C NEWLINEPTR:00000000 RECLENGTH:00000050 FLAGS:81000000

OSIO: 2135B848
+000000 OSIO_EYE:OSIO DCBW:00012CA0 DCBRU:00000000
+00000C JFCB:00012D08 CURRMBUF:00000000 MBUFCOUNT:00000001
+000018 READMAX:00000000 CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF BLKSPERTRK:0000 OSIO_ACCESS_METHOD:02
+000027 OSIO_NOSEEK_TO_SEEK:00 FIRSTPOS:00000000
+00002C LASTPOS:00000000 NEWPOS:00000000 READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005 FCB:2135B640 PARENT:2135B848
+000044 FLAGS1:81000000 DCBERU:00000000 DCBEW:2135B8B0
+000050 OSIO_VOLSEQ:0000 OSIO_NEWVOLSEQ:0000 OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000 APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00012CA0
+000000 DCBRELAD:2135B8B0 DCBFDAD:00000000 F1000200
+000014 DCBBUFNO:05 DCBSRG1:40 DCBEODAD:00000000 DCBRECFM:80
+000025 DCBEXLSA:0127D0 DCBDDNAM:.\.&..a DCBMACR1:A0
+000033 DCBMACR2:58 DCBSYNAD:00000000 DCBBLKS1:0050 DCBNCP:21
+000052 DCBLRECL:0050

DCBE: 2135B8B0
+000000 DCBEID:DCBE DCBELEN:0038 RESERVED0:0000
+000008 DCBEDCB:00012CA0 DCBERELA:00000000 DCBEFLG1:C0
+000011 DCBEFLG2:88 DCBENSTR:0000 DCBEFLG3:80
+000024 DCBESIZE:00000000 DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C MULTSDN:00

JFCB: 00012D08
+000000 JFCBDSNM:CHARUM.G.H
+00002C JFCBELNM: JFCBTSDM:80 JFCBDCB:01391F
+000046 JFCBVLSQ:0000 JFCBIND1:00 JFCBIND2:40
+000058 JFCBUFNO:00 JFCDSRG1:00 JFCDSRG2:00
+000064 JFCRECFM:00 JFCBLKS1:0000 JFCFLRECL:0000 JFCNCP:00
+000075 JFCBNVOL:01 JFCBNVOLS:SL4A04
+000094 JFCBEXTL:00 JFCBEXAD:000AEF JFCFLGS1:00
+0000AE JFCBVLC:01
```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 12 of 22)

```

FFIL: 2135B8F0
+000000 MARKER1:AFCB      FILE:00000000      _FP:2135B910
+000000 MARKER2:AFCAFCB      FF_FLAGS:01000000
+000014 FCBMUTEX:00000000      THREADID:00000000 00000000

File name: CHARUM.H.I

FCB: 2135B910
+000000 BUFPTR:00000000  COUNTIN:00000000  COUNTOUT:00000000
+000000 READFUNC:2135B9E0  WRITEFUNC:2135BA00  FLAGS1:8000
+000016 DEPTH:0000  NAME:2135BACC  _LENGTH:0000000A
+000020 _BUFSIZE:000000044  MEMBER:.....  NEXT:2135BBE0
+000030 PREV:2135B640  PARENT:2135B910  CHILD:00000000
+00003C DDNAME:SYS00016  FD:FFFFFFFF  DEVTYPE:00  FCBTYPE:002A
+00004C FSCE:2135BA44  UNGETBUF:2135BA44  REPOS:20F5A418
+000058 GETPOS:20F5AAC8  CLOSE:20F862E0  FLUSH:20F5A3A0
+000064 UTILITY:20F85D60  USERBUF:00000000  LRECL:00000050
+000070 BLKSIZE:00000050  REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000  BUFSIZE:00000051  BUF:00000000
+000084 CURSOR:00000000  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:20F481E8
+0000B4 DBCSTART:00000000  UTILITYAREA:00000000
+0000BC INTERCEPT:00000000  FLAGS2:02120020 40440100
+0000C8 DBCSTATE:0000  FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000  READ:20F48580
+0000DC RADDR_WSA:00000000  _GETFN:00000000  RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000  RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000  WRITE:20F5B168
+0000FC WADDR_WSA:00000000  _GETFN:00000000  WDLL_INDEX:00000000
+000108 WCEESG003:00000000  WWSA:00000000

OSFS: 2135BA44
+000000 OSFS_EYE:OSFS  READ:20F48580  WRITE:20F89A10
+00000C REPOS:20F87590  GETPOS:20F89070  CLOSE:20F862E0
+000018 FLUSH:20F86D50  UTILITY:20F85D60  EXITFTELL:00000000
+000024 EXITUNGETC:20F481E8  OSIOBLK:2135BB18  SAVECURSOR:00
+00002D NEWLINEPTR:00000000  ENDOFBLOCK:00000000
+000035 CURRBLKSIZE:00000000  LASTBLKSIZE:00000000
+00003D HIGHMAJOR:00000000  RELRECNUM:00000000
+000045 MAXFTELLBLK:00000001  RECBITS:FFFFFFFF00
+00004D RECSPERBLOCK:00000700  SAVECHAR:00000100  FLAGS1:72000000

OSIO: 2135BB18
+000000 OSIO_EYE:OSIO  DCBW:00012DC0  DCBRU:00000000
+00000C JFCB:00012E28  CURRMBUF:00000000  MBUFCOUNT:00000001
+000018 READMAX:00000000  CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF  BLKSPERTRK:0000  OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00  FIRSTPOS:00000100
+00002C LASTPOS:00000100  NEWPOS:00000000  READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005  FCB:2135B910  PARENT:2135BB18
+000044 FLAGS1:85020000  DCBERU:00000000  DCBEW:2135BB80
+000050 OSIO_VOLSEQ:0000  OSIO_NEWVOLSEQ:0000  OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000  APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00012DC0
+000000 DCBRELAD:2135BB80  DCBFDAD:00000000 09000D00
+000014 DCBBUFNO:00  DCBSRG1:40  DCBEODAD:000000  DCBRECFCM:80
+000025 DCBEXLSA:0127D0  DCBDDNAM:.4...a.  DCBMACR1:65
+000033 DCBMACR2:B8  DCBSYNAD:000000  DCBBLKSI:0050  DCBNCP:01
+000052 DCBLRECL:0050

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 13 of 22)

```

DCBE: 2135BB80
+000000 DCBEID:DCBE DCBELEN:0038 RESERVED0:0000
+000008 DCBEDCB:00012DC0 DCBERELA:00000000 DCBEFLG1:C0
+000011 DCBEFLG2:88 DCBENSTR:0000 DCBEFLG3:00
+000024 DCBESIZE:00000000 DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C MULTSDN:00

JFCB: 00012E28
+000000 JFCBDSNM:CHARUM.H.I
+00002C JFCBELNM: JFCBTSDM:80 JFCBDSCB:002214
+000046 JFCBVLSQ:0000 JFCBIND1:00 JFCBIND2:40
+000058 JFCBUFNO:00 JFCDSRG1:00 JFCDSRG2:00
+000064 JFCRECFM:00 JFCBLKSI:0000 JFCLRECL:0000 JFCNCP:00
+000075 JFCBNVOL:01 JFCBNVOLS:SL8B14
+000094 JFCBEXTL:00 JFCBEXAD:000B3F JFCFLGS1:00
+0000AE JFCBVLCT:01

FFIL: 2135BBC0
+000000 MARKER1:AFCB FILE:00000000 _FP:2135BBE0
+00000C MARKER2:AFCAFCB FF_FLAGS:01000000
+000014 FCBMUTEX:00000000 THREADID:00000000 00000000

File name: CHARUM.I.J

FCB: 2135BBE0
+000000 BUFPTR:00000000 COUNTIN:00000000 COUNTOUT:00000000
+00000C READFUNC:2135BCB0 WRITEFUNC:2135BCD0 FLAGS1:8000
+000016 DEPTH:0000 NAME:2135BD9C _LENGTH:0000000A
+000020 _BUFSIZE:00000044 MEMBER:..... NEXT:2135BEB0
+000030 PREV:2135B910 PARENT:2135BBE0 CHILD:00000000
+00003C DDNAME:SYS00017 FD:FFFFFFFF DEVTYPE:00 FCBTYPE:0036
+00004C FSCE:2135BD14 UNGETBUF:2135BD14 REPOS:20F5BFD0
+000058 GETPOS:20F5C2E0 CLOSE:20FA16C0 FLUSH:20F5BF58
+000064 UTILITY:20FA0550 USERBUF:00000000 LRECL:00000404
+000070 BLKSIZE:00000408 REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000 BUFSIZE:00000409 BUF:00000000
+000084 CURSOR:00000000 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:20F481E8
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:02120020 20440100
+0000C8 DBCSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:20F48580
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 RWSA:00000000
+0000F0 WRITGLUE:58FF0008 07FF0000 WRITE:20F5C620
+0000FC WADDR_WSA:00000000 _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 WWSA:00000000

OSVF: 2135BD14
+000000 OSVF_EYE:OSVF READ:20F48580 WRITE:20FA8D58
+00000C REPOS:20FA4BD8 GETPOS:20FA84E0 CLOSE:20FA16C0
+000018 FLUSH:20FA3FD0 UTILITY:20FA0550 EXITFTELL:00000000
+000024 EXITUNGETC:20F481E8 OSIOBLK:2135BDE8
+00002C SAVECURSOR:00000000 NEWLINEPTR:7FFFFFFF
+000034 CURBLKSIZE:00000000 LASTBLKSIZE:00000000
+00003C HIGHMAJOR:00000000 MAXFTELLBLK:001FFFFFFF
+000044 RECBITS:0000000B FLAGS:73100000 RECLEN:00000000
+000050 CURRBLKNUM:00000000 LASTBLKNUM:00000000
+000058 OSWRITETOK:00000000 RECCOUNTOUT1:00000000
+000060 BLKCOUNTOUT1:00000000 SAVECOUNTOUT:00000000 SDWCODE:0000
+00006A SAVECHAR:.. LASTTTTRBLT:00000000 LASTWRTSIZE:00000000
+000073 LRECLUSED:00000000 OLDLRECLUSED:00000000
+00007B READERCOUNT:00000000 UNWRITTENDATA:00000000
+000083 MINRECLEN:00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 14 of 22)

```

OSIO: 2135BDE8
+000000 OSIO_EYE:OSIO      DCBW:00012EE0      DCBRU:00000000
+00000C JFCB:00012F48      CURRMBUF:00000000      MBOFCOUNT:00000001
+000018 READMAX:00000000      CURBLKNUM:FFFFFFF
+000020 LASTBLKNUM:FFFFFFF      BLKSPERTRK:0000      OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00      FIRSTPOS:00000100
+00002C LASTPOS:00000100      NEWPOS:00000000      READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005      FCB:2135BBE0      PARENT:2135BDE8
+000044 FLAGS1:81020000      DCBERU:00000000      DCBEW:2135BE50
+000050 OSIO_VOLSEQ:0000      OSIO_NEWVOLSEQ:0000      OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000      APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00012EE0
+000000 DCBRELAD:2135BE50      DCBFDAD:00000000      23000300
+000014 DCBBUFNO:00      DCBSRG1:40      DCBEODAD:000000      DCBRCFM:40
+000025 DCBEXLSA:0127D0      DCBDDNAM:.....bm      DCBMACR1:65
+000033 DCBMACR2:B8      DCBSYNAD:000000      DCBBLKSI:0408      DCBNCP:01
+000052 DCBLRECL:0404

DCBE: 2135BE50
+000000 DCBEID:DCBE      DCBELEN:0038      RESERVED0:0000
+000008 DCBEDCB:00012EE0      DCBERELA:00000000      DCBEFLG1:C0
+000011 DCBEFLG2:88      DCBENSTR:0000      DCBEFLG3:00
+000024 DCBESIZE:00000000      DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C      MULTSDN:00

JFCB: 00012F48
+000000 JFCBDSNM:CHARUM.I.J
+00002C JFCBELNM:      JFCBTSDM:80      JFCBDCB:002C16
+000046 JFCBVLSQ:0000      JFCBIND1:00      JFCBIND2:40
+000058 JFCBUFNO:00      JFCDSRG1:00      JFCDSRG2:00
+000064 JFCRCFM:00      JFCBLKSI:0000      JFCLRECL:0000      JFCNCP:00
+000075 JFCBNVOL:01      JFCBNVOLS:SL8D1E
+000094 JFCBEXTL:00      JFCBEXAD:000B8F      JFCFLG1:00
+0000AE JFCBVLCT:01

FFIL: 2135BE90
+000000 MARKER1:AFCB      FILE:00000000      FP:2135BEB0
+00000C MARKER2:AFCBAFCB      FF_FLAGS:01000000
+000014 FCBMUTEX:00000000      THREADID:00000000      00000000

File name: CHARUM.J.K

FCB: 2135BEB0
+000000 BUFPTR:00000000      COUNTIN:00000000      COUNTOUT:00000000
+00000C READFUNC:2135BF80      WRITEFUNC:2135BFA0      FLAGS1:8000
+000016 DEPTH:0000      NAME:2135C06C      _LENGTH:0000000A
+000020 _BUFSIZE:00000044      MEMBER:.....      NEXT:2135C180
+000030 PREV:2135BBE0      PARENT:2135BEB0      CHILD:00000000
+00003C DDNAME:SYS00018      FD:FFFFFFF      DEVTYPE:00      FCBTYP:0042
+00004C FSCE:2135BFE4      UNGETBUF:2135BFE4      REPOS:20F5D0A0
+000058 GETPOS:20F5D2D0      CLOSE:20FBBE0      FLUSH:20F5D028
+000064 UTILITY:20FBB708      USERBUF:00000000      LRECL:00000000
+000070 BLKSIZE:00001800      REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000      BUFSIZE:00001801      BUF:00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 15 of 22)

```

+000084 CURSOR:00000000 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:20F481E8
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:02220020 60440100
+0000C8 DBCSSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:20F48580
+0000DC RADDR_WSA:00000000 _GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000 RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:20F5D4E8
+0000FC WADDR_WSA:00000000 _GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000 WWSA:00000000

OSUT: 2135BFE4
+000000 OSUT_EYE:OSUT READ:20F48580 WRITE:20FBE8B0
+00000C REPOS:20FBCAD0 GETPOS:20FBDEC8 CLOSE:00000000
+000018 FLUSH:20FBC5C0 UTILITY:20FBB708 EXITFTELL:00000000
+000024 EXITUNGETC:20F481E8 OSIOBLK:2135C0B8
+00002C NEWLINEPTR:00000000 MAXFTELLBLK:0007FFFF
+000034 RECBITS:00000000 FLAGS:E0000000

OSIO: 2135C0B8
+000000 OSIO_EYE:OSIO DCBW:00014020 DCBRU:00000000
+00000C JFCB:00014088 CURRMBUF:00000000 MBUF_COUNT:00000001
+000018 READMAX:00000000 CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF BLKSPERTRK:0000 OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00 FIRSTPOS:00000100
+00002C LASTPOS:00000100 NEWPOS:00000000 READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005 FCB:2135BEB0 PARENT:2135C0B8
+000044 FLAGS1:81020000 DCBERU:00000000 DCBEW:2135C120
+000050 OSIO_VOLSEQ:0000 OSIO_NEWVOLSEQ:0000 OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000 APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00014020
+000000 DCBRELAD:2135C120 DCBFDAD:00000000 10000C00
+000014 DCBBUFNO:00 DCBSRG1:40 DCBEODAD:000000 DCBRECFCM:C0
+000025 DCBEXLSA:0127D0 DCBDDNAM:.....c. DCBMACR1:65
+000033 DCBMACR2:B8 DCBSYNAD:000000 DCBBLKS1:1800 DCBNCP:01
+000052 DCBLRECL:0000

DCBE: 2135C120
+000000 DCBEID:DCBE DCBELEN:0038 RESERVED0:0000
+000008 DCBEDCB:00014020 DCBERELA:00000000 DCBEFLG1:C0
+000011 DCBEFLG2:88 DCBENSTR:0000 DCBEFLG3:00
+000024 DCBESIZE:00000000 DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C MULTSDN:00

JFCB: 00014088
+000000 JFCBDSNM:CHARUM.J.K
+00002C JFCBELNM: JFCBTSDM:80 JFCBDCB:002B0D
+000046 JFCBVLSQ:0000 JFCBIND1:00 JFCBIND2:40
+000058 JFCBUFNO:00 JFCDSRG1:00 JFCDSRG2:00
+000064 JFCRECFM:00 JFCBLKS1:0000 JFCLRECL:0000 JFCNCP:00
+000075 JFCBNVOL:01 JFCBNVOLS:SL8D1C
+000094 JFCBEXTL:00 JFCBEXAD:000BDF JFCFLGS1:00
+0000AE JFCBVLCT:01

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 16 of 22)


```

FFIL: 2135C160
+000000 MARKER1:AFCB      FILE:00000000      _FP:2135C180
+000000 MARKER2:AFCB      FF_FLAGS:01000000
+000014 FCBMUTEX:00000000  THREADID:00000000 00000000

File name: CHARUM.K.L

FCB: 2135C180
+000000 BUFPTR:00000000  COUNTIN:00000000  COUNTOUT:00000000
+000000 READFUNC:2135C250  WRITEFUNC:2135C270  FLAGS1:0000
+000016 DEPTH:0000  NAME:2135C33C  _LENGTH:0000000A
+000020 _BUFSIZE:000000044  MEMBER:.....  NEXT:2135C450
+000030 PREV:2135BEB0  PARENT:2135C180  CHILD:00000000
+00003C DDNAME:SYS00019  FD:FFFFFFFF  DEVTYPE:00  FCBTYPE:0053
+00004C FSCE:2135C2B4  UNGETBUF:2135C2B4  REPOS:20F5F2A0
+000058 GETPOS:20F48330  CLOSE:20FD8658  FLUSH:20F5F3A8
+000064 UTILITY:20F5F160  USERBUF:00000000  LRECL:00000050
+000070 BLKSIZE:000000050  REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000  BUFSIZE:000000051  BUF:00000000
+000084 CURSOR:00000000  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:20F482A8  EXITUNGETC:20F481E8
+0000B4 DBCSTART:00000000  UTILITYAREA:00000000
+0000BC INTERCEPT:00000000  FLAGS2:02120020 404A9100
+0000C8 DBCSTATE:0000  FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000  READ:20F48580
+0000DC RADDR_WSA:00000000  _GETFN:00000000  RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000  RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000  WRITE:20FD8E88
+0000FC WADDR_WSA:00000000  _GETFN:00000000  WDLL_INDEX:00000000
+000108 WCEESG003:00000000  WWSA:00000000

OSNS: 2135C2B4
+000000 OSNS_EYE:OSNS  READ:20F48580  WRITE:20FD88B8
+00000C REPOS:20F5F588  GETPOS:20F48330  CLOSE:20FD8658
+000018 FLUSH:20FD87F0  UTILITY:213126A0  EXITFTELL:20F482A8
+000024 EXITUNGETC:20F481E8  OSIOBLK:2135C388
+00002C NEWLINEPTR:00000000  RECLENGTH:00000050  FLAGS:81000000

OSIO: 2135C388
+000000 OSIO_EYE:OSIO  DCBW:00014140  DCBRU:00000000
+00000C JFCB:000141A8  CURRMBUF:00000000  MBUF_COUNT:00000000
+000018 READMAX:00000000  CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF  BLKSPERTRK:0000  OSIO_ACCESS_METHOD:02
+000027 OSIO_NOSEEK_TO_SEEK:00  FIRSTPOS:00000000
+00002C LASTPOS:00000000  NEWPOS:00000000  READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005  FCB:2135C180  PARENT:2135C388
+000044 FLAGS1:81000000  DCBERU:00000000  DCBEW:2135C3F0
+000050 OSIO_VOLSEQ:0000  OSIO_NEWVOLSEQ:0000  OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000  APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00014140
+000000 DCBRELAD:2135C3F0  DCBFDAD:00000000 0A000A00
+000014 DCBBUFNO:05  DCBSRG1:40  DCBEODAD:000000  DCBREFCM:80
+000025 DCBEXLSA:0127D0  DCBDDNAM:...&..gM  DCBMACR1:A0
+000033 DCBMACR2:58  DCBSYNAD:000000  DCBBLKSI:0050  DCBNCP:21
+000052 DCBLRECL:0050

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 17 of 22)

```

DCBE: 2135C3F0
+000000 DCBEID:DCBE          DCBELEN:0038      RESERVED0:0000
+000008 DCBEDCB:00014140      DCBERELA:00000000      DCBEFLG1:C0
+000011 DCBEFLG2:88          DCBENSTR:0000      DCBEFLG3:80
+000024 DCBESIZE:00000000      DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C      MULTSDN:00

JFCB: 000141A8
+000000 JFCBDSNM:CHARUM.K.L
+00002C JFCBELNM:          JFCBTSDM:80          JFCBDSCB:004610
+000046 JFCBVLSQ:0000      JFCBIND1:00      JFCBIND2:40
+000058 JFCBUFNO:00          JFCDSRG1:00      JFCDSRG2:00
+000064 JFCRECFM:00          JFCBLKSI:0000      JFCLRECL:0000      JFCNCP:00
+000075 JFCBNVOL:01          JFCBNVOLS:SL8818
+000094 JFCBEXTL:00          JFCBEXAD:000C2F      JFCFLGS1:00
+0000AE JFCBVLCT:01

FFIL: 2135C430
+000000 MARKER1:AFCB          FILE:00000000      _FP:2135C450
+00000C MARKER2:AFCBAFCB      FF_FLAGS:01000000
+000014 FCBMUTEX:00000000      THREADID:00000000 00000000

File name: CHARUM.L.M

FCB: 2135C450
+000000 BUFPTR:00000000      COUNTIN:00000000      COUNTOUT:00000000
+00000C READFUNC:2135C520      WRITEFUNC:2135C540      FLAGS1:0000
+000016 DEPTH:0000      NAME:2135C60C      _LENGTH:0000000A
+000020 _BUFSIZE:00000044      MEMBER:.....      NEXT:2135C720
+000030 PREV:2135C180      PARENT:2135C450      CHILD:00000000
+00003C DDNAME:SYS00020      FD:FFFFFFFF      DEVTYPE:00      FCBTYPE:0046
+00004C FSCE:2135C584      UNGETBUF:2135C584      REPOS:20FD0D10
+000058 GETPOS:20FD1240      CLOSE:20FCD070      FLUSH:20FD0CB0
+000064 UTILITY:20FD07B0      USERBUF:00000000      LRECL:00000050
+000070 BLKSIZE:00000050      REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000      BUFSIZE:00000050      BUF:00000000
+000084 CURSOR:00000000      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:20F481E8
+0000B4 DBCSTART:00000000      UTILITYAREA:00000000
+0000BC INTERCEPT:00000000      FLAGS2:00120020 40461100
+0000C8 DBCSSSTATE:0000      FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000      READ:20F48580
+0000DC RADDR_WSA:00000000      _GETFN:00000000      RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000      _RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000      WRITE:20FD16E0
+0000FC WADDR_WSA:00000000      _GETFN:00000000      WDLL_INDEX:00000000
+000108 WCEESG003:00000000      _WWSA:00000000

OSFS: 2135C584
+000000 OSFS_EYE:OSFS          READ:20F48580      WRITE:20FCFA28
+00000C REPOS:20FCD838      GETPOS:20FCF4C8      CLOSE:00000000
+000018 FLUSH:20FCD3E8      UTILITY:213126A0      EXITFTELL:00000000
+000024 EXITUNGETC:20F481E8      OSIOBLK:2135C658      SAVECURSOR:00
+00002D NEWLINEPTR:00000000      ENDOFBLOCK:00000000
+000035 CURRBLKSIZE:00000000      LASTBLKSIZE:00000000
+00003D HIGHMAJOR:00000000      RELRECNUM:00000000
+000045 MAXFTELLBLK:00000000      RECBITS:00000000
+00004D RECSPERBLOCK:00000000      SAVECHAR:00000100      FLAGS1:72000000

OSIO: 2135C658
+000000 OSIO_EYE:OSIO          DCBW:00014260      DCBRU:00000000
+00000C JFCB:000142C8      CURRMBUF:00000000      MBUFCOUNT:00000001
+000018 READMAX:00000000      CURBLKNUM:FFFFFFF
+000020 _LASTBLKNUM:FFFFFFF      BLKSPERTRK:0000      OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00      FIRSTPOS:00000100

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 18 of 22)

```

+00002C LASTPOS:00000100      NEWPOS:00000000  READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005  FCB:2135C450    PARENT:2135C658
+000044 FLAGS1:85020000  DCBERU:00000000  DCBEW:2135C6C0
+000050 OSIO_VOLSEQ:0000  OSIO_NEWVOLSEQ:0000  OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000  APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

      DCB: 00014260
+000000 DCBRELAD:2135C6C0      DCBFDAD:00000000 22000100
+000014 DCBBUFNO:00      DCBSRG1:40  DCBEODAD:000000  DCBRECFM:80
+000025 DCBEXLSA:0127D0  DCBDDNAM:.....g<      DCBMACR1:65
+000033 DCBMACR2:B8      DCBSYNAD:000000  DCBBLKS1:0050  DCBNCP:01
+000052 DCBLRECL:0050

      DCBE: 2135C6C0
+000000 DCBEID:DCBE      DCBELEN:0038  RESERVED0:0000
+000008 DCBEDCB:00014260  DCBERELA:00000000  DCBEFLG1:C0
+000011 DCBEFLG2:88      DCBENSTR:0000  DCBEFLG3:00
+000024 DCBESIZE:00000000  DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C  MULTSDN:00

      JFCB: 000142C8
+000000 JFCBDSNM:CHARUM.L.M
+00002C JFCBELNM:      JFCBTSDM:80      JFCBDCB:004309
+000046 JFCBVLQ:0000  JFCBIND1:00  JFCBIND2:40
+000058 JFCBUFNO:00  JFCDSRG1:00  JFCDSRG2:00
+000064 JFCRECFM:00  JFCBLKS1:0000  JFCLRECL:0000  JFCNCP:00
+000075 JFCBNVOL:01  JFCBNVOLS:SL620C
+000094 JFCBEXTL:00  JFCBEXAD:000C7F  JFCFLGS1:00
+0000AE JFCBVLCT:01

      FFIL: 2135C700
+000000 MARKER1:AFCB  FILE:00000000  __FP:2135C720
+00000C MARKER2:AFCBAFCB  FF_FLAGS:01000000
+000014 FCBMUTEX:00000000  THREADID:00000000 00000000

      File name: CHARUM.M.N

      FCB: 2135C720
+000000 BUFPTR:00000000  COUNTIN:00000000  COUNTOUT:00000000
+00000C READFUNC:2135C7F0  WRITEFUNC:2135C810  FLAGS1:0000
+000016 DEPTH:0000  NAME:2135C8DC  _LENGTH:0000000A
+000020 _BUFSIZE:00000044  MEMBER:.....  NEXT:2135C9F0
+000030 PREV:2135C450  PARENT:2135C720  CHILD:00000000
+00003C DDNAME:SYS00021  FD:FFFFFFFF  DEVTYPE:00  FCBTYPE:004C
+00004C FSCE:2135C854  UNGETBUF:2135C854  REPOS:20F5E880
+000058 GETPOS:20F5EA80  CLOSE:20FD3118  FLUSH:20F5E818
+000064 UTILITY:213126A0  USERBUF:00000000  LRECL:00000404
+000070 BLKSIZE:00000408  REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000  BUFSIZE:00000408  BUF:00000000
+000084 CURSOR:00000000  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:20F481E8
+0000B4 DBCSTART:00000000  UTILITYAREA:00000000
+0000BC INTERCEPT:00000000  FLAGS2:02120000 20461100
+0000C8 DBCSSTATE:0000  FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000  READ:20F48580
+0000DC RADDR_WSA:00000000  _GETFN:00000000  RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000  RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000  WRITE:20F5EC68
+0000FC WADDR_WSA:00000000  _GETFN:00000000  WDLL_INDEX:00000000
+000108 WCEESG003:00000000  WWSA:00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 19 of 22)

```

OSFS: 2135C854
+000000 OSFS_EYE:OSFS      READ:20F48580      WRITE:20FD5EC0
+00000C REPOS:20FD3788     GETPOS:20FD5890   CLOSE:00000000
+000018 FLUSH:20FD3428        UTILITY:213126A0  EXITFTELL:00000000
+000024 EXITUNGETC:20F481E8  OSIOBLK:2135C928  SAVECURSOR:00
+00002D NEWLINEPTR:00000000     ENDOFBLOCK:00000000
+000035 CURRBLKSIZE:00000000  LASTBLKSIZE:00000000
+00003D HIGHMAJOR:00000000    RELRECNUM:00000000
+000045 MAXFTELLBLK:00000000  RECBITS:00000000
+00004D RECSPERBLOCK:00000000  SAVECHAR:00000000  FLAGS1:72000000

OSIO: 2135C928
+000000 OSIO_EYE:OSIO      DCBW:00014380     DCBRU:00000000
+00000C JFCB:000143E8     CURRMBUF:00000000  MBUFCOUNT:00000001
+000018 READMAX:00000000     CURBLKNUM:FFFFFFF
+000020 LASTBLKNUM:FFFFFFF      BLKSPERTRK:0000  OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00  FIRSTPOS:00000100
+00002C LASTPOS:00000100     NEWPOS:00000000  READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005  FCB:2135C720     PARENT:2135C928
+000044 FLAGS1:81020000     DCBERU:00000000  DCBEW:2135C990
+000050 OSIO_VOLSEQ:0000  OSIO_NEWVOLSEQ:0000  OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000     APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 00014380
+000000 DCBRELAD:2135C990     DCBFDAD:00000000  D9000200
+000014 DCBBUFNO:00     DCBSRG1:40  DCBEODAD:000000  DCBRECFM:40
+000025 DCBEXLSA:0127D0    DCBDDNAM:.....h.  DCBMACR1:65
+000033 DCBMACR2:B8     DCBSYNAD:000000  DCBBLKSI:0408  DCBNCP:01
+000052 DCBLRECL:0404

DCBE: 2135C990
+000000 DCBEID:DCBE      DCBELEN:0038     RESERVED0:0000
+000008 DCBEDCB:00014380    DCBERELA:00000000  DCBEFLG1:C0
+000011 DCBEFLG2:88     DCBENSTR:0000    DCBEFLG3:00
+000024 DCBESIZE:00000000    DCBEEODA:20E6269A
+00002C DCBESYNA:20E6263C    MULTSDN:00

JFCB: 000143E8
+000000 JFCBDSNM:CHARUM.M.N
+00002C JFCBELNM:      JFCBTSDM:80     JFCBDSCB:013609
+000046 JFCBVLSQ:0000     JFCBIND1:00     JFCBIND2:40
+000058 JFCBUFNO:00     JFCDSRG1:00     JFCDSRG2:00
+000064 JFCRECFM:00     JFCBLKSI:0000  JFCLRECL:0000  JFCNCP:00
+000075 JFCBNVOL:01     JFCBNVOLS:SL4A05
+000094 JFCBEXTL:00     JFCBEXAD:000CCF  JFCFLGS1:00
+0000AE JFCBVLCT:01

FFIL: 2135C9D0
+000000 MARKER1:AFCB      FILE:00000000     _FP:2135C9F0
+00000C MARKER2:AFCBAFCB    FF_FLAGS:01000000
+000014 FCBMUTEX:00000000    THREADID:00000000 00000000
File name: CHARUM.N.0

FCB: 2135C9F0
+000000 BUFPTR:00000000    COUNTIN:00000000  COUNTOUT:00000000
+00000C READFUNC:2135CAC0    WRITEFUNC:2135CAE0  FLAGS1:0000
+000016 DEPTH:0000  NAME:2135CBAC    _LENGTH:0000000A
+000020 _BUFSIZE:00000044     MEMBER:.....  NEXT:20C0E148
+000030 PREV:2135C720     PARENT:2135C9F0  CHILD:00000000
+00003C DDNAME:SYS00022  FD:FFFFFFF      DEVTYPE:00  FCBTYPE:0054
+00004C FSCE:2135CB24     UNGETBUF:2135CB24  REPOS:20FDB888
+000058 GETPOS:20FDBAA0    CLOSE:20FD9EC0   FLUSH:20FDB828
+000064 UTILITY:20FDB5D8     USERBUF:00000000  LRECL:00000000
+000070 BLKSIZE:00001800    REALBUFPTR:00000000
+000078 UNGETCOUNT:00000000  BUFSIZE:00001800  BUF:00000000
+000084 CURSOR:00000000  ENDOFDATA:00000000  SAVEDBUF:00000000
+000090 REALCOUNTIN:00000000  REALCOUNTOUT:00000000
+000098 POSMAJOR:00000000  SAVEMAJOR:00000000

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 20 of 22)

```

+0000A0 POSMINOR:00000000 SAVEMINOR:00000000 STATE:0000
+0000AA SAVESTATE:0000 EXITFTTELL:00000000 EXITUNGETC:20F481E8
+0000B4 DBCSTART:00000000 UTILITYAREA:00000000
+0000BC INTERCEPT:00000000 FLAGS2:02120020 60461100
+0000C8 DBCSSTATE:0000 FCB_CPCB:20C0DEA8
+0000D0 READGLUE:58FF0008 07FF0000 READ:20F48580
+0000DC RADDR_WSA:00000000_GETFN:00000000 RDLL_INDEX:00000000
+0000E8 RCEESG003:00000000_RWSA:00000000
+0000F0 WRITEGLUE:58FF0008 07FF0000 WRITE:20FDBC88
+0000FC WADDR_WSA:00000000_GETFN:00000000 WDLL_INDEX:00000000
+000108 WCEESG003:00000000_WWSA:00000000

OSFS: 2135CB24
+000000 OSFS_EYE:OSFS READ:20F48580 WRITE:20FDAF60
+00000C REPOS:20FDA1A0 GETPOS:20FDAC30 CLOSE:00000000
+000018 FLUSH:20FDA078 UTILITY:213126A0 EXITFTTELL:00000000
+000024 EXITUNGETC:20F481E8 OSIOBLK:2135CBF8 SAVECURSOR:00
+00002D NEWLINEPTR:00000000 ENDOFBLOCK:00000000
+000035 CURRBLKSIZE:00000000 LASTBLKSIZE:00000000
+00003D HIGHMAJOR:00000000 RELRECNUM:00000000
+000045 MAXFTTELLBLK:00000000 RECBITS:00000000
+00004D RECSPERBLOCK:00000000 SAVECHAR:00000000 FLAGS1:72000000

OSIO: 2135CBF8
+000000 OSIO_EYE:OSIO DCBW:000144A0 DCBRU:00000000
+00000C JFCB:00014508 CURRMBUF:00000000 MBUFCOUNT:00000001
+000018 READMAX:00000000 CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF BLKSPERTRK:0000 OSIO_ACCESS_METHOD:01
+000027 OSIO_NOSEEK_TO_SEEK:00 FIRSTPOS:00000100
+00002C LASTPOS:00000100 NEWPOS:00000000 READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005 FCB:2135C9F0 PARENT:2135CBF8
+000044 FLAGS1:81020000 DCBERU:00000000 DCBEW:2135CC60
+000050 OSIO_VOLSEQ:0000 OSIO_NEWVOLSEQ:0000 OSIO_EXT:00000000
+000058 OSIO_HIGHVOL:0000 APPENDEDLASTVOLSEQ:0000
+00005C OSIO_JFCBX:00000000

DCB: 000144A0
+000000 DCBRELAD:2135CC60 DCBFDAD:00000000 22000600
+000014 DCBBUFNO:00 DCBSRG1:40 DCBEODAD:000000 DCBRECFCM:C0
+000025 DCBEXLSA:0127D0 DCBDDNAM:.%...im DCBMACR1:65
+000033 DCBMACR2:B8 DCBSYNAD:000000 DCBBLKSI:1800 DCBNCP:01
+000052 DCBLRECL:0000

DCBE: 2135CC60
+000000 DCBEID:DCBE DCBELEN:0038 RESERVED0:0000
+000008 DCBEDCB:000144A0 DCBERELA:00000000 DCBEFLG1:C0
+000011 DCBEFLG2:88 DCBENSTR:0000 DCBEFLG3:00
+000024 DCBESIZE:00000000 DCBEODA:20E6269A
+00002C DCBESYNA:20E6263C MULTSDN:00

JFCB: 00014508
+000000 JFCBDSNM:CHARUM.N.O
+00002C JFCBELNM: JFCBTSDM:80 JFCBDCSB:00471E
+000046 JFCBVLQS:0000 JFCBIND1:00 JFCBIND2:40
+000058 JFCBUFNO:00 JFCDSRG1:00 JFCDSRG2:00
+000064 JFCRECFM:00 JFCBLKSI:0000 JFCLRECL:0000 JFCNCP:00
+000075 JFCBNVOL:01 JFCBNVOLS:SL6206
+000094 JFCBEXTL:00 JFCBEXAD:000D1F JFCFLGS1:00
+0000AE JFCBVLCT:01

```

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 21 of 22)

Dummy FCB encountered at location 20C0E148

```
AMRC: 20C0E770
+000000 CODE:00021708 RBA:00000000 LASTOP:00000032
+00000C FILL_LEN:00000000 MSG_LEN:00000000
+000014 STR1:.....
+000050 STR1_CONT:.....
+00008C PARM0R0:00000000 PARMR1:00000000
+00009C STR2:.....
+0000DC RPLFDBWD:00000000 XRBA:00000000 00000000
+0000E8 AMRC_NOSEEK_TO_SEEK:00

AMRC2: 20C0E878
+000000 __ERROR2:00000000 __FILEPTR:00000000
```

File name: CHARUM.A.B

```
[8]MFCB: 2135A470
+000000 FIRSTNEBULA:2135A4E4 REFCNT:00000001 NEXTMFCB:2135A970
+000010 WRITEFCB:2135A268 FLAG1:0001 DEPTH:0000 NAME:2135A5F0
+00001C NAMELENGTH:0000000A NAMEBUFSIZE:00000044 MEMBER:.....
+00002C NEXTFCB:00000000 PREVFCB:00000000
+000034 PARENTFCB:2135A268 CHILDFCB:00000000
+00003C PREVMFCB:00000000 PARENTMFCB:2135A470
+000044 CHILDMFCB:00000000 HIPERKEY:00000000 00000000
+000050 CURHSPBYTES:00000000 LASTBYTE:0000 CREATELEVEL:0000
+000058 FLAG2:00000000 LASTNEBULA:2135A4E4 LASTNEBINDEX:0001
+000064 LASTDS:00000000 MAXHSPBYTES:00000000
+00006C LASTBLKOFFSET:00000000 MFCB_CPCB:20C0DEA8
```

File name: CHARUM.B.C

```
MFCB: 2135A970
+000000 FIRSTNEBULA:00000000 REFCNT:00000001 NEXTMFCB:00000000
+000010 WRITEFCB:2135A660 FLAG1:0001 DEPTH:0000 NAME:2135A9F0
+00001C NAMELENGTH:0000000A NAMEBUFSIZE:00000044 MEMBER:.....
+00002C NEXTFCB:00000000 PREVFCB:00000000
+000034 PARENTFCB:2135A660 CHILDFCB:00000000
+00003C PREVMFCB:2135A470 PARENTMFCB:2135A970
+000044 CHILDMFCB:00000000 HIPERKEY:80007400 00011F4C
+000050 CURHSPBYTES:00020000 LASTBYTE:0000 CREATELEVEL:0000
+000058 FLAG2:40000000 LASTNEBULA:00000000 LASTNEBINDEX:0000
+000064 LASTDS:00000000 MAXHSPBYTES:80000000
+00006C LASTBLKOFFSET:00000000 MFCB_CPCB:20C0DEA8
```

Exiting CRTL I/O Control Blocks
Exiting Language Environment Data

Figure 18. Example formatted C/C++ output from LEDATA Verbexit (Part 22 of 22)

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

FFIL This section formats the header of the C/C++ file control block (FCB).

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.

HFSF — UNIX file system file

HSPF — Hiper-Space file

INTC — Intercept file

MEMF — Memory file

OSNS — OS no seek

OSFS — OS fixed text

OSVF — OS variable text

OSUT — OS undefined format text

TDQF — CICS Transient Data Queue file

TERM — Terminal file

VSAM — VSAM file

Other FSCEs will be displayed using a generic overlay.

OSIO The OS IO interface control block.

OSIOE

The OS IO extended interface control block.

DCB The data control block. For more information about the DCB, see *z/OS DFSMS Macro Instructions for Data Sets*.

DCBE The data control block extension. For more information about the DCBE, see *z/OS DFSMS Macro Instructions for Data Sets*.

JFCB The job file control block (JFCB). For more information about the JFCB, see *z/OS MVS Data Areas, Vol 3 (IVT-RCWK)*.

JFCBX

The job file control block extension (JFCBX).

MBUF

The message buffer control block (MBUF).

[8] Memory File Control Blocks

This section formats the C/C++ memory file control block (MFCB).

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 19 on page 141 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 142 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 00000008 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 19. 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:80007DEB
+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 19. 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 144. 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 20 on page 144 :

```
CBF 15890 struct(CEECAA)
```

```

CEECAA: 00015890
+000000 FLAG0:00 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 SECTSIZ:00000000
+0002E8 PARTSUM:00000000 SSEXPT: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 DANCHR:00000000 CTCOC: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 ERRRCM:06412AB0 MIB_PTR:00000000
+000434 THDSTATUS:00000000 TICB_PTR:06413840
+00047C FWD_CHAIN:00015890 BKWD_CHAIN:00015890

```

Figure 20. 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 that 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
CEEDLLF	DLL Failure Control Block
CEEDSATR	XPLINK Transition Area
CEEDSAX	Dynamic Storage Area (XPLINK style)
CEEEDB	Enclave Data Block
CEEENSM	Enclave Level Storage Management

Table 8. Language Environment Control blocks that can be individually formatted (continued)

Control Block	Description
CEEHANC	Heap Anchor Node
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 (`/`), the directory is writable, and the CEEDUMP path name does not exceed 1024 characters
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, see 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 21. 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
- 20 Trace all XPLINK/non-XPLINK transitions for AMODE 31 only. If #pragma linkage (xxxxxxx, OS_UPSTACK) is specified, no transitions are recorded.

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 199.

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

Table 10. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
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)
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

Table 10. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
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)
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

Table 10. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
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	EJO	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

Table 10. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
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 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 149.

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 199.

When LE=20 is specified: The following C/C++ records might be generated.

Table 13. LE=20 entry records

Member ID	Record Type	Description
03	00000007	XPLINK calls non-XPLINK entry
03	00000008	non-XPLINK calls XPLINK entry

For a detailed description of these records, see “C/C++ contents of the Language Environment trace tables” on page 199.

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_istream_|
+000038 60606E4D F1F9F35D 406D6D87 85A38382 4D5D4040 40404040 40404040 40404040 |-->(193) __getcb()|
+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 |.....|
+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_istream_|
+000138 60606E4D F1F1F35D 406D6D89 A2D796A2 89A7D695 4D5D4040 40404040 40404040 |-->(113) __isPosixOn()|
+000158 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 F1F1F35D 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 |.....|
+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 |.....|
+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) __getcb()|
+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 |.....|
+0003F8 00000000 00000000
:

```

Figure 22. 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, see *z/OS XL C/C++ Programming Guide* to ensure that you are creating the fetchable module correctly.
- If you are using DLLs, see *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++ programs

You can use C/C++ conventions such as `__amrc` and `perror()` when you debug C/C++ programs.

Using the `__amrc` and `__amrc2` structures to debug input/output

`__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 23 shows the structure as it appears in `stdio.h`.

```
typedef struct __amrctype {
[1]  union {
[2]      long int __error;
          struct {
              unsigned short __syscode,
                              __rc;
[3]      } __abend;
          struct {
              unsigned char __fdbk_fill,
                              __rc,
                              __ftncd,
                              __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;
[9]  #if __EDC_TARGET >= 0x22080000
          unsigned char __rplfdbwd[4];
          #endif
[10] #if __EDC_TARGET >= 0x41080000
          #ifdef __LP64
              unsigned long __XRBA;
          #elif defined(__LL)
              unsigned long long __XRBA;
          #else
              unsigned int __XRBA1;
              unsigned int __XRBA2;
          #endif
          unsigned char __amrc_noseek_to_seek;
          char __amrc_pad[23];
          #endif
    } __amrc_type;

```

Figure 23. `__amrc` structure

Figure 24 on page 161 shows the `__amrc2` structure as it appears in `stdio.h`.

```

    struct {
[11]     long int   __error2;
                char       __pad__error2[4];
[12]     FILE      *__fileptr;
[13]     long int   __reserved{6};
    }

```

Figure 24. `__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. 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 14 on page 162.

[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] __amrc_noseek_to_seek

This field contains the reason for the switch from QSAM (noseek) to BSAM with `NOTE` and `POINT` macros requested (seek) by the XL C/C++ Run-Time Library. This field is set when system-level I/O macro processing triggers an ABEND condition. The macro name values (defined in `stdio.h`) for this field are as follows:

Macro	Definition
<code>__AM_BSAM_NOSWITCH</code>	No switch was made.
<code>__AM_BSAM_UPDATE</code>	The data set is open for update

Macro	Definition
__AM_BSAM_BSAMWRITE	The data set is already open for write (or update) in the same C process.
__AM_BSAM_FBS_APPEND	The data set is recfm=FBS and open for append
__AM_BSAM_LRECLX	The data set is recfm=LRECLX (used for VBS data sets where records span the largest blocksize allowed on the device)
__AM_BSAM_PARTITIONED_DIRECTORY	The data set is the directory for a regular or extended partitioned data set
__AM_BSAM_PARTITIONED_INDIRECT	The data set is a member of a partitioned data set, and the member name was not specified at allocation

- [10] **__XRBA** This is the 8 byte relative byte address 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 may be used in subsequent calls to `flocate()`.
- [11] **__error2** A secondary error code. For example, an unsuccessful rename or remove operation places its reason code here.
- [12] **__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.
- [13] **__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 14 lists `__last_op` values you could receive and where to look for further information.

Table 14. `__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 <code>__error</code> with return code from OS OPEN macro.
__BSAM_CLOSE	Sets <code>__error</code> with return code from OS CLOSE macro.
__BSAM_READ	No return code (either <code>__abend</code> (<code>errno == 92</code>) or <code>__msg</code> (<code>errno == 66</code>) filled in).
__BSAM_NOTE	NOTE returned 0 unexpectedly, no return code.
__BSAM_POINT	This will not appear as an error lastop.
__BSAM_WRITE	No return code (either <code>__abend</code> (<code>errno == 92</code>) or <code>__msg</code> (<code>errno == 65</code>) filled in).
__BSAM_CLOSE_T	Sets <code>__error</code> with return code from OS CLOSE TYPE=T.
__BSAM_BLDL	Sets <code>__error</code> with return code from OS BLDL macro.
__BSAM_STOW	Sets <code>__error</code> with return code from OS STOW macro.
__TGET_READ	Sets <code>__error</code> with return code from TSO TGET macro.
__TPUT_WRITE	Sets <code>__error</code> with return code from TSO TPUT macro.

Table 14. `__last_op` values and diagnosis information (continued)

Value	Further Information
<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.
<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 <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 RRDS.

Table 14. *__last_op* values and diagnosis information (continued)

Value	Further Information
<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 abend (errno == 92), <code>__abend</code> is set, otherwise if read error (errno == 66), look at <code>__msg</code>).
<code>__QSAM_PUT</code>	<code>__error</code> is not set (if abend (errno == 92), <code>__abend</code> is set, otherwise if write error (errno == 65), 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_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.

Table 14. *__last_op* values and diagnosis information (continued)

Value	Further Information
<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> .
<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`). By default, the `errno2` value will be appended to the end of the `perror()` string.

If you do not want the `errno2` value appended to the `perror()` string, set the `_EDC_ADD_ERRNO2` environment variable to 0.

Figure 25 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 25. Example of a routine using `perror()`

Using `__errno2()` to diagnose application problems

Use the `__errno2()` function when diagnosing problems in an application program. This function enables z/OS XL C/C++ application programs to access additional diagnostic information, `errno2` (`errnojr`), associated with `errno`. The `__errno2` may be set by the z/OS XL C/C++ run-time library, z/OS UNIX callable services, or other callable services. The `errno2` is intended for diagnostic display purposes only and is not a programming interface.

Note: Not all functions set `errno2` when `errno` is set. In the cases where `errno2` is not set, the `__errno2()` function may return a residual value. You may use the `__err2ad()` function to clear `errno2` to reduce the possibility of a residual value being returned.

Figure 26 on page 167 is an example of a routine using `__errno2()` and Figure 27 on page 167 shows the sample output from that routine.

```

#pragma runopts(posix(on))
#define _EXT
#include <stdio.h>
#include <errno.h>

int main(void) {
    FILE *f;
    f = fopen("testfile.dat", "r")
    if (f=NULL) {
        perror("fopen() failed");
        printf("__errno2 = %08x\n", __errno2());
    }
    return 0;
}

```

Figure 26. Example of a routine using `__errno2()`

```

fopen() failed: EDC5129I No such file or directory. (errno2=0x05620062)
__errno2 = 05620062

```

Figure 27. Sample output of a routine using `__errno2()`

Figure 28 is an example of a routine using the environment variable `_EDC_ADD_ERRNO2` and Figure 29 shows the sample output from that routine.

```

#pragma runopts(posix(on))
#define _EXT
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int main(void) {
    FILE *fp;
    /* do NOT add errno2 to perror message */
    setenv("_EDC_ADD_ERRNO2", "0", 1);
    fp = fopen("testfile.dat", "r");
    if (fp == NULL)
        perror("fopen() failed");
    return 0;
}

```

Figure 28. Example of a routine using `_EDC_ADD_ERRNO2`

```

fopen() failed: EDC5129I No such file or directory.

```

Figure 29. Sample output of a routine using `_EDC_ADD_ERRNO2`

Figure 30 on page 168 is an example of a routine using `__err2ad()` in combination with `__errno2()` and Figure 31 on page 168 shows the sample output from that routine.

```

#pragma runopts(posix(on))
#define _EXT
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int main(void) {
    FILE *f;
    setenv("_EDC_ADD_ERRNO2", "0", 1);
    f = fopen("testfile.dat", "r");
    if (f == NULL) {
        perror("fopen() failed");
        printf("__errno2 = %08x\n", __errno2());
    }
    /* reset errno2 to zero */
    *_err2ad() = 0x0;
    printf("__errno2 = %08x\n", __errno2());
    f = fopen(*testfile.dat, "r");

    if (fp == NULL) {
        perror("fopen() failed");
        printf(*__errno2 = %08x\n", __errno2());
    }
    return 0;
}

```

Figure 30. Example of a routine using `__err2ad()` in combination with `__errno2()`

```

fopen() failed: EDC5129I No such file or directory.
__errno2 = 05620062
__errno2 = 00000000
fopen() failed: EDC5129I No such file or directory.
__errno2 = 05620062

```

Figure 31. Sample output of routine using `__err2ad()` in combination with `__errno2()`

For more information about `_EDC_ADD_ERRNO2`, see *z/OS XL C/C++ Programming Guide*.

For more information about `__errno2()` and `__err2ad()`, see *z/OS XL C/C++ Run-Time Library Reference*.

Diagnosing DLL problems

Use the `_EDC_DLL_DIAG` environment variable to diagnose DLL problems. For more information, see *z/OS XL C/C++ Programming Guide*

You can also see the diagnosis output in CEEDUMP and Verbexit LEDATA reports. For more information, see “Using the DLL failure control block” on page 78.

Using C/C++ listings

For a detailed description of available listings, see *z/OS XL C/C++ User's Guide*.

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 15. 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 177.

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 169. For this procedure’s example, assume that 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 32. 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 33 on page 172 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.

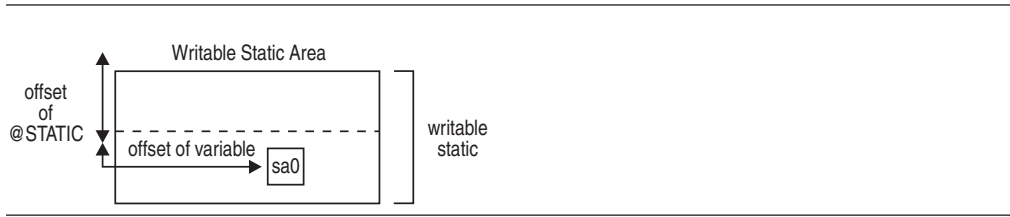


Figure 33. Location of RENT static variable in storage

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 169. 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 34. 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 170. 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 35 shows how to locate NORENT C static variables by adding the Static Storage Area CSECT address to the variable offset.

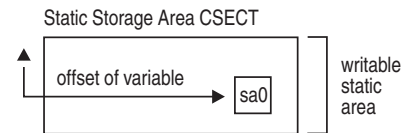


Figure 35. Location of NORENT static variable in storage

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 170. 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 36 on page 174 shows an example code for the parameter variable.

Example:

```

func0() {
    ⋮
    func1(a1,a2);
    ⋮
}

func1(int ppx, int pp0) {
    ⋮
}

```

Figure 36. 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 173.

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 37 shows an example code for the parameter variable.

Example:

```

func0() {
    ⋮
    func1(a1,a2);
    ⋮
}

func1(int ppx, int pp0) {
    ⋮
}

```

Figure 37. 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 38. 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 39 shows an example of a static aggregate.

```
static struct {
    short int ss01;
    char      ss02[56];
    int       sz0[6];
    int       ss03;
} ss0;
```

Figure 39. Example code for structure variable

Figure 40 on page 176 shows an example aggregate map.

```

=====
| Aggregate map for:  ss0
=====
|
|  Offset      Length      Member Name
|  Bytes(Bits) Bytes(Bits)
|-----|-----|
|    0          2          ss01
|    2         56          ss02[56]
|   58          2          ***PADDING***
|   60         24          sz0[6]
|   84          4          ss03
|-----|-----|
=====

```

Figure 40. 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 41 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 41. Writable static map produced by prelinker

- 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.
- Find the offset of sz0 in the Aggregate Map, shown in Figure 40 on page 176. 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:

```
YYYYMMDDHHMSSSS
```

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.

`cdump()`, `csnap()`, and `ctrace()` all return a 1 code in the SPC environment because they are not supported in SPC.

See 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 42 on page 179 shows a sample C routine that uses the `cdump` function to generate a dump.

Figure 47 on page 183 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 42. 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 42. Example C routine using `cdump()` to generate a dump (Part 2 of 2)

Figure 43 shows a fetched C module:

```

#include <ctest.h>

#pragma linkage(func1, fetchable)
int func1(void) {
    cdump("This is a sample dump");
    return(0);
}

```

Figure 43. Fetched module for C routine

Sample C++ routine that generates a Language Environment dump

Figure 44 on page 181 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 44. Example C++ routine with protection exception generating a dump

Figure 45 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 45. Template file STACK.C

Figure 46 on page 182 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 46. Header file STACK.H

Sample Language Environment dump with C/C++-specific information

This sample dump was produced by compiling the routine in Figure 42 on page 179 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 188.

CEE3845I CEEDUMP Processing started.
 CEE3DMP called by program unit (entry point __cdump) at offset +0000017C.

Snap was unsuccessful

Registers on Entry to CEE3DMP:

```
PM..... 0100
GPR0..... 20E56AF4 GPR1..... 20FCB5A0 GPR2..... 00000001 GPR3..... 20E56752
GPR4..... A0FCB5B8 GPR5..... 20FCB5C4 GPR6..... 00000014 GPR7..... 20902760
GPR8..... 00000030 GPR9..... 20FC7038 GPR10.... A0900310 GPR11.... A0900310
GPR12.... 209139B0 GPR13.... 20FCB508 GPR14.... A0E56896 GPR15.... A09F0898
FPR0..... 26100000 00000000 FPR2..... 18000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
```

Information for enclave main

Information for thread 8000000000000000

Registers on Entry to CEE3DMP:

```
PM..... 0100
GPR0..... 20E56AF4 GPR1..... 20FCB5A0 GPR2..... 00000001 GPR3..... 20E56752
GPR4..... A0FCB5B8 GPR5..... 20FCB5C4 GPR6..... 00000014 GPR7..... 20902760
GPR8..... 00000030 GPR9..... 20FC7038 GPR10.... A0900310 GPR11.... A0900310
GPR12.... 209139B0 GPR13.... 20FCB508 GPR14.... A0E56896 GPR15.... A09F0898
FPR0..... 26100000 00000000 FPR2..... 18000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
```

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	__cdump	+0000017C		CEEEV003		D1908:e	Call
2	func1	+0000006E	5	MODULE1	MODULE1		Call
3	@@FECBMODULE1						Call
		-002BD99E					Call
4	@@GETFN	+000000C2		CEEEV003			Call
5	main	+00000392	64	CSAMPLE	CSAMPLE		Call
6	EDCZMINV	+000000C2		CEEEV003			Call
7	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	20FCB508	20E56718	20E56718	+0000017C	20061215	LIBRARY EBCDIC HFP
2	20FCB468	20900310	20900310	+0000006E	19970314	C/C++
3	20FCB378	20FEBF90	20FEBF90	-002BD99E		LIBRARY
4	20FCB2C0	20C0E3C0	20C0E468	+0000001A	12/15/06	LIBRARY
5	20FCB208	20901078	20901078	+00000392	19970314	C/C++
6	20FCB0F0	20E699EE	20E699EE	+000000C2	20061215	LIBRARY
7	20FCB030	20992208	20992208	+000001B6	20061215	CEL

Fully Qualified Names

DSA	Entry	Program Unit	Load Module
2	func1	POSIX.CRTL.C(MODULE1)	MODULE1
5	main	POSIX.CRTL.C(CSAMPLE)	CSAMPLE

Parameters, Registers, and Variables for Active Routines:

main (DSA address 20FCB208):

UPSTACK DSA

Saved Registers:

```
GPR0..... 20FEBE30 GPR1..... A0D2FE64 GPR2..... A0E69AB0 GPR3..... A09010C6
GPR4..... A09922EC GPR5..... 20FC7098 GPR6..... 20FEBE30 GPR7..... 20902760
GPR8..... 00000030 GPR9..... 80000000 GPR10.... A0E699E2 GPR11.... A0992208
GPR12.... 209139B0 GPR13.... 20FCB208 GPR14.... A090140C GPR15.... 20C0E3C0
```

Local Variables:

```
fetchPtr      signed int (*) (void)
               0x20FEBE30
fp2           struct __ffile * 0x20FF5AFC
fp1           struct __ffile * 0x20FF4024
rc           signed int 0
```

Figure 47. Example dump from sample C routine (Part 1 of 6)

```

Control Blocks for Active Routines:
:
:
DSA for func1: 20FCB468
+000000  FLAGS.... 1000      member... 0000      BKC..... 20FCB378  FWC..... D4D6C4E4  R14..... A0900380
+000010  R15..... 20E56718  R0..... 20FCB508  R1..... 20FCB500  R2..... A0D2E5F2  R3..... A090035E
+000024  R4.....  A09922EC  R5..... 20FEBF70  R6..... 20FEBE30  R7..... 20902760  R8..... 00000030
+000038  R9..... 20FC7038  R10..... A0900310  R11..... A0900310  R12..... 209139B0  reserved. 00000000
+00004C  NAB..... 20FCB508  PNAB..... A099DD88  reserved. 20FCB378 20900B08
+000064  reserved. 20900990  reserved. 00000000  MODE..... 20FCB390  reserved. 00000000
+000078  reserved. 20991D52  reserved. A0915770
:
:
[1]Storage for Active Routines:
:
:
DSA frame: 20FCB468
+000000 20FCB468 10000000 20FCB378 D4D6C4E4 A0900380 20E56718 20FCB508 20FCB500 A0D2E5F2 |.....MODU.....V.....KV2|
+000020 20FCB488 A090035E A09922EC 20FEBF70 20FEBE30 20902760 00000030 20FC7038 A0900310 |...;r.....-.....|
+000040 20FCB4A8 A0900310 209139B0 00000000 20FCB508 A099DD88 20FCB378 20FCB534 20900B08 |.....j.....r.h.....|
+000060 20FCB4C8 20FCB378 20900990 00000000 20FCB390 00000000 00000000 20991D52 A0915770 |.....r...j..|
+000080 20FCB4E8 A0991A28 209139B0 00000000 20FCB5A0 20C4D1F2 00000400 20FEBF70 00000009 |r...j.....DJ2.....|
:
:
[2]Control Blocks Associated with the Thread:
CAA: 209139B0
+000000 209139B0 00000800 00000000 20FCB018 20FEB018 00000000 00000000 00000000 00000000 |.....|
+000020 209139D0 00000000 00000000 20910A58 00000000 00000000 00000000 00000000 00000000 |.....j.....|
+000040 209139F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 20913A10 00000000 00000000 00000000 00000000 00000000 80014608 00000000 00000000 |.....|
+000080 20913A30 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
:
:
[2A]C-C++ CAA information :
C-C++ Specific CTHD..... 2090E858
C-C++ Specific CEDB..... 2090F8D8

C-C++ Specific Thread block: 2090E858
+000000 2090E858 C3E3C8C4 00000380 2090E858 00000000 20D15090 00000000 00000000 00000000 |CTHD.....Y.....J&.....|
+000020 2090E878 00000000 00000000 00000000 000000C4 2090EDED 00000000 00000000 00000000 |.....D.....|
+000040 2090E898 00000000 00000000 00000000 2090E728 2090E830 00000000 00000000 00000001 |.....X...Y.....|
+000060 2090E8B8 00000001 00000000 00000000 00000000 00000000 00000000 20FEBF50 20FEBF4C |.....&.<|
+000080 2090E8D8 20FEBF48 20FEBF44 20FEBF38 20FEBF3C 20FEBF58 00000000 00000000 00000000 |.....|
:
+000320 2090E878 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000340 2090E898 - +00037F 2090EBD7 same as above
C-C++ Specific EDB block: 2090F8D8
+000000 2090F8D8 C3C5C4C2 000007C0 2090F8D8 20902B40 00030000 20911210 20911560 20901078 |CEDB.....8Q... ..j...j.-....|
+000020 2090F8F8 00000080 20900FD8 00000000 00000000 00000000 209100A0 20D164F0 20D15A18 |.....Q.....j...J.0.J!|
+000040 2090F918 20FEBF90 2090F9B0 2090F9C4 20FC71B8 2090F6BC 2090F28C 2090F4A4 20D1B242 |.....9...9D.....6...2...4u.J..|
+000060 2090F938 00004650 20910130 40400000 00000000 00100000 00000000 00000000 00000000 |...&.j.. ..|
+000080 2090F958 7FFFFFFF FFFFFFFF 71FFFFFF FFFFFFFF 3C100000 00000000 34100000 00000000 |".....|
:
+000780 20910058 00000000 00000000 00000000 0001D100 00002E80 00000000 00000000 00000000 |.....J.....|
+0007A0 20910078 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

```

Figure 47. Example dump from sample C routine (Part 2 of 6)

```

[2B]errno value..... 0
memory file block chain.... 20FF5D18
open FCB chain..... 20FF5B10
GTAB table..... 2090EBE0

[3]signal information :
SIGFPE :
function pointer... 20901D20   WSA address... A0FC7038   function name... hsigfpe

SIGTERM :
function pointer... 20901E90   WSA address... A0FC7038   function name... hsigterm

Enclave variables:
*.*.C(CSAMPLE):>hsigterm
    void ()                0x20901E90
*.*.C(CSAMPLE):>hsigfpe
    void ()                0x20901D20
*.*.C(CSAMPLE):>xcount
    signed int             0
*.*.C(CSAMPLE):>main
    signed int (void)      0x20901078
*.*.C(CSAMPLE):>atf1
    void (void)            0x20901FF8
*.*.C(CSAMPLE):>st2
    signed int             255
*.*.C(CSAMPLE):>st1
    signed int             99
*.*.C(MODULE1):>func1
    signed int (void)      0x20900310

Enclave Control Blocks:
EDB: 20912648
+000000 20912648 C3C5C5C5 C4C24040 C0000001 20913870 20912D50 00000000 00000000 00000000 | CEEEDB .....j...j.&.....
+000020 20912668 20912B58 20912B88 A0915770 20912198 00000000 00000000 20912768 00000000 | .j...j.h.j...j.q.....j.....
+000040 20912688 00000000 00000000 00006F58 00000000 00000000 A0A33B58 2090A880 20FEBF60 | .....?.....t...y...-
+000060 209126A8 0000F460 00000000 20911E58 00000000 20914550 00000000 20AF4660 209139B0 | ..4-.....j.....j.&.....-j..
+000080 209126C8 50000000 0000FAF6 00000000 00000000 00000001 00000000 00006FF0 008FF4E8 | &.....6.....?0..4Y
+0000A0 209126E8 00000001 00000100 2090A988 209126F0 00000000 00000000 00000000 00000001 | .....zh.j.0.....
MEML: 20913870
+000000 20913870 00000000 00000000 209945B0 00000000 00000000 00000000 209945B0 00000000 | .....r.....r.....
+000020 20913890 00000000 00000000 209945B0 00000000 2090F8D8 00000000 A0B07F78 00000000 | .....r.....8Q.....".....
+000040 209138B0 00000000 00000000 209945B0 00000000 00000000 00000000 209945B0 00000000 | .....r.....r.....
+000060 209138D0 - +00011F 2091398F same as above

[4]WSA address.....20FC7038

[5]atexit information :
function pointer... A0901FF8   WSA address... 20FC7038   function name... atf1

[6]fetch information :
fetch pointer : 20FEBF90
function pointer... A0900310   WSA address... 20FEBF18

Enclave Storage:
Initial (User) Heap : 20FE8018
+000000 20FE8018 C8C1D5C3 20912B28 20912B28 00000000 A0FE8018 20FEC070 00008000 00006FA8 | HANC.j...j.....?y
+000020 20FE8038 20FE8018 00000120 20FE8160 20FE8348 20FE8385 20FE83C2 20FE83FF 20FE843C | .....e..B.....
+000040 20FE8058 20FE8479 20FE84B6 20FE84F3 20FE8900 20FE893D 00000000 00000000 00000000 | .....3.....
:
+001080 20FEC098 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+0010A0 20FEC0B8 - +007FFF 20FF3017 same as above
LE/370 Anywhere Heap : 20FC7000
+000000 20FC7000 C8C1D5C3 21007000 20912B58 20912B58 A0FC7000 20FCAE60 00004000 000001A0 | HANC....j...j.....-.....
+000020 20FC7020 20FC7000 00000190 00000000 00000000 00000000 00000000 00000000 2090F4A4 | .....4u
+000040 20FC7040 2090F28C 2090F6BC 00000000 00000000 00000000 00000000 00000000 00000000 | ..2..6.....
:
+003F40 20FCAF40 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+003F60 20FCAF60 - +003FFF 20CAFAFF same as above
LE/370 Anywhere Heap : 21007000
+000000 21007000 C8C1D5C3 20912B58 20FC7000 20912B58 21007000 00000000 00004A50 00000000 | HANC.j.....j.....&....
+000020 21007020 21007000 00004A30 01000080 F2404040 404086A4 9583F140 40404040 4040D7D6 | .....2 func1 PO
+000040 21007040 E2C9E74B C3D9E3D3 4BC34DD4 D6C4E4D3 C5F15D40 40404040 40404040 40404040 | SIX.CRTL.C(MODULE1)
:
+000240 21007240 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000260 21007260 - +004A5F 2100BA5F same as above

```

Figure 47. Example dump from sample C routine (Part 3 of 6)

```

Additional Heap, heapid = 20FCAE2C : 2100C000
+000000 2100C000 C8C1D5C3 20FCAE2C 20FCAE2C 20FCAE2C 2100C000 2100C3D0 000003E8 00000018 | HANC.....C...Y...|
+000020 2100C020 2100C000 00000050 00000000 209003AC 20900310 2100C098 20FEBF90 46F11000 | .....&.....q...l..|
+000040 2100C040 00000003 20900580 00020000 2100C078 00000000 20900310 20900310 00000000 | .....|
:
+0003C0 2100C3C0 00000000 209014F8 20901C78 00000000 00000000 00000000 00000000 00000000 | .....8.....|
+0003E0 2100C3E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|

[7]File Status and Attributes:
File Control Block: 20FF5B10
+000000 20FF5B10 20FF5F0D 00000000 000003DB 20FF5BE0 20FF5C00 10000000 20FF5CCC 00000011 | ..^.....$.*.....*....|
+000020 20FF5B30 00000044 00000000 00000000 20FF4038 00000000 20FF5B10 00000000 00000000 | .....$.|
+000040 20FF5B50 00000000 FFFFFFFF 00080055 20FF5C44 20FF5C44 20CCDA0 20CCCB90 20CC7B0 | .....*...*.....G..|
+000060 20FF5B70 20CC558 20C41548 00000000 00000400 00000400 00000000 00000000 00000400 | ..E..D.....|
+000080 20FF5B90 20FF5EE8 20FF5EE8 00000000 00000000 00000000 00000000 00000000 00000000 | ..;Y..;Y.....|
+0000A0 20FF5BB0 00000000 00000000 00000000 00000000 20C3C4C8 00000000 00000000 00000000 | .....CDH.....|
+0000C0 20FF5BD0 43020008 40001100 00000000 2090DE68 58FF0008 07FF0000 20C3C830 00000000 | .....CH.....|
+0000E0 20FF5BF0 00000000 00000000 00000000 00000000 58FF0008 07FF0000 20CCE000 00000000 | .....|
+000100 20FF5C10 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|
+000120 20FF5C30 00000000 00000000 00000000 00000000 00000000 D4C5D4D6 20FF5D18 20FF5D8C | .....MEMO...)..|

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
__reserve2:4..... 0
__device..... 8
__blksize..... 1024
__maxreclen..... 1024
__access_method... 0(0)
__noseek_to_seek.. 0(0)
__dsname..... HEALY.MEMORY.DATA

FILE pointer..... 20FF5AFC

Buffer at current file position: 20FF5EE8
+000000 20FF5EE8 A2969485 408481A3 81A29694 85409496 99854084 81A38185 A5859540 94969985 | some datasome more dataeven more|
+000020 20FF5F08 408481A3 81000000 00000000 00000000 00000000 00000000 00000000 00000000 | data.....|
+000040 20FF5F28 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|
+000060 20FF5F48 - +0003FF 20FF62E7 same as above | .....|
Saved Buffer..... NULL

File Control Block: 20FF4038
+000000 20FF4038 20FF4314 00000000 00000400 20FF4108 20FF4128 90000000 20FF41F4 00000011 | .....4....|
+000020 20FF4058 00000044 00000000 00000000 2090E100 20FF5B10 20FF4038 00000000 E2E8E2F0 | .....$.SYS0|
+000040 20FF4078 F0F0F0F1 FFFFFFFF 0000003C 20FF416C 20FF416C 20CB4490 20CB7A58 20CB1240 | 0001.....%...%.....|
+000060 20FF4098 20CB3928 20CB0270 00000000 00000404 00001800 20FF5AEA 00000000 00001801 | .....!|
+000080 20FF40B8 20FF42E8 20FF4314 20FF4314 00000000 00000000 00000000 00000000 00000000 | ..Y.....|
+0000A0 20FF40D8 0000001B 00000000 00000000 00000000 20C3C4C8 00000000 00000000 00000000 | .....CDH.....|
+0000C0 20FF40F8 43120020 28440100 00000000 2090DE68 58FF0008 07FF0000 20C3C830 00000000 | .....CH.....|
+0000E0 20FF4118 00000000 00000000 00000000 00000000 58FF0008 07FF0000 20CB8278 00000000 | .....b.....|
+000100 20FF4138 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....|
+000120 20FF4158 00000000 00000000 00000000 00000000 00000000 D6E2E5C6 20C3C830 20CB8278 | .....OSVF.CH...b..|

```

Figure 47. 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
__reserve2:4..... 0
__device..... 0
__blksize..... 6144
__maxreclen..... 1024
__access_method... 1(1)
__noseek_to_seek.. 0(0)
__dsname..... HEALY.MYFILE.DATA

FILE pointer..... 20FF4024
ddname..... SYS00001
Buffer at current file position: 20FF42E8
+000000 20FF42E8 00280000 000C0000 99858396 998440F1 000C0000 99858396 998440F2 000C0000 |.....record 1....record 2....|
+000020 20FF4308 99858396 998440F3 00040000 00000000 00000000 00000000 00000000 00000000 |record 3.....|
+000040 20FF4328 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 20FF4348 - +0003FF 20FF46E7 same as above
Saved Buffer..... NULL

Write Data Control Block: 00015020
+000000 00015020 20FF42A8 00000000 000B0003 00E02026 002FE5A2 00000001 00004000 00006D40 |...y.....Vs....._|
+000020 00015040 86000000 500157D0 00CC2424 008CED84 12B745B8 00B7E220 0A000000 00001800 |f...&.....d.....S.....|
+000040 00015060 30013030 00006DB0 01C45470 00C45470 00000404 00C45A08 90ECD00C 18BF58A0 |....._..D...D!.....D!.....|
read/update DCB..... NULL

Write Data Control Block Extension: 20FF42A8
+000000 20FF42A8 C4C3C2C5 00380000 00015020 00000000 C0880000 00000000 00000000 00000000 |DCBE.....&.....h.....|
+000020 20FF42C8 00000000 00000000 20B74AB2 20B74A54 00000000 00000100 80001810 20FF4000 |.....|
read/update DCBE..... NULL

Job File Control Block: 000157E8
+000000 000157E8 C8C5C1D3 E84BD4E8 C6C9D3C5 4BC4C1E3 C1404040 40404040 40404040 40404040 |HEALY.MYFILE.DATA|
+000020 00015808 40404040 40404040 40404040 40404040 40404040 80000000 00000000 00000000 |.....|
+000040 00015828 00000200 00000000 00000000 00000000 6B000A00 00000040 00000000 00000000 |.....|
+000060 00015848 00000000 00000000 00000000 00000000 00000000 0001E2D3 F8C2F1F3 40404040 |.....SL8B13|
+000080 00015868 40404040 40404040 40404040 40404040 40404040 000003AF 00000000 00000000 |.....|
+0000A0 00015888 00000000 00000000 00000000 00000100 80000038 00015000 000158A0 20FF42E8 |.....&.....Y|

```

Figure 47. Example dump from sample C routine (Part 5 of 6)

```

[8]_amrc_type structure: 20FCDAB8
+000000 20FCDAB8 00000000 00000000 00000007 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 20FCDAD8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 20FCDAF8 - +0000FF 20FCDBB7          same as above
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
__rplfdbwd..... 0x00000000
__amrc_noseek_to_seek... 0(0)
__XRBA..... 0(0000000000000000)
__amrc2_type structure: 20FCD9B8
+000000 20FCD9B8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
__error2..... 0(0)
__fileptr..... NULL

```

Process Control Blocks:

```

PCB: 20912198
+000000 20912198 C3C5C5D7 C3C24040 03030288 00000000 00000000 00000000 209123D0 A0AF74D0 |CEEPCB ...h.....j.....|
+000020 209121B8 A0AEDB50 A0AF4A08 A0AF4528 20908AE8 20911F68 00000000 00000000 20912648 |...&.....Y.j.....|
+000040 209121D8 A0AF4858 7F800000 00000000 000141D4 00000000 80000000 00000000 00000000 |..."......M.....|

MEML: 209123D0
+000000 209123D0 00000000 00000000 209945B0 00000000 00000000 00000000 209945B0 00000000 |.....r.....r.....|
+000020 209123F0 00000000 00000000 209945B0 00000000 2090DE68 00000000 A0B07F78 20FC7000 |.....r.....".....|
+000040 20912410 00000000 00000000 209945B0 00000000 00000000 00000000 209945B0 00000000 |.....r.....r.....|
+000060 20912430 - +00011F 209124EF          same as above

```

Additional Language Specific Information:

```

[9]errno information :
Thread Id .... 8000000000000000 Errno ..... 0 Errnojr .... 00000000
CEE3846I CEEDUMP Processing completed.

```

Figure 47. 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'20FCB468'.

[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 190.
Open FCB chain	A pointer to the start of a linked list of open file control blocks (FCBs). For more information about FCBs, see 190.

[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, see *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 48.

```

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..... TSOID.MEMORY.DATA
First memory data space: 046F5E68
+000000 046F5E68 93899585 40F19389 958540F2 93899585 40F30000 00000000 00000000 00000000 |line 1line 2line 3.....|

```

Figure 48. 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`, see “Debugging C/C++ programs” on page 159 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, see *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 CEEDUMP; 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 49. 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 50. 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 51. Condition information for active routines

Sample Language Environment dump with XPLINK-specific information

The programs `tranmain` shown in Figure 52 on page 194 and `trandll` shown in Figure 53 on page 195 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 54 on page 196. Explanations for some of the sections are in “Finding XPLINK information in a Language Environment dump” on page 198.

```

#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 52. 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 53. Sample NOXPLINK-compiled program (trandll) which calls an XPLINK-compiled program

CEE3845I CEEDUMP Processing started.

Information for enclave main

Information for thread 8000000000000000

[1]Traceback:

DSA	Entry	E	Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHDSP	+00004030			CEEPLPKA	CEEHDSP	D1908	Call
2	CEL4ABD0	+0000024C			CEEPLPKA	CEL4ABD0	D1908	Exception
3	CEEHABD	+00000074			CEEPLPKA	CEEHABD	D1908	Call
4	tran3	+000000D6	26	XNTDLL		trandll.c		Call
5	CEEVRONU	+00001026			CEEPLPKA	CEEVRONU	D1908	Call
6	tran2	+00000070	27	XNTRAN		tranmain.c		Call
7	CEEVROND	+000011FA			CEEPLPKA			Call
8	tran1	+000000F2	14	XNTDLL		trandll.c		Call
9	CEEVRONU	+00001026			CEEPLPKA	CEEVRONU	D1908	Call
10	main	+0000008C	18	XNTRAN		tranmain.c		Call
11	CEEVROND	+000011FA			CEEPLPKA			Call
12	EDCZHINV	+000000B4			CELHV003	EDCZHINV	D1908	Call
13	CEEBBEXT	+000001B6			CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Compile	Attributes
1	2110CB08	209C4238	209C4238	209C4238	+00004030	20061215	CEL	
2	2110CA60	20AFCA08	20AFCA08	20AFCA08	+0000024C	20061215	CEL	
3	2110C9C8	209BFC00	209BFC00	209BFC00	+00000074	20061215	CEL	
4	2110C910	212BB8C0	212BB8C0	212BB8C0	+000000D6	20000505	C/C++	
5	2110C750	20ACCA58	20ACCA58	20ACCA58	+00001026	20061215	CEL	
6	212B6530	209000E8	209000E8	209000E8	+00000070	20000421	C/C++	XPLINK EBCDIC HFP
7	212B65B0	20ACAAA0	20ACAAA0	20ACAAA0	+00001252	20061215	CEL	XPLINK EBCDIC HFP
8	2110C500	212BBA40	212BBA40	212BBA40	+000000F2	20000505	C/C++	
9	2110C340	20ACCA58	20ACCA58	20ACCA58	+00001026	20061215	CEL	
10	212B6680	20900218	20900218	20900218	+0000008C	20000421	C/C++	XPLINK EBCDIC HFP
11	212B6720	20ACAAA0	20ACAAA0	20ACAAA0	+00001252	20061215	CEL	XPLINK EBCDIC HFP
12	2110C0F0	20C386A8	20C386A8	20C386A8	+000000B4	20061214	LIBRARY	
13	2110C030	20992208	20992208	20992208	+000001B6	20061215	CEL	

Fully Qualified Names

DSA	Entry	Program Unit	Load Module
4	tran3	./trandll.c	XNTDLL
6	tran2	./tranmain.c	XNTRAN
8	tran1	./trandll.c	XNTDLL
10	main	./tranmain.c	XNTRAN

Condition Information for Active Routines

Condition Information for CEL4ABD0 (DSA address 2110CA60)

CIB Address: 2110D428

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: CEL4ABD0 Entry: CEL4ABD0 Statement: Offset: +0000024C

Machine State:

ILC.... 0002 Interruption Code.... 0000

PSW.... 078D1400 A0AFCC54

GPR0.... 84000000 GPR1.... 840003E9 GPR2.... 00000000 GPR3.... 2110C9B0

GPR4.... 00000001 GPR5.... 20914E10 GPR6.... 00000002 GPR7.... 00000000

GPR8.... A0900003 GPR9.... 212BB868 GPR10.... 209BFD4C GPR11.... 20AFCA08

GPR12.... 209139B0 GPR13.... 2110CA60 GPR14.... A09BFD46 GPR15.... 00000000

ABEND code: 000003E9 Reason code: 00000000

Storage dump near condition, beginning at location: 20AFCC44

+000000 20AFCC44 88100008 41000084 89000018 16100A0D 58D00004 98ECD00C 07FE0000 D7C1E3C3 |h.....di.....q.....PATC|

GPREG STORAGE:

Storage around GPR0 (04000000)
 -0020 03FFFFFF0 Inaccessible storage.
 +0000 040000000 Inaccessible storage.
 +0020 04000020 Inaccessible storage.

:

Figure 54. Example dump of calling between XPLINK and non-XPLINK programs (Part 1 of 3)


```

[2]Parameters, Registers, and Variables for Active Routines:
:
tran3 (DSA address 2110C910):
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..... 20914D70 GPR1..... 2110C9A8 GPR2..... 2110C9B4 GPR3..... 212BB8FA
GPR4..... 2110C9B0 GPR5..... 20914E10 GPR6..... 00000000 GPR7..... 00000000
GPR8..... A0900003 GPR9..... 212BB868 GPR10.... 212BB8C0 GPR11.... 20ACDF1C
GPR12.... 209139B0 GPR13.... 2110C910 GPR14.... A12BB998 GPR15.... A09BFCD0
GPREG STORAGE:
Storage around GPR0 (20914D70)
-0020 20914D50 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 20914D70 180F58FF 001007FF 212BB3A0 20914D70 212BB3B0 00000000 00000000 00000000 |.....j(.....|
+0020 20914D90 180F58FF 001007FF 212BB0B0 20914D70 212BB3B0 00000000 00000000 00000000 |.....j(.....|
:
Storage around GPR15(209BFCD0)
-0020 209BFCD0 209BFA00 F2F0F0F6 F1F2F1F5 F1F1F5F2 F0F0F0F1 F0F9F0F0 0005C4F1 F9F0F800 |...20061215115200010900..D1908.|
+0000 209BFCD0 47F0F014 00C3C5C5 00000098 000000E0 47F0F001 90ECD00C 18BF5800 B0D05810 |.00..CEE...q.....00.....|
+0020 209BFCD0 D04C1E01 5500C00C 47D0B034 58F0C2BC 05EF181F 5000104C D7011000 100018FD |.<.....0B.....&..<P.....|
Local Variables:
timing      signed long int 1
code       signed long int 1001
CEEVRONU (DSA address 2110C750):
TRANSITION DSA
Saved Registers:
GPR0..... 20914D70 GPR1..... 212B6D70 GPR2..... 212BBE18 GPR3..... 0000001F
GPR4..... 212B6530 GPR5..... 21109718 GPR6..... 00000000 GPR7..... 00000000
GPR8..... A0900003 GPR9..... 212BB868 GPR10.... 212BB8C0 GPR11.... 20ACDF1C
GPR12.... 209139B0 GPR13.... 2110C750 GPR14.... A0ACDA80 GPR15.... 212BB8C0
:
tran2 (DSA address 212B6530):
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..... 212B6530 GPR5..... 21109718 GPR6..... 20ACCAD8 GPR7..... A090015A
GPR8..... A09000F2 GPR9..... 20914E80 GPR10.... 209000B8 GPR11.... A0ACAA48
GPR12.... 209139B0 GPR13.... 2110C5C8 GPR14.... 209000B8 GPR15.... 0000000C
:
Local Variables:
retval     signed int      -455613482
CEEVROND (DSA address 212B65B0):
TRANSITION DSA
Saved Registers:
GPR0..... ***** GPR1..... ***** GPR2..... ***** GPR3..... *****
GPR4..... 212B65B0 GPR5..... 20914E80 GPR6..... 209000E8 GPR7..... A0ACBC9C
GPR8..... 209000B8 GPR9..... 20ACBA47 GPR10.... ***** GPR11.... *****
GPR12.... ***** GPR13.... ***** GPR14.... ***** GPR15.... *****
:
tran1 (DSA address 2110C500):
UPSTACK DSA
Saved Registers:
GPR0..... 211080A8 GPR1..... 2110C598 GPR2..... 55555555 GPR3..... 212BBA7A
GPR4..... 33333333 GPR5..... 20914E10 GPR6..... 22222222 GPR7..... 11111111
GPR8..... A0900203 GPR9..... 212BB9E8 GPR10.... 212BBA40 GPR11.... 20ACDF1C
GPR12.... 209139B0 GPR13.... 2110C500 GPR14.... A12BBB34 GPR15.... A0ACAA48
:

```

Figure 54. Example dump of calling between XPLINK and non-XPLINK programs (Part 2 of 3)

[3]Control Blocks for Active Routines:

```
DSAs for tran3: 2110C910
+000000  FLAGS.... 1010      member... 803C      BKC..... 2110C750  FWC..... 2110C9C8  R14..... A12BB998
+000010  R15..... A09BFC00  R0..... 20914D70  R1..... 2110C9A8  R2..... 2110C9B4  R3..... 212BB8FA
+000024  R4..... 2110C980  R5..... 20914E10  R6..... 00000000  R7..... 00000000  R8..... A0900003
+000038  R9..... 212BB868  R10..... 212BB8C0  R11..... 20ACDF1C  R12..... 209139B0  reserved. 00000000
+00004C  NAB..... 2110C9C8  PNAB.... 00000001  reserved. 20914E70 209011DC
+000064  reserved. A0A9AED0  reserved. 209139B0  MODE..... 20A9CD06  reserved. 20912668
+000078  reserved. A0A8299C  reserved. 00000000

DSAs for CEEVRONU: 2110C750
+000000  FLAGS.... 0000      member... 0000      BKC..... FFFFFFFF  FWC..... 2110C910  R14..... A0ACDA80
+000010  R15..... 212BB8C0  R0..... 20914D70  R1..... 212B6D70  R2..... 212BBE18  R3..... 0000001F
+000024  R4..... 212B6530  R5..... 21109718  R6..... 00000000  R7..... 00000000  R8..... A0900003
+000038  R9..... 212BB868  R10..... 212BB8C0  R11..... 20ACDF1C  R12..... 209139B0  reserved. 00000000
+00004C  NAB..... 2110C910  PNAB.... 2110C910  reserved. 00000000 00000000
+000064  reserved. 2110C7D0  reserved. 00000000  MODE..... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000

DSAs for CEEVRONU: 2110C7D0
+000000  EYE..... DOWNTOUNP  TRTYPE... 00000003  BOS..... 00000000  STACKFLR. 00000000  SSTOPD... 212B6680
+000018  SSDSAU... 2110C750  TRANEP... 20ACCA58  TR_R0... 55555555  TR_R1... 11111111  TR_R2... 22222222
+00002C  TR_R3... 33333333  TR_R4... 212B6530  TR_R5... 21109718  TR_R6... 20ACCAD8  TR_R7... A090015A
+000040  TR_R8... A09000F2  TR_R9... 20914E80  TR_R10... 209000B8  TR_R11... A0ACAA48  TR_R12... 209139B0
+000054  TR_R13... 2110C5C8  TR_R14... 209000B8  TR_R15... 0000000C  CRENT.... 00000000  ROND_DSA. 2110C5C8
+000068  INTF MAP. 018F1000

DSAs for tran2: 212B6D30
+000000  R4..... 212B65B0  R5..... 20914E80  R6..... 209000E8  R7..... A0ACBC9C  R8..... 209000B8
+000014  R9..... 20ACBA47  R10..... 209000B8  R11..... A0ACAA48  R12..... 209139B0  R13..... 2110C5C8
+000028  R14..... 209000B8  R15..... 0000000C  reserved. 00000A68  reserved. 20F2E0C7  HPTRAN... 00000000
+00003C  reserved. 55555555  reserved. 11111111

DSAs for CEEVROND: 212B6DB0
+000000  R4..... E3D9C1D5  R5..... 00000000  R6..... 20ACAAA0  R7..... 00000000  R8..... 2110C8B0
+000014  R9..... 00000000  R10..... 00000000  R11..... A0A82AE8  R12..... 00000000  R13..... 00000000
+000028  R14..... 212B6E10  R15..... 00000000  reserved. 20914E1C  reserved. 2110C5C0  HPTRAN... 212B6E10
+00003C  reserved. 00000000  reserved. 11111111

DSAs for CEEVROND: 212B6E10
+000000  EYE..... UPTODOWN  TRTYPE... 00000002  BOS..... 00000000  STACKFLR. 00000000  SSTOPD... 212B6680
+000018  SSDSAU... 2110C340  TRANEP... 20ACAAA0  TR_R0... 00000000  TR_R1... 00000000  TR_R2... 00000000
+00002C  TR_R3... 00000000  TR_R4... 2110C500  TR_R5... 00000000  TR_R6... 00000000  TR_R7... A12BBB34
+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.... A0ACAA48  ROND_DSA. 00000000
+000068  INTF MAP. 00000000

DSAs for tran1: 2110C500
+000000  FLAGS.... 1000      member... 0000      BKC..... 2110C340  FWC..... 2110C8B0  R14..... A12BBB34
+000010  R15..... A0ACAA48  R0..... 211080A8  R1..... 2110C598  R2..... 55555555  R3..... 212BBA7A
+000024  R4..... 33333333  R5..... 20914E10  R6..... 22222222  R7..... 11111111  R8..... A0900203
+000038  R9..... 212BB9E8  R10..... 212BBA40  R11..... 20ACDF1C  R12..... 209139B0  reserved. 00000000
+00004C  NAB..... 2110C5C8  PNAB.... 00000000  reserved. 00000000 2110C4A8
+000064  reserved. 2110C584  reserved. 20912658  MODE..... 2110C5A4  reserved. 21135000
+000078  reserved. 00000001  reserved. 21108020

:
CEE3846I CEEUDMP Processing completed.
```

Figure 54. 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.
- Trace entry 7 occurs when an XPLINK function calls a non-XPLINK function.
- Trace entry 8 occurs when a non-XPLINK function calls an XPLINK function.

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 ERRNO=value
```

The format for trace table entry 3 is:

```
NameOfCallingFunction  
-->(xxx) NameOfCalledFunction
```

The format for trace table entry 4 is:

```
<--(xxx) R15=value ERRNO=value ERRNO2=value
```

The format for trace table entry 5 is:

```
NameOfCallingFunction  
-->(xxxx) NameOfCalledFunction<(input_parameters)>
```

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=xxxxxxxx R2=xxxxxxxx R3=xxxxxxxx ERRNO=xxxxxxxx ERRNO2=xxxxxxxx
```

In all 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.

The format for trace table entry 7 is:

```
ModuleNameOfCallingFunction:NameOfCallingXplinkFunction  
-->ModuleNameOfCalledFunction:NameOfCalledNonXplinkFunction
```

The format for trace table entry 8 is:

```
ModuleNameOfCallingFunction:NameOfCallingNonXplinkFunction  
-->ModuleNameOfCalledFunction:NameOfCalledXplinkFunction
```

For entry types 7 and 8, 16 bytes is for the module name and 32 bytes is for the function name. If the name is longer than 16 or 32 bytes, an extra trace entry is taken. The name is truncated and only the first 32/64(16/32) bytes will appear in the trace table entry. Also, a module name might not always be located, such as when a DLL is freed. If that occurs, "UNKNOWN" appears for the module name in the trace table entry.

Figure 55 shows a non-XPLINK trace which has examples of C/C++ trace table entry types 1 thru 4.

```

:
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 |main
+000038  60606E4D F1F3F95D 40A2A399 8397A84D 5D404040 40404040 40404040 40404040 40404040 |-->(139) strcpy()
+000058  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 |0000.....
+0000D8  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 |main
+000138  60606E4D F1F3F95D 40A2A399 8397A84D 5D404040 40404040 40404040 40404040 |-->(139) strcpy()
+000158  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 |0000.....
+0001D8  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 |Igetparms
+000238  60606E4D F0F5F25D 4089A281 A3A3A84D 5D404040 40404040 40404040 40404040 |-->(052) isatty()
+000258  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 |.....
+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 |Igetparms
+000338  60606E4D F0F5F25D 4089A281 A3A3A84D 5D404040 40404040 40404040 40404040 |-->(052) isatty()
+000358  40404040 40404040 40404040 40404040 40404040 40404040 40000000 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 |.....
+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 |Igetparms
+000438  60606E4D F0F5F25D 4089A281 A3A3A84D 5D404040 40404040 40404040 40404040 |-->(052) isatty()
+000458  40404040 40404040 40404040 40404040 40404040 40404040 40000000 00000000 |.....
+000478  00000000 00000000

```

Figure 55. 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 55. Trace table with C/C++ trace table entry types 1 thru 4 (Part 2 of 2)

Figure 56 on page 203 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 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 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 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 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 00000000	
+000478	00000000 00000000	

Figure 56. Trace table with XPLINK trace table entries 5 and 6.

The following is an example of the format of the trace table entry type 7 and 8 :

```

Calculator      :calculatethesumoftwointegrs    -->
exceptionhandler:exceptionhandlersub1

```

The following is an example of a dump of the trace table when you specify the LE=20 suboption:

```

:
Language Environment Trace Table:
Most recent trace entry is at displacement: 000800

```

Displacement	Trace Entry in Hexadecimal	Trace Entry in EBCDIC
+000000	Time 22.10.56.799195 Date 2005.05.01 Thread ID... 21DEC83000000000	
+000010	Member ID.... 03 Flags..... 000000 Entry Type..... 00000008	
+000018	C3C5D3C8 E5F0F0F3 40404040 40404040 7AC5C4C3 E9C8C9D5 E5404040 40404040	CELHV003 :EDCZHINW
+000038	40404040 40404040 40404040 40404040 4060606E 81F8F5F9 83F4F1A7 40404040	-->a859c41x
+000058	40404040 7A948189 95404040 40404040 40404040 40404040 40404040 40404040	:main
+000078	40404040 404040F1	1
+000080	Time 22.10.56.804695 Date 2005.05.01 Thread ID... 21DEC83000000000	
+000090	Member ID.... 03 Flags..... 000000 Entry Type..... 00000007	
+000098	C3C5C5D7 D3D7D2C1 40404040 40404040 7AC3C5C5 D7C8E3D3 C3404040 40404040	CEEPLPKA :CEEPHTLC
+0000B8	40404040 40404040 40404040 40404040 4060606E C3C5C5D7 D3D7D2C1 40404040	-->CEEPLPKA
+0000D8	40404040 7AC3C5C5 D7E3D3D6 D9404040 40404040 40404040 40404040 40404040	:CEEPTLOR
+0000F8	40404040 404040F1	1
+000100	Time 22.10.56.825094 Date 2005.05.01 Thread ID... 21DEC83000000000	
+000110	Member ID.... 03 Flags..... 000000 Entry Type..... 00000007	
+000118	81F8F5F9 83F4F1A7 40404040 40404040 7A948189 95404040 40404040 40404040	a859c41x :main
+000138	40404040 40404040 40404040 40404040 4060606E 81F8F5F9 83F4F186 9593F3F4	-->a859c41fn134
+000158	40404040 7A86A495 83A38996 956D9581 94856D93 859587A3 886D8598 A48193A2	:function_name_length_equals
+000178	6DA3966D F04040F1	_to_0 1
+000180	Time 22.10.56.825094 Date 2005.05.01 Thread ID... 21DEC83000000000	
+000190	Member ID.... 03 Flags..... 000000 Entry Type..... 00000007	
+000198	40404040 40404040 40404040 40404040 7A404040 40404040 40404040 40404040	:
+0001B8	40404040 40404040 40404040 40404040 4060606E 40404040 40404040 40404040	-->
+0001D8	40404040 7AF3F440 40404040 40404040 40404040 40404040 40404040 40404040	:34
+0001F8	40404040 404040F2	2
+000200	Time 22.10.56.826629 Date 2005.05.01 Thread ID... 21DEC83000000000	
+000210	Member ID.... 03 Flags..... 000000 Entry Type..... 00000008	
+000218	81F8F5F9 83F4F186 9593F3F4 40404040 7A86A495 83A38996 956D9581 94856D93	a859c41fn134 :function_name_
+000238	859587A3 886D8598 A48193A2 6DA3966D F060606E C3C5D3C8 E5F0F0F3 40404040	engh_equals_to_0-->CELHV003
+000258	40404040 7A979989 95A38640 40404040 40404040 40404040 40404040 40404040	:printf
+000278	40404040 404040F1	1
+000280	Time 22.10.56.826629 Date 2005.05.01 Thread ID... 21DEC83000000000	
+000290	Member ID.... 03 Flags..... 000000 Entry Type..... 00000008	
+000298	40404040 40404040 40404040 40404040 7AF3F440 40404040 40404040 40404040	:34
+0002B8	40404040 40404040 40404040 40404040 4060606E 40404040 40404040 40404040	-->
+0002D8	40404040 7A404040 40404040 40404040 40404040 40404040 40404040 40404040	:
+0002F8	40404040 404040F2	2
+000300	Time 22.10.56.826670 Date 2005.05.01 Thread ID... 21DEC83000000000	
+000310	Member ID.... 03 Flags..... 000000 Entry Type..... 00000008	
+000318	81F8F5F9 83F4F186 9593F3F4 40404040 7A86A495 83A38996 956D9581 94856D93	a859c41fn134 :function_name_
+000338	859587A3 886D8598 A48193A2 6DA3966D F060606E C3C5D3C8 E5F0F0F3 40404040	engh_equals_to_0-->CELHV003
+000358	40404040 7A979989 95A38640 40404040 40404040 40404040 40404040 40404040	:printf
+000378	40404040 404040F1	1
+000380	Time 22.10.56.826670 Date 2005.05.01 Thread ID... 21DEC83000000000	
+000390	Member ID.... 03 Flags..... 000000 Entry Type..... 00000008	
+000398	40404040 40404040 40404040 40404040 7AF3F440 40404040 40404040 40404040	:34
+0003B8	40404040 40404040 40404040 40404040 4060606E 40404040 40404040 40404040	-->
+0003D8	40404040 7A404040 40404040 40404040 40404040 40404040 40404040 40404040	:
+0003F8	40404040 404040F2	2

Figure 57. Trace table with trace table entry types 7 and 8 (Part 1 of 2)


```

+000400 Time 22.10.56.826697 Date 2005.05.01 Thread ID... 21DEC83000000000
+000410 Member ID.... 03 Flags..... 000000 Entry Type..... 00000008
+000418 81F8F5F9 83F4F186 9593F3F4 40404040 7A86A495 83A38996 956D9581 94856D93 |a859c41fn134 :function_name_|
+000438 859587A3 886D8598 A48193A2 6DA3966D F060606E C3C5D3C8 E5F0F0F3 40404040 |ength_equals_to_0-->CELHV003|
+000458 40404040 7A979989 95A38640 40404040 40404040 40404040 40404040 |:printf|
+000478 40404040 404040F1 |1|

+000480 Time 22.10.56.826697 Date 2005.05.01 Thread ID... 21DEC83000000000
+000490 Member ID.... 03 Flags..... 000000 Entry Type..... 00000008
+000498 40404040 40404040 40404040 40404040 7AF3F440 40404040 40404040 40404040 |:34|
+0004B8 40404040 40404040 40404040 40404040 4060606E 40404040 40404040 40404040 |-->|
+0004D8 40404040 7A404040 40404040 40404040 40404040 40404040 40404040 |:|
+0004F8 40404040 404040F2 |2|

+000500 Time 22.10.56.836542 Date 2005.05.01 Thread ID... 21DEC83000000000
+000510 Member ID.... 03 Flags..... 000000 Entry Type..... 00000008
+000518 81F8F5F9 83F4F186 9593F3F4 40404040 7A86A495 83A38996 956D9581 94856D93 |a859c41fn134 :function_name_|
+000538 859587A3 886D8598 A48193A2 6DA3966D F060606E 81F8F5F9 83F4F186 9593F3F5 |ength_equals_to_0-->a859c41fn135|
+000558 40404040 7A86A495 83A38996 956D9581 94856D93 859587A3 886D8598 A48193F3 |:function_name_length_equal3|
+000578 F56DA285 834040F1 |5_sec 1|

+000580 Time 22.10.56.836543 Date 2005.05.01 Thread ID... 21DEC83000000000
+000590 Member ID.... 03 Flags..... 000000 Entry Type..... 00000008
+000598 40404040 40404040 40404040 40404040 7AF3F440 40404040 40404040 40404040 |:34|
+0005B8 40404040 40404040 40404040 40404040 4060606E 40404040 40404040 40404040 |-->|
+0005D8 40404040 7A969584 40404040 40404040 40404040 40404040 40404040 |:ond|
+0005F8 40404040 404040F2 |2|

+000600 Time 22.10.56.836579 Date 2005.05.01 Thread ID... 21DEC83000000000
+000610 Member ID.... 03 Flags..... 000000 Entry Type..... 00000008
+000618 81F8F5F9 83F4F186 9593F3F4 40404040 7A86A495 83A38996 956D9581 94856D93 |a859c41fn134 :function_name_|
+000638 859587A3 886D8598 A48193A2 6DA3966D F060606E C3C5D3C8 E5F0F0F3 40404040 |ength_equals_to_0-->CELHV003|
+000658 40404040 7A979989 95A38640 40404040 40404040 40404040 40404040 |:printf|
+000678 40404040 404040F1 |1|

+000680 Time 22.10.56.836579 Date 2005.05.01 Thread ID... 21DEC83000000000
+000690 Member ID.... 03 Flags..... 000000 Entry Type..... 00000008
+000698 40404040 40404040 40404040 40404040 7AF3F440 40404040 40404040 40404040 |:34|
+0006B8 40404040 40404040 40404040 40404040 4060606E 40404040 40404040 40404040 |-->|
+0006D8 40404040 7A404040 40404040 40404040 40404040 40404040 40404040 |:|
+0006F8 40404040 404040F2 |2|

+000700 Time 22.10.56.836605 Date 2005.05.01 Thread ID... 21DEC83000000000
+000710 Member ID.... 03 Flags..... 000000 Entry Type..... 00000008
+000718 81F8F5F9 83F4F186 9593F3F4 40404040 7A86A495 83A38996 956D9581 94856D93 |a859c41fn134 :function_name_|
+000738 859587A3 886D8598 A48193A2 6DA3966D F060606E C3C5D3C8 E5F0F0F3 40404040 |ength_equals_to_0-->CELHV003|
+000758 40404040 7A979989 95A38640 40404040 40404040 40404040 40404040 |:printf|
+000778 40404040 404040F1 |1|

+000780 Time 22.10.56.836605 Date 2005.05.01 Thread ID... 21DEC83000000000
+000790 Member ID.... 03 Flags..... 000000 Entry Type..... 00000008
+000798 40404040 40404040 40404040 40404040 7AF3F440 40404040 40404040 40404040 |:34|
+0007B8 40404040 40404040 40404040 40404040 4060606E 40404040 40404040 40404040 |-->|
+0007D8 40404040 7A404040 40404040 40404040 40404040 40404040 40404040 |:|
+0007F8 40404040 404040F2 |2|

+000800 Time 22.10.56.836671 Date 2005.05.01 Thread ID... 21DEC83000000000
+000810 Member ID.... 01 Flags..... 000000 Entry Type..... 00001800
+000818 0125D23C A04F07F0 2033E150 20C121B8 00000001 00000010 00000000 20C121B8 |...K..|.0...&.A.....A..|
+000838 01000000 00000000 00000000 00000000 00000000 00000000 000000F2 00000000 |.....2....|
+000858 00000000 202D92B8 03000000 00000000 BC2F210D 2130666C 40404040 40404040 |.....k.....2.....%|
+000878 40404040 40404040

```

Additional Language Specific Information:

```

errno information :
Thread Id .... 21DEC83000000000 Errno ..... 0 Errnojr .... 00000000
:

```

Figure 57. Trace table with trace table entry types 7 and 8 (Part 2 of 2)

For more information about the Language Environment trace table format, see “Understanding the trace table entry (TTE)” on page 147.

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 58 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 58. C routine with a divide-by-zero error

To debug this routine, use the following steps:

1. Locate the Original 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'76' 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 59 on page 207 shows the generated traceback from the dump.

CEE3845I CEEDUMP Processing started.

Information for enclave main

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHDSP	+00004030		CEEPLPKA	CEEHDSP	D1908	Call
2	funcb	+00000076	18	CDIVZERO			Exception
3	main	+00000064	12	CDIVZERO			Call
4	EDCZMINV	+000000C2		CEEV003			Call
5	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	20FCB358	209C4238	209C4238	+00004030	20061215	CEL
2	20FCB2B0	20900960	20900960	+00000076	20070115	C/C++
3	20FCB208	209008E0	209008E0	+00000064	20070115	C/C++
4	20FCB0F0	20E699EE	20E699EE	+000000C2	20061215	LIBRARY
5	20FCB030	20992208	20992208	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for (DSA address 20FCB2B0)

CIB Address: 20FCBC78

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: 18 Offset: +00000076

Machine State:

```
ILC..... 0002      Interruption Code..... 0009
PSW..... 078D2400 A09009D8
GPR0..... 20900A08  GPR1..... 00000000  GPR2..... 20FCB2A8  GPR3..... 2090099A
GPR4..... 00000000  GPR5..... 00000001  GPR6..... 20900B24  GPR7..... 20900098
GPR8..... 00000030  GPR9..... 80000000  GPR10..... A0E699E2  GPR11..... A0992208
GPR12..... 209139B0  GPR13..... 20FCB2B0  GPR14..... 20914F50  GPR15..... 00000008
```

Storage dump near condition, beginning at location: 209009C6

```
+000000 209009C6 5811E000 504FE000 A71AFFB7 8E400020 1D4158F0 306A4110 D0985000 D0985050 |...&|...x.... .....0.....q&..q&|
```

GPREG STORAGE:

Storage around GPR0 (20900A08)

```
-0020 209009E8 0DEF18F5 58D0D004 58E0D00C 9825D01C 051E0707 00000000 00000008 209000B8 |...5.....q.....|
+0000 20900A08 D985A2A4 93A3407E 406C8415 00000000 1CCEA106 00000228 00000178 00000000 |Result = %d.....|
```

:

Figure 59. 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 60 on page 208.

The offset (within funcb) of the exception from the traceback (X'76') reveals the divide instruction: DR r4,r1 at that location. Instructions X'66' through X'76' 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          000015 |      * int funcb(int *pp) {
:          000015 |      funcb   DS      00
:
:
000046 50D0 E004          000015 |      ST      r13,4(,r14)
00004A 18DE          000015 |      LR      r13,r14
00004C          End of Prolog

00004C 58E0 C1F4          000000 |      L      r14,_CEECAA_(,r12,500)
000016      * int result;
000017      * fa = *pp;
000050 5820 1000          000017 |      L      r2,pp(,r1,0)
000054 5810 3062          000018 |      L      r1,=Q(statint)(,r3,98)
000058 58F0 3066          000017 |      L      r15,=Q(fa)(,r3,102)
00005C C000 0000 0026 000000 |      LARL   r0,'38'
000062 5840 2000          000017 |      L      r4,(*)int(,r2,0)
000018      * result = fa/(statint-73);
000018      L      r1,statint(r1,r14,0)
00006A 504F E000          000017 |      ST      r4,fa(r15,r14,0)
00006E A71A FFB7          000018 |      AHI    r1,H'-73'
000072 8E40 0020          000018 |      SRDA   r4,32
000076 1D41          000018 |      DR     r4,r1
000019      * printf("Result = %d\n",result);
000019      L      r15,=V(printf)(,r3,106)
00007C 4110 D098          000019 |      LA     r1,#MX_TEMP2(,r13,152)
000080 5000 D098          000019 |      ST     r0,#MX_TEMP2(,r13,152)
000084 5050 D09C          000019 |      ST     r5,#MX_TEMP2(,r13,156)
000088 0DEF          000019 |      BASR  r14,r15
000020      * return result;
00008A 18F5          000020 |      LR     r15,r5
000021      * }
00008C          000021 |      @2L3  DS      0H

00008C          Start of Epilog
00008C 58D0 D004          000021 |      L      r13,4(,r13)
000090 58E0 D00C          000021 |      L      r14,12(,r13)
000094 9825 D01C          000021 |      LM     r2,r5,28(r13)
000098 051E          000021 |      BALR  r1,r14
00009A 0707          000021 |      NOPR   7

00009C          Start of Literals
00009C 00000000          =Q(statint)
0000A0 00000000          =Q(fa)
0000A4 00000000          =V(printf)
0000A8          End of Literals

*** General purpose registers used: 1111110000001111
*** Floating point registers used: 1111111100000000
*** Size of register spill area: 128(max) 0(used)
*** Size of dynamic storage: 168
*** Size of executable code: 156

Constant Area
000000 D985A2A4 93A3407E 406C8415 00 |Result = %d.. |

PPA1: Entry Point Constants
000000 1CCEA106          =F'483303686'      Flags
000004 00000228          =A(PPA2-main)
000008 00000178          =A(PPA3-main)
00000C 00000000          =F'0'              No EPD
000010 FE000000          =F'-33554432'     Register save mask
000014 00000000          =F'0'              Member flags
000018 90              =AL1(144)         Flags
000019 000000          =AL3(0)           Callee's DSA use/8
00001C 0040          =H'64'            Flags
00001E 0012          =H'18'            Offset/2 to CDL
000020 00000000          =F'0'              Reserved
000024 5000003C          =F'1342177340'   CDL function length/2
000028 FFFFEC8          =F'-312'          CDL function EP offset
00002C 38260000          =F'942014464'    CDL prolog
000030 40090033          =F'1074331699'   CDL epilog
000034 00000000          =F'0'             CDL end
000038 0004 ****          AL2(4),C'main'

PPA1 End
:

```

Figure 60. Pseudo assembly listing

3. 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 169.

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'20914F50'. Figure 61 shows the WSA address.

```

Enclave Control Blocks:
:
:   WSA address.....20914F50
:
:

```

Figure 61. 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 62. In this example, the offset is X'0'.

```

:
:
:-----
CLASS  C_WSA          LENGTH =      AC  ATTRIBUTES = MRG, DEFER , RMODE=ANY
:          OFFSET =      0 IN SEGMENT 002          ALIGN = DBLWORD
:-----
:
:          CLASS
:          OFFSET  NAME          TYPE    LENGTH  SECTION
:          0      statint        PART    4       statint
:          8      fa             PART    4       fa
:          10     environ        PART    4       environ
:          18     errno          PART    4       errno
:
:

```

Figure 62. Writable static map

- Add the WSA address of X'20914F50' to the offset of `statint`. The result is X'20914F50'. This is the address of the variable `statint`, which is in the writable static area.

The writable static area is shown in the Enclave Storage section of the dump.

For a load module, 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. For a program object, the writable static area is storage allocated by the loader and is shown in the WSA for Program Object(s) section of the dump.

For this example, the program was built as a program object. The writable static area is displayed in the Enclave Storage section of the dump, shown in Figure 63 on page 210.

- To find the variable `statint` in the writable static area, locate the closest address listed that is before the address of `statint`. In this case, that address is X'20914F50'. Count across X'00' to location X'20914F50'. The value at that location is X'49' (that is, `statint` is 73), and hence the fixed point divide exception.

```

Enclave Storage:
:
WSA for Program Object(s)
WSA: 20914F50
+000000 20914F50 00000049 00000000 00000001 00000000 2090A880 00000000 00000000 00000000 |.....y.....|
+000020 20914F70 20910260 2091026A 00000000 00000000 00000000 00000000 00000000 00000000 |.j.-.j.....|
+000040 20914F90 00000001 00000000 00000001 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 20914FB0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 20914FD0 00000000 00000000 00000000 00000000 2090F6BC 00000000 2090F28C 00000000 |.....6.....2.....|
+0000A0 20914FF0 2090F4A4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.4u.....|
:

```

Figure 63. Enclave storage section of dump

Calling a nonexistent non-XPLINK function

Figure 64 demonstrates the error of calling a nonexistent function. This routine was compiled with the compiler options LIST, OFFSET, 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.

```

#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 Original Condition message in the Condition Information for Active Routines section of the dump, shown in Figure 65 on page 211. 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 Location section of the dump indicates that the exception occurred at offset X'-20900978' within function funca and that there may have been a bad branch from offset X'+0000005A' 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'80000002' in the instruction

address of the PSW. This address indicates that an instruction in the routine branched outside the bounds of the routine.

CEE3DMP V1 R9.0: Condition processing resulted in the unhandled condition. 01/15/07 5:38:23 PM Page: 1
 ASID: 0049 Job ID: JOB21060 Job name: EXIST Step name: STEP1 UserID: HEALY

CEE3845I CEEDUMP Processing started.

Information for enclave main

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHDSP	+00004030		CEEPLPKA	CEEHDSP	D1908	Call
2	funca	-20900978		EXIST			Exception
3	main	+0000005C		EXIST			Call
4	EDCZMINV	+000000C2		CEEEV003			Call
5	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	20FCB350	209D2B08	209D2B08	+00004030	20061215	CEL
2	20FCB2B0	20900978	20900978	-20900978	20070115	C/C++
3	20FCB208	209008E0	209008E0	+0000005C	20070115	C/C++
4	20FCB0F0	20E699EE	20E699EE	+000000C2	20061215	LIBRARY
5	20FCB030	209A0AD8	209A0AD8	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for (DSA address 20FCB2B0)

CIB Address: 20FCBC70

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: -20900978

Possible Bad Branch: Statement: Offset: +0000005A

Machine State:

ILC..... 0002 Interruption Code..... 0001

PSW..... 078D1400 80000002

GPR0..... 20FCB350 GPR1..... 20FCB2A0 GPR2..... 20FCB2A0 GPR3..... 209009B2

GPR4..... A09A0BBC GPR5..... 20912648 GPR6..... 20900AA4 GPR7..... 20900098

GPR8..... 00000030 GPR9..... 80000000 GPR10.... A0E699E2 GPR11.... A09A0AD8

GPR12.... 209139B0 GPR13.... 20FCB2B0 GPR14.... A09009D4 GPR15.... 00000000

Storage dump near condition, beginning at location: 00000000

+000000 00000000 Inaccessible storage.

GPREG STORAGE:

Storage around GPR0 (20FCB350)

-0020 20FCB330 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

+0000 20FCB350 0808CEE1 20FCB2B0 20FCE470 A09D6B3A A09EFFD8 20FCB350 20FCB7A8 20912648

.....
U.....Q...&...y.j..

Parameters, Registers, and Variables for Active Routines:

CEEHDSP (DSA address 20FCB350):

UPSTACK DSA

Saved Registers:

GPR0..... 20FCB350 GPR1..... 20FCB7A8 GPR2..... 20912648 GPR3..... 00000080

GPR4..... 209D7734 GPR5..... A0915000 GPR6..... 2090C2A8 GPR7..... 20FCBC70

GPR8..... A09D665A GPR9..... 20FCD34E GPR10.... 20FCC34F GPR11.... 209D2B08

GPR12.... 209139B0 GPR13.... 20FCB350 GPR14.... A09D6B3A GPR15.... A09EFFD8

funca (DSA address 20FCB2B0):

UPSTACK DSA

Saved Registers:

GPR0..... 20FCB350 GPR1..... 20FCB2A0 GPR2..... 20FCB2A0 GPR3..... 209009B2

GPR4..... A09A0BBC GPR5..... 20912648 GPR6..... 20900AA4 GPR7..... 20900098

GPR8..... 00000030 GPR9..... 80000000 GPR10.... A0E699E2 GPR11.... A09A0AD8

GPR12.... 209139B0 GPR13.... 20FCB2B0 GPR14.... A09009D4 GPR15.... 00000000

Figure 65. Sections of the dump from example C routine

- Find the branch instructions at offset X'+0000005A' of funca in the listing in Figure 66 on page 212. The instruction is BASR r14,r15. This branch is part of

the source statement *aa = func_ptr();

OFFSET	OBJECT CODE	LINE#	FILE#	PSEUDO	ASSEMBLY	LISTING
000000		000016		* void funca(int* aa) {		
.		000016		funca DS	0D	
.						
000046	50D0 E004	000016		ST	r13,4(,r14)	
00004A	18DE	000016		LR	r13,r14	
00004C				End of Prolog		
00004C	58E0 C1F4	000000		L	r14,_CEECAA_(,r12,500)	
000050	58F0 303A	000017		* *aa = func_ptr();		
000054	1821	000016		L	r15,=Q(func_ptr)(,r3,58)	
000056	58FF E000	000017		LR	r2,r1	
00005A	0DEF	000017		L	r15,func_ptr(r15,r14,0)	
00005C	5810 2000	000017		BASR	r14,r15	
000060	50F0 1000	000017		L	r1,aa(,r2,0)	
		000017		ST	r15,(*)int(,r1,0)	
		000018		* return;		
		000019		* }		
000064		000019		@2L3 DS	0H	
000064				Start of Epilog		
000064	58D0 D004	000019		L	r13,4(,r13)	
000068	58E0 D00C	000019		L	r14,12(,r13)	
00006C	9824 D01C	000019		LM	r2,r4,28(r13)	
000070	051E	000019		BALR	r1,r14	
000072	0707	000019		NOPR	7	

Figure 66. Pseudo assembly listing

- Find the offset of func_ptr in the Writable Static Map, shown in Figure 67, as produced by the binder.

```

:
-----
CLASS C_WSA          LENGTH =      A4  ATTRIBUTES = MRG, DEFER , RMODE=ANY
                     OFFSET =      0  IN SEGMENT 002      ALIGN = DBLWORD
-----

```

CLASS	OFFSET	NAME	TYPE	LENGTH	SECTION
	0	func_ptr	PART	4	func_ptr
	8	environ	PART	4	environ
	10	errno	PART	4	errno
	18	tzname	PART	8	tzname

```

:

```

Figure 67. Writable static map

- Add the offset of FUNC@PTR (X'0') to the address of WSA (X'20914F58'). The result (X'20914F58') 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 213 shows the sections of the dump.


```

:
: Enclave Control Blocks:
:
: WSA address.....20914F58
:
: Enclave Storage:
:
: WSA for Program Object(s)
: WSA: 20914F58
+000000 20914F58 00000000 00000000 2090A880 00000000 00000000 00000000 20910260 2091026A |.....y.....j.-.j..|
+000020 20914F78 00000000 00000000 00000000 00000000 00000000 00000000 00000001 00000000 |.....|
+000040 20914F98 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 20914FB8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 20914FD8 00000000 00000000 2090F6BC 00000000 2090F28C 00000000 2090F4A4 00000000 |.....6.....2.....4u...|
+0000A0 20914FF8 00000000 00000000 A099FF10 A09C4A58 A09D0FD8 A09D7E98 A09D2B08 A09D9A78 |.....r.....Q..=q.....|
:

```

Figure 68. Enclave control blocks and storage sections in dump

Calling a nonexistent XPLINK function

Figure 69 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 69. C/C++ example of calling a nonexistent subroutine

To debug this routine, use the following steps:

1. Locate the Original Condition message in the Condition Information for Active Routines section of the dump, shown in Figure 70 on page 214. 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 location section of the dump indicates that the exception occurred at offset X'-20900158' within function funca and that there may have been a bad branch from offset X'+0000001C'. 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.

CEE3DMP V1 R9.0: Condition processing resulted in the unhandled condition. 01/15/07 1:41:48 PM Page: 1
 ASID: 0051 Job ID: JOB06606 Job name: XEXIST Step name: STEP1 UserID: HEALY

CEE3845I CEEDUMP Processing started.

Information for enclave main

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHDSP	+00004030		CEEPLPKA	CEEHDSP	D1908	Call
2	CEEHRNUH	+00000092		CEEPLPKA	CEEHRNUH	D1908	Call
3	funca	-20900158		XEXIST			Exception
4	main	+00000012		XEXIST			Call
5	CEEVROND	+000011FA		CEEPLPKA			Call
6	EDCZHINV	+000000B4		CELHV003	EDCZHINV	D1908	Call
7	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	2110C500	209D2B08	209D2B08	+00004030	20061215	CEL
2	2110C340	209E0B80	209E0B80	+00000092	20061215	CEL
3	211B5620	20900158	20900158	-20900158	20000404	C/C++ XPLINK EBCDIC HFP
4	211B56A0	209000D0	209000D0	+00000012	20000404	C/C++ XPLINK EBCDIC HFP
5	211B5720	20ACA1E0	20ACA188	+00001252	20061215	CEL XPLINK EBCDIC HFP
6	2110C0F0	20C36CE8	20C36CE8	+000000B4	20061214	LIBRARY
7	2110C030	209A0AD8	209A0AD8	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for (DSA address 211B5620)

CIB Address: 2110CE20

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: -20900158

Possible Bad Branch: Statement: Offset: +0000001C

Machine State:

ILC..... 0002 Interruption Code..... 0001

PSW.... 078D2400 80000002

GPR0..... 2110C500 GPR1..... 211B5F00 GPR2..... 2110C294 GPR3..... 2110C298

GPR4..... 211B5620 GPR5..... 00FDD100 GPR6..... 00000000 GPR7..... A0900176

GPR8..... A09000DA GPR9..... 20ACB187 GPR10.... 2110C1A0 GPR11.... A0ACA188

GPR12.... 209139B0 GPR13.... 2110C1B8 GPR14.... 00000000 GPR15.... 00000000

Storage dump near condition, beginning at location: 00000000

+000000 00000000 Inaccessible storage.

GPREG STORAGE:

Storage around GPR0 (2110C500)

-0020 2110C4E0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
+0000 2110C500	0808CEE1	2110C340	2110F620	A09D6B3A	A09EFD8	2110C500	2110C958	20912648	C .6....Q.E...I..j..
+0020 2110C520	00000080	209D7734	A0915000	2090C2A8	2110CE20	A09D665A	2110E4FE	2110D4FF	j&...By.....!..U...M.

Figure 70. Sections of the dump from example C routine (Part 1 of 2)

```

Parameters, Registers, and Variables for Active Routines:
CEEHDSP (DSA address 2110C500):
UPSTACK DSA
Saved Registers:
GPR0..... 2110C500 GPR1..... 2110C958 GPR2..... 20912648 GPR3..... 00000080
GPR4..... 209D7734 GPR5..... A0915000 GPR6..... 2090C2A8 GPR7..... 2110CE20
GPR8..... A09D665A GPR9..... 2110E4FE GPR10.... 2110D4FF GPR11.... 209D2B08
GPR12.... 209139B0 GPR13.... 2110C500 GPR14.... A09D6B3A GPR15.... A09EFD8
:
CEEHRNUH (DSA address 2110C340):
TRANSITION DSA
Saved Registers:
GPR0..... 2110C500 GPR1..... 00000000 GPR2..... 2090B3B0 GPR3..... 2090B458
GPR4..... 209E0B80 GPR5..... 211B5620 GPR6..... 2110C340 GPR7..... 209E625C
GPR8..... 940C1000 GPR9..... 00000000 GPR10.... 2090B430 GPR11.... 209E0B80
GPR12.... 209139B0 GPR13.... 2110C340 GPR14.... A09E0C14 GPR15.... 209D2B08
:
funcna (DSA address 211B5620):
DOWNSTACK DSA
Saved Registers:
GPR0..... 2110C500 GPR1..... 211B5F00 GPR2..... 2110C294 GPR3..... 2110C298
GPR4..... 211B5620 GPR5..... 00FDD100 GPR6..... 00000000 GPR7..... A0900176
GPR8..... A09000DA GPR9..... 20ACB187 GPR10.... 2110C1A0 GPR11.... A0ACA188
GPR12.... 209139B0 GPR13.... 2110C1B8 GPR14.... 00000000 GPR15.... 00000000
:

```

Figure 70. Sections of the dump from example C routine (Part 2 of 2)

- Find the branch instruction at offset X'+0000001C' of funcna in the listing in Figure 71. This instruction is BASR r7,r6. This branch is part of the source statement *aa = func_ptr().

<pre> : 000020 000020 00C300C5 000024 00C500F1 000028 FFFFFFFE0 00002C 00000080 000000 000000 9057 4784 000004 A74A FF80 000008 000008 5010 48C0 00000C 5860 4804 000010 5860 6018 000014 5860 6000 000018 9856 6010 00001C 0D76 00001E 4700 0004 000022 1803 000024 5860 48C0 000028 5000 6000 : 00002C 00002C 5870 480C 000030 4140 4080 000034 07F7 : </pre>	<pre> 00015 * void funcna(int* aa) { : @2L0 DS 0D =F'12779717' XPLink entrypoint marker =F'12910833' =F'-32' =F'128' 00015 funcna DS 0D 00015 STM r5,r7,1924(r4) 00015 AHI r4,H'-128' End of Prolog 00015 ST r1,aa(,r4,2240) 00016 * *aa = func_ptr(); 00016 L r6,#Save_ADA_Ptr_2(,r4,2052) 00016 L r6,=A(func_ptr)(,r6,24) 00016 L r6,func_ptr(,r6,0) 00016 LM r5,r6,&ADA_&EPA(r6,16) 00016 BASR r7,r6 00016 NOP 4 00016 LR r0,r3 00016 L r6,aa(,r4,2240) 00016 ST r0,(*)int(,r6,0) 00017 * return; 00018 * } 00018 @2L3 DS 0H Start of Epilog 00018 L r7,2060(,r4) 00018 LA r4,128(,r4) 00018 BR r7 </pre>
---	--

Figure 71. Pseudo assembly listing

- Find the offset of func_ptr in the Writable Static Map, shown in Figure 72 on page 216.

```

:
-----
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 72. Writable static map

4. Add the offset of func_ptr (X'38') to the address of WSA (X'20914FC0'). The result (X'20914FF8') 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 73 shows the sections of the dump.

```

Enclave Control Blocks:
:
DLL Information:
WSA Addr  Module Addr  Thread ID      Use Count  Name
20914FC0  00000001  main
WSA address.....20914FC0

Enclave Storage:
:
WSA for Program Object(s)
WSA: 20914FC0
+000000 20914FC0 C36DE6E2 C1404040 40404040 40404040 9985A2A4 93A34096 864086A4 95838140 |C_WSA      result of funca
+000020 20914FE0 7E406C84 15000000 20914FF8 00000000 00000060 20F23280 00000000 00000000 |=%d.....j|8.....-2.....
:

```

Figure 73. 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.

Alternatively, you can specify the `DYNDUMP` run-time option to generate a system dump. For more information, see *z/OS Language Environment Programming Reference*.

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 82.

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 . . . : DPXDU1
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 74. 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. See “Formatting and analyzing system dumps” on page 83 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 14.

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* Get requests: *gggg*
 - *p* — the number of the pool. When there are multiple pools for a cell size, the pools are numbered using the format *aa.bbb*.
 - *aa* — the number for the cell size.
 - *bbb* — the number for the pool within the cell size.
 - *ssss* — the cell size specified for the pool.
 - *gggg* — the number of storage requests that were satisfied from this 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.

- Specified Cell Size — the size of the cell specified in the HEAPPOOLS run-time option

- Element Size — the size of the cell plus any additional storage needed for control information or to maintain alignment
- 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:

$$\text{Initial Heap Size} * (\text{Extent Percent}/100) / (\text{Element Size})$$

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:

$$(\text{Maximum Cells Used} * (\text{Element Size}) * 100) / \text{Initial Heap Size}$$

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.

User-created 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 range

- *yyyy* — the high side of the range
- *n* — the number of requests in the 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 on the `__ucreate()` call
- Extent Percent — the cell pool percent specified on the `__ucreate()` call
- 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.

- Extents Allocated — the number of times that each pool allocated an extent.
- 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 `__umalloc/__ufree` 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*.

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 75 on page 221.

```
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 75. 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` or `_CEE_ENVFILE_S`. 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 generate 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 76. 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 76. 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 CEEPGETFN
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 CEEPGETFN
Called from: 25725c08 +000000a0 dlinit
Called from: 05d49c88 +000007dc (unknown)
Unmatched ALLOCATE of 2074 bytes at address 0x25a2e1f8
- sequence 2
Called from: 25a44330 +000000d2 MemAlloc
Called from: 05cd9918 +0000005c CEEPGETFN
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 CEEPGETFN
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 CEEPGETFN
Called from: 25601ae8 +00000084 function3
Called from: 25601bb8 +0000008c function2
Called from: 25601c68 +000000ca function1
Called from: 25601a60 +00000062 main

```

Figure 77. 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*, SC23-8528 or *Enterprise COBOL for z/OS Language Reference*, SC27-1408.

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*, SC23-8529
- *Enterprise COBOL for z/OS Programming Guide*, SC27-1412
- *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 the following documents:

- *Enterprise COBOL for z/OS Language Reference*, SC23-8528
- *Enterprise COBOL for z/OS Language Reference*, SC27-1408
- *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 the following documents:

- *Enterprise COBOL for z/OS Language Reference*, SC23-8528
- *Enterprise COBOL for z/OS Language Reference*, SC27-1408
- *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*, SC23-8529
- *Enterprise COBOL for z/OS Programming Guide*, SC27-1412
- *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*, SC23-8529
- *Enterprise COBOL for z/OS Programming Guide*, SC27-1412
- *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 the following documents:

- *Enterprise COBOL for z/OS Language Reference, SC23-8528*
- *Enterprise COBOL for z/OS Language Reference, SC27-1408*
- *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 78 on page 232 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 78. Example of using the WITH DEBUGGING MODE clause

In the example in Figure 78, 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 the following documents:

- *Enterprise COBOL for z/OS Programming Guide*, SC23-8529
- *Enterprise COBOL for z/OS Programming Guide*, SC27-1412
- *COBOL for OS/390 & VM Programming Guide*

Table 16. 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 79 on page 234 and Figure 80 on page 235 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 79. COBOL program COBDUMP1 calling COBDUMP2

COBOL program that calls the Language Environment CEE3DMP callable service

In the example in Figure 80 on page 235, 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 80. 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 81 on page 236. 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 237.

CEE3845I CEEDUMP Processing started.
 CEE3DMP called by program unit COBDUMP2 at statement 40 (offset +00000496).

Registers on Entry to CEE3DMP:

```
PM..... 0000
GPR0..... 11480BDC GPR1..... 114842C0 GPR2..... 114A4340 GPR3..... 11202BBC
GPR4..... 11202818 GPR5..... 11480100 GPR6..... 00000000 GPR7..... 00FDD100
GPR8..... 114A41D8 GPR9..... 11480AA0 GPR10.... 11202908 GPR11.... 11202AD4
GPR12.... 112129C0 GPR13.... 114841D0 GPR14.... 91202C78 GPR15.... 912EF898
FPR0..... 00000000 00000000 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
GPREG STORAGE:
Storage around GPR0 (11480BDC)
-0020 11480BBC 00000000 00000000 00000000 114A4158 114A41D8 00000000 114A4120 0001E038 .....Q.....
+0000 11480BDC 00000000 00000000 11202D2C 07FE07FE 00000000 00000000 00001FFF 07FE0000 .....
+0020 11480BFC 00000000 00000000 00000000 11480C50 40000000 00000000 00000000 11480D68 .....& .....
```

Information for enclave COBDUMP1

Information for thread 8000000000000000

Registers on Entry to CEE3DMP:

```
PM..... 0000
GPR0..... 11480BDC GPR1..... 114842C0 GPR2..... 114A4340 GPR3..... 11202BBC
GPR4..... 11202818 GPR5..... 11480100 GPR6..... 00000000 GPR7..... 00FDD100
GPR8..... 114A41D8 GPR9..... 11480AA0 GPR10.... 11202908 GPR11.... 11202AD4
GPR12.... 112129C0 GPR13.... 114841D0 GPR14.... 91202C78 GPR15.... 912EF898
FPR0..... 00000000 00000000 FPR2..... 00000000 00000000
FPR4..... 00000000 00000000 FPR6..... 00000000 00000000
GPREG STORAGE:
Storage around GPR0 (11480BDC)
-0020 11480BBC 00000000 00000000 00000000 114A4158 114A41D8 00000000 114A4120 0001E038 .....Q.....
+0000 11480BDC 00000000 00000000 11202D2C 07FE07FE 00000000 00000000 00001FFF 07FE0000 .....
+0020 11480BFC 00000000 00000000 00000000 11480C50 40000000 00000000 00000000 11480D68 .....& .....
```

Traceback:

DSA	Entry	E	Offset	Statement	Load	Mod	Program Unit	Service	Status
1	COBDUMP2		+00000496	40	GO		COBDUMP2		Call
2	COBDUMP1		+000003A4	23	GO		COBDUMP1		Call

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	114841D0		112027E0	112027E0	+00000496	20070214	COBOL
2	11484030		11200398	11200398	+000003A4	20070214	COBOL

Parameters, Registers, and Variables for Active Routines:

```
COBDUMP2 (DSA address 114841D0):
UPSTACK DSA
Saved Registers:
GPR0..... 11480BDC GPR1..... 114842C0 GPR2..... 114A4340 GPR3..... 11202BBC
GPR4..... 11202818 GPR5..... 11480100 GPR6..... 00000000 GPR7..... 00FDD100
GPR8..... 114A41D8 GPR9..... 11480AA0 GPR10.... 11202908 GPR11.... 11202AD4
GPR12.... 112129C0 GPR13.... 114841D0 GPR14.... 91202C78 GPR15.... 912EF898
GPREG STORAGE:
Storage around GPR0 (11480BDC)
-0020 11480BBC 00000000 00000000 00000000 114A4158 114A41D8 00000000 114A4120 0001E038 .....Q.....
+0000 11480BDC 00000000 00000000 11202D2C 07FE07FE 00000000 00000000 00001FFF 07FE0000 .....
+0020 11480BFC 00000000 00000000 00000000 11480C50 40000000 00000000 00000000 11480D68 .....& .....
```

Local Variables:

```
13 IOFSS1 FILE SPECIFIED AS: OPTIONAL, ORGANIZATION=VSAM SEQUENTIAL,
ACCESS MODE=SEQUENTIAL, RECFM=FIXED. CURRENT STATUS OF
FILE IS: NOT OPEN, FILE STATUS CODE=00, VSAM FEEDBACK=000,
VSAM RET CODE=000, VSAM FUNCTION CODE=000.

14 01 IOFSS1R X(40) DISP
17 01 TEMP4 AN-GR
18 02 A-1 AN-GR OCCURS 2
19 03 A-2 AN-GR OCCURS 2
```

Figure 81. Sections of the Language Environment dump called from COBDUMP2 (Part 1 of 2)


```

20 04 A-3V          XXX
    SUB(1,1)        DISP          '  '
    SUB(1,2) to SUB(2,2) elements same as above.
21 04 A-6          XXX
    SUB(1,1)        DISP          'XXX'
    SUB(1,2)        DISP          'YYY'
    SUB(2,1)        DISP          'ZZZ'
    SUB(2,2)        DISP          '  '
22 77 DMPTITLE     X(80) DISP     'COBOL DUMP
23 77 OPTIONS      X(255) DISP    'BLOCKS STORAGE PAGE(55) FILES

24 77 FC          X(12) DISP     '  '
27 01 SALARY-RECORD AN-GR
28 02 NAME        X(10) DISP     '  '
29 02 DEPT        9999 DISP
30 02 SALARY      9(6) DISP

COBDUMP1 (DSA address 11484030):
UPSTACK DSA
Saved Registers:
GPR0..... 1148057C GPR1..... 11484120 GPR2..... 114A4120 GPR3..... 112006C4
GPR4..... 112003D0 GPR5..... 11211778 GPR6..... 00000000 GPR7..... 00000000
GPR8..... 114A40D0 GPR9..... 11480448 GPR10..... 112004C0 GPR11..... 112005E8
GPR12.... 112129C0 GPR13.... 11484030 GPR14.... 9120073E GPR15.... 112027E0
GPREG STORAGE:
Storage around GPR0 (1148057C)
-0020 1148055C 114A40D0 00000000 00000000 00000000 00000000 114A4050 114A40D0 00000000 |..... &.. .....|
+0000 1148057C 11480A48 00000000 11200824 07FE07FE 00000000 00000000 00001FFF 07FE0000 |.....|
+0020 1148059C 00000000 00000000 00000000 40000000 00000000 00000000 00000000 00000001 |.....|
:
Local Variables:
10 01 SOME-WORKINGSTG AN-GR
11 02 SUB-LEVEL      X(80) DISP     'THIS IS IN WORKING STORAGE
13 01 SALARY-RECORDA AN-GR
14 02 NAMEA         X(10) DISP     '  '
15 02 DEPTA        9999 DISP
16 02 SALARYA      9(6) DISP
:

```

Figure 81. 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 82 on page 238, 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 compile Options
- 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: 114841D0
+000000  FLAGS.... 0010      member... 4001      BKC..... 11484030  FWC..... 11484370  R14..... 91202C78
+000010  R15..... 912EF898  R0..... 11480BDC  R1..... 114842C0  R2..... 114A4340  R3..... 11202BBC
+000024  R4..... 11202818  R5..... 11480100  R6..... 00000000  R7..... 00FDD100  R8..... 114A41D8
+000038  R9..... 11480AA0  R10..... 11202908  R11..... 11202AD4  R12..... 112129C0  reserved. 00000000
+00004C  NAB..... 11484370  PNAB..... 00000000  reserved. 00000000  11480AA0
+000064  reserved. 11200398  reserved. 114803F0  MODE..... 00016108  reserved. 00000000
+000078  reserved. 11205F98  reserved. 00000000

Program COBDUMP2 was compiled 02/14/07 2:59:20 PM
COBOL Version = 03 Release = 04 Modification = 01      User Level = ' '

COBOL compile options

Compile Options for COBDUMP2:
ADV, ARITH(COMPAT), NOAWO, CODEPAGE(01140), DATA(31), NODATEPROC, DBCS,
NODLL, NODYNAM, NOEXPORTALL, NOFASTSRT, INTDATE(ANSI), NUMPROC(NOPFD),
NOOPTIMIZE, OUTDD(SYSOUT), PGMNAME(COMPAT), RENT, RMODE(ANY), NOSSRANGE,
TEST(STMT,SYM,NOSEPARATE), NOTHREAD, TRUNC(STD), YEARWINDOW(1900), ZWB
TGT for COBDUMP2: 11480AA0
+000000 11480AA0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 11480AC0 - +00003F 11480ADF same as above |.....|
+000040 11480AE0 00000000 00000000 F3E3C7E3 00000000 06000000 42430260 11480100 000167FC |.....3TGT.....|
+000060 11480B00 11480C20 00000001 00000174 00000000 00000000 114A4148 00000000 00000000 |.....|
+000080 11480B20 112129C0 00000180 00000000 00000000 00000000 00000001 E2E8E2D6 E4E34040 |.....SYSOUT|
+0000A0 11480B40 C9C7E9E2 D9E3C3C4 00000000 00000000 00000000 00000000 00000000 00000000 |IGZSRCD.....|
+0000C0 11480B60 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000E0 11480B80 00000000 00000000 112028DC 00000000 11480C0C 11480A48 11202988 11480BE0 |.....|
+000100 11480BA0 112027E0 11202910 11480C08 11202904 11480C08 114A41D8 00000000 00000000 |.....Q.....|
+000120 11480BC0 00000000 00000000 114A4158 114A41D8 00000000 114A4120 0001E038 00000000 |.....Q.....|
+000140 11480BE0 00000000 11202D2C 07FE07FE 00000000 00000000 00001FFF 07FE0000 00000000 |.....|
+000160 11480C00 00000000 00000000 11480C50 40000000 00000000 00000000 11480D68 00000001 |.....&.....|

CLLE for COBDUMP2: 11480A48
+000000 11480A48 C3D6C2C4 E4D4D7F2 00004100 00000000 84810000 112027E0 11480AA0 00000000 |COBDUMP2.....da.....|
+000020 11480A68 00000000 114808E4 114809F0 114803F0 00000001 00000000 00000000 00000000 |.....U...0...0.....|
:
:

```

Figure 82. Control block information for active COBOL routines

Storage for each active routine

The Storage for Active Routines section of the dump, shown in Figure 83 on page 239, 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:
    0-0001E038

  Contents of base locators for WORKING-STORAGE are:
    0-114A41D8

  Contents of base locators for the LINKAGE SECTION are:
    0-00000000    1-114A4120

No indexes were used in this program.

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 FACTORY data was used in this program.

No XML data was used in this program.

File record contents for COBDUMP2
ESDS1DD (BLF-0): 0001E038
+000000 0001E038 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 0001E058 - +00003F 0001E077 same as above

WORKING-STORAGE for COBDUMP2
BLW-0: 114A41D8
+000000 114A41D8 000000E7 E7E70000 00E8E8E8 000000E9 E9E90000 00000000 C3D6C2D6 D340C4E4 |...XXX...YYY...ZZZ.....COBOL DU
+000020 114A41F8 D4D74040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |MP
+000040 114A4218 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+000060 114A4238 40404040 40404040 40C2D3D6 C3D2E240 E2E3D6D9 C1C7C540 D7C1C7C5 4DF5F55D |BLOCKS STORAGE PAGE(55)
+000080 114A4258 40C6C9D3 C5E24040 40404040 40404040 40404040 40404040 40404040 40404040 |FILES
+0000A0 114A4278 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+0000C0 114A4298 - +00015F 114A4337 same as above
+000160 114A4338 40404040 40404000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

LINKAGE SECTION for COBDUMP2
BLL-1: 114A4120
+000000 114A4120 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 114A4140 114A4018 00000218 00000210 00000000 00000000 11480C40 00000000 00000000 |.....|
+000040 114A4160 00000000 00000000 00000000 00000000 C9C7E9E2 D9E3C3C4 00000000 00000000 |.....IGZSRCTD.....|
+000060 114A4180 00000000 00000000 00000000 00000000 E2E8E2D6 E4E34040 00000000 00000000 |.....SYSOUT.....|
+000080 114A41A0 0E000000 00000000 0F000000 00000000 00000000 00000000 40404040 40404040 |.....|
+0000A0 114A41C0 40404040 40404040 40404040 40404040 40404040 40400000 000000E7 E7E70000 |.....XXX..|
+0000C0 114A41E0 00E8E8E8 000000E9 E9E90000 00000000 C3D6C2D6 D340C4E4 D4D74040 40404040 |.YYY...ZZZ.....COBOL DUMP|
+0000E0 114A4200 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+000100 114A4220 - +00011F 114A423F same as above
+000120 114A4240 40C2D3D6 C3D2E240 E2E3D6D9 C1C7C540 D7C1C7C5 4DF5F55D 40C6C9D3 C5E24040 |BLOCKS STORAGE PAGE(55) FILES|
+000140 114A4260 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+000160 114A4280 - +0001FF 114A431F same as above
+000200 114A4320 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404000 |.....|
+000220 114A4340 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000240 114A4360 - +000FFF 114A511F same as above

Program class storage: 114A4148
+000000 114A4148 00000210 00000000 00000000 11480C40 00000000 00000000 00000000 00000000 |.....|
+000020 114A4168 00000000 00000000 C9C7E9E2 D9E3C3C4 00000000 00000000 00000000 00000000 |.....IGZSRCTD.....|
+000040 114A4188 00000000 00000000 E2E8E2D6 E4E34040 00000000 00000000 0E000000 00000000 |.....SYSOUT.....|
+000060 114A41A8 0F000000 00000000 00000000 00000000 40404040 40404040 40404040 40404040 |.....|
+000080 114A41C8 40404040 40404040 40404040 40400000 000000E7 E7E70000 00E8E8E8 000000E9 |.....XXX...YYY...Z|
+0000A0 114A41E8 E9E90000 00000000 C3D6C2D6 D340C4E4 D4D74040 40404040 40404040 40404040 |ZZ.....COBOL DUMP|
+0000C0 114A4208 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+0000E0 114A4228 40404040 40404040 40404040 40404040 40404040 40404040 40C2D3D6 C3D2E240 |BLOCKS|
+000100 114A4248 E2E3D6D9 C1C7C540 D7C1C7C5 4DF5F55D 40C6C9D3 C5E24040 40404040 40404040 |STORAGE PAGE(55) FILES|
+000120 114A4268 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |
+000140 114A4288 - +0001DF 114A4327 same as above
+0001E0 114A4328 40404040 40404040 40404040 40404040 40404000 00000000 00000000 |.....|

```

Figure 83. Storage for active COBOL programs (Part 1 of 2)

```

+000200 114A4348 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
Program class storage: 11480C40
+000000 11480C40 000001D2 00000000 114A4148 0001E028 C6C3C200 01020000 FFFFFFFF FFFFFFFF |...K.....FCB.....|
+000020 11480C60 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 00000000 00000000 |.....|
+000040 11480C80 00000000 00000000 00000000 800082C8 800082C8 800082C8 914754F8 800082C8 |......bH..bHj..8..bH|
+000060 11480CA0 800082C8 800082C8 914754F8 00000000 00000000 00000000 00000000 00000000 |..bH..bHj..8.....|
+000080 11480CC0 00000000 00000000 00000000 00000000 00000000 00000000 C3D6C2C4 E4D4D7F2 |.....COBDUMP2|
+0000A0 11480CE0 C5E2C4E2 F1C4C440 00000000 00000000 00000000 11202A34 00000000 00000000 |ESDS1DD .....|
+0000C0 11480D00 00000000 00000000 00000000 00000000 00000000 00000000 00008800 00000000 |.....h.....|
+0000E0 11480D20 00000000 00000000 00000028 00000000 00000000 00000000 00000000 00000000 |.....|
+000100 11480D40 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000120 11480D60 - +0001BF 11480DFF same as above |.....|
+0001C0 11480E00 00000000 00000000 00000000 00000000 00000000 00000000 11480000 00000E20 |.....|
Program class storage: 0001E028
+000000 0001E028 0000003F 00000000 11480C40 00000000 00000000 00000000 00000000 00000000 |.....|
+000020 0001E048 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
:

```

Figure 83. Storage for active COBOL programs (Part 2 of 2)

Enclave-level data

The Enclave Control Blocks section of the dump, shown in Figure 84 on page 241, 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: 11480100										
+000000	11480100	C3F3D9E4	D5C3D6D4	000002D8	04C60000	11211658	00000002	00006F50	00000000	C3RUNCOM...Q.F.....?&....
+000020	11480120	00000000	11200398	114803F0	00000000	11211778	00000000	00000000	00000000q...0.....
+000040	11480140	00016A80	000161BC	00000000	000167FC	00000000	00000000	112129C0	00000000/.....
+000060	11480160	00000000	00000000	00000000	00000000	00000000	F0F0F0F0	F0F0F0F0	114809F000000000...0
Enclave Storage:										
Initial (User) Heap : 114A4018										
+000000	114A4018	C8C1D5C3	0001E000	11211B38	00000000	914A4018	114A4358	00008000	00007CC0	HANC.....j.....@.
+000020	114A4038	114A4018	00000108	00000100	00000000	00000000	00000000	00000000	00000000IGZSRITCD.....
+000040	114A4058	00000000	00000000	00000000	00000000	C9C7E9E2	D9E3C3C4	00000000	00000000SYSOUT.....
+000060	114A4078	00000000	00000000	00000000	00000000	E2E8E2D6	E4E34040	00000000	00000000THIS IS
+000080	114A4098	0E000000	00000000	0F000000	00000000	00000000	00000000	40404040	40404040	IN WORKING STORAGE
+0000A0	114A40B8	40404040	40404040	40404040	40404040	40404040	40400000	E3C8C9E2	40C9E240
+0000C0	114A40D8	C9D540E6	D6D9D2C9	D5C740E2	E3D6D9C1	C7C54040	40404040	40404040	40404040
+0000E0	114A40F8	40404040	40404040	40404040	40404040	40404040	40404040	40404040	40404040
+000100	114A4118	40404040	40404040	00000000	00000000	00000000	00000000	00000000	00000000
+000120	114A4138	00000000	00000000	114A4018	00000218	00000210	00000000	00000000	11480C40
+000140	114A4158	00000000	00000000	00000000	00000000	00000000	00000000	C9C7E9E2	D9E3C3C4IGZSRITCD
+000160	114A4178	00000000	00000000	00000000	00000000	00000000	00000000	E2E8E2D6	E4E34040SYSOUT
+000180	114A4198	00000000	00000000	0E000000	00000000	0F000000	00000000	00000000	00000000
+0001A0	114A41B8	40404040	40404040	40404040	40404040	40404040	40404040	40404040	40400000
+0001C0	114A41D8	000000E7	E7E70000	00E8E8E8	000000E9	E9E90000	00000000	C3D6C2D6	D340C4E4	...XXX...YYY...ZZZ.....COBOL DU
+0001E0	114A41F8	D4D74040	40404040	40404040	40404040	40404040	40404040	40404040	40404040	MP
+000200	114A4218	40404040	40404040	40404040	40404040	40404040	40404040	40404040	40404040
+000220	114A4238	40404040	40404040	40C2D3D6	C3D2E240	E2E3D6D9	C1C7C540	D7C1C7C5	4DF5F55DBLOCKS STORAGE PAGE(55)
+000240	114A4258	40C6C9D3	C5E24040	40404040	40404040	40404040	40404040	40404040	40404040	FILES
+000260	114A4278	40404040	40404040	40404040	40404040	40404040	40404040	40404040	40404040
+000280	114A4298	- +00031F	114A4337		same as above				
+000320	114A4338	40404040	40404000	00000000	00000000	00000000	00000000	00000000	00000000
+000340	114A4358	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
+000360	114A4378	- +007FFF	114AC017		same as above				
LE/370 Anywhere Heap : 11480000										
+0003E0	114803E0	00000050	00000000	00000000	00000000	C3D6C2C4	E4D4D7F1	00004100	00000000	...&.....COBDUMP1.....
+000400	11480400	94810000	91200398	11480448	00000000	00000000	114808E0	114809F0	00000000	ma..j..q.....0.....
+000420	11480420	00000001	00000000	00000000	00000000	11480000	000001A0	00000194	00000000m.....
+000440	11480440	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
+000460	11480460	- +00047F	1148047F		same as above				
+000480	11480480	00000000	00000000	00000000	00000000	F3E3C7E3	00000000	06000000	604302603TGT.....-
+0004A0	114804A0	11480100	000167FC	114805C0	00000000	00000064	00000000	00000000	00000000
+000A40	11480A40	00000000	00000000	C3D6C2C4	E4D4D7F2	00004100	00000000	84810000	112027E0COBDUMP2.....da.....
+000A60	11480A60	11480AA0	00000000	00000000	114808E4	114809F0	114803F0	00000001	00000000U...0...0.....
+000A80	11480A80	00000000	00000000	11480000	000001B0	000001A8	00000000	00000000	00000000y.....
+000AA0	11480AA0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
+000AC0	11480AC0	- +000ADF	11480ADF		same as above				
+000AE0	11480AE0	00000000	00000000	F3E3C7E3	00000000	06000000	42430260	11480100	000167FC3TGT.....-
+000B00	11480B00	11480C20	00000001	00000174	00000000	00000000	00000000	00000000	00000000
File Control Blocks:										
FCB for file ESDS1DD in program COBDUMP2: 11480C50										
+000000	11480C50	C6C3C200	01020000	FFFFFFF	FFFFFFF	FFFFFFF	FFFFFFF	FFFFFFF	FFFFFFF	FCB.....
+000020	11480C70	FFFFFFF	00000000	00000000	00000000	00000000	00000000	00000000	800082C8bH
+000040	11480C90	800082C8	800082C8	914754F8	800082C8	800082C8	800082C8	914754F8	00000000	..bH..bHj..8.....
+000060	11480CB0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000COBDUMP2ESDS1DD.....
+000080	11480CD0	00000000	00000000	C3D6C2C4	E4D4D7F2	C5E2C4E2	F1C4C440	00000000	00000000
+0000A0	11480CF0	00000000	11202A34	00000000	00000000	00000000	00000000	00000000	00000000
+0000C0	11480D10	00000000	00000000	00008800	00000000	00000000	00000000	00000028	00000000h.....
+0000E0	11480D30	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
+000100	11480D50	- +00011F	11480D6F		same as above				
FIB for file ESDS1DD in program COBDUMP2: 11202A34										
+000000	11202A34	C6C9C200	0103C5E2	C4E2F1C4	C4400088	8080A000	00008000	00000000	00000028	FIB...ESDS1DD .h.....
+000020	11202A54	00010000	00000000	00000000	00000000	00000000	00000000	11202A2D	00000000
+000040	11202A74	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
+000060	11202A94	- +00007F	11202AB3		same as above				
+000080	11202AB4	0000C9D6	C6E2E2F1	40404040	40404040	40404040	40404040	40404040	40404040	..IOFSS1
GMAREA for file ESDS1DD in program COBDUMP2: 00000000										
+000000	00000000	Inaccessible storage.								

Figure 84. Enclave-level data for COBOL programs

Process-level data

The Process Control Block section of the dump, shown in Figure 85, 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: 00016A80
+000000 00016A80 C3F3E3C8 C4C3D6D4 000001E8 81000000 00000100 00000000 00016108 000161BC |C3THDCOM...Ya...../.../|
+000020 00016AA0 11480100 00000000 C3D6C2C4 E4D4D7F1 00000000 00000000 00000000 00000000 |.....COBDUMP1.....|
+000040 00016AC0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 00016AE0 - +00007F 00016AFF same as above
:
:
COBCOM: 00016108
+000000 00016108 C3F3C3D6 C2C3D6D4 00000978 00000000 00000000 00000000 00000000 00000000 |C3COBCOM.....|
+000020 00016128 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 00016148 00000000 00000000 00000000 00000000 00000000 1140AE68 906000D6 000161BC |.....-0.../|
+000060 00016168 000167FC 00000100 00820000 00000000 00000800 08000000 00016A80 00000000 |.....b.....|
:
:
COBVEC: 000161BC
+000000 000161BC 0001643C 00016442 00016448 0001644E 00016454 0001645A 00016460 00016466 |.....+.....!...-.....|
+000020 000161DC 0001646C 00016472 00016478 0001647E 00016484 0001648A 00016490 00016496 |...%.....=.d.....o|
+000040 000161FC 0001649C 000164A2 000164A8 000164AE 000164B4 000164BA 000164C0 000164C6 |.....S...y.....F|
+000060 0001621C 000164CC 000164D2 000164D8 000164DE 000164E4 000164EA 000164F0 000164F6 |.....K...Q.....U.....0...6|
:
:
```

Figure 85. 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 86 on page 243 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 87 on page 244.

```
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 86. 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 87 on page 244. 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.

CEE3845I CEEDUMP Processing started.

Information for enclave COBOLX

Information for thread 8000000000000000

Traceback:

DSA	Entry	E	Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHDSP		+000049D6		CEEPLPKA	CEEHDSP	D1908	Call
2	CEEHSGLT		+0000005C		CEEPLPKA	CEEHSGLT	D1908	Exception
3	IGZCMSG		+000003C2		IGZCPAC	IGZCMSG		Call
4	COBOLX		+000002BC	11	GO	COBOLX		Call

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	1147E880		112BD238	112BD238	+000049D6	20061215	CEL
2	1147E6E8		112CF960	112CF960	+0000005C	20061215	CEL
3	1147E1D0		11455AA0	11455AA0	+000003C2	20061213	LIBRARY
4	1147E030		00008398	00008398	+000002BC	20070214	COBOL

Condition Information for Active Routines

Condition Information for CEEHSGLT (DSA address 1147E6E8)

CIB Address: 1147F1A0

Current Condition:

IGZ0006S The reference to table SLO7 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: 112CF9AC

+000000 112CF9AC F010D20B D0801000 58A0C2B8 58F0A01C 05EFD20B D098B108 41A0D098 50A0D08C |0.K.....B..0....K..q.....q&...|

Parameters, Registers, and Variables for Active Routines:

CEEHDSP (DSA address 1147E880):

UPSTACK DSA

Saved Registers:

GPR0..... 00008550 GPR1..... 1147ECD8 GPR2..... 1147F1A0 GPR3..... 1120BD60
GPR4..... 112C1E64 GPR5..... 9120E770 GPR6..... 00000000 GPR7..... 112092A8
GPR8..... 912C1988 GPR9..... 1148087E GPR10.... 1147F87F GPR11.... 912BD238
GPR12.... 1120C9C0 GPR13.... 1147E880 GPR14.... 912C1C10 GPR15.... 912E9898

GPREG STORAGE:

Storage around GPR0 (00008550)

-0020 00008530 08000004 00224000 00000006 C0000140 00040800 00040028 02400002 08000004 |.....|
+0000 00008550 001F03C0 00060800 0004001C 40000000 0040C000 01400006 08000004 002202C0 |.....|
+0020 00008570 00060800 00040022 183F4100 11A05500 C00C05F0 47D0F00C 58F0C300 05EF181F |.....0..0..0C.....|

IGZCMSG (DSA address 1147E1D0):

UPSTACK DSA

Saved Registers:

GPR0..... 00008550 GPR1..... 1147E3A8 GPR2..... 1147E3A8 GPR3..... 00000000
GPR4..... 9120E770 GPR5..... 9120E770 GPR6..... 1147E3EA GPR7..... 00000005
GPR8..... 00019A80 GPR9..... 0000A1A8 GPR10.... 1147A100 GPR11.... 91455AA0
GPR12.... 1120C9C0 GPR13.... 1147E1D0 GPR14.... 91455E64 GPR15.... 912CF960

GPREG STORAGE:

Storage around GPR0 (00008550)

-0020 00008530 08000004 00224000 00000006 C0000140 00040800 00040028 02400002 08000004 |.....|
+0000 00008550 001F03C0 00060800 0004001C 40000000 0040C000 01400006 08000004 002202C0 |.....|
+0020 00008570 00060800 00040022 183F4100 11A05500 C00C05F0 47D0F00C 58F0C300 05EF181F |.....0..0..0C.....|

COBOLX (DSA address 1147E030):

UPSTACK DSA

Saved Registers:

GPR0..... 1147E1D0 GPR1..... 00008536 GPR2..... 00000010 GPR3..... 000197FC
GPR4..... 000083D0 GPR5..... 1120B778 GPR6..... 00000000 GPR7..... 00000000
GPR8..... 0000A398 GPR9..... 0000A1A8 GPR10.... 000084A0 GPR11.... 000085E4
GPR12.... 00008494 GPR13.... 1147E030 GPR14.... 80008656 GPR15.... 91455AA0

GPREG STORAGE:

Storage around GPR0 (1147E1D0)

-0020 1147E1B0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 1147E1D0 00101001 1147E030 1147E6E8 91455E64 912CF960 00008550 1147E3A8 1147E3A8 |.....WYj.;.j.9-.e&..Ty..Ty|
+0020 1147E1F0 00000000 9120E770 9120E770 1147E3EA 00000005 00019A80 0000A1A8 1147A100 |...j.X.j.X...T.....y.....|

Figure 87. Sections of Language Environment dump for COBOLX (Part 1 of 2)

```

Local Variables:
  6 77 J          9999 COMP          00009
  7 01 TABLE-X  AN-GR
  8 02 SLOT      9999 OCCURS 8
                SUB(1)          COMP          00000
                SUB(2) to SUB(8) elements same as above.
:

```

Figure 87. 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 88. This statement moves the 1 value to the array SLOT (J).

```

PP 5655-G53 IBM Enterprise COBOL for z/OS 3.4.1          COBOLX   Date 02/14/2007  Time 14:58:58  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 88. 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 87 on page 244. 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 89 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 89 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 89. COBOL example of calling a nonexistent subroutine

To understand the traceback information and debug this program, use the following steps:

1. Locate the error message for the original condition under the Condition Information for Active Routines section of the dump, shown in Figure 90. The message is CEE3501S The module UNKNOWN was not found. For more information about this message, see *z/OS Language Environment Run-Time Messages*.
2. 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 +0000036E.

```

CEE3DMP V1 R9.0: Condition processing resulted in the unhandled condition.      02/14/07 2:59:09 PM      Page: 1
ASID: 0099 Job ID: J0005737 Job name: COBOLY Step name: GO      UserID: BARBARA

CEE3845I CEEDUMP Processing started.

Information for enclave COBOLY

Information for thread 8000000000000000

Traceback:
 DSA  Entry      E  Offset  Statement  Load Mod  Program Unit  Service  Status
  1  CEEHDSP    +00004030  CEEPLPKA  CEEHDSP    D1908  Call
  2  CEEHSGLT   +0000005C  CEEPLPKA  CEEHSGLT   D1908  Exception
  3  IGZCLDL   +0000012A  IGZCPAC   IGZCLDL    Call
  4  IGZCFCC   +000003EE  IGZCPAC   IGZCFCC    Call
  5  COBOLY    +0000036E  8         GO         COBOLY     Call

 DSA  DSA Addr  E  Addr  PU Addr  PU Offset  Comp Date  Compile Attributes
  1  1147E6F0  112BD238 112BD238 +00004030 20061215  CEL
  2  1147E558  112CF960 112CF960 +0000005C 20061215  CEL
  3  1147E3B0  11453680 11453680 +0000012A 20061213  LIBRARY
  4  1147E1D0  1140A5F8 1140A5F8 +000003EE 20061213  LIBRARY
  5  1147E030  000083F0 000083F0 +0000036E 20070214  COBOL

Condition Information for Active Routines
Condition Information for CEEHSGLT (DSA address 1147E558)
CIB Address: 1147F010
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: 112CF9AC
+000000 112CF9AC F010D20B D0801000 58A0C2B8 58F0A01C 05EFD20B D098B108 41A0D098 50A0D08C |0.K.....B..0....K..q....q&...|
:

```

Figure 90. Sections of Language Environment dump for COBOLY

3. Use the offset of X'36E' from the COBOL listing, shown in Figure 91 on page 247, to locate the statement that caused the exception in the COBOLY program. At offset X'36E' is an instruction for statement 8. Statement 8 is a call with the identifier SUBNAME specified.

```

PP 5655-G53 IBM Enterprise COBOL for z/OS 3.4.1          COBOLY   Date 02/14/2007   Time 14:59:07   Page   11
0002B0          START      EQU      *
0002B0 183F          LR       3,15
0002B2 4100 11A0     LA       0,416(0,1)
0002B6 5500 C00C     CL       0,12(0,12)
0002BA 05F0          BALR     15,0
0002BC 47D0 F00C     BC       13,12(0,15)
0002C0 58F0 C300     L        15,768(0,12)
0002C4 05EF          BALR     14,15
0002C6 181F          LR       1,15
0002C8 50D0 1004     ST       13,4(0,1)
0002CC 5000 104C     ST       0,76(0,1)
0002D0 D203 1000 3058 MVC      0(4,1),88(3)
0002D6 D703 1084 1084 XC       132(4,1),132(1)
0002DC 5090 105C     ST       9,92(0,1)
0002E0 18D1          LR       13,1
0002E2 58C0 90E8     L        12,232(0,9)      TGTFIXD+232
0002E6 1812          LR       1,2
0002E8 50D0 D058     ST       13,88(0,13)
0002EC 58A0 C004     L        10,4(0,12)      CBL=1
0002F0 5880 912C     L        8,300(0,9)      BLW=0
0002F4 D203 D088 A010 MVC      136(4,13),16(10)  DSAFIXD+136      PGMLIT AT +8
0002FA 4120 D100     LA       2,256(0,13)      CALLname+0
0002FE 5020 D08C     ST       2,140(0,13)      DSAFIXD+140
000302 BF2F 916C     ICM      2,15,364(9)      IPCB=1+16
000306 58B0 C008     L        11,8(0,12)      PBL=1
00030A 4780 B000     BC       8,0(0,11)      GN=7(00031C)
00030E 5830 905C     L        3,92(0,9)      TGTFIXD+92
000312 58F0 30F4     L        15,244(0,3)      V(IGZCMMSG )
000316 4110 A188     LA       1,392(0,10)      PGMLIT AT +384
00031A 05EF          BALR     14,15
00031C          GN=7      EQU      *
00031C 5A20 C000     A        2,0(0,12)      SYSLIT AT +0
000320 5020 916C     ST       2,364(0,9)      IPCB=1+16
000324 9140 9057     TM       87(9),X'40'      TGTFIXD+87
000328 4710 B024     BC       1,36(0,11)      GN=8(000340)
00032C 9120 9054     TM       84(9),X'20'      TGTFIXD+84
000330 47E0 B01C     BC       14,28(0,11)      GN=9(000338)
000334 9620 D084     OI      132(13),X'20'      DSAFIXD+132
000338          GN=9      EQU      *
000338 9640 9057     OI      87(9),X'40'      TGTFIXD+87
00033C 47F0 B024     BC       15,36(0,11)      GN=10(000340)
000340          GN=8      EQU      *
000340          GN=10     EQU      *
000340 9640 915C     OI      348(9),X'40'      IPCB=1
000344 9601 D084     OI      132(13),X'01'      DSAFIXD+132
000008 *
000008 CALL
000348          GN=13     EQU      *
000348 D207 D0F0 8000 MVC      240(8,13),0(8)      TS2=0
00034E DC07 D0F0 A01A TR       240(8,13),26(10)  TS2=0
000354 D203 D0F8 A162 MVC      248(4,13),354(10)  TS2=8
00035A 4120 D0F0     LA       2,240(0,13)      TS2=0
00035E 5020 D0FC     ST       2,252(0,13)      TS2=12
000362 4110 D0F8     LA       1,248(0,13)      TS2=8
000366 5820 905C     L        2,92(0,9)      TGTFIXD+92
00036A 58F0 2100     L        15,256(0,2)      V(IGZCFCC )
00036E 05EF          BALR     14,15
000370 5830 9128     L        3,296(0,9)      BL=1
PP 5655-G53 IBM Enterprise COBOL for z/OS 3.4.1          COBOLY   Date 02/14/2007   Time 14:59:07   Page   12
000374 40F0 3008     STH     15,8(0,3)      RETURN-CODE
000009 GOBACK
000378          GN=14     EQU      *
000378 58B0 C008     L        11,8(0,12)      PBL=1
00037C 47F0 B07C     BC       15,124(0,11)      GN=2(000398)
000380 0700          BCR      0,0
000382 9120 D084     TM      132(13),X'20'      DSAFIXD+132
000386 47E0 B07C     BC       14,124(0,11)      GN=2(000398)
00038A 5820 905C     L        2,92(0,9)      TGTFIXD+92
00038E 58F0 20F4     L        15,244(0,2)      V(IGZCMMSG )
000392 4110 A176     LA       1,374(0,10)      PGMLIT AT +366
000396 05EF          BALR     14,15
000398          GN=2      EQU      *
000398 5820 916C     L        2,364(0,9)      IPCB=1+16
00039C 5B20 C000     S        2,0(0,12)      SYSLIT AT +0
0003A0 5020 916C     ST       2,364(0,9)      IPCB=1+16
0003A4 9140 9055     TM       85(9),X'40'      TGTFIXD+85

```

Figure 91. COBOL Listing for COBOLY (Part 1 of 2)

```

0003A8 47E0 B09E          BC  14,158(0,11)      GN=11(0003BA)
0003AC 4110 0008          LA  1,8(0,0)
0003B0 5820 905C          L   2,92(0,9)        TGTFIXD+92
0003B4 58F0 2020          L   15,32(0,2)       V(IGZCCTL )
0003B8 05EF          BALR 14,15
0003BA          GN=11      EQU  *
0003BA 9128 9054          TM  84(9),X'28'      TGTFIXD+84
0003BE 4770 B0BC          BC  7,188(0,11)     GN=12(0003D8)
0003C2 5820 9128          L   2,296(0,9)      BL=1
0003C6 48F0 2008          LH  15,8(0,2)       RETURN-CODE
0003CA 58D0 D004          L   13,4(0,13)
0003CE 58E0 D00C          L   14,12(0,13)
0003D2 980C D014          LM  0,12,20(13)
0003D6 07FE          BCR  15,14
0003D8          GN=12      EQU  *
0003D8 D20B D0F0 A156      MVC 240(12,13),342(10) TS2=0          PGMLIT AT +334
0003DE 5830 9128          L   3,296(0,9)      BL=1
0003E2 4820 3008          LH  2,8(0,3)        RETURN-CODE
0003E6 5020 D0FC          ST  2,252(0,13)     TS2=12
0003EA 4110 D0F0          LA  1,240(0,13)     TS2=0
0003EE 5820 905C          L   2,92(0,9)        TGTFIXD+92
0003F2 58F0 2224          L   15,548(0,2)     V(IGZETRM )
0003F6 05EF          BALR 14,15
.
.
.

```

Figure 91. 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 92. 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 1147E030):
UPSTACK DSA
Saved Registers:
GPR0..... 1147E1D0  GPR1..... 1147E128  GPR2..... 000197FC  GPR3..... 000083F0
GPR4..... 00008428  GPR5..... 1120B778  GPR6..... 00000000  GPR7..... 00000000
GPR8..... 0000A3A8  GPR9..... 0000A1B8  GPR10.... 000084F8  GPR11.... 0000870C
GPR12.... 000084EC  GPR13.... 1147E030  GPR14.... 80008760  GPR15.... 9140A5F8
GPREG STORAGE:
Storage around GPR0 (1147E1D0)
-0020 1147E1B0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 1147E1D0 00102001 1147E030 00000000 9140A9E8 91453680 1147E3B0 1147E368 000197FC |.....j zYj.....T...T...p|
+0020 1147E1F0 000083F0 1147E128 000197FC 00000000 1147E358 00019A80 0000A1B8 1147A100 |..c0.....p.....T.....|
:
: Local Variables:
:      6 77 SUBNAME          X(8) DISP          'UNKNOWN '
:
:

```

Figure 92. 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 93 on page 249 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 93. 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 94 on page 250. 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.

CEE3845I CEEDUMP Processing started.

Information for enclave COBOLZ1

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHDSP	+000049D6		CEEPLPKA	CEEHDSP	D1908	Call
2	ASSEMZ3	+00000064		ASSEMZ3	ASSEMZ3		Exception
3	IGZCFCC	+000002CA		IGZCPAC	IGZCFCC		Call
4	COBOLZ2	+000002A4	11	COBOLZ2	COBOLZ2		Call
5	IGZCFCC	+000002CA		IGZCPAC	IGZCFCC		Call
6	COBOLZ1	+0000028E	8	GO	COBOLZ1		Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	1147E7D0	112BD238	112BD238	+000049D6	20061215	CEL
2	1147E750	1120E448	1120E448	+00000064	20070214	ASM
3	1147E570	1140A5F8	1140A5F8	+000002CA	20061213	LIBRARY
4	1147E3C0	0001F320	0001F320	+000002A4	20070214	COBOL
5	1147E1E0	1140A5F8	1140A5F8	+000002CA	20061213	LIBRARY
6	1147E030	000083D0	000083D0	+0000028E	20070214	COBOL

Condition Information for Active Routines

Condition Information for ASSEMZ3 (DSA address 1147E750)

CIB Address: 1147F0F0

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 9120E4B0

GPR0..... 1147E7D0 GPR1..... 1147E4B8 GPR2..... 1147AE54 GPR3..... 000212E8
GPR4..... 00000000 GPR5..... 0000092C GPR6..... 000213A8 GPR7..... 1147E4B8
GPR8..... 00000002 GPR9..... 000211B0 GPR10.... 1147A100 GPR11.... 9120E448
GPR12.... 1120C9C0 GPR13.... 1147E750 GPR14.... 9140A8C4 GPR15.... 9120E448

Storage dump near condition, beginning at location: 1120E49C

```
+000000 1120E49C 10184150 092C1B44 58610000 41660000 5D460000 58F0B0C8 5800B0C8 58DD0004 |...&...../.....)....0.H...H...|
GPREG STORAGE:
Storage around GPR0 (1147E7D0)
-0020 1147E7B0 1120C9C0 00000000 00000000 00000000 00008408 00000000 00000100 1147E958 |..I.....d.....Z.|
+0000 1147E7D0 0808CEE1 1147E750 114818F0 912C1C10 912E9898 1147E7D0 1147EC28 1147F0F0 |.....X&...0j...j.qq..X.....00|
+0020 1147E7F0 1120BD60 112C1E64 9120E770 00000000 112092A8 912C1988 114807CE 1147F7CF |...-.....j.X.....kyj..h.....7.|
```

Parameters, Registers, and Variables for Active Routines:

COBOLZ2 (DSA address 1147E3C0):

UPSTACK DSA

Saved Registers:

GPR0..... 1147E570 GPR1..... 1147E4C0 GPR2..... 000197FC GPR3..... 000212E8
GPR4..... 1147E4B8 GPR5..... 000191BC GPR6..... 1147AFD0 GPR7..... 00FDD100
GPR8..... 000213A8 GPR9..... 000211B0 GPR10.... 0001F428 GPR11.... 0001F54C
GPR12.... 0001F41C GPR13.... 1147E3C0 GPR14.... 8001F5C6 GPR15.... 9140A5F8

GPREG STORAGE:

Storage around GPR0 (1147E570)

```
-0020 1147E550 1147E30C 1147E44C 1120B658 00000000 1147E470 00000000 00000011 00000005 |..T...U<.....U.....|
+0000 1147E570 00102401 1147E3C0 1147E750 9140A8C4 9120E448 1147E750 1147E4B8 1147AE54 |.....T...X&j yDj.U...X&..U....|
+0020 1147E590 000212E8 1147E4C0 000212E8 1147B028 1147E4B8 00000002 000211B0 1147A100 |...Y..U...Y.....U.....|
```

Local Variables:

6 77 DV-VAL 9999 COMP 00000
8 77 D-VAL 9999 COMP 00000

Figure 94. Sections of Language Environment dump for program COBOLZ1 (Part 1 of 2)

```

COBOLZ1 (DSA address 1147E030):
UPSTACK DSA
Saved Registers:
GPR0..... 1147E1E0  GPR1..... 1147E130  GPR2..... 000197FC  GPR3..... 0000A2E4
GPR4..... 1147E128  GPR5..... 1120B778  GPR6..... 00000000  GPR7..... 00000000
GPR8..... 0000A3A8  GPR9..... 0000A1B0  GPR10.... 000084D8  GPR11.... 000085F4
GPR12.... 000084CC  GPR13.... 1147E030  GPR14.... 80008660  GPR15.... 9140A5F8
GPREG STORAGE:
Storage around GPR0 (1147E1E0)
-0020 1147E1C0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 1147E1E0  00102401 1147E030 1147E3C0 9140A8C4 0001F320 1147E3C0 1147E128 000197FC |.....T.j yD..3...T.....p.
+0020 1147E200  0000A2E4 1147E130 0000A2E4 1147AFD0 1147E128 00000002 0000A1B0 1147A100 |..sU.....sU.....|
:
Local Variables:
      6 77 D-VAL          9999 COMP          00000
:

```

Figure 94. Sections of Language Environment dump for program COBOLZ1 (Part 2 of 2)

- Locate statement 11 in the COBOL listing for the COBOLZ2 program, shown in Figure 95. This is a call to the assembler routine ASSEMZ3.

```

PP 5655-G53 IBM Enterprise COBOL for z/OS 3.4.1          COBOLZ2  Date 02/14/2007  Time 14:59:15  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=00000+000          2C
000007          LINKAGE SECTION.
000008          77 D-VAL PIC 9(4) USAGE COMP.          BLL=00001+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 95. COBOL listing for COBOLZ2

- Check offset +64 in the listing for the assembler routine ASSEMZ3, shown in Figure 96 on page 252.

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*.

No Overriding ASMAOPT Parameters
Overriding Parameters- NODECK,LIST
No Process Statements
Options for this Assembly

External Symbol Dictionary Page 2
Symbol Type Id Address Length Owner Id Flags Alias-of HLASM R5.0 2007/02/14 14.59
ASSEMZ3 SD 00000001 00000000 000000CC 07
CEESTART ER 00000002

Active Usings: None Page 3
Loc Object Code Addr1 Addr2 Stmt Source Statement HLASM R5.0 2007/02/14 14.59
000000 47F0 F014 00014 1 PRINT NOGEN
000056 4150 092C 0092C 2 ASSEMZ3 CEEENTRY MAIN=NO,PPA=MAINPPA
00005A 1B44 39 SR 4,4 Low order part of quotient
00005C 5861 0000 00000 40 L 6,0(1) Hi order part of quotient
000060 4166 0000 00000 41 LA 6,0(6) Get pointer to divisor
000064 5D46 0000 00000 42 D 4,0(6) Clear hi bit
000068 58F0 B0C8 000C8 43 CEETERM RC=0 Do division
50 * Terminate with return code zero
000080 10 51 MAINPPA CEEPPA Constants describing the code block
123**,Time Stamp = 2007/02/14 14:59:00 01-CEEPP
124**,Version 1 Release 1 Modification 0 01-CEEPP
135 CEEDSA Mapping of the Dynamic Save Area
228 CEECAA Mapping of the Common Anchor Area
000000 551 END ASSEMZ3
0000C8 00000000 552 =A(0)

Figure 96. Listing for ASSEMZ3

5. Check local variables for COBOLZ2 in the Local Variables section of the dump shown in Figure 97. 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 97. Variables section of Language Environment dump for COBOLZ2

6. 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 98 on page 253, indicating that the value did pass from COBOLZ1 to COBOLZ2.

```

PP 5655-G53 IBM Enterprise COBOL for z/OS 3.4.1          COBOLZ2  Date 02/14/2007  Time 14:59:15  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=00000+000      2C
000007      LINKAGE SECTION.
000008      77  D-VAL PIC 9(4) USAGE COMP.                BLL=00001+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 98. Listing for COBOLZ2

- In the Local Variables section of the dump for program COBOLZ1, shown in Figure 99, 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 99. 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

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 TERMTHDACT(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 257.
- 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 17. 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 100 on page 262 shows a sample program calling SDUMP and Figure 101 on page 263 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 NOCBCS 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#)
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 NOCBCS 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#)
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 NOCBCS 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#)
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 NOCBCS 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#)
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 100. Example program that calls SDUMP

Figure 101 on page 263 shows the resulting output generated by the example in Figure 100.

```

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 101. 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	AFHOOPNR	05A11638	+00001EDE	AFHOOPNR	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...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	00000000lg;.l.-el.-.....
+000000	0002D3B4	0002DFD7	0002E027	0002DF94	0002DF94	0002DDF4	0002DEC4	0002E158	0002D018	...P.....m...m...4...D.....
+000020	0002D3D4	0002D468	00000000	00000000	00000007	859D67E0	00000000	00000000	05914848	..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 102. Sections of the Language Environment dump

Understanding the Language Environment traceback table

Examine the traceback section of the dump, labeled with [1] in Figure 102, 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 102 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 102 on page 264, 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 102 on page 264 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 103 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 NOBACS 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#)
1      PROGRAM CALLNON
2      INTEGER*4 ARRAY_END
      C
3      CALL SUBNAM
4      STOP
5      END
```

Figure 103. Example of calling a nonexistent routine

Figure 104 on page 266 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 104. 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 103 on page 265. 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 105 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 NOBBS 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#)
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 NOBBS 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#)
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 105. Fortran routine with a divide-by-zero error

Figure 106 on page 268 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..... 0593B75E 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 |.....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..|

:

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 106. 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 106. 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 DIVZEROSUB.

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 106 on page 268. 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 106 on page 268. 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 for MVS & VM routines

This chapter contains information that can help you debug applications that contain one or more PL/I for MVS & VM routines. Following a discussion about potential errors in PL/I for MVS & VM routines, the first part of this chapter discusses how to use compiler-generated listings to obtain information about PL/I for MVS & VM routines, and how to use PLIDUMP to generate a Language Environment dump of a PL/I for MVS & VM routine. The last part of the chapter provides examples of PL/I for MVS & VM 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 for MVS & VM routines
- Using PL/I for MVS & VM compiler listings
- Generating a Language Environment dump of a PL/I for MVS & VM routine
- Finding PL/I for MVS & VM information in a dump
- Debugging example of PL/I for MVS & VM routines

Determining the source of errors in PL/I for MVS & VM routines

Most errors in PL/I for MVS & VM 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 for MVS & VM routine can be caused by:

- Logic errors in the source routine
- Invalid use of PL/I for MVS & VM
- 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 for MVS & VM

A misunderstanding of the language or a failure to provide the correct environment for using PL/I for MVS & VM can result in an apparent malfunction of a PL/I for MVS & VM 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 *PL/I for MVS & VM 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 for MVS & VM 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 for MVS & VM 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 for MVS & VM 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 for MVS & VM compiler listings

The following sections explain how to generate listings that contain information about your routine. PL/I for MVS & VM 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 for MVS & VM routine.

The PL/I compiler listings included below are from the PL/I for MVS & VM product.

Generating PL/I for MVS & VM 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 for MVS & VM routines. For more information about supported compiler options that generate listings, reference the *PL/I for MVS & VM Programming Guide*.

Table 18. Compiler-generated PL/I for MVS & VM 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 18. Compiler-generated PL/I for MVS & VM 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 for MVS & VM listings

Figure 107 shows an example PL/I for MVS & VM 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 107. PL/I for MVS & VM routine compiled with LIST and MAP

Figure 108 on page 276 shows the output generated by the LIST and MAP options for 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 0000000000000044 LOCATOR..B
000028 0000008800140000 LOCATOR..C
000030 91E091E0      CONSTANT
000034 0A000000C5E7E3D9 FECD..EXTR
          40404040
000040 80000034      A..FECD..EXTR
000044 0000000C00000008 DESCRIPTOR
          0000000200000001
          0000000400000002
          00000001
000060 80000034      A..FECD..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

```

          000096 58 B0 C 004      L 11,4(0,12)
          00009A 58 FB 0 000      L 15,PR..EXTR
          00009E 59 F0 C 064      C 15,100(0,12)
000000      DC C'EXAMPLE'          0000A2 47 70 2 01E      BNE CL.5
000007      DC AL1(7)          0000A6 41 10 3 040      LA 1,64(0,3)
          0000AA 58 F0 3 018      L 15,A..IBMSPFRA
* PROCEDURE          EXAMPLE          0000AE 05 EF      BALR 14,15
          0000B0 58 FB 0 000      L 15,PR..EXTR
* REAL ENTRY          0000B4      CL.5 EQU *
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(COMPI LATION 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 108. Compiler-generated listings from example PL/I for MVS & VM 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               * STATEMENT NUMBER 10
000062 55 00 C 00C    CL   0,12(0,12)        0000F0 41 10 3 080      LA  1,128(0,3)
000066 47 D0 F 068    BNH  ++10             0000F4 58 F0 3 014      L   15,A..IBMSJDSA
00006A 58 F0 C 074    L   15,116(0,12)      0000F8 05 EF           BALR 14,15
00006E 05 EF            BALR 14,15
000070 58 E0 D 048    L   14,72(0,13)
000074 18 F0            LR   15,0              * STATEMENT NUMBER 11
000076 90 E0 1 048    STM  14,0,72(1)        0000FA 18 0D           LR   0,13
00007A 50 D0 1 004    ST   13,4(0,1)         0000FC 58 D0 D 004      L   13,4(0,13)
00007E 92 80 1 000    MVI  0(1),X'80'        000100 58 E0 D 00C      L   14,12(0,13)
000082 92 25 1 001    MVI  1(1),X'25'        000104 98 2C D 01C      LM   2,12,28(13)
000086 92 02 1 076    MVI  118(1),X'02'      000108 05 1E           BALR 1,14
00008A 41 D1 0 000    LA   13,0(1,0)
00008E D2 03 D 054 3 030 MVC  84(4,13),48(3)    * END PROCEDURE
000094 05 20            BALR 2,0              00010A 07 07           NOPR 7
* PROCEDURE BASE
* END PROGRAM

```

Figure 108. Compiler-generated listings from example PL/I for MVS & VM 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 19. Typical comments in a PL/I for MVS & VM 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 19. Typical comments in a PL/I for MVS & VM 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 for MVS & VM identifier name
- Level
- Storage class
- Name of the PL/I for MVS & VM 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 for MVS & VM source program listing. Generally, only executable statements appear in the listing. DECLARE statements are not normally included. The names of PL/I for MVS & VM 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 for MVS & VM source program.

Following are the comments used in the listing:

Table 20. Comments in a PL/I for MVS & VM 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 20. Comments in a PL/I for MVS & VM 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 for MVS & VM variable.

Table 21. PL/I for MVS & VM 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 for MVS & VM routine

To generate a dump of a PL/I for MVS & VM 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

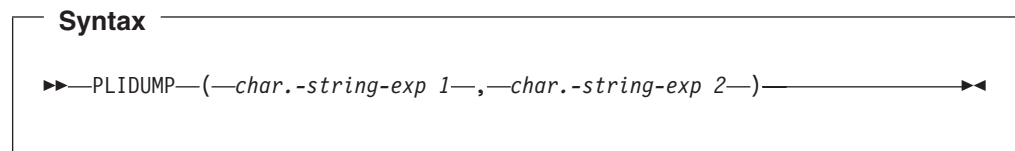
PLIDUMP calls intermediate PL/I for MVS & VM library routines, which convert most PLIDUMP options to CEE3DMP options. The following list contains PLIDUMP options and the corresponding CEE3DMP option, if applicable.

Some PLIDUMP options do not have corresponding CEE3DMP options, but continue to function as PL/I for MVS & VM default options. The list following the syntax diagram provides a description of those options.

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.



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 for MVS & VM hexadecimal dump). Dumps the control blocks used in Language Environment and member language libraries. For PL/I for MVS & VM, this includes the DSA for every routine on the call chain and PL/I for MVS & VM “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 Enterprise 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 for MVS & VM 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 for MVS & VM routine.

Finding PL/I for MVS & VM 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 109 on page 284, for condition information about your routine and information about the statement number and address where the exception occurred.

CEE3845I CEEDUMP Processing started.
PLIDUMP was called from statement number 6 at offset +000000D6 from ERR ON-unit with entry address 20900C58

Information for enclave EXAMPLE

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEKCMRA	+0000081C		CEEPLPKA	CEEKCMRA	D1908	Call
2	IBMRKDM	+000000C2		IBMREV10	IBMRKDM		Call
3	ERR ON-unit+000000D6		6	EXAMPLE	EXAMPLE		Call
4	IBMRERPL	+0000065A		IBMRLIB1	IBMRERPL		Call
5	CEEEV010	+0000013A		IBMREV10	CEEEV010		Call
6	CEEHDSP	+000017D0		CEEPLPKA	CEEHDSP	D1908	Call
7	IBMRERRI	+0000045A		IBMRLIB1	IBMRERRI		Exception
8	LABL1: BEGIN+000000BE		11	EXAMPLE	EXAMPLE		Call
9	EXAMPLE	+000000C8	8	EXAMPLE	EXAMPLE		Call
10	IBMRPMIA	+0000051E		IBMRLIB1	IBMRPMIA		Call
11	CEEEV010	+00000310		IBMREV10	CEEEV010		Call
12	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	20B45B88	209F0420	209F0420	+0000081C	20061214	CEL
2	00025670	20B1C0A0	20B1C0A0	+000000C2	*****	OS PL/I
3	20B45A88	20900C58	20900B70	+000001BE	*****	OS PL/I
4	20B45850	00019F50	00019F50	+0000065A	20061213	LIBRARY
5	20B457C8	20B02998	20B02998	+0000013A	20061213	LIBRARY
6	20B426A8	209BF068	209BF068	+000017D0	20061215	CEL
7	20B42500	0001B328	0001B328	+0000045A	20061213	LIBRARY
8	20B42430	20900D48	20900B70	+00000296	*****	OS PL/I
9	20B42330	20900B78	20900B70	+000000D0	*****	OS PL/I
10	20B42178	000201D0	000201D0	+0000051E	20061214	LIBRARY
11	20B420F0	20B02998	20B02998	+00000310	20061213	LIBRARY
12	20B42030	2098DDB8	2098DDB8	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for IBMRERRI (DSA address 20B42500)

CIB Address: 20B42FC8

Current Condition:

IBM0281S A prior condition was promoted to the ERROR condition.

Original Condition:

IBM0421S ONCODE=520 The SUBSCRIPTRANGE condition was raised.

Location:

Program Unit: IBMRERRI Entry: IBMRERRI Statement: Offset: +0000045A

Storage dump near condition, beginning at location: 0001B772

+000000 0001B772 5050D080 58A0C2B8 58F0A01C 4110D080 05EF9108 404F4710 B4709104 404F47E0 |&&....B..0.....j. |....j. |..|

Figure 109. Traceback section of dump

PL/I for MVS & VM task traceback

A task traceback table is produced for multitasking programs showing the task invocation sequence (trace). For each task, the thread ID, CAA address (identified by TCA address in the dump), event variable address, task variable address, and absolute priority appear in the traceback table. An example is shown in Figure 110 on page 285.

CEE3845I CEEDUMP Processing started.
 PLIDUMP was called from statement number 5 at offset +000000E6 from SUBTSK2 within task SUBTSK2

PL/I Task Traceback:

Task	Attached by	Thread ID	TCA Addr	EV Addr	TV Addr	Absolute Priority
SUBTSK2	SUBTSK1	1171C16000000003	11489BD8	11200AA0	00035290	000
SUBTSK1	SUBTASK	1171943000000002	11480BD8	11200AE0	000266F8	000
SUBTASK	TASKING	1171761000000001	11440D48	11200B20	00025D70	000
TASKING		1170C15000000000	1120F9C0	000255D4	0002523C	254

Information for enclave TASKING

Information for thread 1171C16000000003

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEKMMRA	+0000081C		CEEPLPKA	CEEKMMRA	D1908	Call
2	IBMRKDM	+000000C2		IBMREV10	IBMRKDM		Call
3	SUBTSK2	+000000E6	5	GO	SUBTSK2		Call
4	IBMUPTMM	+0000014E		IBMLIB1	IBMUPTMM		Call
5	CEEOPCMM	+00000908		CEEBINIT	CEEOPCMM	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	00036430	112FD268	112FD268	+0000081C	20061214	CEL POSIX
2	000352E8	1141D0A0	1141D0A0	+000000C2	*****	OS PL/I POSIX
3	00036308	11201048	11201040	+000000EE	*****	OS PL/I POSIX
4	00036040	00021900	00021900	+0000014E	20061214	LIBRARY POSIX
5	1148DEE0	0000E460	0000E460	+00000908	20061215	CEL POSIX

:

Information for thread 1170C15000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	IBMUJWTP	+00000146		IBMREV10	IBMUJWTP		Call
2	IBMUJWT	+0000027A		IBMLIB1	IBMUJWT		Call
3	TASKING	+00000106	11	GO	TASKING		Call
4	IBMRPMIA	+0000051E		IBMLIB1	IBMRPMIA		Call
5	CEEEV010	+00000310		IBMREV10	CEEEV010		Call
6	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	11443530	1143A850	1143A850	+00000146	20061214	LIBRARY POSIX
2	11443438	00020E28	00020E28	+0000027A	*****	OS PL/I POSIX
3	11443330	11200728	11200720	+0000010E	*****	OS PL/I POSIX
4	11443178	000201D0	000201D0	+0000051E	20061214	LIBRARY POSIX
5	114430F0	11403998	11403998	+00000310	20061213	LIBRARY POSIX
6	11443030	112B1BF8	112B1BF8	+000001B6	20061215	CEL POSIX

:

Information for thread 1171943000000002

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	IBMUJWTP	+00000146		IBMREV10	IBMUJWTP		Call
2	IBMUJWT	+0000027A		IBMLIB1	IBMUJWT		Call
3	SUBTSK1	+000000D6	13	GO	SUBTSK1		Call
4	IBMUPTMM	+0000014E		IBMLIB1	IBMUPTMM		Call
5	CEEOPCMM	+00000908		CEEBINIT	CEEOPCMM	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	00030508	1143A850	1143A850	+00000146	20061214	LIBRARY POSIX
2	00030410	00020E28	00020E28	+0000027A	*****	OS PL/I POSIX
3	00030308	1120DD08	1120DD00	+000000DE	*****	OS PL/I POSIX
4	00030040	00021900	00021900	+0000014E	20061214	LIBRARY POSIX
5	11485EE0	0000E460	0000E460	+00000908	20061215	CEL POSIX

:

Figure 110. Task traceback section (Part 1 of 2)

Information for thread 1171761000000001

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	IBMUJWTP	+00000146		IBMREV10	IBMUJWTP		Call
2	IBMUJWT	+0000027A		IBMRLIB1	IBMUJWT		Call
3	SUBTASK	+000000D6	13	GO	SUBTASK		Call
4	IBMUPTMM	+0000014E		IBMRLIB1	IBMUPTMM		Call
5	CEEOPCMM	+00000908		CEEBINIT	CEEOPCMM	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	00029508	1143A850	1143A850	+00000146	20061214	LIBRARY POSIX
2	00029410	00020E28	00020E28	+0000027A	*****	OS PL/I POSIX
3	00029308	11200B68	11200B60	+000000DE	*****	OS PL/I POSIX
4	00029040	00021900	00021900	+0000014E	20061214	LIBRARY POSIX
5	11479EE0	0000E460	0000E460	+00000908	20061215	CEL POSIX

:

Figure 110. Task traceback section (Part 2 of 2)

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 111 on page 287 shows this section of the dump.

Control Blocks for Active Routines:

```

DSA for CEEKMRA: 20B45B88
+000000  FLAGS.... 0000  member... 4040  BKC..... 00025670  FWC..... 20B46080  R14..... A09F0C3E
+000010  R15..... A09EB5B8  R0..... 00000000  R1..... 20B45C18  R2..... 000251E8  R3..... 00000000
+000024  R4..... 00000001  R5..... 20B45E17  R6..... 00025030  R7..... 00000000  R8..... A09104D0
+000038  R9..... 00000000  R10..... 209F141F  R11..... A09F0420  R12..... 2090E9C0  reserved. 00025760
+00004C  NAB..... 20B46080  PNAB..... 20B40050  reserved. 00500007  00000000
+000064  reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 8000C3C5

DSA for IBMRKDM: 00025670
+000000  FLAGS.... 0800  member... 01E0  BKC..... 20B45A88  FWC..... 20B45B88  R14..... A0B1C164
+000010  R15..... A09F0420  R0..... 80007810  R1..... 20900EBC  R2..... 20B45A88  R3..... A0B1C0A6
+000024  R4..... 00000064  R5..... 00000000  R6..... 20900EBC  R7..... 20B45938  R8..... 20B45B58
+000038  R9..... 20B45B7C  R10..... 00000000  R11..... 0001AF4F  R12..... 2090E9C0  reserved. 00025760
+00004C  NAB..... 20B45B88  PNAB..... 00000000  reserved. 00000000  00000000
+000064  reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000

Library Work Space: 00025670
+000000  00025670  080001E0  20B45A88  20B45B88  A0B1C164  A09F0420  80007810  20900EBC  20B45A88  |.....!h..$.h..A.....!h
+000020  00025690  A0B1C0A6  00000064  00000000  20900EBC  20B45938  20B45B58  20B45B7C  00000000  |...w.....$.$.$.@....
+000040  00025680  0001AF4F  2090E9C0  00025760  20B45B88  00000000  00000000  00000000  00000000  |...|.Z.....$.h.....
+000060  00025600  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  |.....
+000080  000256F0 - +00009F  0002570F  same as above

DSA for ERR ON-unit: 20B45A88
+000000  FLAGS.... CC25  member... 4040  BKC..... 20B45850  FWC..... 40404040  R14..... A0900D30
+000010  R15..... A0B1C0A0  R0..... 20B45B88  R1..... 20900EBC  R2..... A0900E4  R3..... 20900E40
+000024  R4..... 00000064  R5..... 00000000  R6..... 20B45B40  R7..... 20B45938  R8..... 20B45B58
+000038  R9..... 20B45B7C  R10..... 00000000  R11..... 0001AF4F  R12..... D6D940C7  reserved. 00025670
+00004C  NAB..... 20B45B88  PNAB..... 20B45B88  reserved. 91E091E0  20900F00
+000064  reserved. 40404040  reserved. 40404040  MODE..... 40404040  reserved. 20B45B40
+000078  reserved. 40404040  reserved. 40404040

Dynamic save area (ERR ON-unit): 20B45A88
+000000  20B45A88  CC254040  20B45850  40404040  A0900D30  A0B1C0A0  20B45B88  20900EBC  A0900CE4  |.. ..& .....$.h.....U
+000020  20B45A88  20900E40  00000064  00000000  20B45B40  20B45938  20B45B58  20B45B7C  00000000  |... ..$.$.$.$.@....
+000040  20B45AC8  0001AF4F  D6D940C7  00025670  20B45B88  20B45B88  91E091E0  20B42330  20900F00  |...|OR G.....$.h..$.h.j.....
+000060  20B45AE8  40404040  40404040  40404040  40404040  20B45B40  40400240  40404040  40404040  |.....$.
+000080  20B45B08  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  |.....
+0000A0  20B45B28  40404040  40404040  40404040  40404040  40404040  40404040  0C014040  40404040  |.....
+0000C0  20B45B48  A3829586  A2404040  20B45B48  00050000  D7938984  A4949740  83819393  85844086  |tbnfs ..$......Plidump called f
+0000E0  20B45B68  99969440  85999996  9940D695  60A49589  A3404040  20B45B58  00210000  40404040  |rom error On-unit ..$.

Static for procedure EXAMPLE Timestamp: 27 FEB 07 11:45:18
Starting from: 20900E40
+000000  20900E40  E00001E0  20900B78  20900C20  20900C30  20900C58  20900CE4  20900D48  20900DBE  |.....U.....
+000020  20900E60  20900DBE  20900DBE  20900DBE  20901E40  B0000002  1F800004  00000000  20900E94  |.....m
+000040  20900E80  00000000  00050000  00000000  00210000  91E091E0  00000004  00000004  0000000A  |.....j.j.....
+000060  20900EA0  00000001  91A091A0  00000014  00000001  0A200000  0000000A  00000002  20B45B50  |.....$.
+000080  20900EC0  A0B45B7C  20901368  A3829586  A2D79389  84A49497  40838193  93858440  86999694  |..$.tbnfsPlidump called from
+0000A0  20900EE0  40859999  969940D6  9560A495  89A30000  20900C34  20900C30  0C160000  20900C58  |error On-unit.....
+0000C0  20900F00  0C960000  00000000  20901038  00000000  00000000  20900F3C  20900F50  20900F6C  |.o.....&.....%
+0000E0  20900F20  20900F84  00000000  20900F08  00000000  20900F14  00000000  20900F14  85000001  |...d.....e.....
+000100  20900F40  20900E72  000000C8  00000000  0001C900  85000001  20900E72  000000CC  00000000  |.....H.....I.e.....
+000120  20900F60  0009C1D9  D9C1E86D  C5D5C400  81000101  20900E72  000000C0  00000000  0005C1D9  |..ARRAY_END.a.....AR
+000140  20900F80  D9C1E800  0D020001  20900E70  20900EF0  00000000  0005D3C1  C2D3F100  00000001  |RAY.....0.....LABL1.....
+000160  20900FA0  20900B78  000000DE  20900FC0  00000001  00B90004  00BD0008  00CF000E  00E00004  |.....
+000180  20900FC0  20900C58  000000E8  20900FE0  00000004  008D0005  00910006  00D90007  00E80008  |.....Y.....j...R...Y..
+0001A0  20900FE0  20900D48  000000F6  20901008  00000008  00770009  0083000A  009F000B  00D1000C  |.....6.....c.....J..
+0001C0  20901000  00E0000D  00EC000D  00E0E0E0  F2F740C6  C5C240F0  F74040F1  F17AF4F5  7AF1F840  |.....27 FEB 07 11:45:18
+0001E0  20901020  80000117  20900AF0  01000001  20900B78  20901340  00000000  1D020000  20901054  |.....0.....
:
DSA for IBMRERRI: 20B42500
+000000  FLAGS.... 8800  member... 0000  BKC..... 20B42430  FWC..... 20B425C0  R14..... 8001B784
+000010  R15..... A09D0B48  R0..... 0000000B  R1..... 20B42580  R2..... 0000000A  R3..... 20900EB0
+000024  R4..... 00025470  R5..... 000254C4  R6..... 20B42330  R7..... 20B42330  R8..... 00000028
+000038  R9..... 00000008  R10..... A09104D0  R11..... 0001B328  R12..... 2090E9C0  reserved. 00025290
+00004C  NAB..... 20B425C0  PNAB..... 008FF4E8  reserved. 2090D658  20AF22BC
+000064  reserved. 2090E9C0  reserved. 20B420F0  MODE..... 20B2FA47  reserved. A09104D0
+000078  reserved. 00000000  reserved. 20B42838

```

Figure 111. Control blocks for active routines section of the dump (Part 1 of 3)

```

CIB for IBMRERRI: 20B42FC8
+000000 20B42FC8 C3C9C240 00000000 00000000 010C0004 00000000 00000000 00030119 59C9C2D4 |CIB .....IBM|
+000020 20B42FE8 00000000 20B430D8 000301A5 59C9C2D4 00000001 00000015 20B42330 A0B02998 |.....Q...v.IBM.....q|
+000040 20B43008 00000000 20B42500 8001B784 2090B6F0 0000000A 20B42430 00000000 00000000 |.....d...0.....|
+000060 20B43028 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 20B43048 - +00009F 20B43067 same as above |.....|
+0000A0 20B43068 00000000 00000000 00000000 00000000 06230000 00000FC6 00000001 00000000 |.....F.....|
+0000C0 20B43088 00000000 00000000 20B42430 20B42500 0001B782 00000000 00000000 00000001 |.....b.....|
+0000E0 20B430A8 20B42330 0000000A 00000064 00000000 FFFFFFFC 00000000 00000000 00000000 |.....|
+000100 20B430C8 00000000 2090B908 00000000 00000000 E9D4C3C8 02000001 0000000B 20B42580 |.....ZMCH.....|

Dynamic save area (IBMRERRI): 20B42500
+000000 20B42500 88000000 20B42430 20B425C0 8001B784 A09D0B48 0000000B 20B42580 0000000A |h.....d.....|
+000020 20B42520 20900E00 00025470 000254C4 20B42330 20B42330 00000028 00000008 A09104D0 |.....D.....j..|
+000040 20B42540 0001B328 2090E9C0 00025290 20B425C0 008FF4E8 2090D658 00000000 20AF22BC |.....Z.....4Y..0.....|
+000060 20B42560 A0997C20 2090E9C0 20B420F0 20B2FA47 A09104D0 2090E9C0 00000000 20B42838 |.r@...Z...0....j...Z.....|
+000080 20B42580 000254C4 00000000 00000000 00000000 00000000 00000000 00000000 |...D.....|
+0000A0 20B425A0 16000000 20909DB0 00000000 80010000 00000000 209D4520 20B2F7B4 2090E200 |.....7...S..|

DSA for LABL1: BEGIN: 20B42430
+000000 FLAGS.... 8425 member... 0000 BKC..... 20B42330 FWC..... 20B42500 R14..... A0900E08
+000010 R15..... 0001B328 R0..... 0000000B R1..... 20900E00 R2..... A0900DBE R3..... 20900E40
+000024 R4..... 00000001 R5..... 20B42330 R6..... 20B42330 R7..... 20B42330 R8..... 00000028
+000038 R9..... 00000008 R10..... 20B420B0 R11..... 2090102C R12..... 2090E9C0 reserved. 00025290
+00004C NAB..... 20B42500 PNAB.... 20B42500 reserved. 91A091A0 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE.... 00000000 reserved. 00000000
+000078 reserved. 00000000 reserved. 20B42030

Dynamic save area (LABL1: BEGIN): 20B42430
+000000 20B42430 84250000 20B42330 20B42500 A0900E08 0001B328 0000000B 20900E00 A0900DBE |d.....|
+000020 20B42450 20900E40 00000001 20B42330 20B42330 20B42330 00000028 00000008 20B420B0 |.....|
+000040 20B42470 2090102C 2090E9C0 00025290 20B42500 20B42500 91A091A0 20B42330 00000000 |.....Z.....j..j.....|
+000060 20B42490 00000000 00000000 00000000 00000000 00000000 00002000 00000000 20B42030 |.....|
+000080 20B424B0 20B42530 A0B02CEC 20B2EA48 00000800 20B420F0 20B420F0 A0B02998 20901340 |.....0...0...q...|
+0000A0 20B424D0 2090E880 0000000A 00000000 00000000 20AF2CD6 00000027 A0AF0CDB 2090E9C0 |...Y.....0.....Q..Z..|
+0000C0 20B424F0 20B42330 20B42618 00000014 00000000 88000000 20B42430 20B425C0 8001B784 |.....h.....d..|

DSA for EXAMPLE: 20B42330
+000000 FLAGS.... C025 member... 0000 BKC..... 20B42178 FWC..... 00000000 R14..... A0900C42
+000010 R15..... 20900D48 R0..... 20B42430 R1..... 20B42330 R2..... A0900C30 R3..... 20900E40
+000024 R4..... 00000001 R5..... 20B42330 R6..... 20B42400 R7..... 00000005 R8..... 20900EF8
+000038 R9..... 00000008 R10..... 20B420B0 R11..... 2090102C R12..... 2090E9C0 reserved. 00025290
+00004C NAB..... 20B42430 PNAB.... 20B42430 reserved. 91E091A0 20900EF8
+000064 reserved. 00000000 reserved. 00000000 MODE.... 00000000 reserved. 20B423E8
+000078 reserved. 00000000 reserved. 00000000

Dynamic save area (EXAMPLE): 20B42330
+000000 20B42330 C0250000 20B42178 00000000 A0900C42 20900D48 20B42430 20B42330 A0900C30 |.....|
+000020 20B42350 20900E40 00000001 20B42330 20B42400 00000005 20900EF8 00000008 20B420B0 |.....8.....|
+000040 20B42370 2090102C 2090E9C0 00025290 20B42430 20B42430 91E091A0 00000000 20900EF8 |.....Z.....j..j.....8|
+000060 20B42390 00000000 00000000 00000000 00000000 20B423E8 00002000 00000000 00000000 |.....Y.....|
+000080 20B423B0 20B42124 20B42128 20B4212C 20B42130 20B42138 20B42134 20B4213C 00000000 |.....|
+0000A0 20B423D0 00000000 00000000 00000000 00000000 00000000 00000000 0C010000 00000000 |.....|
+0000C0 20B423F0 20B42400 20900E94 0000000B 00000014 00000002 00000002 00000002 00000002 |.....m.....|
+0000E0 20B42410 00000002 00000002 00000002 00000002 00000002 00000002 00000000 00000000 |.....|

:
Dynamic save area (IBMRERRI): 20B42500
+000000 20B42500 88000000 20B42430 20B425C0 8001B784 A09D0B48 0000000B 20B42580 0000000A |h.....d.....|
+000020 20B42520 20900E00 00025470 000254C4 20B42330 20B42330 00000028 00000008 A09104D0 |.....D.....j..|
+000040 20B42540 0001B328 2090E9C0 00025290 20B425C0 008FF4E8 2090D658 00000000 20AF22BC |.....Z.....4Y..0.....|
+000060 20B42560 A0997C20 2090E9C0 20B420F0 20B2FA47 A09104D0 2090E9C0 00000000 20B42838 |.r@...Z...0....j...Z.....|
+000080 20B42580 000254C4 00000000 00000000 00000000 00000000 00000000 00000000 |...D.....|
+0000A0 20B425A0 16000000 20909DB0 00000000 80010000 00000000 209D4520 20B2F7B4 2090E200 |.....7...S..|

DSA for LABL1: BEGIN: 20B42430
+000000 FLAGS.... 8425 member... 0000 BKC..... 20B42330 FWC..... 20B42500 R14..... A0900E08
+000010 R15..... 0001B328 R0..... 0000000B R1..... 20900E00 R2..... A0900DBE R3..... 20900E40
+000024 R4..... 00000001 R5..... 20B42330 R6..... 20B42330 R7..... 20B42330 R8..... 00000028
+000038 R9..... 00000008 R10..... 20B420B0 R11..... 2090102C R12..... 2090E9C0 reserved. 00025290
+00004C NAB..... 20B42500 PNAB.... 20B42500 reserved. 91A091A0 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE.... 00000000 reserved. 00000000
+000078 reserved. 00000000 reserved. 20B42030

```

Figure 111. Control blocks for active routines section of the dump (Part 2 of 3)

```

Dynamic save area (LABL1: BEGIN): 20B42430
+000000 20B42430 84250000 20B42330 20B42500 A0900E08 0001B328 0000000B 20900EB0 A0900DBE |d.....|
+000020 20B42450 20900E40 00000001 20B42330 20B42330 20B42330 00000028 00000008 20B420B0 |.....8.....|
+000040 20B42470 2090102C 2090E9C0 00025290 20B42500 20B42500 91A091A0 20B42330 00000000 |.....Z.....j.j.....|
+000060 20B42490 00000000 00000000 00000000 00000000 00000000 00000200 00000000 20B42030 |.....|
+000080 20B424B0 20B42530 A0B02CEC 20B2EA48 00000800 20B420F0 20B420F0 A0B02998 20901340 |......0...0...q...|
+0000A0 20B424D0 2090E880 0000000A 00000000 00000000 20AF2CD6 00000027 A0AF0CD8 2090E9C0 |.Y......0.....Q..Z..|
+0000C0 20B424F0 20B42330 20B42618 00000014 00000000 88000000 20B42430 20B425C0 8001B784 |.....h.....d.....|
DSA for EXAMPLE: 20B42330
+000000 FLAGS.... C025      member... 0000      BKC..... 20B42178  FWC..... 00000000  R14..... A0900C42
+000010 R15..... 20900D48  R0..... 20B42430  R1..... 20B42330  R2..... A0900C30  R3..... 20900E40
+000024 R4..... 00000001  R5..... 20B42330  R6..... 20B42400  R7..... 00000005  R8..... 20900EF8
+000038 R9..... 00000008  R10..... 20B420B0  R11..... 2090102C  R12..... 2090E9C0  reserved. 00025290
+00004C NAB..... 20B42430  PNAB..... 20B42430  reserved. 91E091A0 20900EF8
+000064 reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 20B423E8
+000078 reserved. 00000000  reserved. 00000000
Dynamic save area (EXAMPLE): 20B42330
+000000 20B42330 C0250000 20B42178 00000000 A0900C42 20900D48 20B42430 20B42330 A0900C30 |.....|
+000020 20B42350 20900E40 00000001 20B42330 20B42400 00000005 20900EF8 00000008 20B420B0 |.....8.....|
+000040 20B42370 2090102C 2090E9C0 00025290 20B42430 20B42430 91E091A0 00000000 20900EF8 |.....Z.....j.j.....8|
+000060 20B42390 00000000 00000000 00000000 00000000 20B423E8 00000200 00000000 00000000 |.....Y.....|
+000080 20B423B0 20B42124 20B42128 20B4212C 20B42130 20B42138 20B42134 20B4213C 00000000 |.....|
+0000A0 20B423D0 00000000 00000000 00000000 00000000 00000000 00000000 0C010000 00000000 |.....|
+0000C0 20B423F0 20B42400 20900E94 0000000B 00000014 00000002 00000002 00000002 00000002 |.....m.....|
+0000E0 20B42410 00000002 00000002 00000002 00000002 00000002 00000002 00000000 00000000 |.....|
:

```

Figure 111. Control blocks for active routines section of the dump (Part 3 of 3)

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 *PL/I for MVS & VM 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 112 on page 291, includes information about PL/I for MVS & VM fields of the CAA and other control block information.

Control Blocks Associated with the Thread:

CAA: 2090E9C0

```
+000000 2090E9C0 00000000 00025658 20B42018 20B62018 00000000 2090E9D0 00000000 000251B0 |.....Z.....|
+000020 2090E9E0 00000000 00000000 00025030 00000000 00025240 00000000 00025208 00000000 |.....&.....|
+000040 2090EA00 000251D0 00000000 00020028 00000000 00020A80 00019D38 00000000 00000000 |.....|
+000060 2090EA20 00000000 00019CB8 00025658 0001FB70 0001FEE0 80014608 0001B328 70002323 |.....|
```

DUMMY DSA: 2090F360

```
+000000 FLAGS.... 0000 member... 0000 BKC..... 00006F60 FWC..... 20B42030 R14..... A0900D66
+000010 R15..... A098DD88 R0..... 7D000008 R1..... 2090D778 R2..... 00000000 R3..... 00000000
+000024 R4..... 00000000 R5..... 00000000 R6..... 00000000 R7..... A09104D0 R8..... 20900808
+000038 R9..... 008DCD0 R10..... 00000000 R11..... A0900C92 R12..... 2090E9C0 reserved. 00025290
+00004C NAB..... 20B42030 PNAB..... 20B42030 reserved. 00000000 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE..... 00000000 reserved. 00000000
+000078 reserved. 00000000 reserved. 00000000
```

PL/I TCA Appendage: 00025030

```
+000000 00025030 00000000 00000000 00000000 00000030 00000000 00000000 00000000 00000000 |.....|
+000020 00025050 00000000 000251E8 00000000 00000000 00000000 00000000 00000000 00000000 |...Y.....|
+000040 00025070 00000000 00000000 00000000 00000000 2090E9C0 00000000 00000000 00000000 |.....Z.....|
+000060 00025090 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 000250B0 - +0000BF 000250EF same as above
+0000C0 000250F0 00000000 00000000 00000000 00000000 00000000 00000000 20900838 00000000 |.....|
+0000E0 00025110 00000000 00000000 00000000 00000000 00000000 00000000 7FFFFFFF 00000000 |.....".....|
+000100 00025130 00000000 00000000 00000000 00000000 00000000 00000000 00000000 20B3E034 |.....|
+000120 00025150 00000000 0000FB00 0002519D 00008000 0002519C 00010000 0002519D 00008000 |.....|
+000140 00025170 0002519D 00008000 0002519D 00008000 00000000 00000000 00000000 00000000 |.....|
+000160 00025190 00000000 00000000 00000000 00000000 40000000 00000000 00000000 00000000 |.....|
+000180 000251B0 00010000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0001A0 000251D0 00000000 000076BC 00000000 00000000 00000000 00000000 80C16800 20B45A68 |.....A...!.....|
+0001C0 000251F0 00000000 209004A8 00000000 00000000 00000000 00000000 00000000 00000000 |.....y.....|
+0001E0 00025210 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000200 00025230 - +00023F 0002526F same as above
+000240 00025270 00000000 00000000 00000000 00000000 00000000 00000000 D7404040 E9D3E6E2 |.....P ZLWS|
+000260 00025290 08001E0 20B42460 00000000 5E007744 00CBA600 00027290 00029F78 A090055C |.....-...;...w.....*|
+000280 000252B0 800076BC 00028004 00000015 00029F78 00028004 00028090 00000008 00000010 |.....|
+0002A0 000252D0 20B42538 2090E9C0 00025380 20B42560 00000000 00000000 000000B4 00000000 |.....Z.....-.....|
+0002C0 000252F0 00000000 00150000 00000000 00000000 00000000 00000000 00000000 00000000 |.....&.....|
+0002E0 00025310 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000300 00025330 - +00033F 0002536F same as above
+000340 00025370 00000000 00000000 00000000 00000000 080000F0 00000000 00000000 00000000 |.....0.....|
+000360 00025390 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000380 000253B0 00000000 00000000 00000000 00000000 00000000 00000000 00025380 00000000 |.....|
+0003A0 000253D0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0003C0 000253F0 - +00043F 0002546F same as above
+000440 00025470 00025150 02000000 00000000 00000000 00010000 00000000 00000000 00000000 |...&.....|
+000460 00025490 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000480 000254B0 00000000 00000000 00000000 40000000 00000000 00030119 59C9C2D4 00000000 |.....IBM.....|
+0004A0 000254D0 00000000 00025670 00000000 A0B1C156 8000FCD8 00025558 00025524 20B45940 |.....A...Q.....|
+0004C0 000254F0 A0B1C0A6 00000064 00000000 209005A8 20B457F0 20B45A30 20B45A60 00000000 |...w.....y...0...!...-...|
+0004E0 00025510 0001AF4F 2090E9C0 00000000 00000000 8000FCD8 00025544 00025548 0002554C |...|.Z.....Q.....<|
+000500 00025530 00025550 00025554 00025558 0002555C 00025560 20B1C220 00000008 00000000 |...&.....*...-...B.....|
+000520 00025550 00000000 800075B0 00000100 00000000 00000000 00000000 00000000 00000000 |.....|
+000540 00025570 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000560 00025590 - +00061F 0002564F same as above
+000620 00025650 00019CB8 00028004 00020AEB 00020AEB 00020AEB 00000000 00025000 00000248 |.....Y...Y...Y...&.....|
```

Enclave Control Blocks:

EDB: 2090D658

```
+000000 2090D658 C3C5C5C5 C4C24040 C0000001 2090E880 2090DD60 00000000 00000000 00000000 |CEEEDB .....Y...-.....|
+000020 2090D678 2090DB68 2090DB98 A09104D0 2090D1A8 00000000 00000000 2090D778 00000000 |.....q.j...Jy.....P....|
+000040 2090D698 00000000 00000000 00006F60 00000000 00000000 A0A2E878 20909880 2090D6B0 |.....?.....sY...q...0..|
+000060 2090D6B8 0000F460 00000000 00000000 00000000 00000000 00000000 20AEF660 2090E9C0 |.4-.....6...Z.....|
+000080 2090D6D8 40000000 0000FAF6 00000000 00000000 00000001 0002A060 00006FF8 008FF4E8 |.....6.....-...?8..4Y|
+0000A0 2090D6F8 00000001 00000100 20909988 2090D700 00000000 00000000 00000000 00000001 |.....rh..P.....|
```

MEML: 2090E880

```
+000000 2090E880 00000000 00000000 20990160 00000000 00000000 00000000 20990160 00000000 |.....r.-.....r.-....|
+000020 2090E8A0 - +00009F 2090E91F same as above
+000040 2090E920 00000000 00000000 A0B02998 00000000 00000000 00000000 20990160 00000000 |.....q.....r.-....|
+000060 2090E940 00000000 00000000 20990160 00000000 00000000 00000000 20990160 00000000 |.....r.-.....r.-....|
+000080 2090E960 - +00011F 2090E99F same as above
```

Figure 112. Control blocks associated with the thread section of the dump (Part 1 of 2)

```

File Status and Attributes:
Attributes of file: SYSPRINT
STREAM OUTPUT PRINT ENVIRONMENT( VB BLKSIZE(129) RECSIZE(125) BUFFERS(1) )
Contents of buffers
BUFFER: 00029F78
+000000 00029F78 000E0000 40E2A482 F140E2A3 8199A389 95874087 40000000 00000000 00000000 |... Sub1 Starting g .....|
+000020 00029F98 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 00029FB8 - +00007F 00029FF7 same as above
+000080 00029FF8 00000000 00000000 04C4E2E3 01004000 00000000 0002C000 E2E8E2D6 E4E34040 |.....MDST.. .....SYSOUT |

File Control Blocks:
FILE CONTROL BLOCK (FCB): 00028004
+000000 00028004 00000000 00000000 0001D170 000076B4 000280F4 00028090 00000000 00000000 |.....J.....4.....|
+000020 00028024 00000000 49291100 50000000 00000004 00810000 00000079 00029F78 E3E50114 |.....&.....a.....TV..|
+000040 00028044 00000000 00000000 00000000 00029F8B 006A0001 003C0078 0002007D 00000000 |.....'|
+000060 00028064 00000000 00000000 00000000 20B42538 00000000 00000000 00000000 00000000 |.....|
DATA CONTROL BLOCK (DCB): 00028090
+000000 00028090 00000000 00000000 00000000 01000000 00020000 01029F70 00814000 00006DE0 |.....a..._|
+000020 000280B0 42007DB0 54000000 00540048 008CC038 92CBA600 00000001 000078D6 08090081 |..'......k.w.....0...a|
+000040 000280D0 00000000 00006C80 00029FF5 00029FF5 0000007D 80000001 00000000 00000001 |.....%...5...5...'|
DECLARE CONTROL BLOCK (DCLCB): 000280F4
+000000 000280F4 FFFFFFFF 41201000 02D70F00 00000000 00000014 0008E2E8 E2D7D9C9 D5E30000 |.....P.....SYSPRINT..|

Process Control Blocks:
PCB: 2090D1A8
+000000 2090D1A8 C3C5C5D7 C3C24040 03030288 00000000 00000000 00000000 2090D3E0 A0AF24D0 |CEEPCB ...h.....L.....|
+000020 2090D1C8 A0AE8B50 A0AEFA08 A0AEF528 20907AE8 2090CF78 00000000 00000000 2090D658 |...&.....5...:Y.....0..|
+000040 2090D1E8 A0AEF858 7F800000 00000000 000141D4 00000000 80000000 00000000 00000000 |..8.".....M.....|

MEML: 2090D3E0
+000000 2090D3E0 00000000 00000000 20990160 00000000 00000000 00000000 20990160 00000000 |.....r-.....r-....|
+000020 2090D400 - +00009F 2090D47F same as above
+0000A0 2090D480 00024038 00000000 A0B02998 00000000 00000000 00000000 20990160 00000000 |.. .....q.....r-....|
+0000C0 2090D4A0 00000000 00000000 20990160 00000000 00000000 00000000 20990160 00000000 |.....r-.....r-....|
+0000E0 2090D4C0 - +00011F 2090D4FF same as above

PL/I Process Control Block: 00024038
+000000 00024038 E9D7D9C2 40404040 00024000 00028004 00000000 00000000 00019CB8 00000000 |ZPRB .. .....|
+000020 00024058 00000000 00000000 00000000 80000000 00000000 00000000 00000000 00000000 |.....|
+000040 00024078 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
CEE3846I CEEDUMP Processing completed.

```

Figure 112. Control blocks associated with the thread section of the dump (Part 2 of 2)

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 for MVS & VM 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 for MVS & VM contents of the Language Environment trace table

Language Environment provides three PL/I for MVS & VM 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 147.

Debugging example of PL/I for MVS & VM routines

This section contains examples of PL/I for MVS & VM 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 113 on page 294 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.

```

5688-235 IBM PL/I for MVS & VM          Ver 1 Rel 1 Mod 1          27 FEB 07  11:45:18  PAGE  1
OPTIONS SPECIFIED
*PROCESS  GOSTMT LIST S STG TEST MAP NOOPTIONS;
5688-235 IBM PL/I for MVS & VM          EXAMPLE:  PROC  OPTIONS(MAIN);          PAGE  2
SOURCE LISTING
STMT
1  EXAMPLE:  PROC  OPTIONS(MAIN);

2  DCL Array(10) Fixed bin(31);
3  DCL (I,Array_End)  Fixed bin(31);
4  On error
   Begin;
5     On error system;
6     Call plidump('tbnfs','Plidump called from error On-unit');
7     End;

8  (subrg):          /* Enable subscriptrange condition */
   Lab11: Begin;
9     Array_End = 20;
10    Do I = 1 to Array_End; /* Loop to initialize array */
11       Array(I) = 2; /* Set array elements to 2 */
12    End;
13    End Lab11;
14  End Example;
:
5688-235 IBM PL/I for MVS & VM          EXAMPLE:  PROC  OPTIONS(MAIN);          PAGE  5
VARIABLE STORAGE MAP
IDENTIFIER          LEVEL          OFFSET          (HEX)  CLASS  BLOCK
I
ARRAY_END          1          204          C8  AUTO  EXAMPLE
ARRAY              1          208          D0  AUTO  EXAMPLE
5688-235 IBM PL/I for MVS & VM          EXAMPLE:  PROC  OPTIONS(MAIN);          PAGE  6

```

Figure 113. Example of moving a value outside an array range

Figure 114 on page 295 shows sections of the dump generated by a call to PLIDUMP.

CEE3845I CEEDUMP Processing started.
PLIDUMP was called from statement number 6 at offset +000000D6 from ERR ON-unit with entry address 20900C58

Information for enclave EXAMPLE

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEKCMRA	+0000081C		CEEPLPKA	CEEKCMRA	D1908	Call
2	IBMRKDM	+000000C2		IBMREV10	IBMRKDM		Call
3	ERR ON-unit	+000000D6	6	EXAMPLE	EXAMPLE		Call
4	IBMRERPL	+0000065A		IBMRLIB1	IBMRERPL		Call
5	CEEEV010	+0000013A		IBMREV10	CEEEV010		Call
6	CEEHDSP	+000017D0		CEEPLPKA	CEEHDSP	D1908	Call
7	IBMRERRI	+0000045A		IBMRLIB1	IBMRERRI		Exception
8	LABL1: BEGIN	+000000BE	11	EXAMPLE	EXAMPLE		Call
9	EXAMPLE	+000000C8	8	EXAMPLE	EXAMPLE		Call
10	IBMRPMIA	+0000051E		IBMRLIB1	IBMRPMIA		Call
11	CEEEV010	+00000310		IBMREV10	CEEEV010		Call
12	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	20B45B88	209F0420	209F0420	+0000081C	20061214	CEL
2	00025670	20B1C0A0	20B1C0A0	+000000C2	*****	OS PL/I
3	20B45A88	20900C58	20900B70	+000001BE	*****	OS PL/I
4	20B45850	00019F50	00019F50	+0000065A	20061213	LIBRARY
5	20B457C8	20B02998	20B02998	+0000013A	20061213	LIBRARY
6	20B426A8	209BF068	209BF068	+000017D0	20061215	CEL
7	20B42500	0001B328	0001B328	+0000045A	20061213	LIBRARY
8	20B42430	20900D48	20900B70	+00000296	*****	OS PL/I
9	20B42330	20900B78	20900B70	+000000D0	*****	OS PL/I
10	20B42178	000201D0	000201D0	+0000051E	20061214	LIBRARY
11	20B420F0	20B02998	20B02998	+00000310	20061213	LIBRARY
12	20B42030	20980DB8	20980DB8	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for IBMRERRI (DSA address 20B42500)

CIB Address: 20B42FC8

Current Condition:

IBM0281S A prior condition was promoted to the ERROR condition.

Original Condition:

IBM0421S ONCODE=520 The SUBSCRIPTRANGE condition was raised.

Location:

Program Unit: IBMRERRI Entry: IBMRERRI Statement: Offset: +0000045A

Storage dump near condition, beginning at location: 0001B772

+000000 0001B772 505D0080 58A0C2B8 58F0A01C 4110D080 05EF9108 404F4710 B4709104 404F47E0 |&&....B..0.....j. |...j. |..|

Control Blocks for Active Routines:

:

DSA for CEEHDSP: 20B426A8

+000000	FLAGS....	0808	member...	CEE1	BKC.....	20B42500	FWC.....	20B457C8	R14.....	A09C083A
+000010	R15.....	A0B02998	R0.....	00000020	R1.....	2090B2E8	R2.....	20B42FC8	R3.....	20B42330
+000024	R4.....	209C3C94	R5.....	FFFFFFF20	R6.....	00000001	R7.....	00000007	R8.....	A09C0542
+000038	R9.....	20B446A6	R10.....	20B436A7	R11.....	A09BF068	R12.....	2090E9C0	reserved.	00025670
+00004C	NAB.....	20B457C8	PNAB.....	00000000	reserved.	00000000	20B4271C			
+000064	reserved.	00000000	reserved.	00000000	MODE.....	00000000	reserved.	00000000		
+000078	reserved.	00000000	reserved.	00000000						

DSA for IBMRERRI: 20B42500

+000000	FLAGS....	8800	member...	0000	BKC.....	20B42430	FWC.....	20B425C0	R14.....	8001B784
+000010	R15.....	A09D0B48	R0.....	0000000B	R1.....	20B42580	R2.....	0000000A	R3.....	20900EB0
+000024	R4.....	00025470	R5.....	000254C4	R6.....	20B42330	R7.....	20B42330	R8.....	00000028
+000038	R9.....	00000008	R10.....	A09104D0	R11.....	0001B328	R12.....	2090E9C0	reserved.	00025290
+00004C	NAB.....	20B425C0	PNAB.....	008FF4E8	reserved.	2090D658	20AF22BC			
+000064	reserved.	2090E9C0	reserved.	20B420F0	MODE.....	20B2FA47	reserved.	A09104D0		
+000078	reserved.	00000000	reserved.	20B42838						

Figure 114. Sections of the Language Environment dump (Part 1 of 2)

```

CIB for IBMRERRI: 20B42FC8
+000000 20B42FC8 C3C9C240 00000000 00000000 010C0004 00000000 00000000 00030119 59C9C2D4 |CIB .....IBM|
+000020 20B42FE8 00000000 20B430D8 000301A5 59C9C2D4 00000001 00000015 20B42330 A0B02998 |.....Q...v.IBM.....q|
+000040 20B43008 00000000 20B42500 8001B784 2090B6F0 0000000A 20B42430 00000000 00000000 |.....d...0.....|
+000060 20B43028 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 20B43048 - +00009F 20B43067 same as above |.....|
+0000A0 20B43068 00000000 00000000 00000000 00000000 06230000 00000FC6 00000001 00000000 |.....F.....|
+0000C0 20B43088 00000000 00000000 20B42430 20B42500 0001B782 00000000 00000000 00000001 |.....b.....|
+0000E0 20B430A8 20B42330 0000000A 00000064 00000000 FFFFFFFC 00000000 00000000 00000000 |.....|
+000100 20B430C8 00000000 2090B908 00000000 00000000 E9D4C3C8 02000001 0000000B 20B42580 |.....ZMCH.....|
Dynamic save area (IBMRERRI): 20B42500
+000000 20B42500 88000000 20B42430 20B425C0 8001B784 A09D0B48 0000000B 20B42580 0000000A |h.....d.....|
+000020 20B42520 20900E80 00025470 000254C4 20B42330 20B42330 00000028 00000008 A09104D0 |.....D.....j..|
+000040 20B42540 0001B328 2090E9C0 00025290 20B425C0 008FF4E8 2090D658 00000000 20AF22BC |.....Z.....4Y..0.....|
+000060 20B42560 A0997C20 2090E9C0 20B420F0 20B2FA47 A09104D0 2090E9C0 00000000 20B42838 |.r@...Z...0....j...Z.....|
+000080 20B42580 000254C4 00000000 00000000 00000000 00000000 00000000 00000000 |...D.....|
+0000A0 20B425A0 16000000 20909DB0 00000000 80010000 00000000 209D4520 20B2F7B4 2090E200 |.....7...S.....|
DSA for LAB1: BEGIN: 20B42430
+000000 FLAGS.... 8425 member... 0000 BKC..... 20B42330 FWC..... 20B42500 R14..... A0900E08
+000010 R15..... 0001B328 R0..... 0000000B R1..... 20900E80 R2..... A0900DBE R3..... 20900E40
+000024 R4..... 00000001 R5..... 20B42330 R6..... 20B42330 R7..... 20B42330 R8..... 00000028
+000038 R9..... 00000008 R10..... 20B420B0 R11..... 2090102C R12..... 2090E9C0 reserved. 00025290
+00004C NAB..... 20B42500 PNAB..... 20B42500 reserved. 91A091A0 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE.... 00000000 reserved. 00000000
+000078 reserved. 00000000 reserved. 20B42030
Dynamic save area (LAB1: BEGIN): 20B42430
+000000 20B42430 84250000 20B42330 20B42500 A0900E08 0001B328 0000000B 20900E80 A0900DBE |d.....|
+000020 20B42450 20900E40 00000001 20B42330 20B42330 20B42330 00000028 00000008 20B420B0 |.....|
+000040 20B42470 2090102C 2090E9C0 00025290 20B42500 20B42500 91A091A0 20B42330 00000000 |.....Z.....j.j.....|
+000060 20B42490 00000000 00000000 00000000 00000000 00000000 00000200 00000000 20B42030 |.....|
+000080 20B424B0 20B42530 A0B02CEC 20B2EA48 00000800 20B420F0 20B420F0 A0B02998 20901340 |.....0...0...q...|
+0000A0 20B424D0 2090E880 0000000A 00000000 00000000 20AF2CD6 00000027 A0AF0CD8 2090E9C0 |.Y.....0.....Q..Z.....|
+0000C0 20B424F0 20B42618 00000014 00000000 88000000 20B42430 20B425C0 8001B784 |.....h.....d.....|
DSA for EXAMPLE: 20B42330
+000000 FLAGS.... C025 member... 0000 BKC..... 20B42178 FWC..... 00000000 R14..... A0900C42
+000010 R15..... 20900D48 R0..... 20B42430 R1..... 20B42330 R2..... A0900C30 R3..... 20900E40
+000024 R4..... 00000001 R5..... 20B42330 R6..... 20B42400 R7..... 00000005 R8..... 20900EF8
+000038 R9..... 00000008 R10..... 20B420B0 R11..... 2090102C R12..... 2090E9C0 reserved. 00025290
+00004C NAB..... 20B42430 PNAB..... 20B42430 reserved. 91E091A0 20900EF8
+000064 reserved. 00000000 reserved. 00000000 MODE.... 00000000 reserved. 20B423E8
+000078 reserved. 00000000 reserved. 00000000
Dynamic save area (EXAMPLE): 20B42330
+000000 20B42330 C0250000 20B42178 00000000 A0900C42 20900D48 20B42430 20B42330 A0900C30 |.....|
+000020 20B42350 20900E40 00000001 20B42330 20B42400 00000005 20900EF8 00000008 20B420B0 |.....8.....|
+000040 20B42370 2090102C 2090E9C0 00025290 20B42430 20B42430 91E091A0 00000000 20900EF8 |.....Z.....j.j.....8|
+000060 20B42390 00000000 00000000 00000000 00000000 20B423E8 00000200 00000000 00000000 |.....y.....|
+000080 20B423B0 20B42124 20B42128 20B4212C 20B42130 20B42138 20B42134 20B4213C 00000000 |.....|
+0000A0 20B423D0 00000000 00000000 00000000 00000000 00000000 00000000 0C010000 00000000 |.....|
+0000C0 20B423F0 20B42400 20900E94 0000000B 00000014 00000002 00000002 00000002 00000002 |.....m.....|
+0000E0 20B42410 00000002 00000002 00000002 00000002 00000002 00000002 00000000 00000000 |.....|

```

Figure 114. Sections of the Language Environment dump (Part 2 of 2)

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 113 on page 294. 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 113 on page 294. 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 for the entry EXAMPLE in Figure 114 on page 295.

The block located at this offset contains the value that exceeded the array range, X'B' or 11.

Calling a nonexistent subroutine

Figure 115 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.

```

5688-235 IBM PL/I for MVS & VM          Ver 1 Rel 1 Mod 1          27 FEB 07  11:45:18  PAGE  1
OPTIONS SPECIFIED
*PROCESS  GOSTMT LIST S STG TEST MAP NOOPTIONS;
5688-235 IBM PL/I for MVS & VM          EXAMPLE1: PROC  OPTIONS(MAIN);          PAGE  2
          SOURCE LISTING
STMT
  1  EXAMPLE1: PROC  OPTIONS(MAIN);
  2      DCL Prog01 entry external;
  3      On error
  4          Begin;
  5          On error system;
  6          Call plidump('tbnfs','Plidump called from error On-unit');
  7      End;
  8      Call prog01;          /* Call external program PROG01 */
  9  End Example1;
5688-235 IBM PL/I for MVS & VM          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
5688-235 IBM PL/I for MVS & VM          EXAMPLE1: PROC  OPTIONS(MAIN);          PAGE  4
          STATIC INTERNAL STORAGE MAP

```

Figure 115. Example of calling a nonexistent subroutine

Figure 116 on page 298 shows the traceback and condition information from the dump.

CEE3845I CEEDUMP Processing started.
 PLIDUMP was called from statement number 5 at offset +000000D6 from ERR ON-unit with entry address 20900DF4

Information for enclave EXAMPLE1

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEKMKRA	+0000081C		CEEPLPKA	CEEKMKRA	D1908	Call
2	IBMRKDM	+000000C2		IBMREV10	IBMRKDM		Call
3	ERR ON-unit	+000000D6	5	EXAMPLE1	EXAMPLE1		Call
4	IBMRERPL	+0000065A		IBMLIB1	IBMRERPL		Call
5	CEEEV010	+0000013A		IBMREV10	CEEEV010		Call
6	CEEHDSP	+000017D0		CEEPLPKA	CEEHDSP	D1908	Call
7	EXAMPLE1	-20900D2C		EXAMPLE1	EXAMPLE1		Exception
8	IBMRPMIA	+0000051E		IBMLIB1	IBMRPMIA		Call
9	CEEEV010	+00000310		IBMREV10	CEEEV010		Call
10	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	20B458D0	209F0420	209F0420	+0000081C	20061214	CEL
2	00025670	20B1C0A0	20B1C0A0	+000000C2	*****	OS PL/I
3	20B457D0	20900DF4	20900D20	+000001AA	*****	OS PL/I
4	20B45598	00019F50	00019F50	+0000065A	20061213	LIBRARY
5	20B45510	20B02998	20B02998	+0000013A	20061213	LIBRARY
6	20B423F0	209BF068	209BF068	+000017D0	20061215	CEL
7	20B42330	20900D2C	20900D20	-20900D20	*****	OS PL/I
8	20B42178	000201D0	000201D0	+0000051E	20061214	LIBRARY
9	20B420F0	20B02998	20B02998	+00000310	20061213	LIBRARY
10	20B42030	2098DD88	2098DD88	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for EXAMPLE1 (DSA address 20B42330)

CIB Address: 20B42D10

Current Condition:

CEE3201S The system detected an operation exception (System Completion Code=0C1).

Location:

Program Unit: EXAMPLE1 Entry: EXAMPLE1 Statement: Offset: -20900D2C

Possible Bad Branch: Statement: 7 Offset: +000000C0

Machine State:

ILC..... 0002 Interruption Code..... 0001

PSW..... 078D0E00 80000002

GPR0.... 20B423F0 GPR1.... 00000000 GPR2.... A0900D4 GPR3.... 20900EE0

GPR4.... 00000001 GPR5.... 00000000 GPR6.... 20B423E8 GPR7.... 00000005

GPR8.... 20900F50 GPR9.... 00000008 GPR10... 20B420B0 GPR11... 2090100C

GPR12... 2090E9C0 GPR13... 20B42330 GPR14... A0900DE2 GPR15... 00000000

Storage dump near condition, beginning at location: 00000000

+000000 00000000 Inaccessible storage.

GPREG STORAGE:

Storage around GPR0 (20B423F0)

```
-0020 20B423D0 00000000 00000000 00000000 00000000 00000000 00000000 0C010000 00000000 |.....|
+0000 20B423F0 0808CEE1 20B42330 20B45510 A09C083A A0B02998 00000020 2090B2E8 20B42D10 |.....q.....Y....|
+0020 20B42410 20B42330 209C3C94 FFFFFFF2 00000001 00000007 A09C0542 20B443EE 20B433EF |.....m.....|
```

Storage around GPR1 (00000000)

```
+0000 00000000 Inaccessible storage.
+0020 00000020 Inaccessible storage.
+0040 00000040 Inaccessible storage.
```

Storage around GPR2 (20900DD4)

```
-0020 20900DB4 92C01000 92251001 92021076 41D10000 41803070 5080D05C D203D054 30300520 |k...k...k...J.....&.*K.....|
+0000 20900DD4 92CD00B8 1B111B55 58F03040 05EF180D 58D0D004 58E0D00C 982CD01C 051E0707 |k......0.....q.....|
+0020 20900DF4 90ECD00C 47F0F02C 20900FC4 00000100 20900EE0 20900F78 20900D58 10000000 |.....00.....D.....|
```

Storage around GPR3 (20900EE0)

```
-0020 20900EC0 1B554110 303458F0 303C05EF 180D5800 D00458E0 D00C982C D01C051E 00000000 |.....0.....q.....|
+0000 20900EE0 E000011C 20900D2C 20900DD4 20900DF4 20900E80 20900E80 20900E80 20900E80 |.....M...4.....|
+0020 20900F00 00000000 00050000 00000000 00210000 91E091E0 20B45898 A0B458C4 20901368 |.....j...j...q...D....|
```

Storage around GPR4 (00000001)

```
-0001 00000000 Inaccessible storage.
+001F 00000020 Inaccessible storage.
+003F 00000040 Inaccessible storage.
```

Figure 116. Sections of the Language Environment dump (Part 1 of 2)

```

Storage around GPR5 (00000000)
+0000 00000000 Inaccessible storage.
+0020 00000020 Inaccessible storage.
+0040 00000040 Inaccessible storage.
Storage around GPR6 (20B423E8)
-0020 20B423C8 20B4213C 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 20B423E8 0C010000 00000000 0808CEE1 20B42330 20B45510 A09C083A A0B02998 00000020 |.....q....|
+0020 20B42408 2090B2E8 20B42D10 20B42330 209C3C94 FFFFFFF0 00000001 00000007 A09C0542 |...Y.....m.....|
Storage around GPR7 (00000005)
-0005 00000000 Inaccessible storage.
+001B 00000020 Inaccessible storage.
+003B 00000040 Inaccessible storage.
Storage around GPR8 (20900F50)
-0020 20900F30 40838193 93858440 86999694 40859999 969940D6 9560A495 89A30000 00000000 | called from error 0n-unit.....|
+0000 20900F50 0C160000 20900DF4 0C960000 00000000 20901038 00000000 00000000 20901018 |.....4.o.....|
+0020 20900F70 00000000 20900F60 00000000 20900F6C 20900F20 00000001 00000000 00000000 |.....-%.....|
Storage around GPR9 (00000008)
-0008 00000000 Inaccessible storage.
+0018 00000020 Inaccessible storage.
+0038 00000040 Inaccessible storage.
Storage around GPR10(20B420B0)
-0020 20B42090 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 20B420B0 2098E080 2090100C 20B420E0 2090D790 20B420D0 20B420D8 20B420D4 00000000 |.q.....P.....Q...M....|
+0020 20B420D0 2090D778 00000000 00000000 00000000 00000001 00000000 00000000 00000000 |..P.....|
Storage around GPR11(2090100C)
-0020 20900FEC C5C240F0 F74040F1 F17AF4F5 7AF1F840 80000117 20900CA0 00000000 01000001 |EB 07 11:45:18 .....|
+0000 2090100C 20900D2C 20901340 00000000 1D020001 20901030 00000000 00000000 0006D7D9 |.....PR|
+0020 2090102C D6C7F0F1 B4000A00 00000000 1D020000 20901054 20900D2C 00000000 0008C5E7 |0G01.....EX|
Storage around GPR12(2090E9C0)
-0020 2090E9A0 00000000 00000000 C3C5C5C3 C1C14040 00000000 E9E3C3C1 000058C0 D0640CCC |.....CEECAA .....ZTCA.....|
+0000 2090E9C0 00000000 00025658 20B42018 20B62018 00000000 2090E9D0 00000000 000251B0 |.....Z.....|
+0020 2090E9E0 00000000 00000000 00025030 00000000 00025240 00000000 00025208 00000000 |.....&.....|
Storage around GPR13(20B42330)
-0020 20B42310 00000000 00000000 00000000 0000000E 0E000000 00000000 00000000 00000000 |.....|
+0000 20B42330 C0250000 20B42178 00000000 80000002 00000000 20B423F0 00000000 A0900DD4 |.....0.....M|
+0020 20B42350 20900EE0 00000001 00000000 20B423E8 00000005 20900F50 00000008 20B420B0 |.....Y.....&.....|
Storage around GPR14(20900DE2)
-0020 20900DC2 00004180 30705080 D05CD203 D0543030 0520920C D0B81B11 1B5558F0 304005EF |.....&.*K.....k.....0. ..|
+0000 20900DE2 180D58D0 D00458E0 D00C982C D01C051E 070790EC D00C47F0 F02C2090 0FC40000 |.....q.....00....D..|
+0020 20900E02 01002090 0EE02090 0F782090 0D581000 00000002 02000000 00000000 00005830 |.....|
Storage around GPR15(00000000)
+0000 00000000 Inaccessible storage.
+0020 00000020 Inaccessible storage.
+0040 00000040 Inaccessible storage.
:

```

Figure 116. 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. The Location section indicates that the exception occurred at offset X'-20900D2C' within entry EXAMPLE1 and that there might have been a bad branch from offset X'+000000C0' statement 7 within entry EXAMPLE1 .

2. Locate statement 7 in the routine (Figure 115 on page 297). 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 117 on page 300 demonstrates a divide-by-zero error. In this example, the main PL/I for MVS & VM routine passed bad data to a PL/I for MVS & VM

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.

```

5688-235 IBM PL/I for MVS & VM          Ver 1 Rel 1 Mod 1          27 FEB 07  13:57:59  PAGE  1
OPTIONS SPECIFIED
*PROCESS  GOSTMT LIST S STG TEST MAP NOOPTIONS;
5688-235 IBM PL/I for MVS & VM          SAMPLE: PROC  OPTIONS(MAIN) ;          PAGE  2
SOURCE LISTING
STMT
  1  SAMPLE: PROC  OPTIONS(MAIN) ;
  2      On error
      begin;
  3      On error system;          /* prevent nested error conditions */
  4      Call PLIDUMP('TBC','PLIDUMP called from error ON-unit');
  5      Put Data;          /* Display variables */
  6      End;

  7  DECLARE
      A_number   Fixed Bin(31),
      My_Name    Char(13),
      An_Array(3) Fixed Bin(31) init(1,3,5);

  8      Put skip list('Sample Starting');
  9      A_number = 0;
 10      My_Name = 'Tery Gillaspy';

 11      Call Sub1(a_number, my_name, an_array);

 12  SUB1: PROC(divisor, name1, Array1);
 13      Declare
          Divisor   Fixed Bin(31),
          Name1     Char(13),
          Array1(3) Fixed Bin(31);
 14      Put skip list('Sub1 Starting');
 15      Array1(1) = Array1(2) / Divisor;
 16      Put skip list('Sub1 Ending');
 17      End SUB1;

 18      Put skip list('Sample Ending');

 19  End;
5688-235 IBM PL/I for MVS & VM          SAMPLE: PROC  OPTIONS(MAIN) ;          PAGE  3
STORAGE REQUIREMENTS
BLOCK, SECTION OR STATEMENT  TYPE          LENGTH  (HEX)  DSA SIZE  (HEX)
*SAMPLE1                     PROGRAM CSECT  1060    424
*SAMPLE2                     STATIC CSECT   860     35C
SAMPLE                       PROCEDURE BLOCK  428    1AC      304     130
BLOCK 2      STMT 2          ON UNIT      298    12A      296     128
SUB1                       PROCEDURE BLOCK  332    14C      256     100
5688-235 IBM PL/I for MVS & VM          SAMPLE: PROC  OPTIONS(MAIN) ;          PAGE  4
STATIC INTERNAL STORAGE MAP          0000F0  80000000          A..LOCATOR

```

Figure 117. PL/I for MVS & VM 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 118 shows this output.

```

1Sample Starting
Sub1 Starting          A_NUMBER=          0 MY_NAME='Tery Gillaspy' AN_ARRAY(1)= 1
AN_ARRAY(2)=          3          AN_ARRAY(3)=          5;

```

Figure 118. Variables from routine SAMPLE

The routine in Figure 117 on page 300 was compiled with the LIST compiler option, which generated the object code listing shown in Figure 119.

```
* STATEMENT NUMBER 15
0003A2 58 B0 D 0C8          L    11,200(0,13)
0003A6 58 40 B 004          L    4,4(0,11)
0003AA 58 90 3 0B4          L    9,180(0,3)
0003AE 5C 80 4 004          M    8,4(0,4)
0003B2 58 70 3 0D4          L    7,212(0,3)
0003B6 5C 60 4 004          M    6,4(0,4)
0003BA 58 80 D 0C0          L    8,192(0,13)
0003BE 58 60 B 000          L    6,0(0,11)
0003C2 5F 60 4 000          SL   6,0(0,4)
0003C6 58 E7 6 000          L    14,VO..ARRAY1(7)
0003CA 8E E0 0 020          SRDA 14,32
0003CE 5D E0 8 000          D    14,DIVISOR
0003D2 50 F9 6 000          ST   15,VO..ARRAY1(9)
```

Figure 119. Object code listing from example PL/I for MVS & VM routine

Figure 120 on page 302 shows the Language Environment dump for routine SAMPLE.

CEE3845I CEEDUMP Processing started.
PLIDUMP was called from statement number 4 at offset +000000D6 from ERR ON-unit with entry address 2090022C

Information for enclave SAMPLE

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEKCMRA	+0000081C		CEEPLPKA	CEEKCMRA	D1908	Call
2	IBMRKDM	+000000C2		IBMRV10	IBMRKDM		Call
3	ERR ON-unit	+000000D6	4	SAMPLE	SAMPLE		Call
4	IBMRERPL	+0000065A		IBMLIB1	IBMRERPL		Call
5	CEEEV010	+0000013A		IBMRV10	CEEEV010		Call
6	CEEHDSP	+000017D0		CEEPLPKA	CEEHDSP	D1908	Call
7	SUB1	+000000EE	15	SAMPLE	SAMPLE		Exception
8	SAMPLE	+00000154	11	SAMPLE	SAMPLE		Call
9	IBMRPMIA	+0000051E		IBMLIB1	IBMRPMIA		Call
10	CEEEV010	+00000310		IBMRV10	CEEEV010		Call
11	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	20B45A68	209F0420	209F0420	+0000081C	20061214	CEL
2	00025670	20B1C0A0	20B1C0A0	+000000C2	*****	OS PL/I
3	20B45940	2090022C	20900080	+00000282	*****	OS PL/I
4	20B45708	00019F50	00019F50	+0000065A	20061213	LIBRARY
5	20B45680	20B02998	20B02998	+0000013A	20061213	LIBRARY
6	20B42560	209BF068	209BF068	+000017D0	20061215	CEL
7	20B42460	20900360	20900080	+000003CE	*****	OS PL/I
8	20B42330	20900088	20900080	+0000015C	*****	OS PL/I
9	20B42178	000201D0	000201D0	+0000051E	20061214	LIBRARY
10	20B420F0	20B02998	20B02998	+00000310	20061213	LIBRARY
11	20B42030	2098DDB8	2098DDB8	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for SAMPLE (DSA address 20B42460)

CIB Address: 20B42E80

Current Condition:

IBM0281S A prior condition was promoted to the ERROR condition.

Original Condition:

CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

Location:

Program Unit: SAMPLE Entry: SUB1 Statement: 15 Offset: +000000EE

Machine State:

ILC..... 0004 Interruption Code..... 0009

PSW..... 078D2E00 A0900452

GPR0..... 20B42560 GPR1..... 20B42538 GPR2..... A09003E4 GPR3..... 209004A8

GPR4..... 2090056C GPR5..... 20B42330 GPR6..... 20B42404 GPR7..... 00000008

GPR8..... 20B42400 GPR9..... 00000004 GPR10.... 20B420B0 GPR11.... 20B423F8

GPR12.... 2090E9C0 GPR13.... 20B42460 GPR14.... 00000000 GPR15.... 00000003

Storage dump near condition, beginning at location: 2090043E

+000000 2090043E 5860B000 5F604000 58E76000 8EE00020 5DE08000 50F96000 41E0D0D8 50E0312C |.-..^- ..X-.....)&9-....Q&...|

GPREG STORAGE:

Storage around GPR0 (20B42560)

-0020 20B42540 20909DB0 A090055C 00409D00 00028004 008FF4E8 2090D658 00010000 00025470 |.....*.4Y..0.....|

+0000 20B42560 0808CEE1 20B42460 20B45680 A09C083A A0B02998 00000020 2090B2E8 20B42E80 |.....-.....q.....Y....|

+0020 20B42580 20B42330 209C3C94 FFFFFFF2 00000001 00000007 A09C0542 20B4455E 20B4355F |.....m.....;.....^|

Figure 120. Language Environment dump from example PL/I for MVS & VM routine (Part 1 of 3)

```

Control Blocks for Active Routines:
:
DSA for ERR ON-unit: 20B45940
+000000 FLAGS.... CC25      member... 4040      BKC..... 20B45708  FWC..... 40404040  R14..... A0900304
+000010 R15..... A0B1C0A0  R0..... 20B45A68  R1..... 209005A8  R2..... A09002B8  R3..... 209004A8
+000024 R4..... 00000064  R5..... 00000000  R6..... 20B459F8  R7..... 20B457F0  R8..... 20B45A30
+000038 R9..... 20B45A60  R10..... 00000000  R11..... 0001AF4F  R12..... D6D940C7  reserved. 00025670
+00004C NAB..... 20B45A68  PNAB..... 20B45A68  reserved. 91E091E0 20900650
+000064 reserved. 40404040  reserved. 40404040  MODE..... 40404040  reserved. 20B459F8
+000078 reserved. 40404040  reserved. 40404040
Dynamic save area (ERR ON-unit): 20B45940
+000000 20B45940  CC254040 20B45708 40404040 A0900304 A0B1C0A0 20B45A68 209005A8 A09002B8 |.. .. .!...y...
+000020 20B45960 209004A8 00000064 00000000 20B459F8 20B457F0 20B45A30 20B45A60 00000000 |.y.....8..0..!...!...
+000040 20B45980 0001AF4F D6D940C7 00025670 20B45A68 20B45A68 91E091E0 20B42330 20900650 |..|OR G.....!...!..j.j.....&
+000060 20B459A0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 |..8 .
+000080 20B459C0 20B42E80 00025470 A0B1120A A0B11190 20B42560 20B457D8 00025470 A0B11190 |.....Q.....
+0000A0 20B459E0 209C3C94 20B42E80 00025470 20B42E80 2090E880 00016038 0C010027 20B121F8 |..m.....Y.....
+0000C0 20B45A00 2090E9C0 40404040 40404040 40404040 40404040 40404040 40404040 |..Z.
+0000E0 20B45A20 40404040 40404040 40404040 40404040 E3C2C340 20B45A30 00030000 D7D3C9C4 |.....TBC ..!.....PLID
+000100 20B45A40 E4D4D740 83819393 85844086 99969440 85999996 9940D6D5 60A49589 A32C0000 |UMP called from error ON-unit...
+000120 20B45A60 20B45A3C 00210000 0000F3C3 00025670 20B45F60 A09F0C3E A09E5B58 00000000 |..!.....3C.....^.....
Static for procedure SAMPLE Timestamp: 27 FEB 07 13:57:59
Starting from: 209004A8
+000000 209004A8 E0000354 20900088 20900130 20900166 2090022C 209002B8 20900360 209003E4 |.....h.....-...U
+000020 209004C8 209003E4 209003E4 209003E4 20900C78 20900C60 20900C48 20900C30 20900C18 |..U..U..U.....-.....
+000040 209004E8 20900C00 20901BF8 20901BE0 20900BE8 20901BC8 20901BB0 20900BD0 20900BB8 |.....8.....Y..H.....
+000060 20900508 B4000A00 20000002 1F800000 00000000 209005DC 000F0000 00000000 000D0000 |.....*.....0..8.....*
+000080 20900528 00000000 2090056C 209005F8 000D0000 00000000 00030000 00000000 00210000 |.....%.8.....
+0000A0 20900548 20900629 000D0000 20900636 000B0000 91E091E0 00000001 00000003 00000005 |.....j.....
+0000C0 20900568 00000000 00000004 00000004 00000003 00000001 00000002 20900838 20900838 |.....*.....
+0000E0 20900588 20B42438 A090055C 20B42400 20B423F0 A0B423F8 20900838 00000000 A090055C |.....*.....0..8.....*
+000100 209005A8 20B45A34 A0B45A60 20900BA0 20900838 80000000 00000000 A0900658 20900838 |..!...!.....
+000120 209005C8 20B42538 A090055C 20900838 00000000 A090055C E2819497 938540E2 A38199A3 |.....*.....*Sample Start
+000140 209005E8 899587E3 8599A840 C7899393 81A297A8 E2819497 938540C5 95848995 87E3C2C3 |ingTery GillaspySample EndingTBC
+000160 20900608 D7D3C9C4 E4D4D740 83819393 85844086 99969440 85999996 9940D6D5 60A49589 |PLIDUMP called from error ON-uni
+000180 20900628 A3E2A482 F140E2A3 8199A389 9587E2A4 82F140C5 95848995 87000000 00000000 |tSub1 StartingSub1 Ending.....
+0001A0 20900648 0C160000 2090022C 0C960000 00000000 2090066C 20900668 209006A4 00000000 |.....o.....%.h..u...
+0001C0 20900668 00000000 85000001 2090050E 00000000 00000000 0008C1D4 D5E4D4C2 C5D90000 |.....e....._A_NUMBER..
+0001E0 20900688 81000001 2090050C 0000000C 00000000 0007D4E8 60D5C1D4 C5000000 81000101 |a.....MY_NAME...a...
+000200 209006A8 2090050E 0000000C 00000000 0008C1D5 6DC1D9D9 C1E80000 20900818 00000000 |.....H.....AN_ARRAY.....
+000220 209006C8 00000000 2090066C 20900688 209006A4 20900750 20900858 00000000 209006C0 |.....%.h..u..&.....
+000240 209006E8 00000000 2090066C 20900704 20900720 20900738 00000000 209006CC A5000002 |.....v...
+000260 20900708 2090050E 0000000C 00000000 0007C4C9 E5C9E2D6 D9000000 A1000002 2090050C |.....DIVISOR.....
+000280 20900728 0000000C 00000000 0005D5C1 D4C5F100 A1000102 2090050E 0000000C 00000000 |..D.....NAME1.....H....
+0002A0 20900748 0006C1D9 D9C1E8F1 00020001 20900508 20900360 00000000 0004E2E4 C2F10000 |..ARRAY1.....SUB1...
+0002C0 20900768 00000001 20900088 000001A4 2090079C 00000001 00DF0002 00E30008 01210009 |.....h..u.....T.....
+0002E0 20900788 0129000A 012F000B 01570012 01950013 01A40002 2090022C 0000012A 209007C0 |.....n..u.....
+000300 209007A8 00000002 008D0003 00910004 00D90005 011B0006 012C000C 20900360 00000144 |.....j...R.....
+000320 209007C8 209007E4 0000000C 0085000E 00C3000F 00F70010 01350011 01340011 0E0E0E0E |..U.....e...C...7.....
+000340 209007E8 F2F740C6 C5C240F0 F74040F1 F37AF5F7 7AF5F940 80000117 20900000 00000000 |27 FEB 07 13:57:59
:
DSA for CEEHDS: 20B42560
+000000 FLAGS.... 0808      member... CEE1      BKC..... 20B42460  FWC..... 20B45680  R14..... A09C083A
+000010 R15..... A0B02998  R0..... 00000020  R1..... 2090B2E8  R2..... 20B42E80  R3..... 20B42330
+000024 R4..... 209C3C94  R5..... FFFFFFF20  R6..... 00000001  R7..... 00000007  R8..... A09C0542
+000038 R9..... 20B4455E  R10..... 20B4355F  R11..... 209BF068  R12..... 2090E9C0  reserved. 00025670
+00004C NAB..... 20B45680  PNAB..... 00000000  reserved. 20B424F0 20B424F0
+000064 reserved. A0AF2F14  reserved. 00010280  MODE..... 000272F0  reserved. 20B4269C
+000078 reserved. 20B42598  reserved. 20900838
DSA for SUB1: 20B42460
+000000 FLAGS.... 8025      member... 0000      BKC..... 20B42330  FWC..... 00000000  R14..... A0900452
+000010 R15..... 00000003  R0..... 20B42560  R1..... 20B42538  R2..... A09003E4  R3..... 209004A8
+000024 R4..... 2090056C  R5..... 20B42330  R6..... 20B42404  R7..... 00000008  R8..... 20B42400
+000038 R9..... 00000004  R10..... 20B420B0  R11..... 20B423F8  R12..... 2090E9C0  reserved. 00025290
+00004C NAB..... 20B42560  PNAB..... 20B42560  reserved. 91E091E0 00000800
+000064 reserved. 20B420F0  reserved. A0B02998  MODE..... 20900B78  reserved. 2090E880
+000078 reserved. 00000000  reserved. 00000000
CIB for SUB1: 20B42E80
+000000 20B42E80 C3C9C240 00000000 00000000 010C0004 00000000 00000000 00030119 59C9C2D4 |CIB .....IBM
+000020 20B42EA0 00000000 20B42F90 00030C89 59C3C5C5 00000001 00000005 20B42330 A0B02998 |.....i.CEE.....q
+000040 20B42EC0 00000000 20B42460 20900452 2090B6F0 0000000A 20B42460 00000000 00000000 |.....-.....0.....
+000060 20B42EE0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+000080 20B42F00 - +00009F 20B42F1F same as above |.....
+0000A0 20B42F20 00000000 00000000 00000000 00000000 44230000 940C9000 00000009 00000000 |.....m.....
+0000C0 20B42F40 00000000 00000000 20B42460 20B42460 2090044E 00000000 00000000 00000001 |.....-.....+.....
+0000E0 20B42F60 20B42330 0000000A 00000064 00000000 FFFFFFFC 00000000 00000000 00000000 |.....
+000100 20B42F80 00000000 2090B908 00000000 00000000 E9D4C3C8 02000001 20B42560 20B42538 |.....ZMCH.....

```

Figure 120. Language Environment dump from example PL/I for MVS & VM routine (Part 2 of 3)

```

Dynamic save area (SUB1): 20B42460
+000000 20B42460 80250000 20B42330 00000000 A0900452 00000003 20B42560 20B42538 A09003E4 |.....U
+000020 20B42480 209004A8 2090056C 20B42330 20B42404 00000008 20B42400 00000004 20B420B0 |...y...%......
+000040 20B424A0 20B423F8 2090E9C0 00025290 20B42560 20B42560 91E091E0 20B42330 00000800 |...8.Z.....-j.j.....
+000060 20B424C0 20B420F0 20B420F0 A0B02998 20900B78 2090E880 0000020A 00000000 00000000 |...0...0...q...Y.....
+000080 20B424E0 20AF2CD6 C9C2D4D9 D6D7C1C1 209005DC 000F0000 00027258 20B425B8 A0B2C18E |...0IBMROPAA.....A.
+0000A0 20B42500 A0AF24D0 000272F0 20B42570 000272F0 A0B2C0F0 20900838 00000001 2090D1A8 |.....0.....0...0.....Jy
+0000C0 20B42520 20B42400 20B423F0 20B423F8 00000001 20B42538 20B420B0 20900548 2090050C |.....0...8.....
+0000E0 20B42540 20909DB0 A090055C 00409D00 00028004 008FF4E8 2090D658 00010000 00025470 |.....*......4Y..0.....

DSA for SAMPLE: 20B42330
+000000 FLAGS.... C025      member... 0000      BKC..... 20B42178  FWC..... 00000000  R14..... A09001DE
+000010 R15..... 20900360  R0..... 20B42460  R1..... 20900590  R2..... A0900166  R3..... 209004A8
+000024 R4..... 00000005  R5..... 20B42330  R6..... 20B42408  R7..... 20B423F8  R8..... 00000001
+000038 R9..... 00000008  R10..... 20B420B0  R11..... 2090080C  R12..... 2090E9C0  reserved. 00025290
+00004C NAB..... 20B42460  PNAB..... 20B42460  reserved. 91E091E0 20900648
+000064 reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 20B423E8
+000078 reserved. 00000000  reserved. 00000000

Dynamic save area (SAMPLE): 20B42330
+000000 20B42330 C0250000 20B42178 00000000 A09001DE 20900360 20B42460 20900590 A0900166 |.....-.....
+000020 20B42350 209004A8 00000005 20B42330 20B42408 20B423F8 00000001 00000008 20B420B0 |...y.....8.....
+000040 20B42370 2090080C 2090E9C0 00025290 20B42460 20B42460 91E091E0 00000000 20900648 |.....Z.....-j.j.....
+000060 20B42390 00000000 00000000 00000000 00000000 20B423E8 00000200 00000000 00000000 |.....Y.....
+000080 20B423B0 20B42124 20B42128 20B4212C 20B42130 20B42138 20B42134 20B4213C 00000000 |.....
+0000A0 20B423D0 00000000 00000000 00000000 00000000 00000000 00000000 0C010000 00000000 |.....
+0000C0 20B423F0 20B42414 000D0000 20B42408 2090056C 00000000 00000000 00000001 00000003 |.....%.
+0000E0 20B42410 00000005 E38599A8 40C78993 9381A297 A8000000 00000000 00000000 00000000 |...Tery Gillaspy.....
+000100 20B42430 20B42438 00000000 20900518 2090050C 00000000 A090055C 00400000 00028004 |.....*.
+000120 20B42450 00000000 00000000 00010000 00025470 80250000 20B42330 00000000 A0900452 |.....

```

Figure 120. Language Environment dump from example PL/I for MVS & VM routine (Part 3 of 3)

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 for MVS & VM 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'3CE'.
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'3CE' in the object listing for this routine, shown in Figure 119 on page 301. 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'20900590'. This is the address of the parameter list, which is located in static storage.

7. Find the parameter list in the stack frame; the address of the first parameter is X'20B42400' and 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 Enterprise PL/I routines

This chapter contains information that can help you debug applications that contain one or more Enterprise PL/I routines. Following a discussion about potential errors in Enterprise PL/I routines, the first part of this chapter discusses how to use compiler-generated listings to obtain information about Enterprise PL/I routines, and how to use PLIDUMP to generate a Language Environment dump of an Enterprise PL/I routine. The last part of the chapter provides examples of Enterprise 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 Enterprise PL/I routines
- Using Enterprise PL/I compiler listings
- Generating a Language Environment dump of an Enterprise PL/I routine
- Finding Enterprise PL/I information in a dump
- Debugging example of Enterprise PL/I routines

Determining the source of errors in Enterprise PL/I routines

Most errors in Enterprise PL/I routines can be identified by the information provided in Enterprise 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 an Enterprise PL/I routine can be caused by:

- Logic errors in the source routine
- Invalid use of Enterprise 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 Enterprise PL/I

A misunderstanding of the language or a failure to provide the correct environment for using Enterprise PL/I can result in an apparent malfunction of an Enterprise 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. Meanwhile, you can try an alternative way to perform the operation that is causing the trouble. A bypass is often feasible, since the Enterprise 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 Enterprise 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 Enterprise 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 Enterprise PL/I compiler listings

The following sections explain how to generate listings that contain information about your routine. Enterprise 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 an Enterprise PL/I routine.

Note: Enterprise PL/I shares a common compiler back-end with C/C++. The Enterprise PL/I assembler listing will, consequently, have a similar form to those from the XL C/C++ compiler.

The compiler listings included below are from the Enterprise PL/I product.

Generating Enterprise 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 Enterprise PL/I routines.

Table 22. 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
Object code	Contents of the program control section in hexadecimal notation and translated into a pseudo-assembler format.	LIST

Table 22. Compiler-generated PL/I listings and their contents (continued)

Name	Contents	Compiler Option
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 Enterprise PL/I listings

Figure 121 shows the first two pages of an example Enterprise PL/I routine that was compiled with the LIST, MAP and SOURCE options.

```

5655-H31  IBM(R) Enterprise PL/I for z/OS      V3.R6.M0 (Built:20070119)
              Options Specified
Install:
Command: s
Line.File Process Statements
      1.0  *PROCESS SOURCE LIST MAP;
Install:

5655-H31  IBM(R) Enterprise PL/I for z/OS
Compiler Source
Line.File
      2.0
      3.0  EXAMPLE: PROC OPTIONS(MAIN);
      4.0  DCL EXTR ENTRY EXTERNAL;
      5.0  DCL A FIXED BIN(31);
      6.0  DCL B(2,2) FIXED BIN(31) STATIC EXTERNAL INIT((4)0);
      7.0  DCL C CHAR(20) STATIC INIT('SAMPLE CONSTANT');
      8.0  DCL D FIXED BIN(31) STATIC;
      9.0  DCL E FIXED BIN(31);
     10.0  FETCH EXTR;
     11.0  CALL EXTR(A,B,C,D,E);
     12.0  DISPLAY(C);
     13.0  END;

```

Figure 121. Enterprise PL/I routine compiled with LIST, MAP, and SOURCE

Figure 122 on page 312 shows the output generated by the LIST and MAP options for this routine, including the pseudo-assembly listing, the external symbol dictionary and reference, the storage offset listing and the static and automatic storage maps. The sections following this example describe the contents of each type of listing.

```

OFFSET OBJECT CODE      LINE# FILE#   P S E U D O   A S S E M B L Y   L I S T I N G

                                Timestamp and Version Information
000000 F2F0 F0F7                                =C'2007'           Compiled Year
000004 F0F2 F0F1                                =C'0201'           Compiled Date MMDD
000008 F1F5 F3F2 F5F0                                =C'153250'         Compiled Time HHMMSS
00000E F0F3 F0F6 F0F0                                =C'030600'         Compiler Version

000014 002C ****                                Service String     20070122

                                Timestamp and Version End

5655-H31 IBM(R) Enterprise PL/I for z/OS                : EXAMPLE

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                                000003 |           EXAMPLE DS  0D
000000 47F0 F024          000003 |           B          36(,r15)
000004 01C3C5C5          000003 |           CEE eyecatcher
000008 000000C8          000003 |           DSA size
00000C 00000180          000003 |           =A(PPA1-EXAMPLE)
000010 47F0 F001          000003 |           B          1(,r15)
000014 58F0 C31C          000003 |           L          r15,796(,r12)
000018 184E              000003 |           LR         r4,r14
00001A 05EF              000003 |           BALR       r14,r15
00001C 00000000          000003 |           =F'0'
000020 A7F4 000C          000003 |           J          ++24
000024 90E8 D00C          000003 |           STM       r14,r8,12(r13)
000028 58E0 D04C          000003 |           L          r14,76(,r13)
00002C 4100 E0C8          000003 |           LA         r0,200(,r14)
000030 5500 C314          000003 |           CL         r0,788(,r12)
000034 A724 FFF0          000003 |           JH         *-32
000038 58F0 C280          000003 |           L          r15,640(,r12)
00003C 90F0 E048          000003 |           STM       r15,r0,72(r14)
000040 9210 E000          000003 |           MVI       0(r14),16
000044 50D0 E004          000003 |           ST        r13,4(,r14)
000048 18DE              000003 |           LR         r13,r14
00004A C030 0000 008A      000003 |           LARL      r3,F'138'
000050                                000003 |           End of Prolog

000050 5860 3002          000000 |           L          r6,=A(EXAMPLE2)(,r3,2)
000054 C070 0000          0092 000000 |           LARL     r7,F'146'
00005A 4110 6008          000003 |           LA         r1,EXTR(,r6,8)
00005E 4100 0000          000003 |           LA         r0,0
000062 5000 1000          000003 |           ST        r0,_shadow3(,r1,0)
000066 4110 6008          000010 |           LA         r1,EXTR(,r6,8)
00006A 5800 1000          000010 |           L          r0,_shadow2(,r1,0)
00006E 1200              000010 |           LTR        r0,r0
000070 A774 0012          000010 |           JNE       @1L2
000074 4100 6018          000010 |           LA         r0,_Dsc_000002(,r6,24)
000078 4120 6008          000010 |           LA         r2,EXTR(,r6,8)
00007C 58F0 3006          000010 |           L          r15,=V(IBMQFRG)(,r3,6)
000080 4110 D098          000010 |           LA         r1,#MX_TEMP1(,r13,152)
000084 5020 D098          000010 |           ST        r2,#MX_TEMP1(,r13,152)
000088 1827              000010 |           LR         r2,r7
00008A 5020 D09C          000010 |           ST        r2,#MX_TEMP1(,r13,156)
00008E 5000 D0A0          000010 |           ST        r0,#MX_TEMP1(,r13,160)
000092 05EF              000010 |           BALR      r14,r15
000094                                000010 |           @1L2
000094 5800 300A          000011 |           DS        0H
000098 5000 D0C0          000011 |           L          r0,=A(B)(,r3,10)
00009C 4100 6040          000011 |           ST        r0,192(,r13)
0000A0 5000 D0C4          000011 |           LA         r0,_Dsc_000005(,r6,64)
0000A4 4100 6028          000011 |           ST        r0,196(,r13)
0000A8 5000 D0B8          000011 |           LA         r0,C(,r6,40)
0000AC 4110 6004          000011 |           ST        r0,184(,r13)
0000B0 5800 1000          000011 |           LA         r1,_Dsc_000003(,r6,4)
0000B4 5000 D0BC          000011 |           L          r0,_shadow1(,r1,0)
0000B8 4110 6008          000011 |           ST        r0,_temp1(,r13,188)
0000BC 5800 1000          000011 |           LA         r1,EXTR(,r6,8)
0000C0 1200              000011 |           L          r0,_shadow2(,r1,0)
                                000011 |           LTR        r0,r0

```

Figure 122. Compiler-generated listings from example Enterprise PL/I routine (Part 1 of 4)

OFFSET	OBJECT	CODE	LINE#	FILE#	PSEUDO	ASSEMBLY	LISTING
0000C2	A774	0012	000011		JNE	@1L3	
0000C6	4100	6018	000011		LA	r0,_Dsc_000002(,r6,24)	
0000CA	4120	6008	000011		LA	r2,EXTR(,r6,8)	
0000CE	58F0	3006	000011		L	r15,=V(IBMQFRG)(,r3,6)	
0000D2	4110	D098	000011		LA	r1,#MX_TEMP1(,r13,152)	
0000D6	5020	D098	000011		ST	r2,#MX_TEMP1(,r13,152)	
0000DA	1827		000011		LR	r2,r7	
0000DC	5020	D09C	000011		ST	r2,#MX_TEMP1(,r13,156)	
0000E0	5000	D0A0	000011		ST	r0,#MX_TEMP1(,r13,160)	
0000E4	05EF		000011		BALR	r14,r15	
0000E6			000011	@1L3	DS	0H	
0000E6	4110	6008	000011		LA	r1,EXTR(,r6,8)	
0000EA	58F0	1000	000011		L	r15,_shadow2(,r1,0)	
0000EE	4100	D0B4	000011		LA	r0,E(,r13,180)	
0000F2	1826		000011		LR	r2,r6	
0000F4	4140	D0B8	000011		LA	r4,_temp1(,r13,184)	
0000F8	4150	D0C0	000011		LA	r5,_temp2(,r13,192)	
0000FC	4180	D0B0	000011		LA	r8,A(,r13,176)	
000100	4110	D098	000011		LA	r1,#MX_TEMP1(,r13,152)	
000104	5080	D098	000011		ST	r8,#MX_TEMP1(,r13,152)	
000108	5050	D09C	000011		ST	r5,#MX_TEMP1(,r13,156)	
00010C	5040	D0A0	000011		ST	r4,#MX_TEMP1(,r13,160)	
000110	5020	D0A4	000011		ST	r2,#MX_TEMP1(,r13,164)	
000114	5000	D0A8	000011		ST	r0,#MX_TEMP1(,r13,168)	
000118	05EF		000011		BALR	r14,r15	
00011A	4120	6010	000012		LA	r2,_Dsc_000001(,r6,16)	
00011E	4100	6020	000012		LA	r0,_Dsc_000004(,r6,32)	
000122	4140	6028	000012		LA	r4,C(,r6,40)	
000126	58F0	300E	000012		L	r15,=V(IBMQJDSB)(,r3,14)	
00012A	4110	D098	000012		LA	r1,#MX_TEMP1(,r13,152)	
00012E	5040	D098	000012		ST	r4,#MX_TEMP1(,r13,152)	
000132	5000	D09C	000012		ST	r0,#MX_TEMP1(,r13,156)	
000136	4100	0000	000012		LA	r0,0	
00013A	5000	D0A0	000012		ST	r0,#MX_TEMP1(,r13,160)	
00013E	5020	D0A4	000012		ST	r2,#MX_TEMP1(,r13,164)	
000142	5000	D0A8	000012		ST	r0,#MX_TEMP1(,r13,168)	
000146	05EF		000012		BALR	r14,r15	
000148			000013	@1L1	DS	0H	
000148	58F0	3012	000013		L	r15,=V(IBMQEFSH)(,r3,18)	
00014C	05EF		000013		BALR	r14,r15	
00014E			000013	@1L4	DS	0H	
00014E					Start of Epilog		
00014E	58D0	D004	000013		L	r13,4(,r13)	
000152	58E0	D00C	000013		L	r14,12(,r13)	
000156	9828	D01C	000013		LM	r2,r8,28(r13)	
00015A	051E		000013		BALR	r1,r14	
00015C	0707		000013		NOPR	7	
00015E	0000						
000160					Start of Literals		
000160	00000000					=A(EXAMPLE2)	
000164	00000000					=V(IBMQFRG)	
000168	00000000					=A(B)	
00016C	00000000					=V(IBMQJDSB)	

Figure 122. Compiler-generated listings from example Enterprise PL/I routine (Part 2 of 4)

5655-H31 IBM(R) Enterprise PL/I for z/OS : EXAMPLE

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

000170 00000000 =V(IBMQEF5H)
000174 End of Literals

*** General purpose registers used: 111111110001111
*** Floating point registers used: 1111111100000000
*** Size of register spill area: 512(max) 0(used)
*** Size of dynamic storage: 200
*** Size of executable code: 350
*** CSECT Offset: 72 : 0x48

0001BC 0000 0000

Constant Area

000000 0004C5E7 E3D9 |..EXTR |
5655-H31 IBM(R) Enterprise PL/I for z/OS

OFFSET OBJECT CODE LINE# FILE# P S E U D O A S S E M B L Y L I S T I N G

PPA1: Entry Point Constants

000000 1CCEA166 =F'483303782' Flags
000004 000001C8 =A(PPA2-EXAMPLE)
000008 00000000 =F'0' No PPA3
00000C 00000000 =F'0' No EPD
000010 FFE00000 =F'-2097152' Register save mask
000014 00000000 =F'0' Member flags
000018 90 =AL1(144) Flags
000019 000000 =AL3(0) Callee's DSA use/8
00001C 0040 =H'64' Flags
00001E 0012 =H'18' Offset/2 to CDL
000020 00000000 =F'0' State variable location
000024 500000AF =F'1342177455' CDL function length/2
000028 FFFFFFFE80 =F'-384' CDL function EP offset
00002C 38280000 =F'942145536' CDL prolog
000030 400800A7 =F'1074266279' CDL epilog
000034 00000000 =F'0' CDL end
000038 0007 **** AL2(7),C'EXAMPLE'

PPA1 End

PPA2: Compile Unit Block

000000 0B00 3203 =F'184562179' Flags
000004 FFFF FDF0 =A(CEESTART-PPA2)
000008 0000 0000 =F'0' No PPA4
00000C FFFF FDF0 =A(TIMESTAMP-PPA2)
000010 0000 0000 =F'0' No primary
000014 0200 0000 =F'33554432' Flags

PPA2 End

5655-H31 IBM(R) Enterprise PL/I for z/OS

E X T E R N A L S Y M B O L D I C T I O N A R Y

NAME	TYPE	ID	ADDR	LENGTH	NAME	TYPE	ID	ADDR	LENGTH
EXAMPLE1	SD	1	000000	000228	EXAMPLE2	SD	2	000000	00005C
@EXAMPLE	SD	3	000000	000004	B	SD	4	000000	000010
EXAMPLE	LD	0	000048	000001	CEESG011	ER	5	000000	
IBMQFRG	ER	6	000000		IBMQJDSB	ER	7	000000	
IBMQEF5H	ER	8	000000		CEESTART	ER	9	000000	
CEEMAIN	SD	10	000000	00000C	IBMPINPL	ER	11	000000	
EXAMPLE	ER	12	000000						

5655-H31 IBM(R) Enterprise PL/I for z/OS

E X T E R N A L S Y M B O L C R O S S R E F E R E N C E

ORIGINAL NAME	EXTERNAL SYMBOL NAME
EXAMPLE1	EXAMPLE1
EXAMPLE2	EXAMPLE2
_EXAMPLE	@EXAMPLE
B	B
EXAMPLE	EXAMPLE
CEESG011	CEESG011
IBMQFRG	IBMQFRG
IBMQJDSB	IBMQJDSB
IBMQEF5H	IBMQEF5H
CEESTART	CEESTART
CEEMAIN	CEEMAIN
IBMPINPL	IBMPINPL

Figure 122. Compiler-generated listings from example Enterprise PL/I routine (Part 3 of 4)

```

5655-H31 IBM(R) Enterprise PL/I for z/OS

          * * * * * S T O R A G E   O F F S E T   L I S T I N G   * * * * *

IDENTIFIER      DEFINITION      ATTRIBUTES
<SEQNBR>-<FILE NO="":<FILE LINE="">

A              1-0:5            Class = automatic,           Location = 176 : 0xB0(r13),           Length = 4
B              1-0:6            Class = external definition, Location = CSECT B,                 Length = 16
C              1-0:7            Class = static,              Location = 40 : 0x28 + CSECT EXAMPLE2, Length = 20
D              1-0:8            Class = static,              Location = 0 : 0x0 + CSECT EXAMPLE2, Length = 4
E              1-0:9            Class = automatic,           Location = 180 : 0xB4(r13),          Length = 4
EXTR          1-0:4            Class = static,              Location = 8 : 0x8 + CSECT EXAMPLE2, Length = 8

          * * * * * E N D   O F   S T O R A G E   O F F S E T   L I S T I N G   * * * * *
5655-H31 IBM(R) Enterprise PL/I for z/OS

          * * * * * S T A T I C   M A P   * * * * *

OFFSET (HEX)   LENGTH (HEX)   NAME
              0                4      D
              4                4      _Dsc_000003
              8                8      EXTR
             10                8      _Dsc_000001
             18                8      _Dsc_000002
             20                8      _Dsc_000004
             28               14      C
             40               1C      _Dsc_000005

          * * * * * E N D   O F   S T A T I C   M A P   * * * * *
5655-H31 IBM(R) Enterprise PL/I for z/OS

          * * * * * A U T O M A T I C   M A P   * * * * *

Block name: EXAMPLE

OFFSET (HEX)   LENGTH (HEX)   NAME
              98               18      #MX_TEMP1
             B0                4      A
             B4                4      E
             B8                8      _temp1
             C0                8      _temp2

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

```

Figure 122. Compiler-generated listings from example Enterprise PL/I routine (Part 4 of 4)

Pseudo assembly listing

The pseudo assembly listing consists of the machine instructions and a translation of these instructions into a form that resembles assembler code.

This listing always starts with a small section of non-executable data that records the date and time when the object was produced as well as the version of the compiler used to produce the object. This section ends with a service string which in the listing is followed by the build date for the compiler back-end that generated this part of the listing (and this date may be different from the build date for the compiler front-end that generated the first pages of the listing).

The majority of the pseudo assembly listing consists of the object code arranged in columns that specify for each instructions:

- Its offset.
- the instruction in object code format.
- Its associated line number.
- Its associated file number if non-zero (i.e. if from an include file).

- the instruction in mnemonic format.

External symbol dictionary

The external symbol dictionary lists all the external symbols generated for this compilation. For each symbol, it also lists its linkage type and size (in hex).

External symbol cross reference

The external symbol dictionary cross reference shows for each external symbol the name that will be visible externally to the linker.

Storage offset listing

Each line of the storage offset listing contains the following information for each user variable:

- Its name.
- the number of the block in which it was declared.
- the number of the file in which it was declared.
- the number of the line in which it was declared.
- Its class (automatic, static, etc).
- Its location (as appropriate for its class).
- Its byte length in decimal.

This list is sorted by block number and then by name within each block.

Static map

Each line of the static storage map contains the following information for each internal static variable:

- Its hexadecimal offset.
- Its byte length in hex.
- Its name.

This list is sorted by the offset of the variables in static.

This list of variables may also include compiler-generated variables.

Automatic map

Each line of the automatic storage map contains the following information, grouped by named block, for each automatic variable in that block:

- Its hexadecimal offset.
- Its byte length in hex.
- Its name.

These lists are sorted by the offset of the variables in automatic for each block.

These lists of variables may also include compiler-generated variables.

Generating a Language Environment dump of an Enterprise PL/I routine

To generate a dump of an Enterprise 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

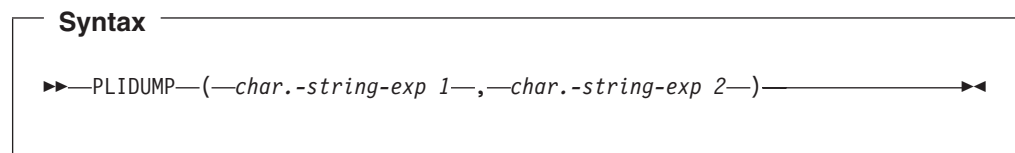
PLIDUMP calls intermediate Enterprise PL/I library routines, which convert most PLIDUMP options to CEE3DMP options. The following list contains PLIDUMP options and the corresponding CEE3DMP option, if applicable.

Some PLIDUMP options do not have corresponding CEE3DMP options, but continue to function as Enterprise PL/I default options. The list following the syntax diagram provides a description of those options.

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.



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 (Enterprise PL/I hexadecimal dump). Dumps the control blocks used in Language Environment and member language libraries. For Enterprise PL/I, this includes the DSA for every routine on the call chain and Enterprise PL/I "global" control blocks, such as Tasking Implementation Appendage (TIA), Task Communication Area (TCA), and the PL/I Tasking Control Block (PTCB). Enterprise 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 Enterprise 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 Enterprise 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 Enterprise PL/I routine.

Finding Enterprise PL/I information in a dump

The following sections discuss Enterprise 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 123 on page 320, for condition information about your routine and information about the statement number and address where the exception occurred.

CEE3845I CEEDUMP Processing started.
 PLIDUMP was called from statement number 9 at offset +000000D2 from _ON_Begin_7_Blk_2 with entry address 11200240

Information for enclave EXAMPLE

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	IBMPDUMP	+000002AE			IBMPEV11	PQ78306	Call
2	_ON_Begin_7_Blk_2	+000000D2	9	EXAMPLE	_Begin_12_Blk_3		Call
3	IBMPEONR	+000002A2			IBMPEV11	PQ76426	Call
4	IBMPEBOP	+000004DC			IBMPEV11	LE19BAS	Call
5	CEEEV011	+00000132			IBMPEV11		Call
6	CEEHDSP	+000017D0			CEEPLPKA	D1908	Call
7	IBMBERRI	+000000AA			IBMPEV11	LE19BAS	Exception
8	ERR_RAISE_COND						
		+00000090			IBMPEV11	LE19BAS	Call
9	IBMPERSU	+00000082			IBMPEV11	LE19BAS	Call
10	_Begin_12_Blk_3						
		+00000100	16	EXAMPLE	_Begin_12_Blk_3		Call
11	EXAMPLE	+000000B0	12	EXAMPLE	_Begin_12_Blk_3		Call
12	IBMPMINV	+000004DE			IBMPEV11	IBMPMINV	Call
13	CEEEV011	+00000202			IBMPEV11	CEEEV011	Call
14	CEEBBEXT	+000001B6			CEEPLPKA	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	11A3DE50	114A4E38	114A4E38	+000002AE	20061214	LIBRARY EBCDIC HFP
2	11A3DD70	11200240	112000D0	+00000242	20070131	ENT PL/I EBCDIC HFP
3	11A3DBD8	114A7B98	114A7B98	+000002A2	20061214	LIBRARY EBCDIC HFP
4	11A3DA08	114AF390	114AF390	+000004DC	20061214	LIBRARY EBCDIC HFP
5	11A3D978	114062E8	114062E8	+00000132	20061214	LIBRARY
6	11A3A858	112C3238	112C3238	+000017D0	20061215	CEL
7	11A3A6B8	114AAA18	114AAA18	+000000AA	20061214	LIBRARY EBCDIC HFP
8	11A3A618	114A9FA8	114A9FA8	+00000090	20061214	LIBRARY EBCDIC HFP
9	11A3A570	114AB120	114AB120	+00000082	20061214	LIBRARY EBCDIC HFP
10	11A3A4A8	112000D0	112000D0	+00000100	20070131	ENT PL/I EBCDIC HFP
11	11A3A3B8	11200340	112000D0	+00000320	20070131	ENT PL/I EBCDIC HFP
12	11A3A180	114DD990	114DD990	+000004DE	20061214	LIBRARY
13	11A3A0F0	114062E8	114062E8	+00000202	20061214	LIBRARY
14	11A3A030	11291208	11291208	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for (DSA address 11A3A6B8)

CIB Address: 11A3B178

Current Condition:

IBM0281S A prior condition was promoted to the ERROR condition.

Original Condition:

IBM0421S ONCODE=520 The SUBSCRIPTRANGE condition was raised.

Location:

Program Unit: Entry: IBMBERRI Statement: Offset: +000000AA

Storage dump near condition, beginning at location: 114AAA82

+000000 114AAA82 4110D098 5050D098 5040D09C 5040D0A0 05EF1F00 43002016 A7010080 A7840009 |...q&&.q& ..&x...xd..|

Figure 123. Traceback section of dump

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 124 on page 322 shows this section of the dump.

```

Control Blocks for Active Routines:
:
DSA for EXAMPLE: 11A3A3B8
+000000  FLAGS.... 10A3      member... A12C      BKC..... 11A3A180  FWC..... 11A3A138  R14..... 912003F2
+000010  R15..... 112000D0  R0..... 11A3A3B8  R1..... 11200240  R2..... 11211768  R3..... 1120037A
+000024  R4..... 11A3A0D8  R5..... 11A3A3B8  R6..... 11200800  R7..... 00000000  R8..... 11211648
+000038  R9..... 00000008  R10..... 11A3A0B0  R11..... 11200ACC  R12..... 112129B0  reserved. 00000000
+00004C  NAB..... 11A3A4A8  PNAB..... 00000000  reserved. 00000000 00000000
+000064  reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000
Dynamic save area (EXAMPLE): 11A3A3B8
+000000  11A3A3B8  10A3A12C  11A3A180  11A3A138  912003F2  112000D0  11A3A3B8  11200240  11211768  |.t...t...t..j..2....tt... ..|
+000020  11A3A3D8  1120037A  11A3A0D8  11A3A3B8  11200800  00000000  11211648  00000008  11A3A0B0  |....t.Q.tt.....t..|
+000040  11A3A3F8  11200ACC  112129B0  00000000  11A3A4A8  00000000  00000000  00000000  00000000  |.....tuy.....|
+000060  11A3A418  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  |.....|
+000080  11A3A438  00000000  00000000  00000000  00000000  00000000  00000000  00100000  00000000  |.....|
+0000A0  11A3A458  00180180  11A3A3B8  00000000  11200980  0B300000  11200240  11A3A3B8  00000000  |....tt.....tt....|
+0000C0  11A3A478  00000002  00000002  00000002  00000002  00000002  00000002  00000002  00000002  |.....|
+0000E0  11A3A498  00000002  00000002  0000000B  00000014  10000000  11A3A3B8  11A3A7D8  912001D2  |.....tt..txQj..K|
ENTERPRISE PL/I OPTIONS:
AFP, ARCH( 5), BACKREG(5), BIFPREC(15), CHECK(NOCONFORMANCE, NOSTORAGE), CMPAT(V2), CODEPAGE( 1140), COMMON,
NOCOMPACT, CSECT, CSECTCUT( 4) CURRENCY( $), NOBCS, DECIMAL( FOFLONASGN, NOFORCEDSIGN), DEFAULT( IBM, ASSIGNABLE,
NOINITFILL, NONCONNECTED, DESCRIPTOR, DESCLOCATOR, DUMMY(ALIGNED), ORDINAL(MIN), BYADDR, RETURNS(BYADDR),
LINKAGE(OPTLINK), NORETCODE, NOINLINE, ORDER, NOOVERLAP, NONRECURSIVE, ALIGNED, NULL370, EVENDEC, SHORT(HEXADEC),
EBCDIC, HEXADEC, NATIVE, NATIVEADDR, E(HEXADEC) ), DISPLAY(WTO), NODLLIMIT, EXTRN(FULL), NOGRAPHIC, NOINITAUTO,
NOINITBASED, NOINITCTL, NOINITSTATIC, NOINTERRUPT, LIMITS( EXTNAME( 7), FIXEDBIN( 31, 31), FIXEDDEC( 15, 15), NAME( 100
)), OPTIMIZE( 0), PREFIX( CONVERSION, FIXEDOVERFLOW, INVALIDDOP, OVERFLOW, PRECTYPE(ANS), NOSIZE, NOSTRINGRANGE,
NOSTRINGSIZE, NOSUBSCRIPTRANGE, UNDERFLOW, ZERODIVIDE), REDUCE, NORENT, RESEXP, RESPECT(), RULES(IBM), NOSTDSYS,
NOSCHEDULER, STRINGOFGGRAPHIC(GRAPHIC), SYSTEM(MVS), TEST(ALL, _SYM, HOOK, NOSEPARATE), TUNE( 5), USAGE( ROUND(IBM),
UNSPEC(IBM)), WIDECHAR(BIGENDIAN), WINDOW( 1950), WRITABLE, XINFO(NODEF, NOXML)
Static for procedure EXAMPLE      Timestamp: 2007.01.31 15:59:36 V03.R06.M00: 11200800
+000000  11200800  02020240  00000005  02020240  00000021  00000000  11200898  00000000  11200830  |... ..q.....|
+000020  11200820  00000000  11200898  00000000  11200830  00000000  11200838  00000000  00000000  |.....q.....|
+000040  11200840  31010001  00000000  00000000  00000000  00000010  00000000  00000000  00000000  |.....|
+000060  11200860  112008B0  00000000  00000000  00000000  00000004  00000004  0000000A  00000001  |.....|
+000080  11200880  0000001B  00000024  0000002D  00000032  00000000  00000000  112008D0  112008F0  |.....0|
+0000A0  112008A0  11200910  11200930  00000000  11200860  02002200  11200340  00000000  00000000  |.....|
+0000C0  112008C0  11200840  11200438  00090107  00000000  42002201  000000A2  00000000  00000000  |... ..s.....|
+0000E0  112008E0  30000000  11200444  00090105  00000000  48012401  D00000C0  11200870  00000000  |.....|
+000100  11200900  00001F80  1120044C  00010105  00000000  40002401  D00000E8  00000000  00000000  |.....<.....Y.....|
+000120  11200920  00001F80  11200454  00010101  00000000  40002401  D00000EC  00000000  00000000  |.....|
+000140  11200940  00001F80  11200458  00010109  00000000  00000003  00000010  00000002  0000002D  |.....|
+000160  11200960  00000024  0000000F  00000002  0000002D  00000032  0000000E  00000001  00000032  |.....|
+000180  11200980  D0000098  00100000  6E3BFEE0  00000000  11200898  11200A80  11200830  90010000  |...q...>.....q.....|
+0001A0  112009A0  11200340  00000000  00000000  00000004  11200880  00000000  112009C0  00000000  |... ..|
+0001C0  112009C0  D00000A8  00100000  6E3BFEE0  00000000  11200820  11200A80  11200828  83010000  |...y...>.....c...|
+0001E0  112009E0  11200240  11200980  11200A00  00000000  00000000  00000000  00000000  00000000  |... ..|
+000200  11200A00  D0000098  00000000  6E7BFEE0  00000000  11200810  11200A80  11200818  01000000  |...q...>#.....|
+000220  11200A20  112000D0  11200980  00000000  00000000  00000000  11200950  00000000  00000000  |.....&.....|
+000240  11200A40  00000000  00000000  00000000  00000000  00010007  C5E7C1D4  D7D3C500  010005D3  |.....EXAMPLE....L|
+000260  11200A60  C1C2D3F1  00010005  C1D9D9C1  E8000100  01C90001  0009C1D9  D9C1E86D  C5D5C400  |ABL1...ARRAY...I...ARRAY_END.|
+000280  11200A80  00000000  11200464  E1100FF2  F0F0F7F0  F1F3F1F1  F5F5F9F3  F6F0F0F0  0000003F  |.....20070131155936000....|
+0002A0  11200AA0  11200A40  11200860  11200838  00000001  11200850  00220000  00000000  11200980  |... ..&.....|
+0002C0  11200AC0  11200438  00000000  01000001  11200340  11200FC0  00000000  18AF47F0  A016C3C5  |.....0..CE|
:

```

Figure 124. 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.

Control blocks associated with the thread

This section of the dump, shown in Figure 125 on page 324, includes information about Enterprise PL/I fields of the CAA and other control block information.

Control Blocks Associated with the Thread:

```

CAA: 112139B0
+000000 112139B0 00000800 00000000 11A3B018 11A5B018 00000000 00000000 00000000 00000000 |.....t...v.....|
+000020 112139D0 00000000 00000000 11210A58 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 112139F0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000060 11213A10 00000000 00000000 00000000 00000000 00000000 80014608 00000000 00000000 |.....|
:
DUMMY DSA: 11214350
+000000 FLAGS.... 0000      member... 0000      BKC..... 00006F60 FWC..... 11A3B030 R14..... 9120132E
+000010 R15..... 91292208 R0..... 7D000008 R1..... 11212768 R2..... 00000000 R3..... 00000000
+000024 R4..... 00000000 R5..... 00000000 R6..... 00000000 R7..... 91215770 R8..... 11200FE0
+000038 R9..... 008FF7D0 R10..... 00000000 R11..... 9120125A R12..... 11213980 reserved. 00000000
+00004C NAB..... 11A3B030 PNAB..... 11A3B030 reserved. 00000000 00000000
+000064 reserved. 00000000 reserved. 00000000 MODE..... 00000000 reserved. 00000000
+000078 reserved. 00000000 reserved. 00000000
:
PL/I TCA Appendage: 11A5BCE8
+000000 11A5BCE8 00000000 00000000 11A5BEF8 0000047B 00000000 00000000 00000000 00000000 |.....v.8...#.....|
+000020 11A5BD08 00000000 00000000 11A5BE48 1154AAC8 00000000 00009600 00000000 00000000 |.....v.....H.....o.....|
+000040 11A5BD28 00000000 00000000 00000000 00000000 11A5C960 11A3EAC0 00004000 00000000 |.....vI-t.....|
+000060 11A5BD48 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000080 11A5BD68 114B38C8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |...H.....|
+0000A0 11A5BD88 00000000 00000000 00000000 00000000 00000000 00000000 00000001 00000000 |.....|
+0000C0 11A5BDA8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000E0 11A5BDC8 - +0000FF 11A5BDE7      same as above
+000100 11A5BDE8 00000000 00000000 00000000 11A5C380 00000002 00000000 00000000 00000000 |.....vC.....|
+000120 11A5BE08 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000140 11A5BE28 - +00015F 11A5BE47      same as above
+000160 11A5BE48 11A5CF80 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.v.....|
+000180 11A5BE68 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0001A0 11A5BE88 - +0001BF 11A5BEA7      same as above
PL/I OCA: 11A3EAC0
+000000 11A3EAC0 D6C3C100 11A5CA28 000000FF 00000000 00000000 00000000 00000000 00000000 |OCA..v.....|
+000020 11A3EAE0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 11A3EB00 - +0000BF 11A3EB7F      same as above
+0000C0 11A3EB80 00000000 00000000 40404040 40404040 40404096 002C0000 A3404EF0 F0F0F0F0 |..... o....t +00000|
PL/I OCA: 11A5CA28
+000000 11A5CA28 D6C3C100 11A5C960 0B0220FF 00000000 01400030 00000200 00000000 00000000 |OCA..vI-.....|
+000020 11A5CA48 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000040 11A5CA68 - +00005F 11A5CA87      same as above
+000060 11A5CA88 00000000 00000000 00000000 00000000 00000000 00000000 00000000 11200952 |.....|
+000080 11A5CAA8 02020000 00000004 11A3E978 00000000 00000000 00000000 00000000 00000000 |.....tZ.....|
+0000A0 11A5CAC8 00000000 00000000 00000000 00000000 00000000 00000000 00030119 59C9C2D4 |.....IBM|
+0000C0 11A5CAE8 00000000 00000000 11A5B018 00000040 C3C4D3D3 00000000 40000000 00000000 |.....v... CDLL.....|
PL/I OCA: 11A5C960
+000000 11A5C960 D6C3C100 00000000 000000FF 00000000 0000FFFF 00000000 00000000 11A5BD40 |OCA.....v.....|
+000020 11A5C980 02040000 00000000 11A5BD42 02020000 00000001 11A5BD40 02040000 00000000 |.....v.....v.....v.....|
+000040 11A5C9A0 11A5BD40 02040000 00000000 11A5BD40 02040000 00000000 11A5BD40 02040000 |.v.....v.....v.....v.....|
+000060 11A5C9C0 00000000 11A5BD40 020D0000 00000000 11A5BE1C 020D0000 00000001 11A5BD40 |.....v.....v.....v.....|
+000080 11A5C9E0 02040000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000A0 11A5CA00 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000C0 11A5CA20 00000000 00000000 D6C3C100 11A5C960 0B0220FF 00000000 01400030 00000200 |.....OCA..vI-.....|
Enclave Control Blocks:
EDB: 11212648
+000000 11212648 C3C5C5C5 C4C24040 C0000001 11213870 11212D50 00000000 00000000 00000000 |CEEEDB .....&.....|
+000020 11212668 11212B58 11212B88 91215770 11212198 00000000 00000000 11212768 00000000 |.....hj.....q.....|
+000040 11212688 00000000 00000000 00006F60 00000000 00000000 91333B58 1120A880 112126A0 |.....?-.....j.....y.....|
+000060 112126A8 0000F460 00000000 11211E58 00000000 11214550 00000000 113F4660 112139B0 |.4-.....&.....-.....|
+000080 112126C8 40000000 0000FAF6 00000000 00000000 00000001 0001F060 00006FF8 008CCE00 |.....6.....0-?8.....|
+0000A0 112126E8 00000001 00000100 1120A988 112126F0 00000000 00000000 00000000 00000001 |.....zh...0.....|
MEML: 11213870
+000000 11213870 00000000 00000000 112945B0 00000000 00000000 00000000 112945B0 00000000 |.....|
+000020 11213890 00000000 00000000 112945B0 00000000 1120F8D8 00000000 91575288 00000000 |.....8Q....j..h....|
+000040 112138B0 00000000 00000000 112945B0 00000000 00000000 00000000 112945B0 00000000 |.....|
+000060 112138D0 - +00009F 1121390F      same as above
+0000A0 11213910 00000000 00000000 112945B0 00000000 00000000 00000000 914072E8 00000000 |.....j..Y....|
+0000C0 11213930 00000000 00000000 112945B0 00000000 00000000 00000000 112945B0 00000000 |.....|
+0000E0 11213950 - +00011F 1121398F      same as above
WSA address.....00000000

```

Figure 125. Control blocks associated with the thread section of the dump (Part 1 of 2)


```

File Status and Attributes:
:
Attributes of file: SYSPRINT
STREAM OUTPUT PRINT ENVIRONMENT( CONSECUTIVE VB BLKSIZE( 129) RECSIZE( 125) LINESIZE( 120) PAGESIZE( 60) CTLASA )
CURRENT RECORD IN BUFFER
BUFFER: 11A5CF20
+000000 11A5CF20 40E2A482 F140E2A3 8199A389 95874000 40000000 00000000 00000000 00000000 | Subl Starting . . . . . |
+000020 11A5CF40 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | ..... |
+000040 11A5CF60 - +00007F 11A5CF9F same as above

File Control Blocks:
FILE CONTROL BLOCK (FCB): 11A5CB38
+000000 11A5CB38 11A5CB38 00000000 11A5CFB0 114F5440 114F53E8 114F4FF0 114F53E8 114F4F50 |.v.....v...|. .|.Y. |0|.Y. |&
+000020 11A5CB58 114F4ED8 1153CA88 114F5440 114F5440 114F5440 114F5440 114F5440 114F53E8 |.+Q...h...|. .|. .|. .|. .|. .|.Y
+000040 11A5CB78 114F53E8 11537648 114F53E8 114F4E20 114F53E8 114F53E8 114F53E8 114F46A0 |.Y.....Y. |+. .|.Y. |.Y. |.Y. |..
+000060 11A5CB98 114F4618 114F4548 114F53E8 11A5CD20 00000000 00000000 00000000 00000000 |. .|. .|. .|.Y.v.....
+000080 11A5CB88 00000000 00454842 11A5D02C 00000000 000001DA 11A64024 11A5CF20 00000081 |.....v.....w .v.....a
+0000A0 11A5CB08 00000000 00000000 00000078 00000010 00000079 00000000 00000000 00000000 |.....&.....-.....
+0000C0 11A5CBF8 00000000 00080150 20000720 000A6080 00000000 00000000 00000000 00000000 |.....v.....t.....H.....
+0000E0 11A5CC18 1154DA70 11A5CF21 00000000 00000000 11A3B5FC 1154AAC8 00000000 0000000E |.....v.....
+000100 11A5CC38 00000000 11A5CF20 00000001 00000001 00000002 003C0078 08000000 00000000 |.....v.....
+000120 11A5CC58 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+000140 11A5CC78 00000000 00000000 00000000 003BA682 6B409985 8386947E E5C2C16B 40939985 |.....wb, recfm=VBA, lre
+000160 11A5CC98 83937EF1 F2F56B40 829392A2 89A9857E F1F2F96B 40A3A897 857E9985 83969984 |cl=125, blksize=129, type=record
+000180 11A5CCB8 6B409596 A2858592 00000000 00000000 00000000 00000000 00000000 00000000 |, noseek.....
+0001A0 11A5CCD8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....
+0001C0 11A5CCF8 - +0001DF 11A5CD17 same as above

CURRENT ACTIVE ICO CONTROL BLOCK:
ICO: 11A5CF21
+000000 11A5CF21 E2A482F1 40E2A381 99A38995 87400040 00000000 00000000 00000000 00000000 | Subl Starting . . . . . |
+000020 11A5CF41 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | ..... |
+000040 11A5CF61 - +00007F 11A5CFA0 same as above

Process Control Blocks:

PCB: 11212198
+000000 11212198 C3C5C5D7 C3C24040 03030288 00000000 00000000 00000000 112123D0 913F74D0 |CEEPCB ...h.....j...|
+000020 112121B8 913EDB50 913F4A08 913F4528 11208AE8 11211F68 00000000 00000000 11212648 |j..&j...j.....Y.....|
+000040 112121D8 913F4858 7F800000 00000000 000141D4 00000000 80000000 00000000 00000000 |j... ".....M.....|

MEML: 112123D0
+000000 112123D0 00000000 00000000 112945B0 00000000 00000000 00000000 112945B0 00000000 |.....|
+000020 112123F0 00000000 00000000 112945B0 00000000 1120DE68 00000000 91575288 11A35000 |.....j..h.t&|.
+000040 11212410 00000000 00000000 112945B0 00000000 00000000 00000000 112945B0 00000000 |.....|
+000060 11212430 - +00009F 1121246F same as above
+0000A0 11212470 00000000 00000000 112945B0 00000000 11A36000 00000000 914072E8 00000000 |.....t-....j .Y....|
+0000C0 11212490 00000000 00000000 112945B0 00000000 00000000 00000000 112945B0 00000000 |.....|
+0000E0 112124B0 - +00011F 112124EF same as above

PL/I Process Control Block: 11212198
+000000 11212198 C3C5C5D7 C3C24040 03030288 00000000 00000000 00000000 112123D0 913F74D0 |CEEPCB ...h.....j...|
+000020 112121B8 913EDB50 913F4A08 913F4528 11208AE8 11211F68 00000000 00000000 11212648 |j..&j...j.....Y.....|
:

```

Figure 125. Control blocks associated with the thread section of the dump (Part 2 of 2)

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 Enterprise 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

Enterprise PL/I contents of the Language Environment trace table

Language Environment provides three Enterprise 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 147.

Debugging example of Enterprise PL/I routines

This section contains examples of Enterprise 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 126 on page 327 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, GONUMBER, and MAP. It was run with the TERMTHDACT(TRACE) option to generate a traceback for the condition.

5655-H31 IBM(R) Enterprise PL/I for z/OS V3.R6.M0 (Built:20070119) 2007.01.31 15:59:36 Page 1

Options Specified
Install:
Command:
Line.File Process Statements
1.0 *PROCESS GONUMBER LIST S STG TEST MAP;

5655-H31 IBM(R) Enterprise PL/I for z/OS EXAMPLE: PROC OPTIONS(M 2007.01.31 15:59:36 Page 2
Compiler Source

Line.File
2.0 EXAMPLE: PROC OPTIONS(MAIN);
3.0
4.0 DCL Array(10) Fixed bin(31);
5.0 DCL (I,Array_End) Fixed bin(31);
6.0 On error
7.0 Begin;
8.0 On error system;
9.0 Call plidump('tbnfs','Plidump called from error On-unit');
10.0 End;
11.0
12.0 (subrg): /* Enable subscriptrange condition */
13.0 Labl1: Begin;
14.0 Array_End = 20;
15.0 Do I = 1 to Array_End; /* Loop to initialize array */
16.0 Array(I) = 2; /* Set array elements to 2 */
17.0 End;
18.0 End Labl1;
19.0 End Example;

5655-H31 IBM(R) Enterprise PL/I for z/OS EXAMPLE: PROC OPTIONS(M 2007.01.31 15:59:36 Page 3
Block Name List

Number Name
1 EXAMPLE
2 _ON_Begin_7_Blk_2
3 _Begin_12_Blk_3

5655-H31 IBM(R) Enterprise PL/I for z/OS EXAMPLE: PROC OPTIONS(M 2007.01.31 15:59:36 Page 4

OFFSET OBJECT CODE LINE# FILE# P S E U D O A S S E M B L Y L I S T I N G

Timestamp and Version Information
000000 F2F0 F0F7 =C'2007' Compiled Year
000004 F0F1 F3F1 =C'0131' Compiled Date MMDD
000008 F1F5 F5F9 F3F6 =C'155936' Compiled Time HHMMSS
00000E F0F3 F0F6 F0F0 =C'030600' Compiler Version
000014 0036 **** Service String 20070122
Timestamp and Version End

5655-H31 IBM(R) Enterprise PL/I for z/OS EXAMPLE: PROC OPTIONS(M :_Begin_12_Blk_3 2007.01.31 15:59:36 Page 5

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

:
5655-H31 IBM(R) Enterprise PL/I for z/OS EXAMPLE: PROC OPTIONS(M 2007.01.31 15:59:36 Page 17

***** STORAGE OFFSET LISTING *****

IDENTIFIER DEFINITION ATTRIBUTES
<SEQNBR>-<FILE NO="">:<FILE LINE="">
ARRAY 1-0:4 Class = automatic, Location = 192 : 0xC0(r13), Length = 40
ARRAY_END 1-0:5 Class = automatic, Location = 236 : 0xEC(r13), Length = 4
I 1-0:5 Class = automatic, Location = 232 : 0xE8(r13), Length = 4

***** END OF STORAGE OFFSET LISTING *****

:

Figure 126. Example of moving a value outside an array range

Figure 127 on page 328 shows sections of the dump generated by a call to PLIDUMP.

CEE3845I CEEDUMP Processing started.
PLIDUMP was called from statement number 9 at offset +000000D2 from _ON_Begin_7_Blk_2 with entry address 11200240

Information for enclave EXAMPLE

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	IBMPDUMP	+000002AE		IBMPEV11		PQ78306	Call
2	_ON_Begin_7_Blk_2						
		+000000D2	9	EXAMPLE	_Begin_12_Blk_3		Call
3	IBMPEONR	+000002A2		IBMPEV11		PQ76426	Call
4	IBMPEBOP	+000004DC		IBMPEV11		LE19BAS	Call
5	CEEEV011	+00000132		IBMPEV11	CEEEV011		Call
6	CEEHDSP	+000017D0		CEEPLPKA	CEEHDSP	D1908	Call
7	IBMBERRI	+000000AA		IBMPEV11		LE19BAS	Exception
8	ERR_RAISE_COND						
		+00000090		IBMPEV11		LE19BAS	Call
9	IBMPERSU	+00000082		IBMPEV11		LE19BAS	Call
10	_Begin_12_Blk_3						
		+00000100	16	EXAMPLE	_Begin_12_Blk_3		Call
11	EXAMPLE	+000000B0	12	EXAMPLE	_Begin_12_Blk_3		Call
12	IBMPMINV	+000004DE		IBMPEV11	IBMPMINV		Call
13	CEEEV011	+00000202		IBMPEV11	CEEEV011		Call
14	CEEBBEXT	+000001B6		CEEPLPKA	CEEBBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	11A3DE50	114A4E38	114A4E38	+000002AE	20061214	LIBRARY EBCDIC HFP
2	11A3DD70	11200240	112000D0	+00000242	20070131	ENT PL/I EBCDIC HFP
3	11A3DBD8	114A7B98	114A7B98	+000002A2	20061214	LIBRARY EBCDIC HFP
4	11A3DA08	114AF390	114AF390	+000004DC	20061214	LIBRARY EBCDIC HFP
5	11A3D978	114062E8	114062E8	+00000132	20061214	LIBRARY
6	11A3A858	112C3238	112C3238	+000017D0	20061215	CEL
7	11A3A6B8	114AAA18	114AAA18	+000000AA	20061214	LIBRARY EBCDIC HFP
8	11A3A618	114A9FA8	114A9FA8	+00000090	20061214	LIBRARY EBCDIC HFP
9	11A3A570	114AB120	114AB120	+00000082	20061214	LIBRARY EBCDIC HFP
10	11A3A4A8	112000D0	112000D0	+00000100	20070131	ENT PL/I EBCDIC HFP
11	11A3A3B8	11200340	112000D0	+00000320	20070131	ENT PL/I EBCDIC HFP
12	11A3A180	114DD990	114DD990	+000004DE	20061214	LIBRARY
13	11A3A0F0	114062E8	114062E8	+00000202	20061214	LIBRARY
14	11A3A030	11291208	11291208	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for (DSA address 11A3A6B8)

CIB Address: 11A3B178

Current Condition:

IBM0281S A prior condition was promoted to the ERROR condition.

Original Condition:

IBM0421S ONCODE=520 The SUBSCRIPTRANGE condition was raised.

Location:

Program Unit: Entry: IBMBERRI Statement: Offset: +000000AA

Storage dump near condition, beginning at location: 114AAAB2

+000000 114AAAB2 4110D098 5050D098 5040D09C 5040D0A0 05EF1F00 43002016 A7010080 A7840009 |...q&&.q& ..&x...xd..|

Control Blocks for Active Routines:

DSA for IBMPDUMP: 11A3DE50

+000000	FLAGS....	1000	member...	0000	BKC.....	11A3DD70	FWC.....	11A3E150	R14.....	914A50E8
+000010	R15.....	912EF898	R0.....	00000000	R1.....	11A3DEE8	R2.....	00000002	R3.....	114A5104
+000024	R4.....	11A3DEF8	R5.....	11A3E04C	R6.....	11200800	R7.....	11200438	R8.....	11A3B264
+000038	R9.....	00000014	R10.....	00000098	R11.....	11A3A468	R12.....	112129B0	reserved.	00000000
+00004C	NAB.....	11A3E150	PNAB....	00000000	reserved.	00000000	11A3DB34			
+000064	reserved.	926DF340	reserved.	81A340A2	MODE....	A381A385	reserved.	948595A3		
+000078	reserved.	81A34083	reserved.	C6C5D5E4						

Dynamic save area (IBMPDUMP): 11A3DE50

+000000	11A3DE50	10000000	11A3DD70	11A3E150	914A50E8	912EF898	00000000	11A3DEE8	00000002t...t.&j.&Yj.8q.....t.Y....
+000020	11A3DE70	114A5104	11A3DEF8	11A3E04C	11200800	11200438	11A3B264	00000014	00000098t...t.<.....t.....q
+000040	11A3DE90	11A3A468	112129B0	00000000	11A3E150	00000000	00000000	11A37B2E	11A3DB34	..tu.....t.&.....t#...t..

Figure 127. Sections of the Language Environment dump (Part 1 of 2)

```

DSA for _Begin_12_Blk_3: 11A3A4A8
+000000  FLAGS.... 1000      member... 0000      BKC..... 11A3A3B8  FWC..... 11A3A7D8  R14..... 912001D2
+000010  R15..... 914AB120  R0..... 0000000A  R1..... 11A3A478  R2..... 0000000B  R3..... 1120010A
+000024  R4..... 11A3A0D8  R5..... 11A3A3B8  R6..... 11200800  R7..... 00000000  R8..... 11211648
+000038  R9..... 00000008  R10..... 11A3A0B0  R11..... 11200ACC  R12..... 112129B0  reserved. 00000000
+00004C  NAB..... 11A3A570  PNAB.... 00000000  reserved. 00000000 00000000
+000064  reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000
Dynamic save area (_Begin_12_Blk_3): 11A3A4A8
+000000  11A3A4A8  10000000  11A3A3B8  11A3A7D8  912001D2  914AB120  0000000A  11A3A478  0000000B  |.....tt.txQj..Kj.....tu....
+000020  11A3A4C8  1120010A  11A3A0D8  11A3A3B8  11200800  00000000  11211648  00000008  11A3A0B0  |.....t.Q.tt.....t..
+000040  11A3A4E8  11200ACC  112129B0  00000000  11A3A570  00000000  00000000  00000000  00000000  |.....tv.....
+000060  11A3A508  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  |.....
+000080  11A3A528  11A3A120  9129B186  11A3A0F0  00000000  10A3A0F0  11A3A4A8  00000000  00000000  |.t..j..f.t.0.....t.0.tuy.....
+0000A0  11A3A548  00180280  11A3A4A8  11A3A450  11200A00  00000014  11A3A3B8  0000000A  0000000B  |.....tuy.tu&.....tt.....
+0000C0  11A3A568  11A3A478  11200340  10A3A4A8  11A3A4A8  11200650  914AB1A4  11A9FA8  00000020  |.tu....tuy.tuy..&j..u...y....
DSA for EXAMPLE: 11A3A3B8
+000000  FLAGS.... 10A3      member... A12C      BKC..... 11A3A180  FWC..... 11A3A138  R14..... 912003F2
+000010  R15..... 112000D0  R0..... 11A3A3B8  R1..... 11200240  R2..... 11211768  R3..... 1120037A
+000024  R4..... 11A3A0D8  R5..... 11A3A3B8  R6..... 11200800  R7..... 00000000  R8..... 11211648
+000038  R9..... 00000008  R10..... 11A3A0B0  R11..... 11200ACC  R12..... 112129B0  reserved. 00000000
+00004C  NAB..... 11A3A4A8  PNAB.... 00000000  reserved. 00000000 00000000
+000064  reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000
Dynamic save area (EXAMPLE): 11A3A3B8
+000000  11A3A3B8  10A3A12C  11A3A180  11A3A138  912003F2  112000D0  11A3A3B8  11200240  11211768  |.t.t...t..j..2....tt....
+000020  11A3A3D8  1120037A  11A3A0D8  11A3A3B8  11200800  00000000  11211648  00000008  11A3A0B0  |.....t.Q.tt.....t..
+000040  11A3A3F8  11200ACC  112129B0  00000000  11A3A4A8  00000000  00000000  00000000  00000000  |.....tuy.....
+000060  11A3A418  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  |.....
+000080  11A3A438  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  |.....
+0000A0  11A3A458  00180180  11A3A3B8  00000000  11200980  0B300000  11200240  11A3A3B8  00000000  |.....tt.....tt.....
+0000C0  11A3A478  00000002  00000002  00000002  00000002  00000002  00000002  00000002  00000002  |.....
+0000E0  11A3A498  00000002  00000002  0000000B  00000014  10000000  11A3A3B8  11A3A7D8  912001D2  |.....tt.txQj..K
ENTERPRISE PL/I OPTIONS:
AFP, ARCH( 5), BACKREG(5), BIFPREC(15), CHECK(NOCONFORMANCE, NOSTORAGE), CMPAT(V2), CODEPAGE( 1140), COMMON,
NOCOMPACT, CSECT, CSECTCUT( 4) CURRENCY( $), NOCBCS, DECIMAL( FOFLONASGN, NOFORCEDSIGN), DEFAULT( IBM, ASSIGNABLE,
NOINITFILL, NONCONNECTED, DESCRIPTOR, DESCLOCATOR, DUMMY(ALIGNED), ORDINAL(MIN), BYADDR, RETURNS(BYADDR),
LINKAGE(OPTLINK), NORETCODE, NOINLINE, ORDER, NOOVERLAP, NONRECURSIVE, ALIGNED, NULL370, EVENDEC, SHORT(HEXADEC),
EBCDIC, HEXADEC, NATIVE, NATIVEADDR, E(HEXADEC) ), DISPLAY(WTO), NODLLIMIT, EXTRN(FULL), NOGRAPHIC, NOINITAUTO,
NOINITBASED, NOINITCTL, NOINITSTATIC, NOINTERRUPT, LIMITS( EXTNAME( 7), FIXEDBIN( 31, 31), FIXEDDEC( 15, 15), NAME( 100
)), OPTIMIZE( 0), PREFIX( CONVERSION, FIXEDOVERFLOW, INVALIDDOP, OVERFLOW, NOSIZE, NOSTRINGRANGE,
NOSTRINGSIZE, NOSUBSCRIPTRANGE, UNDERFLOW, ZERODIVIDE), REDUCE, NORENT, RESEXP, RESPECT(), RULES(IBM), NOSTDSYS,
NOSCHEDULER, STRINGOFGGRAPHIC(GRAPHIC), SYSTEM(MVS), TEST(ALL, SYM,HOOK,NOSEPARATE), TUNE( 5), USAGE( ROUND(IBM),
UNSPEC(IBM)), WIDECCHAR(BIGENDIAN), WINDOW( 1950), WRITABLE, XINFO(NODEF, NOXML)
Static for procedure EXAMPLE Timestamp: 2007.01.31 15:59:36 V03.R06.M00: 11200800
+000000  11200800  02020240  00000005  02020240  00000021  00000000  11200898  00000000  11200830  |.. .....q.....
+000020  11200820  00000000  11200898  00000000  11200830  00000000  11200838  00000000  00000000  |.....q.....
+000040  11200840  31010001  00000000  00000000  00000000  00000010  00000000  00000000  00000000  |.....
+000060  11200860  11200880  00000000  00000000  00000000  00000004  00000004  0000000A  00000001  |.....
+000080  11200880  0000001B  00000024  0000002D  00000032  00000000  00000000  112008D0  112008F0  |.....0
+0000A0  112008A0  11200910  11200930  00000000  11200860  02002200  11200340  00000000  00000000  |.....
+0000C0  112008C0  11200840  11200438  00090107  00000000  42002201  000000A2  00000000  00000000  |.....s.....
+0000E0  112008E0  30000000  11200444  00090105  00000000  48012401  D00000C0  11200870  00000000  |.....
+000100  11200900  00001F80  1120044C  00010105  00000000  40002401  D00000E8  00000000  00000000  |.....<.....Y.....
+000120  11200920  00001F80  11200454  00010101  00000000  40002401  D00000EC  00000000  00000000  |.....
+000140  11200940  00001F80  11200458  00010109  00000000  00000003  00000010  00000002  0000002D  |.....
+000160  11200960  00000024  0000000F  00000002  0000002D  00000032  0000000E  00000001  00000032  |.....
+000180  11200980  D0000098  00100000  6E3BFF00  00000000  11200898  11200A80  11200830  90010000  |..q...>.....q.....
+0001A0  112009A0  11200340  00000000  00000000  00000004  11200880  00000000  112009C0  00000000  |.....
+0001C0  112009C0  D00000A8  00100000  6E3BFF00  00000000  11200820  11200A80  11200828  83010000  |..y...>.....c...
+0001E0  112009E0  11200240  11200980  11200A00  00000000  00000000  00000000  00000000  00000000  |.....
+000200  11200A00  D0000098  00000000  6E7BFF00  00000000  11200810  11200A80  11200818  01000000  |..q...>#.....
+000220  11200A20  112000D0  11200980  00000000  00000000  00000000  11200950  00000000  00000000  |.....&.....
+000240  11200A40  00000000  00000000  00000000  00000000  00010007  C5E7C1D4  D7D3C500  010005D3  |.....EXAMPLE....L
+000260  11200A60  C1CD3F1  00010005  C1D9D9C1  E8000100  01C90001  0009C1D9  D9C1E86D  C5D5C400  |ABL1...ARRAY...I...ARRAY_END.
+000280  11200A80  00000000  11200464  E1100FF2  F0F0F7F0  F1F3F1F1  F5F5F9F3  F6F0F0F0  0000003F  |.....20070131155936000....
+0002A0  11200AA0  11200A40  11200860  11200838  00000001  11200850  00220000  00000000  11200980  |..-.....&.....
+0002C0  11200AC0  11200438  00000000  01000001  11200340  11200FC0  00000000  18AF47F0  A016C3C5  |.....0..CE

```

Figure 127. Sections of the Language Environment dump (Part 2 of 2)

To debug this routine, use the following steps:

1. In the dump, PLIDUMP was called by the ERROR ON-unit in statement 9. The traceback information in the dump shows that the exception occurred following statement 16.

Note: In the LE dumps, the columns and messages refer to "statements", but the numbers are actually (for Enterprise PL/I) the line numbers from the source file.

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 14 in the routine in Figure 126 on page 327. The instruction is Array_End = 20. This statement assigns a 20 value to the variable Array_End.
4. Statement 15 begins the DO-loop instruction Do I = 1 to Array_End. Since the previous instruction (statement 14) specified that Array_End = 20, the loop in statement 10 should run until I reaches a 20 value.

The instruction in statement 4, 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 storage offset listing in Figure 126 on page 327. Use this offset to find the I value at the time of the dump. In this example, the offset is X'E8'.
2. Now find offset X'E8' from the start of the stack frame for the entry EXAMPLE in Figure 127 on page 328.

The block located at this offset contains the value that exceeded the array range, X'B' or 11.

Calling a nonexistent subroutine

Figure 128 on page 331 demonstrates the error of calling a nonexistent subroutine. This routine was compiled with the LIST, MAP, and GONUMBER compiler options. It was run with the TERMTHDACT(DUMP) run-time option to generate a traceback.

```

5655-H31  IBM(R) Enterprise PL/I for z/OS      V3.R6.M0 (Built:20070119)      2007.01.31 16:02:29  Page    1
          Options Specified
Install:
Command:
Line.File Process Statements
1.0      *PROCESS  GONUMBER LIST S STG TEST MAP;
Install:
5655-H31  IBM(R) Enterprise PL/I for z/OS      EXAMPLE1: PROC  OPTIONS(      2007.01.31 16:02:29  Page    2
Compiler Source
Line.File
2.0      EXAMPLE1: PROC  OPTIONS(MAIN);
3.0
4.0      DCL Prog01 entry external;
5.0
6.0      On error
7.0      Begin;
8.0      On error system;
9.0      Call plidump('tbnfs','Plidump called from error On-unit');
10.0     End;
11.0
12.0     Call prog01;          /* Call external program PROG01 */
13.0
14.0     End Example1;
5655-H31  IBM(R) Enterprise PL/I for z/OS      EXAMPLE1: PROC  OPTIONS(      2007.01.31 16:02:29  Page    3
          Block Name List
Number  Name
1      EXAMPLE1
2      _ON_Begin_7_Blk_2
5655-H31  IBM(R) Enterprise PL/I for z/OS      EXAMPLE1: PROC  OPTIONS(      2007.01.31 16:02:29  Page    4

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
:

```

Figure 128. Example of calling a nonexistent subroutine

Figure 129 on page 332 shows the traceback and condition information from the dump.

CEE3845I CEEDUMP Processing started.
 PLIDUMP was called from statement number 9 at offset +000000D2 from _ON_Begin_7_Blk_2 with entry address 0B9008A8

Information for enclave EXAMPLE1

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	IBMPDUMP	+000002AE		IBMPEV11		PQ78306	Call
2	_ON_Begin_7_Blk_2	+000000D2	9	EXAMPLE1	_ON_Begin_7_Blk_2		Call
3	IBMPEONR	+000002A2		IBMPEV11		PQ76426	Call
4	IBMPEBOP	+000004DC		IBMPEV11		LE19BAS	Call
5	CEEEV011	+00000132		IBMPEV11	CEEEV011		Call
6	CEEHDSP	+000017D0		CEEPLPKA	CEEHDSP	D1908	Call
7	EXAMPLE1	-0B9009A8		EXAMPLE1	_ON_Begin_7_Blk_2		Exception
8	IBMPMINV	+000004DE		IBMPEV11	IBMPMINV		Call
9	CEEEV011	+00000202		IBMPEV11	CEEEV011		Call
10	CEEBEXT	+000001B6		CEEPLPKA	CEEBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	0C13DA70	0BBA4E38	0BBA4E38	+000002AE	20061214	LIBRARY EBCDIC HFP
2	0C13D990	0B9008A8	0B9008A8	+000000D2	20070131	ENT PL/I EBCDIC HFP
3	0C13D7F8	0BBA7B98	0BBA7B98	+000002A2	20061214	LIBRARY EBCDIC HFP
4	0C13D628	0BBAF390	0BBAF390	+000004DC	20061214	LIBRARY EBCDIC HFP
5	0C13D598	0BB062E8	0BB062E8	+00000132	20061214	LIBRARY
6	0C13A478	0B9C3238	0B9C3238	+000017D0	20061215	CEL
7	0C13A3B8	0B9009A8	0B9009A8	-0B9009A8	20070131	ENT PL/I EBCDIC HFP
8	0C13A180	0BBD0990	0BBD0990	+000004DE	20061214	LIBRARY
9	0C13A0F0	0BB062E8	0BB062E8	+00000202	20061214	LIBRARY
10	0C13A030	0B991208	0B991208	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for _ON_Begin_7_Blk_2 (DSA address 0C13A3B8)

CIB Address: 0C13AD98

Current Condition:

CEE3201S The system detected an operation exception (System Completion Code=0C1).

Location:

Program Unit: _ON_Begin_7_Blk_2

Entry: EXAMPLE1 Statement: Offset: -0B9009A8

Possible Bad Branch: Statement: 12 Offset: +000001AE

Machine State:

```
ILC..... 0002      Interruption Code..... 0001
PSW..... 078D0600 80000002
GPR0..... 0C13A3B8  GPR1..... 0B9008A8  GPR2..... 0B911768  GPR3..... 0B9009E2
GPR4..... 0C13A0D8  GPR5..... 00000000  GPR6..... 0B900DA0  GPR7..... 00000000
GPR8..... 0B911648  GPR9..... 00000008  GPR10..... 0C13A0B0  GPR11..... 0B900F1C
GPR12..... 0B9129B0  GPR13..... 0C13A3B8  GPR14..... 8B900A58  GPR15..... 00000000
FPC..... F0000000
FPR0..... 26100000  00000000          FPR1..... 00000000  00000000
FPR2..... 18000000  00000000          FPR3..... 00000000  00000000
FPR4..... 00000000  00000000          FPR5..... 00000000  00000000
FPR6..... 00000000  00000000          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: 00000000

+000000 00000000 Inaccessible storage.

GPREG STORAGE:

Storage around GPR0 (0C13A3B8)

```
-0020 0C13A398 00000000 00000000 00000000 00000000 00000000 00000000 0C13A124 0C13A128 |.....|
+0000 0C13A3B8 1013A12C 0C13A180 0C13A138 8B900A30 8BBA4490 0B000000 0C13A3B8 0B911768 |.....t.j..|
+0020 0C13A3D8 0B9009E2 0C13A0D8 00000000 0B900DA0 00000000 0B911648 00000000 00000000 |...S...Q.....j.....|
```

Storage around GPR1 (0B9008A8)

```
-0020 0B900888 36000301 0FC00000 00240008 C5E7C1D4 D7D3C5F1 0B900DA0 0000016C 00000000 |.....EXAMPLE1.....%...|
+0000 0B9008A8 47F0F022 01C3C5C5 000000E0 00000330 47F0F001 58F0C31C 184E05EF 00000000 |.00..CEE.....00..0C..+.....|
+0020 0B9008C8 07F390E7 D00C58E0 D04C4100 E0E05500 C3144130 F03A4720 F01458F0 C28090F0 |.3.X.....<.....C...0...0..0B..0|
```

Storage around GPR2 (0B911768)

```
-0020 0B911748 00000000 00000000 0B9095A8 0B909500 0B909450 00000000 00000106 00000000 |.....ny..n..m&.....|
+0000 0B911768 8B910E58 00000000 00000000 00000000 0B901410 00000000 8B9094C8 00030000 0003000B |.j.....mH.....|
+0020 0B911788 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
```

Figure 129. Sections of the Language Environment dump (Part 1 of 2)


```

Storage around GPR3 (0B9009E2)
-0020 0B9009C2 05EF0000 000007F3 90E6D00C 58E0D04C 4100E0C0 5500C314 4130F03A 4720F014 |.....3.W.....<.....C...0...0.|
+0000 0B9009E2 58F0C280 90F0E048 9210E000 50D0E004 18DE5860 30AA5800 309E5000 D0984100 |.0B..0..k...&.....-.....&..q..|
+0020 0B900A02 00005000 D09C5810 30A25010 D0A04110 60A85010 D0AC181D 5010D0A4 5000D0A8 |..&.....s&.....-y&.....&..u&..y|
Storage around GPR4 (0C13A0D8)
-0020 0C13A0B8 0C13A0E0 0B911780 0C13A0D0 0C13A0D8 0C13A0D4 00000000 0B911768 00000000 |.....j.....Q...M.....j.....|
+0000 0C13A0D8 00000000 00000000 00000001 00000000 00000000 00000000 880062CC 0C13A030 |.....h.....|
+0020 0C13A0F8 0C13A180 8BB064EC 0BBDD990 7D000008 0C13A0B0 0C13A0B0 8BB062E8 8B9912EC |.....R.'.....Y.r..|
Storage around GPR5 (00000000)
+0000 00000000 Inaccessible storage.
+0020 00000020 Inaccessible storage.
+0040 00000040 Inaccessible storage.
Storage around GPR6 (0B900DA0)
-0020 0B900D80 14380000 00000000 0B003203 FFFFA50 FFFFFFFB8 FFFFFAD0 00000000 02000000 |.....&.....|
+0000 0B900DA0 02020240 00000005 02020240 00000021 00000000 0B900DC0 00000000 0B900DC8 |.....H|
+0020 0B900DC0 00000000 0B900DF8 00000000 0B900DD0 00000000 00000000 31010001 00000000 |.....8.....|
Storage around GPR7 (00000000)
+0000 00000000 Inaccessible storage.
+0020 00000020 Inaccessible storage.
+0040 00000040 Inaccessible storage.
Storage around GPR8 (0B911648)
-0020 0B911628 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 0B911648 C3C5C5C5 C4C24040 C0000001 0B912870 0B911D50 00000000 00000000 00000000 |CEEEDB .....j...j.&.....|
+0020 0B911668 0B911B58 0B911B88 8B914770 0B911198 00000000 00000000 0B911768 00000000 |.j...j.h.j...j.q.....j.....|
Storage around GPR9 (00000000)
-0008 00000000 Inaccessible storage.
+0018 00000020 Inaccessible storage.
+0038 00000040 Inaccessible storage.
Storage around GPR10(0C13A0B0)
-0020 0C13A090 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0000 0C13A0B0 0B9914D0 0B900F1C 0C13A0E0 0B911780 0C13A0D0 0C13A0D8 0C13A0D4 00000000 |.r.....j.....Q...M...|
+0020 0C13A0D0 0B911768 00000000 00000000 00000000 00000001 00000000 00000000 00000000 |.j.....|
Storage around GPR11(0B900F1C)
-0020 0B900EFC 00220000 00000000 0B900E48 0B900AA0 00000000 00000000 00000000 01000001 |.....|
+0000 0B900F1C 0B9009A8 0B901410 00000000 18AF47F0 A016C3C5 C5D6D7C9 D7C9F1F2 61F1F561 |...y.....0..CEE0PIPI12/15/|
+0020 0B900F3C F0F65830 20005830 30009547 30004770 A06895CE 30054770 A0365830 300847F0 |06.....n.....n.....0|
Storage around GPR12(0B9129B0)
-0020 0B912990 00000000 00000000 C3C5C5C3 C1C14040 00000000 00000000 000058C0 D0640CCC |.....CEECAA .....|
+0000 0B9129B0 00000800 00000000 0C13A018 0C15A018 00000000 00000000 00000000 00000000 |.....|
+0020 0B9129D0 00000000 00000000 0B90FA58 00000000 00000000 00000000 00000000 00000000 |.....|
Storage around GPR13(0C13A3B8)
-0020 0C13A398 00000000 00000000 00000000 00000000 00000000 00000000 0C13A124 0C13A128 |.....|
+0000 0C13A3B8 1013A12C 0C13A180 0C13A138 8B900A30 8BBA4490 0B000000 0C13A3B8 0B911768 |.....t..j..|
+0020 0C13A3D8 0B9009E2 0C13A0D8 00000000 0B900DA0 00000000 0B911648 00000000 00000000 |..S...Q.....j.....|
Storage around GPR14(0B900A58)
-0020 0B900A38 9230D0B1 581030B2 180D5010 D0B45000 D0B84400 C1AC58F0 30B64400 C1C005EF |k.....&...&....A..0...A...|
+0000 0B900A58 4400C1C4 4400C1AC 4400C1B4 58F030BA 05EF4400 C1B858D0 D00458E0 D00C9826 |..AD..A...A..0...A.....q..|
+0020 0B900A78 D01C051E 07070000 00100000 00180180 0B000000 0B900DA0 0B901438 0B9008A8 |.....y|
Storage around GPR15(00000000)
+0000 00000000 Inaccessible storage.
+0020 00000020 Inaccessible storage.
+0040 00000040 Inaccessible storage.
:

```

Figure 129. 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. The Location section indicates that the exception occurred at offset X'-0B9009A8' within entry EXAMPLE1 and that there may have been a bad branch from offset X'+000001AE' statement 12 within entry EXAMPLE1 .

2. Locate statement 12 in the routine (Figure 128 on page 331). 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 130 demonstrates a divide-by-zero error. In this example, the main Enterprise PL/I routine passed bad data to an Enterprise 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.

```

5655-H31 IBM(R) Enterprise PL/I for z/OS      V3.R6.M0 (Built:20070119)      2007.01.31 16:02:31  Page  1
          Options Specified
Install:
Command:
Line.File Process Statements
  1.0  *PROCESS  GONUMBER LIST S STG TEST MAP;
Install:
5655-H31 IBM(R) Enterprise PL/I for z/OS      SAMPLE: PROC  OPTIONS(MAI      2007.01.31 16:02:31  Page  2
Compiler Source
Line.File
  2.0  SAMPLE: PROC  OPTIONS(MAIN) ;
  3.0  On error
  4.0  begin;
  5.0  On error system;          /* prevent nested error conditions */
  6.0  Call PLIDUMP('TBC','PLIDUMP called from error ON-unit');
  7.0  Put Data;                /* Display variables */
  8.0  End;
  9.0
 10.0  DECLARE
 11.0  A_number  Fixed Bin(31),
 12.0  My_Name   Char(13),
 13.0  An_Array(3) Fixed Bin(31) init(1,3,5);
 14.0
 15.0  Put skip list('Sample Starting');
 16.0  A_number = 0;
 17.0  My_Name = 'Tery Gillaspy';
 18.0
 19.0  Call Sub1(a_number, my_name, an_array);
 20.0
 21.0  SUB1: PROC(divisor, name1, Array1);
 22.0  Declare
 23.0  Divisor   Fixed Bin(31),
 24.0  Name1     Char(13),
 25.0  Array1(3) Fixed Bin(31);
 26.0  Put skip list('Sub1 Starting');
 27.0  Array1(1) = Array1(2) / Divisor;
 28.0  Put skip list('Sub1 Ending');
 29.0  End SUB1;
 30.0
 31.0  Put skip list('Sample Ending');
 32.0
 33.0  End;
:

```

Figure 130. Enterprise 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 131 shows this output.

```

1Sample Starting
Sub1 Starting      A_NUMBER=          0 MY_NAME='Tery Gillaspy' AN_ARRAY(1)=  1
AN_ARRAY(2)=      3          AN_ARRAY(3)=          5;

```

Figure 131. Variables from routine SAMPLE

The routine in Figure 130 on page 334 was compiled with the LIST compiler option, which generated the object code listing shown in Figure 132.

```

:
:
5655-H31 IBM(R) Enterprise PL/I for z/OS  SAMPLE: PROC OPTIONS(MAI : SUB1  2007.01.31 16:02:31  Page  5

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                000021  |      SUB1  DS  0D
000000  47F0  F022      000021  |          B  34(,r15)
000004  01C3C5C5                CEE eyecatcher
000008  00000160                DSA size
00000C  00000848                =A(PPA1-SUB1)
:
000136  5820  D14C      000027  |          L  r2,#SR_PARM_3(,r13,332)
00013A  5820  2008      000027  |          L  r2,_addrARRAY1(,r2,8)
00013E  5840  2000      000027  |          L  r4,_shadow1(,r2,0)
000142  5820  D14C      000027  |          L  r2,#SR_PARM_3(,r13,332)
000146  5820  2008      000027  |          L  r2,_addrARRAY1(,r2,8)
00014A  5820  2004      000027  |          L  r2,_shadow1(,r2,4)
00014E  5040  D150      000027  |          ST r4,#wtemp_1(,r13,336)
000152  5020  D144      000027  |          ST r2,_temp15(,r13,324)
000156  5820  D144      000027  |          L  r2,_temp15(,r13,324)
00015A  5850  2004      000027  |          L  r5,_shadow2(,r2,4)
00015E  5840  D150      000027  |          L  r4,#wtemp_1(,r13,336)
000162  5820  D144      000027  |          L  r2,_temp15(,r13,324)
000166  5820  2000      000027  |          L  r2,_shadow2(,r2,0)
00016A  1382                000027  |          LCR  r8,r2
00016C  5820  D14C      000027  |          L  r2,#SR_PARM_3(,r13,332)
000170  5820  2008      000027  |          L  r2,_addrARRAY1(,r2,8)
000174  5890  2000      000027  |          L  r9,_shadow1(,r2,0)
000178  5820  D14C      000027  |          L  r2,#SR_PARM_3(,r13,332)
00017C  5820  2008      000027  |          L  r2,_addrARRAY1(,r2,8)
000180  5820  2004      000027  |          L  r2,_shadow1(,r2,4)
000184  5090  D154      000027  |          ST r9,#wtemp_2(,r13,340)
000188  1E58                000027  |          ALR r5,r8
00018A  4145  4000      000027  |          LA r4,#AddressShadow(r5,r4,0)
00018E  5040  D158      000027  |          ST r4,#wtemp_3(,r13,344)
000192  5020  D140      000027  |          ST r2,_temp14(,r13,320)
000196  5820  D140      000027  |          L  r2,_temp14(,r13,320)
00019A  5840  2004      000027  |          L  r4,_shadow2(,r2,4)
00019E  8940  0001      000027  |          SLL r4,1
0001A2  5820  D154      000027  |          L  r2,#wtemp_2(,r13,340)
0001A6  5850  D140      000027  |          L  r5,_temp14(,r13,320)
0001AA  5850  5000      000027  |          L  r5,_shadow2(,r5,0)
0001AE  1355                000027  |          LCR  r5,r5
0001B0  1E45                000027  |          ALR  r4,r5
0001B2  5884  2000      000027  |          L  r8,_shadow2(r4,r2,0)
0001B6  5820  D14C      000027  |          L  r2,#SR_PARM_3(,r13,332)
0001BA  5820  2000      000027  |          L  r2,_addrDIVISOR(,r2,0)
0001BE  5820  2000      000027  |          L  r2,_shadow2(,r2,0)
0001C2  8E80  0020      000027  |          SRDA r8,32
0001C6  1D82                000027  |          DR  r8,r2
0001C8  1849                000027  |          LR  r4,r9
0001CA  5820  D158      000027  |          L  r2,#wtemp_3(,r13,344)
0001CE  5040  2000      000027  |          ST  r4,_shadow2(,r2,0)
:
:

```

Figure 132. Object code listing from example Enterprise PL/I routine

Figure 133 on page 336 shows the Language Environment dump for routine SAMPLE.

CEE3845I CEEDUMP Processing started.
 PLIDUMP was called from statement number 6 at offset +000000D4 from _ON_Begin_4_Blk_2 with entry address 11200340

Information for enclave SAMPLE

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	IBMPDUMP	+000002AE		IBMPEV11		PQ78306	Call
2	_ON_Begin_4_Blk_2						
		+000000D4	6	SAMPLE	SUB1		Call
3	IBMPEONR	+000002A2		IBMPEV11		PQ76426	Call
4	IBMPEBOP	+000004DC		IBMPEV11		LE19BAS	Call
5	CEEEV011	+00000132		IBMPEV11	CEEEV011		Call
6	CEEHDSB	+000017D0		CEEPLPKA	CEEHDSB	D1908	Call
7	SUB1	+000001C6	27	SAMPLE	SUB1		Exception
8	SAMPLE	+000001CA	19	SAMPLE	SUB1		Call
9	IBMPMINV	+000004DE		IBMPEV11	IBMPMINV		Call
10	CEEEV011	+00000202		IBMPEV11	CEEEV011		Call
11	CEEBEXT	+000001B6		CEEPLPKA	CEEBEXT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	11A3ED08	11A45E38	11A45E38	+000002AE	20061214	LIBRARY EBCDIC HFP
2	11A3EBB0	11200340	112000D0	+00000344	20070131	ENT PL/I EBCDIC HFP
3	11A3EA18	11A48B98	11A48B98	+000002A2	20061214	LIBRARY EBCDIC HFP
4	11A3E848	114B0390	114B0390	+000004DC	20061214	LIBRARY EBCDIC HFP
5	11A3E7B8	114072E8	114072E8	+00000132	20061214	LIBRARY
6	11A3B698	112C4238	112C4238	+000017D0	20061215	CEL
7	11A3B538	112000D0	112000D0	+000001C6	20070131	ENT PL/I EBCDIC HFP
8	11A3B3B8	112004B8	112000D0	+000005B2	20070131	ENT PL/I EBCDIC HFP
9	11A3B180	114DE990	114DE990	+000004DE	20061214	LIBRARY
10	11A3B0F0	114072E8	114072E8	+00000202	20061214	LIBRARY
11	11A3B030	11292208	11292208	+000001B6	20061215	CEL

Condition Information for Active Routines

Condition Information for SUB1 (DSA address 11A3B538)

CIB Address: 11A3BF88

Current Condition:

IBM0281S A prior condition was promoted to the ERROR condition.

Original Condition:

CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

Location:

Program Unit: SUB1 Entry: SUB1 Statement: 27 Offset: +000001C6

Machine State:

ILC.....	0002	Interruption Code.....	0009
PSW.....	078D2600	91200298	
GPR0.....	00000001	GPR1.....	00004A48
GPR2.....	00000000	GPR3.....	1120010A
GPR4.....	00000004	GPR5.....	FFFFFFFC
GPR6.....	11200BA8	GPR7.....	11200748
GPR8.....	00000000	GPR9.....	00000003
GPR10.....	11A3B0B0	GPR11.....	11200FE4
GPR12.....	112139B0	GPR13.....	11A3B538
GPR14.....	912001FE	GPR15.....	1153CF40
FPC.....	F0000000		
FPR0.....	26100000	00000000	FPR1.....
FPR2.....	18000000	00000000	FPR3.....
FPR4.....	00000000	00000000	FPR5.....
FPR6.....	00000000	00000000	FPR7.....
FPR8.....	00000000	00000000	FPR9.....
FPR10.....	00000000	00000000	FPR11.....
FPR12.....	00000000	00000000	FPR13.....
FPR14.....	00000000	00000000	FPR15.....

Storage dump near condition, beginning at location: 11200286

+000000 11200286 5820D14C 58202000 58202000 8E800020 1D821849 5820D158 50402000 4400C1AC |..J<.....b....J.&A.|

Figure 133. Language Environment dump from example Enterprise PL/I routine (Part 1 of 2)

Control Blocks for Active Routines:

DSA for CEEHDSP: 11A3B698

```
+000000  FLAGS.... 0808      member... CEE1      BKC..... 11A3B538  FWC..... 11A3E7B8  R14..... 912C5A0A
+000010  R15..... 914072E8  R0..... 00000010  R1..... 1120C2E8  R2..... 11A3BF88  R3..... 11A3B3B8
+000024  R4..... 112C8E64  R5..... FFFFFFF30  R6..... 00000001  R7..... 00000007  R8..... 912C5712
+000038  R9..... 11A3D696  R10..... 11A3C697  R11..... 112C4238  R12..... 112139B0  reserved. 00000000
+00004C  NAB..... 11A3E7B8  PNAB..... 11A3B874  reserved. 11A3B9D8 00000000
+000064  reserved. 11A3BE58  reserved. 00000000  MODE..... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000
```

DSA for SUB1: 11A3B538

```
+000000  FLAGS.... 10A3      member... B0F0      BKC..... 11A3B3B8  FWC..... 11A3B5F8  R14..... 912001FE
+000010  R15..... 11537648  R0..... 11A3B668  R1..... 11A3B5D0  R2..... 00000001  R3..... 1120010A
+000024  R4..... 00004A48  R5..... 11A3B3B8  R6..... 11200BA8  R7..... 11200748  R8..... 11A3B528
+000038  R9..... 11A3B484  R10..... 11A3B0B0  R11..... 11200FE4  R12..... 112139B0  reserved. 00000000
+00004C  NAB..... 11A3B698  PNAB..... 00000000  reserved. 00000000 00000000
+000064  reserved. 00000000  reserved. 00000002  MODE..... 1120ADB0  reserved. 00000000
+000078  reserved. 00000000  reserved. 112D96E0
```

CIB for SUB1: 11A3BF88

```
+000000  11A3BF88  C3C9C240 00000000 00000000 010C0004 00000001 00000000 00030119 59C9C2D4 | CIB .....IBM
+000020  11A3BFD8 00000000 11A3C0C8 00030C89 59C3C5C5 00000001 00000005 11A3B3B8 914072E8 | .....t.H...i.CEE.....t.j.Y
+000040  11A3BFF8 00000000 11A3B538 11200298 1120C6F0 0000000B 11A3B538 00000000 00000000 | .....t.....q..F0.....t.....
+000060  11A3C018 00000000 00000000 11A64240 00000015 00000011 00000001 117537B0 00000011 | .....w.....
+000080  11A3C038 11200FE4 112139B0 00000000 11A3C0B0 114F9EBA 11A3C138 114F4338 00000007 | ..U.....t...|...tA..|.....
+0000A0  11A3C058 00000000 11A3C0F8 11A3C104 004F4327 44230000 940C9000 00000009 00000000 | .....t.8.tA..|.....m.....
+0000C0  11A3C078 00000000 11200918 11A3B538 11A3B538 11200296 00000000 0000000A 00000001 | .....t.....t.....o.....
+0000E0  11A3C098 11A3B3B8 0000000B 00000064 00000014 00000003 00000000 10000000 00000000 | .t.....
+000100  11A3C0B8 00000000 1120C908 11A3C104 11A3C0F8  E9D4C3C8 02000001 00000001 00004A48 | .....I..tA..t.8ZMCH.....
```

Dynamic save area (SUB1): 11A3B538

```
+000000  11A3B538 10A3B0F0 11A3B3B8 11A3B5F8 912001FE 11537648 11A3B668 11A3B5D0 00000001 | .t.0.t...t.8j.....t...t.....
+000020  11A3B558 1120010A 00004A48 11A3B3B8 11200BA8 11200748 11A3B528 11A3B484 11A3B0B0 | .....t.....y.....t...t.d.t...
+000040  11A3B578 11200FE4 112139B0 00000000 11A3B698 00000000 00000000 00000000 00000000 | ..U.....t.q.....
+000060  11A3B598 00000000 00000000 00000002 1120ADB0 00000000 80010000 00000000 112D96E0 | .....o.....
+000080  11A3B5B8 00000000 00000000 00000000 00000000 00000000 00000000 11A3B668 00000000 | .....
+0000A0  11A3B5D8 00000000 00000000 00180280 11A3B538 11A3B460 11200EE8 11A3B520 11A3B528 | .....t...t...Y...t...
+0000C0  11A3B5F8 11A3B484 00000001 00000000 00000000 00000000 000005D8 112008D8 112008A4 | .t.d.....Q...Q...u
+0000E0  11A3B618 11200BC8 9153CE54 115476F8 00004A48 4A480100 01A30000 11A5CB38 00000001 | ..Hj.....8.....t...v.....
+000100  11A3B638 11A5CB38 00000000 11200748 11212648 00000008 11A3B0B0 11200FE4 40404040 | .v.....t.....t.....U
+000120  11A3B658 00000000 11A3B7E0 40404040 40404040 40404040 11200C8C 11A3B5FC 40404040 | .....t.....t.....
+000140  11A3B678 11200C50 11200C50 11A3B3B8 11A3B450 11A3B498 11A3B498 11A3B498 00000010 | ...&...&t...t.&t.q.t.q.t.q....
```

DSA for SAMPLE: 11A3B3B8

```
+000000  FLAGS.... 10A3      member... B12C      BKC..... 11A3B180  FWC..... 11A3B138  R14..... 91200684
+000010  R15..... 112000D0  R0..... 11A3B520  R1..... 11A3B450  R2..... 00000001  R3..... 112004F2
+000024  R4..... 00004A48  R5..... 11A3B3B8  R6..... 11200BA8  R7..... 11200748  R8..... 11A3B528
+000038  R9..... 11A3B484  R10..... 11A3B0B0  R11..... 11200FE4  R12..... 112139B0  reserved. 00000000
+00004C  NAB..... 11A3B538  PNAB..... 00000000  reserved. 00000000 00000000
+000064  reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 00000000
+000078  reserved. 00000000  reserved. 00000000
```

Dynamic save area (SAMPLE): 11A3B3B8

```
+000000  11A3B3B8 10A3B12C 11A3B180 11A3B138 91200684 112000D0 11A3B520 11A3B450 00000001 | .t...t...t.j.d.....t...t.&....
+000020  11A3B3D8 112004F2 00004A48 11A3B3B8 11200BA8 11200748 11A3B528 11A3B484 11A3B0B0 | ...2.....t.....y.....t...t.d.t...
+000040  11A3B3F8 11200FE4 112139B0 00000000 11A3B538 00000000 00000000 00000000 00000000 | ..U.....t.....
+000060  11A3B418 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 | .....
+000080  11A3B438 00000000 00000000 00000000 00000000 00000000 00000000 11A3B484 11A3B528 | .....t.d.t...
+0000A0  11A3B458 11A3B520 00000000 00100000 00000000 00180180 11A3B3B8 00000000 11200E68 | .t.....t.....
+0000C0  11A3B478 0B300000 11200340 11A3B3B8 00000000 E38599A8 40C78993 9381A297 A8000000 | .....t.....Tery Gillaspay...
+0000E0  11A3B498 00000001 00000003 00000005 00000001 88000000 11A3B030 11A3B7D8 9140755E | .....h.....t...t.Qj. ;
+000100  11A3B4B8 112008A8 112008A4 112008D0 11A3B0F0 914072E8 112014D8 4A480100 01000000 | ...y...u.....t.0j.Y...Q.....
+000120  11A3B4D8 11A5CB38 00000001 112014D8 00000000 11A3B308 112139B0 00000000 11A3B538 | .v.....Q.....t.....t...
+000140  11A3B4F8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 11200C8C | .....
+000160  11A3B518 11A3B4A4 00000000 11A3B498 11200C50 11A3B488 000D0002 11A3B4A0 00000000 | .t.u....t.q...&t.h.....t.....
```

ENTERPRISE PL/I OPTIONS:

```
AFP, ARCH( 5), BACKREG(5), BIFPREC(15), CHECK(NOCONFORMANCE, NOSTORAGE), CMPAT(V2), CODEPAGE( 1140), COMMON,
NOCOMPACT, CSECT, CSECTCUT( 4) CURRENCY( $), NODBCS, DECIMAL( FOFLONASGN, NOFORCEDSIGN), DEFAULT( IBM, ASSIGNABLE,
NOINITFILL, NONCONNECTED, DESCRIPTOR, DESCLOCATOR, DUMMY(ALIGNED), ORDINAL(MIN), BYADDR, RETURNS(BYADDR),
LINKAGE(OPTLINK), NORETCODE, NOINLINE, ORDER, NOOVERLAP, NONRECURSIVE, ALIGNED, NULL370, EVENDEC, SHORT(HEXADEC),
EBCDIC, HEXADEC, NATIVE, NATIVEADDR, E(HEXADEC) ), DISPLAY(WTO), NODLLIMIT, EXTRN(FULL), NOGRAPHIC, NOINITAUTO,
NOINITBASED, NOINITCTL, NOINITSTATIC, NOINTERRUPT, LIMITS( EXTNAME( 7), FIXEDBIN( 31, 31), FIXEDDEC( 15, 15), NAME( 100
)), OPTIMIZE( 0), PREFIX( CONVERSION, FIXEDOVERFLOW, INVALIDOP, OVERFLOW, PRECTYPE(ANS), NOSIZE, NOSTRINGRANGE,
NOSTRINGSIZE, NOSUBSCRIPTRANGE, UNDERFLOW, ZERODIVIDE), REDUCE, NORENT, RESEXP, RESPECT(), RULES(IBM), NOSTDSYS,
NOSCHEDULER, STRINGOFGGRAPHIC(GRAPHIC), SYSTEM(MVS), TEST(ALL, $SYM, HOOK, NOSEPARATE), TUNE( 5), USAGE( ROUND(IBM),
UNSPEC(IBM)), WIDEXCHAR(BIGENDIAN), WINDOW( 1950), WRITABLE, XINFO(NODEF, NOXML)
```

Static for procedure SAMPLE Timestamp: 2007.01.31 16:02:31 V03.R06.M00: 11200BA8

```
+000000  11200BA8 11201750 11200C8C 000D0002 00000000 02020240 00000003 02020240 0000000B | ...&.....
+000020  11200BC8 02020240 00000000 02020240 0000000F 02020240 00000021 00000000 11200C00 | .....
+000040  11200BF8 00000000 11200CA8 00000000 11200E18 00000000 11200C00 00000000 11200C08 | .....y.....
+000060  11200C08 00000000 00000000 31010001 00000000 00000000 00000000 31000001 00000000 | .....
```

Figure 133. Language Environment dump from example Enterprise 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 Enterprise 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 19, and SUB1 raised an exception at statement 27, PU offset X'1C6'.
4. Find the statement in the listing for SUB1 that raised the ZERODIVIDE condition. If SUB1 was compiled with GOSTMT and SOURCE, find statement 27 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'1C6' in the object listing for this routine, shown in Figure 132 on page 335. Either method shows that *divisor* was loaded into register 2 (r2) and 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'11A3B450'. This is the address of the parameter list, which is located in static storage.
7. Find the parameter list in the stack frame; the address of the first parameter is X'11A3B484' and 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 9. 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 134 on page 340 shows an example Language Environment traceback

written to the CESE transient data queue. Data unnecessary for this example has been replaced by ellipses.

```

L320DIVZ 20070116180722 CEE3211S The system detected a decimal-divide exception (System Completion Code=0CB).
L320DIVZ 20070116180722     From compile unit DIVZERO2 at entry point DIVZERO2 at compile unit offset +0000039A
                             at entry offset +0000039A at address
L320DIVZ 20070116180722     262E139A.
L320DIVZ 20070116180722 CEE3DMP V1 R9.0: Condition processing resulted in the unhandled condition. 01/16/07 6:07:22 PM Page:1
L320DIVZ 20070116180722 Task Number: 0860 Transaction ID: DIVZ
L320DIVZ 20070116180722
L320DIVZ 20070116180722 CEE3845I CEEDUMP Processing started.
L320DIVZ 20070116180722
L320DIVZ 20070116180722 Information for enclave DIVZERO2
L320DIVZ 20070116180722
L320DIVZ 20070116180722 Information for thread 8000000000000000
L320DIVZ 20070116180722
L320DIVZ 20070116180722 Traceback:
L320DIVZ 20070116180722 DSA Entry E Offset Statement Load Mod Program Unit Service Status
L320DIVZ 20070116180722 1 CEEHDSP +000049D6 CEEHDSP D1908 Call
L320DIVZ 20070116180722 2 IGZCDSP -007BDDF6 IGZCDSP Call
L320DIVZ 20070116180722 3 DIVZERO2 +0000039A DIVZERO2 Exception
L320DIVZ 20070116180722 4 IGZCEV5 +0000066A IGZCEV5 Call
L320DIVZ 20070116180722 5 CEECRINV +00000480 CEECRINV D1908 Call
L320DIVZ 20070116180722 6 CEECRINI +00000AF6 CEECRINI D1908 Call
L320DIVZ 20070116180722
L320DIVZ 20070116180722 DSA DSA Addr E Addr PU Addr PU Offset Comp Date Compile Attributes
L320DIVZ 20070116180722 1 2610CDB0 266BDB08 266BDB08 +000049D6 20061215 CEL
L320DIVZ 20070116180722 2 26108B90 26E72398 26E72398 -007BDDF6 20061213 LIBRARY
L320DIVZ 20070116180722 3 26109BA8 262E1000 262E1000 +0000039A ***** COBOL
L320DIVZ 20070116180722 4 261089B8 25DDE000 25DDE000 +0000066A 20061213 LIBRARY
L320DIVZ 20070116180722 5 26108810 266B6DE0 266B6DE0 +00000480 20061215 CEL
L320DIVZ 20070116180722 6 26108790 266B60B8 266B60B8 +00000AF6 20061214 CEL
L320DIVZ 20070116180722
L320DIVZ 20070116180722 Condition Information for Active Routines
L320DIVZ 20070116180722 Condition Information for DIVZERO2 (DSA address 26109BA8)
L320DIVZ 20070116180722 CIB Address: 2610D6D0
L320DIVZ 20070116180722 Current Condition:
L320DIVZ 20070116180722 CEE3211S The system detected a decimal-divide exception (System Completion Code=0CB).
L320DIVZ 20070116180722 Location:
L320DIVZ 20070116180722 Program Unit: DIVZERO2 Entry: DIVZERO2 Statement: Offset: +0000039A
L320DIVZ 20070116180722 Machine State:
L320DIVZ 20070116180722 ILC..... 0006 Interruption Code..... 000B
L320DIVZ 20070116180722 PSW..... 079D0000 A62E13A0
L320DIVZ 20070116180722 GPR0..... 00000000 GPR1..... 262E11E1 GPR2..... 000617FC GPR3..... A62E1366
L320DIVZ 20070116180722 GPR4..... 262E1054 GPR5..... 00000000 GPR6..... 26105778 GPR7..... 25DDEFFF
L320DIVZ 20070116180722 GPR8..... 261031B8 GPR9..... 26109DB0 GPR10..... 262E1088 GPR11..... 262E1262
L320DIVZ 20070116180722 GPR12..... 262E1080 GPR13..... 26109BA8 GPR14..... A62E1374 GPR15..... A6E72398
L320DIVZ 20070116180722
L320DIVZ 20070116180722 Storage dump near condition, beginning at location: 262E138A
L320DIVZ 20070116180722 +000000 262E138A 92409169 D2CC916A 9169D201 D180A155 FD10D180 A157F300 9090D180 96F09090 =>
|k j.K.j.j.K.J.....J...J.o0..|
L320DIVZ 20070116180722 GPREG STORAGE:
L320DIVZ 20070116180722 Storage around GPR0 (00000000)
L320DIVZ 20070116180722 +0000 00000000 Inaccessible storage.
L320DIVZ 20070116180722 +0020 00000020 Inaccessible storage.
L320DIVZ 20070116180722 +0040 00000040 Inaccessible storage.
.
.
.

```

Figure 134. 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 DFHDMPA 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(OPTP) 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 135. 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-05:58:55.264333923 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 00001B00 87500020 00000000 06C03800 00000000 *.....&.....g&.....{.....*
0020 00140050 00000000 00000000 0678FABC *..&..... *

FT Lang.Env. 1014 CEEZCREN EVENT - CEEVNT-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 - CEEVNT-ID(OTPT) 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 *..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 *..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 - CEEVNT-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 0000C9C2 D4404040 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.....QCESO..... *
0020 D8C3C5E2 C5000000 00000000 00000000 *QCESE..... *

FT Lang.Env. 1008 CEECRINV EVENT - CEEVNT-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 *..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 06878DEC 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 *..... *

```

Figure 136. 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 23 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 23. 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 24. 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 24. 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 25. 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 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 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:
 - ALL31(ONIOFF)
 - CBLPSHPOP(ONIOFF)
 - CHECK(ONIOFF)
 - INFMSGFILTER(ONIOFF)
 - RPTOPTS(ONIOFF)
 - RPTSTG(ONIOFF)
 - TERMTHDACT(QUIETIMSGITRACE|DUMPIUAONLY|UATRACE|UADUMPIUAIMM)
 - TRAP(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. When the stack is modified to BELOW, it will remain below for the duration of the region, even if you set ALL31 back to ON. A warning message, asking if you want to continue, is presented on the panel if the run-time option ALL31 is set to OFF or CBLPSHPOP, RPTOPTS, and RPTSTG are set 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 10. 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, see 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.

When the GONUMBER compiler option is used with LP64, it will produce executables with additional debug information. This is used by Language Environment to produce statement numbers in the Language Environment dump (CEEDUMP). Statement numbers in the CEEDUMP are also produced if the DEBUG compiler option or the c89 -g option is used.

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 DYNDUMP, HEAPCHK, TERMTHDACT, TRACE, and TRAP options affect exception handling.

The following Language Environment run-time options affect debugging:

CEEDUMP	Specifies options to control the processing of the Language Environment dump report.
----------------	--

DYNDUMP	Provides a way to obtain IPCS readable dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement
HEAPCHK	Determines whether additional heap check tests are performed.
INFMSGFILTER	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 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 353.
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 137 on page 353 shows a sample options report.

```

Options Report for Enclave main Wed Dec 13 19:21:04 2006
Language Environment V01 R10.00
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    HEAPPOOLS(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    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    INFMSGFILTER(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 137. 64-bit options report

Controlling storage allocation

The following run-time options control storage allocation:

- HEAP64
- HEAPPOOLS
- 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 138 on page 354 shows a sample storage report.

Storage Report for Enclave main Tue Jan 8 16:52:36 2008
Language Environment V01 R10.00

```
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
  Largest used by any thread:  0M
  Number of increments allocated: 0
64bit User HEAP statistics:
  Initial size:                1M
  Increment size:              1M
  Total heap storage used:      983808
  Suggested initial size:      1M
  Successful Get Heap requests: 11
  Successful Free Heap requests: 0
  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): 243352
  Successful Get Heap requests: 58
  Successful Free Heap requests: 0
  Number of segments allocated: 9
  Number of segments freed:    0
24bit User HEAP statistics:
  Initial size:                4096
  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 Library HEAP statistics:
  Initial size:                1M
  Increment size:              1M
  Total heap storage used:      3795584
  Suggested initial size:      4M
  Successful Get Heap requests: 384
  Successful Free Heap requests: 337
  Number of segments allocated: 2
  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 138. 64-bit storage report (Part 1 of 4)

```

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): 9616
  Successful Get Heap requests:        27
  Successful Free Heap requests:       19
  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): 3032
  Successful Get Heap requests:        14
  Successful Free Heap requests:       6
  Number of segments allocated:        1
  Number of segments freed:           0
HEAPOOLS Statistics:
  Pool 1 size: 8 Get Requests:         0
  Pool 2 size: 32 Get Requests:        1
  Successful Get Heap requests: 17- 24 1
  Pool 3 size: 128 Get Requests:       0
  Pool 4 size: 256 Get Requests:       0
  Pool 5.1 size: 1024 Get Requests:    225
  Pool 5.2 size: 1024 Get Requests:    0
  Pool 5.3 size: 1024 Get Requests:    0
  Successful Get Heap requests: 273- 280 225
  Pool 6 size: 2048 Get Requests:      0
  Requests greater than the largest cell size: 0
HEAPOOLS Summary:
  Specified Cell Size  Element Size  Extent Percent  Cells Per Extent  Extents Allocated  Maximum Cells Used  Cells In Use
-----
      8          16         10         409         0         0         0         0
     32          40         10         163         1         1         1         1
    128         136         10          48         0         0         0         0
    256         264         10          24         0         0         0         0
   1024        1032         10           4         57        225        225
   1024        1032         10           4         0         0         0
   1024        1032         10           4         0         0         0
   2048        2056         10           4         0         0         0
-----
Suggested Percentages for current Cell Sizes:
HEAPP(ON,8,1,32,1,128,1,256,1,(1024,3),90,2048,1,0)
Suggested Cell Sizes:
HEAPP(ON,24,,280,,2048,,0)

```

Figure 138. 64-bit storage report (Part 2 of 4)

```

HEAPPOOLS64 Statistics:
Pool 1 size: 8 Get Requests: 2
  Successful Get Heap requests: 1- 8 2
Pool 2 size: 32 Get Requests: 240
  Successful Get Heap requests: 9- 16 12
  Successful Get Heap requests: 17- 24 227
  Successful Get Heap requests: 25- 32 1
  Successful Get Heap requests: 33- 40 4
  Successful Get Heap requests: 41- 48 1
  Successful Get Heap requests: 49- 56 1
  Successful Get Heap requests: 57- 64 4
  Successful Get Heap requests: 65- 72 3
  Successful Get Heap requests: 73- 80 7
  Successful Get Heap requests: 81- 88 2
  Successful Get Heap requests: 89- 96 88
  Successful Get Heap requests: 105- 112 3
  Successful Get Heap requests: 113- 120 8
  Successful Get Heap requests: 121- 128 4
Pool 4 size: 256 Get Requests: 53
  Successful Get Heap requests: 129- 136 2
  Successful Get Heap requests: 137- 144 2
  Successful Get Heap requests: 145- 152 2
  Successful Get Heap requests: 153- 160 8
  Successful Get Heap requests: 161- 168 1
  Successful Get Heap requests: 169- 176 2
  Successful Get Heap requests: 177- 184 3
  Successful Get Heap requests: 185- 192 6
  Successful Get Heap requests: 193- 200 5
  Successful Get Heap requests: 201- 208 5
  Successful Get Heap requests: 209- 216 2
  Successful Get Heap requests: 217- 224 1
  Successful Get Heap requests: 225- 232 5
  Successful Get Heap requests: 233- 240 6
  Successful Get Heap requests: 249- 256 3
Pool 5.1 size: 1024 Get Requests: 2
Pool 5.2 size: 1024 Get Requests: 2
Pool 5.3 size: 1024 Get Requests: 0
  Successful Get Heap requests: 281- 288 2
  Successful Get Heap requests: 521- 528 1
  Successful Get Heap requests: 713- 720 1
Pool 6 size: 2048 Get Requests: 2
  Successful Get Heap requests: 1505- 1512 1
  Successful Get Heap requests: 1641- 1648 1
Pool 7 size: 3072 Get Requests: 2
  Successful Get Heap requests: 2073- 2080 1
  Successful Get Heap requests: 2105- 2112 1
Pool 8 size: 4096 Get Requests: 1
  Successful Get Heap requests: 3681- 3688 1
Pool 9 size: 8192 Get Requests: 0
Pool 10 size: 16384 Get Requests: 0
Pool 11 size: 32768 Get Requests: 0
Pool 12 size: 65536 Get Requests: 0
Requests greater than the largest cell size: 0

```

Figure 138. 64-bit storage report (Part 3 of 4)

```

HEAPPOOLS64 Summary:
Specified Element Cells Per Extents Maximum Cells In
Cell Size Size Extent Allocated Cells Used Use
-----
      8      32      4000         1         1         0
     32      48      2000         1       226       225
    128     144       700         1        83        77
    256     272       350         1         1         0
   1024    1040        34         1         2         2
   1024    1040        34         1         2         0
   1024    1040        34         0         0         0
   2048    2064        50         1         2         2
   3072    3088        50         1         2         2
   4096    4112        50         1         1         0
   8192    8208        25         0         0         0
  16384   16400        10         0         0         0
  32768   32784         5         0         0         0
 65536   65552         5         0         0         0
-----
Suggested Cell Sizes:
HP64(ON,
    40,,80,,96,,128,,168,,224,,
    288,,528,,720,,1648,,2112,,3688,)
Largest number of threads concurrently active:      6
End of Storage Report

```

Figure 138. 64-bit storage report (Part 4 of 4)

Storage statistics for AMODE 64 applications

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. See the descriptions of the run-time options in *z/OS Language Environment Programming Reference* for information about rounding.

Stack storage statistics for AMODE 64 applications

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

STACK64 and THREADSTACK64 statistics:

- Initial size—the actual size of the initial stack area assigned to each thread. If a pthread-attributes-table is provided on the invocation of pthread-create, the stack size specified in the pthread-attributes-table takes precedence over the stack run-time options.
- Increment size—the size of each incremental stack area made available, 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 increments allocated—the number of incremental segments allocated by all threads.

Determining the applicable threads: If the application is not a multithreading application, the STACK64 statistics are for the one and only thread that executed, and the THREADSTACK64 statistics are all zero.

If the application is a multithreading application, and `THREADSTACK64` was not suppressed, the `STACK64` statistics are for the initial thread (IPT), and the `THREADSTACK64` statistics are for the other threads. However, if `THREADSTACK64` was suppressed, the `STACK64` statistics are for all of the threads, initial and other.

Allocating stack storage: The allocation of the stack for each thread, including the initial processing thread (IPT), is part of a storage request to the system when the thread is first created. Other storage, not part of the stack, is also acquired at this time. These storage allocations are not shown in the storage report. The size of the stack portion of this storage is the stack maximum size plus a one megabyte (1M) guard area. After allocation, the guard area follows the stack initial size and runs through the end of the stack maximum size plus the 1M guard area. Increments to the stack for each thread do not result in additional storage requests to the system. They result in the movement of the beginning of the guard area no further than the maximum size of the stack. The stack initial, increment, and maximum sizes are controlled through the `STACK64` and `THREADSTACK64` run-time options.

Heap storage statistics

Language Environment heap storage 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. The heap resources have 64-bit, 31-bit, and 24-bit addressable areas, each of which can be tuned separately.

HEAP64, LIBHEAP64, and IOHEAP64 statistics:

- Initial size—the default initial allocation, as specified by the corresponding run-time option.
- Increment size—the minimum incremental allocation, as specified by the corresponding run-time 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 explicitly freed, but were freed implicitly during enclave termination. The number of incremental segments individually freed could be less than the number allocated if the segments were not explicitly freed, but were freed implicitly during enclave termination. The initial segment is included in *Number of segments allocated* for each 31-bit and 24-bit addressable heap resource, and for the 64-bit addressable `IOHEAP64` resource. A disposition of `KEEP` always causes 0 to be reported for the *Number of segments freed*. These statistics, in all cases, specify totals for the entire enclave.

Heap pools storage statistics

The `HEAPPOOLS` and `HEAPPOOLS64` run-time options 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 heap pools storage statistics in the storage report, see “Language Environment storage report with heap pools statistics” on page 501.

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 <code>abend</code> .
<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: <ul style="list-style-type: none">• 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 <code>abend</code> 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 *z/OS Language Environment Programming Reference*.

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 362).

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 139. 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 11. 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. For more information, see *z/OS MVS Programming: Extended Addressability Guide*.
- 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 353 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 26 on page 364 lists common error symptoms, possible causes, and programmer responses.

Table 26. 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. See the messages and codes book of the operating system.
System abend with TRAP(ON)	System-detected error	See 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`. If the `DYNDUMP` run-time option is used in combination with the `TERMTHDACT` run-time option, the system dump can be written without the ddname specified. System dumps show the memory state at the time of the condition. See “Generating a system dump” on page 386 for more information about system dumps.

Chapter 12. 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 27. 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 27. 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:

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

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

- BLOCKS
- CONDITION
- ENCLAVE(ALL)
- FILES

- FNAME(CEEDUMP)
- GENOPTS
- NOENTRY
- STACKFRAME(ALL)
- STORAGE
- THREAD(CURRENT)
- TRACEBACK
- VARIABLES

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=00000009`
(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 472.

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 142 on page 373 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 381.

The CEE3DMP was generated by the C program CELQSAMP shown in Figure 140 on page 371. CELQSAMP uses the DLL CELQDLL shown in Figure 141 on page 373.

```

#pragma options(SERVICE("1.8"),NOOPT,GONUM)
#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 140. 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 140. The C program CELQSAMP (Part 2 of 2)


```

/* DLL containing div_zero */
#pragma options(SERVICE("1.8"),NOOPT,GONUM)
#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 141. 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 381.

```

[1] CEE3DMP V1 R9.0: Condition processing resulted in the unhandled condition.          Wed Jan 17 21:19:59 2007          Page: 1
    ASID: 0028  Job ID: JOB00051  Job name: CELQSAMP  Step name: STEP1  PID: 67108872  Parent PID: 1  User name: IBMUSER

[2] CEE3845I CEEDUMP Processing started.

[3] Information for enclave main

[4] Information for thread 253E019000000000

[5] Traceback:
    DSA  Entry      E Offset  Statement  Load Mod  Program Unit  Service  Status
    1    CEEHDSP    +00000000  CELQLIB    CELQLIB    CEEHDSP      D1908   Call
    2    CEEOSIGJ   +0000094E  CELQLIB    CELQLIB    CEEOSIGJ     D1908   Call
    3    CELQHR0D   +0000024E  CELQLIB    CELQLIB    CELQHR0D     D1908   Call
    4    CEEOSIGG   +1B032E48  CELQLIB    CELQLIB    CEEOSIGG     D1908   Call
    5    CELQHR0D   +0000024E  CELQLIB    CELQLIB    CELQHR0D     D1908   Call
    6    div_zero   +0000004E  15         CELQDLL    CELQDLL     1.4.f   Exception
    7    main       +00000468  98         CELQSAMP   CELQSAMP    1.2.d   Call
    8    CELQINIT   +0000134C  CELQLIB    CELQLIB    CELQINIT     D1908   Call

    DSA  DSA Addr  E Addr  PU Addr  PU Offset  Comp Date  Compile Attributes
    1    00000001082FAAC0  00000000251B6060  00000000251B6060  00000000  20061215  CEL  POSIX  XPLINK  EBCDIC  HFP
    2    00000001082FD3E0  000000002504AAB0  000000002504AAB0  0000094E  20070109  CEL  POSIX  XPLINK  EBCDIC  HFP
    3    00000001082FDDE0  00000000251C9480  00000000251C9480  0000024E  20061215  CEL  POSIX  XPLINK  EBCDIC  HFP
    4    00000001082FE020  00000000253D11F8  00000000253D11F8  1B032E48  20061215  CEL  POSIX  XPLINK  EBCDIC  HFP
    5    00000001082FEE40  00000000251C9480  00000000251C9480  0000024E  20061215  CEL  POSIX  XPLINK  EBCDIC  HFP
    6    00000001082FF080  000000002575B5A0  0000000000000000  *****  20070117  C/C++  POSIX  XPLINK  EBCDIC  IEEE
    7    00000001082FF180  00000000250000D8  0000000000000000  *****  20070117  C/C++  POSIX  XPLINK  EBCDIC  IEEE
    8    00000001082FF280  0000000025005010  0000000025005010  0000134C  20061215  CEL  POSIX  XPLINK  EBCDIC  HFP

    Fully Qualified Names
    DSA  Entry      Program Unit  Load Module
    6    div_zero    PLPSC:/'POSIX.CRTL.C(CELQDLL)'  CELQDLL
    7    main       PLPSC:/'POSIX.CRTL.C(CELQSAMP)'  CELQSAMP

```

Figure 142. Example dump using CEE3DMP (Part 1 of 9)

```

[6] Condition Information for Active Routines
Condition Information for PLPSC:/'POSIX.CRTL.C(CELQDLL)' (DSA address 0000001082FF080)
CIB Address: 0000001082FBE00
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: PLPSC:/'POSIX.CRTL.C(CELQDLL)'
  Entry:      div_zero Statement: 15 Offset: +0000004E
Machine State:
  ILC..... 0002      Interruption Code..... 0009
  PSW..... 0785240180000000 00000002575B5F0
  GPR0.... 0000000000000000 GPR1.... 000000100009DF0 GPR2.... 0000001082FF278 GPR3.... 000000000000012
  GPR4.... 0000001082FF080 GPR5.... 0000000000000C0 GPR6.... 000000000000000 GPR7.... 000000000000001
  GPR8.... 00000002575B5AC GPR9.... 00000002575B638 GPR10... 0000000250014B0 GPR11... 000000108FC5E70
  GPR12... 000000100005340 GPR13... 000000000006F58 GPR14... 000000025250098 GPR15... 00000000000001F

Storage dump near condition, beginning at location(00000002575B5DE)
+0000 000000002575B5DE 0700E300 48C00014 A7690001 8E600020 |..T....x....-..|
+0010 000000002575B5EE 1D00B904 00075000 48C0E320 48C00014 |.-...&...T....|
GPREG STORAGE:
Storage around GPR0 (0000000000000000)
+0000 0000000000000000 Inaccessible storage.
+0010 0000000000000010 Inaccessible storage.
+0020 0000000000000020 Inaccessible storage.
+0030 0000000000000030 Inaccessible storage.
+0040 0000000000000040 Inaccessible storage.
+0050 0000000000000050 Inaccessible storage.
Storage around GPR1 (000000100009DF0)
-0020 000000100009DD0 00000000 00000000 00000000 00000000 |.....|
-0010 000000100009DE0 - +FFFFFF 000000100009DEF same as above
:
Storage around GPR15(000000000000001F)
-001F 0000000000000000 Inaccessible storage.
-000F 0000000000000010 Inaccessible storage.
+0001 0000000000000020 Inaccessible storage.
+0011 0000000000000030 Inaccessible storage.
+0021 0000000000000040 Inaccessible storage.
+0031 0000000000000050 Inaccessible storage.

[7] Parameters, Registers, and Variables for Active Routines:
div_zero (DSA address 0000001082FF080):
DOWNSTACK DSA
Saved Registers:
  GPR0.... ***** GPR1.... ***** GPR2.... ***** GPR3.... *****
  GPR4.... 0000001082FF080 GPR5.... BBBB BBBB BBBB BBBB GPR6.... 0000000251C9480 GPR7.... 000000000000001
  GPR8.... 00000002575B5AC GPR9.... 00000002575B638 GPR10... 0000000250014B0 GPR11... 000000108FC5E70
  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 (0000001082FF080)
+0800 0000001082FF880 00000001 082FF180 00000001 083710A0 |.....1.....|
+0810 0000001082FF890 00000000 2575B5A0 00000000 25000542 |.....|
+0820 0000001082FF8A0 00000000 250000E4 00000000 25000658 |.....U.....|
+0830 0000001082FF8B0 00000000 250014B0 00000001 08FC5E70 |.....;.....|
+0840 0000001082FF8C0 00000001 00005340 00000000 00006F58 |.....?.....|
+0850 0000001082FF8D0 00000000 25250098 00000000 0000001F |.....q.....|
:
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 0000001082FF180):
DOWNSTACK DSA
Saved Registers:
  GPR0.... ***** GPR1.... ***** GPR2.... ***** GPR3.... *****
  GPR4.... 0000001082FF180 GPR5.... 0000001083710A0 GPR6.... 00000002575B5A0 GPR7.... 000000025000542
  GPR8.... 0000000250000E4 GPR9.... 000000025000658 GPR10... ***** GPR11... *****
  GPR12... ***** GPR13... ***** GPR14... ***** GPR15... *****

```

Figure 142. Example dump using CEE3DMP (Part 2 of 9)

```

GPREG STORAGE:
Storage around GPR0 is invalid.
Storage around GPR1 is invalid.
:
:
CELQINIT (DSA address 0000001082FF280):
DOWNSTACK DSA
Saved Registers:
GPR0..... ***** GPR1..... ***** GPR2..... ***** GPR3..... *****
GPR4..... 0000001082FF280 GPR5..... 000000108300070 GPR6..... 0000000250000D8 GPR7..... 00000002500635E
GPR8..... 000000000006FF0 GPR9..... 000000025002E98 GPR10..... ***** GPR11..... *****
GPR12..... ***** GPR13..... ***** GPR14..... ***** GPR15..... *****
GPREG STORAGE:
Storage around GPR0 is invalid.
Storage around GPR1 is invalid.
:
:
[8] Control Blocks for Active Routines:
DSA for CEEHDSP: 0000001082FB2C0
+000000 R4..... 0000001082FD3E0 R5..... 0000000251BABA0 R6..... 0000000251B6060
+000018 R7..... 00000002504B400 R8..... 0000001089135B0 R9..... 000000000000005
+000030 R10..... 0000001082FE3DF R11..... 0000001082FE0C0 R12..... 0000001089135B0
+000048 R13..... 0000001082FE680 R14..... 00000002504B300 R15..... 00000002530A300
+000060 reserved. 000000000000000 reserved. 000000000000000 HPTRAN... 000000000000000
+000078 reserved. 000000000000000 reserved. 000000000000000
DSA for CEEOSIGJ: 0000001082FDBE0
+000000 R4..... 0000001082FDDE0 R5..... 00000002504C3EC R6..... 00000002504AAB0
+000018 R7..... 0000000251C96D0 R8..... 000000025754AD8 R9..... 000000025754B00
+000030 R10..... 000000025754A30 R11..... 000000000000020 R12..... 000000100007B18
+000048 R13..... 0000001082FE680 R14..... 0000000253D6504 R15..... 000000000000003
+000060 reserved. 082FE3C400000001 reserved. 082FDD800000001 HPTRAN... 082FDDF00000001
+000078 reserved. 082FE13C00000001 reserved. 082FDDF00000001
DSA for CELQHROD: 0000001082FE5E0
+000000 R4..... E3D9C1D5E3D9C1D5 R5..... CCCCCCCCCCCCCC R6..... 0000000251C9480
+000018 R7..... 000000000000000 R8..... 000000000000000 R9..... 00000000119B648
+000030 R10..... 0000001082FF01F R11..... 000000119B00050 R12..... 000000100007B18
+000048 R13..... 0000001082FF6E0 R14..... 0000000253D3012 R15..... 0000000007FF050
+000060 reserved. 000000100007B18 reserved. 0000001082FF6E0 HPTRAN... 0000001082FE708
+000078 reserved. 00000000284F9CA reserved. 0000001082FE708
DSA for CELQHROD: 0000001082FE708
+000000 EYE..... 64INTRPT TRTYPE... 0000010 reserved. 4040404 reserved. 404040404040404
+000018 reserved. 404040404040404 reserved. 404040404040404 reserved. 404040404040404
+000030 TRANEP... 0000000251C9480 TR_R0... 404040404040404 TR_R1... 404040404040404
+000048 TR_R2... 404040404040404 TR_R3... 404040404040404 TR_R4... 0000001082FE020
+000060 TR_R5... 404040404040404 TR_R6... 404040404040404 TR_R7... 000000000000003
+000078 TR_R8... 0000001082FEBD0 TR_R9... 0000001082FEBD8 TR_R10... 0000001082FECB0
+000090 TR_R11... 00000007F754910 TR_R12... 0000001082FEBD0 TR_R13... 0000001082FEB08
+0000A8 TR_R14... 0000001082FE7D8 TR_R15... 404040404040404 reserved. 000000000000000
+0000C0 ROND_DSA. 0000001082FDDE0 ROND_R13. 0000000257549A0 ROND_R14. 0000000253D6504
DSA for CEEOSIGG: 0000001082FE820
+000000 R4..... 0000001082FEE40 R5..... 0000000253D4114 R6..... 0000000253D11F8
+000018 R7..... 0000000251C96D0 R8..... 000000025754190 R9..... 000000025754198
+000030 R10..... 000000025754110 R11..... 000000000000020 R12..... 000000100007B18
+000048 R13..... 0000001082FF6E0 R14..... 0000000251CCBF8 R15..... 000000000000001
+000060 reserved. 0000001082FEC28 reserved. 0000001082FEAA4 HPTRAN... 0000001082FEB30
+000078 reserved. 0000001082FEB38 reserved. 0000001082FEB38
DSA for CELQHROD: 0000001082FF640
+000000 R4..... E3D9C1D5E3D9C1D5 R5..... CCCCCCCCCCCCCC R6..... 0000000251C9480
+000018 R7..... 000000000000000 R8..... 00000002575B5AC R9..... 00000002575B638
+000030 R10..... 0000000250014B0 R11..... 000000108CF5E70 R12..... 000000100005340
+000048 R13..... 000000000006F58 R14..... 000000025250098 R15..... 00000000000001F
+000060 reserved. 0000001082FF080 reserved. 0000000000000C0 HPTRAN... 0000001082FF768
+000078 reserved. 00000002575B5DE reserved. 0000000000000C0
DSA for CELQHROD: 0000001082FF768
+000000 EYE..... 64INTRPT TRTYPE... 0000010 reserved. 0000000 reserved. 3C1000000000000
+000018 reserved. 341000000000000 reserved. 000008000000000 reserved. 0000000000000048
+000030 TRANEP... 0000000251C9480 TR_R0... 0000001082FF180 TR_R1... 0000000000000C0
+000048 TR_R2... 000000025573FF0 TR_R3... 00000002500052E TR_R4... 0000001082FF080
+000060 TR_R5... 000000025000658 TR_R6... 0000000250014B0 TR_R7... 000000000000001
+000078 TR_R8... 000000100005340 TR_R9... 000000000006F58 TR_R10... 000000025250098
+000090 TR_R11... 00000000000001F TR_R12... 000024000000001 TR_R13... 199036000000001
+0000A8 TR_R14... 199013C800000001 TR_R15... 19901F3800000001 reserved. 000000100009DF0
+0000C0 ROND_DSA. 0000001082FEE40 ROND_R13. 000000025754040 ROND_R14. 0000000251CCBF8
DSA for div_zero: 0000001082FF880
+000000 R4..... 0000001082FF180 R5..... 0000001083710A0 R6..... 00000002575B5A0
+000018 R7..... 000000025000542 R8..... 0000000250000E4 R9..... 000000025000658
+000030 R10..... 0000000250014B0 R11..... 000000108CF5E70 R12..... 000000100005340
+000048 R13..... 000000000006F58 R14..... 000000025250098 R15..... 00000000000001F
+000060 reserved. 0000001082FF918 reserved. 00000007F7547D8 HPTRAN... 000000000000000
+000078 reserved. 000000000000000 reserved. 000000000000000

```

Figure 142. Example dump using CEE3DMP (Part 3 of 9)

```

CIB for div_zero(0000001082FBE0)
+0000 00000001082FBE00 C3C9C240 00000000 00000000 00000000 |CIB .....|
+0010 00000001082FBE10 00000000 00000000 01900004 00000000 |.....|
+0020 00000001082FBE20 00000000 00000000 000300C6 59C3C5C5 |.....F.CEE|
+0030 00000001082FBE30 00000000 00000000 00000001 082FBF90 |.....|
+0040 00000001082FBE40 00030C89 59C3C5C5 00000000 00000004 |...i.CEE.....|
+0050 00000001082FBE50 00000004 00000000 00000001 082FF180 |.....1.|
+0060 00000001082FBE60 00000000 251BB1D0 00000000 00000000 |.....|
+0070 00000001082FBE70 00000001 082FF080 00000000 2575B5F0 |.....0.....0|
+0080 00000001082FBE80 00000001 00007000 00000003 00000000 |.....|
+0090 00000001082FBE90 00000001 082FF080 01010000 00000000 |.....0.....|
+00A0 00000001082FBEA0 00000000 00000000 00000000 00000000 |.....|
+00B0 00000001082FBEB0 - +0000FF 00000001082FBEFF |same as above|
+0100 00000001082FBF00 48220400 00000000 940C9000 00000009 |.....m.....|
+0110 00000001082FBF10 00000000 00000000 00000000 250008C0 |.....|
+0120 00000001082FBF20 00000001 082FF080 00000001 082FF080 |.....0.....0|
+0130 00000001082FBF30 00000000 2575B5EE 00000000 00000000 |.....|
+0140 00000001082FBF40 00000000 00000000 00000067 00000000 |.....|
+0150 00000001082FBF50 00000001 082FF180 00000003 00000008 |.....1.....|
+0160 00000001082FBF60 00000014 00000004 00000000 00000000 |.....|
+0170 00000001082FBF70 00000000 00000000 00000008 00000000 |.....|
+0180 00000001082FBF80 00000001 00007220 00000000 00000000 |.....|

DSA for main: 00000001082FF980
+000000 R4..... 00000001082FF280 R5..... 0000000108300070 R6..... 00000000250000D8
+000018 R7..... 000000002500635E R8..... 0000000000006FF0 R9..... 0000000025002E98
+000030 R10..... 00000000250014B0 R11..... 0000000108FC5E70 R12..... 0000000100005340
+000048 R13..... 0000000000006F58 R14..... 0000000025250098 R15..... 000000000000001F
+000060 reserved. 0000000000000000 reserved. 0000000000000000 HPTRAN... 0000000000000000
+000078 reserved. 0000000000000000

DSA for CELQINIT: 00000001082FFA80
+000000 R4..... 00000001082FF760 R5..... 0000000000000000 R6..... 0000000025005010
+000018 R7..... 00000000250064F8 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

[9] Storage for Active Routines:
DSA frame(00000001082FAAC0)
+0800 00000001082FB2C0 00000001 082FD3E0 00000000 251BABA0 |.....L.....|
+0810 00000001082FB2D0 00000000 251B6060 00000000 2504B400 |.....-.....|
+0820 00000001082FB2E0 00000001 089135B0 00000000 00000005 |.....j.....|
+0830 00000001082FB2F0 00000001 082FE3DF 00000001 082FE0C0 |.....T.....|
+0840 00000001082FB300 00000001 089135B0 00000001 082FE680 |.....j.....W.|
+0850 00000001082FB310 00000000 2504B300 00000000 2530A300 |.....t.....|
+0860 00000001082FB320 00000000 00000000 00000000 00000000 |.....|
+0870 00000001082FB330 - +00087F 00000001082FB33F |same as above|
:
+2540 00000001082FD000 00000000 00000000 00000000 00000000 |.....|
+2550 00000001082FD010 - +00311F 00000001082FDBDF |same as above|

DSA frame(00000001082FD3E0)
+0800 00000001082FDBE0 00000001 082FDDE0 00000000 2504C3EC |.....C.....|
+0810 00000001082FDBF0 00000000 2504AAB0 00000000 251C96D0 |.....o.....|
+0820 00000001082FDC00 00000000 25754AD8 00000000 25754BD0 |.....Q.....|
+0830 00000001082FDC10 00000000 25754A30 00000000 00000020 |.....|
:
+1180 00000001082FE560 - +0011EF 00000001082FE5CF |same as above|
+11F0 00000001082FE5D0 00000000 00000000 00000001 082FEBD8 |.....Q.....|

DSA frame(00000001082FDDE0)
+0800 00000001082FE5E0 E3D9C1D5 E3D9C1D5 CCCCCCCC CCCCCCCC |TRANTRAN.....|

DSA frame(00000001082FE020)
+0800 00000001082FE820 00000001 082FEE40 00000000 253D4114 |.....|
+0810 00000001082FE830 00000000 253D11F8 00000000 251C96D0 |.....8.....o.|
+0820 00000001082FE840 00000000 25754190 00000000 25754198 |.....q.....|
:
+1600 00000001082FF620 00000000 00000000 00000000 00000013 |.....|
+1610 00000001082FF630 00000000 252B27D0 00000000 252B2810 |.....|

DSA frame(00000001082FEE40)
+0800 00000001082FF640 E3D9C1D5 E3D9C1D5 CCCCCCCC CCCCCCCC |TRANTRAN.....|

DSA frame(00000001082FF080)
+0800 00000001082FF880 00000001 082FF180 00000001 083710A0 |.....1.....|
+0810 00000001082FF890 00000000 2575B5A0 00000000 25000542 |.....|
+0820 00000001082FF8A0 00000000 250000E4 00000000 25000658 |.....U.....|
+0830 00000001082FF8B0 00000000 250014B0 00000001 08FC5E70 |.....j.....|
+0840 00000001082FF8C0 00000001 00005340 00000000 00006F58 |.....?.....|
+0850 00000001082FF8D0 00000000 25250098 00000000 0000001F |.....q.....|
:
+08E0 00000001082FF960 34100000 00000000 00000080 00000000 |.....|
+08F0 00000001082FF970 00000000 00000048 00000000 00000000 |.....|

```

Figure 142. Example dump using CEE3DMP (Part 4 of 9)
376 z/OS V1R10.0 Language Environment Debugging Guide

```

DSA frame(00000001082FF180)
+0800 00000001082FF980 00000001 082FF280 00000001 08300070 |.....2.....|
+0810 00000001082FF990 00000000 250000D8 00000000 2500635E |.....Q.....;|
+0820 00000001082FF9A0 00000000 00006FF0 00000000 25002E98 |.....?0.....q|
+0830 00000001082FF9B0 00000000 250014B0 00000001 08FC5E70 |.....;.....|
:
+08E0 00000001082FFA60 00000001 08300070 00000000 00000000 |.....|
+08F0 00000001082FFA70 00000000 00000000 00000000 00000000 |.....|

DSA frame(00000001082FF280)
+0800 00000001082FFA80 00000001 082FF760 00000000 00000000 |.....7-.....|
+0810 00000001082FFA90 00000000 25005010 00000000 250064F8 |.....&.....8|
+0820 00000001082FFAA0 00000000 00000000 00000000 00000000 |.....|
+0830 00000001082FFAB0 - +00087F 00000001082FFAFF          same as above
:
+0C00 00000001082FFE80 00000000 00000000 00000000 00000000 |.....|
+0C10 00000001082FFE90 - +000CDF 00000001082FFF5F          same as above

[4] Information for thread 253E10A000000001

[5] Traceback:
DSA  Entry      E Offset Statement      Load Mod      Program Unit      Service Status
1    CEEOPML2    +00000000      CELQLIB        CEEOPML2        D1908  Call
2    thread_func +0000005A 24      CELQSAMP        CELQSAMP        1.2.d  Call
3    CELQPCMM    +00000DEA      CELQLIB        CELQPCMM        D1908  Call

DSA DSA Addr      E Addr      PU Addr      PU Offset  Comp Date  Compile Attributes
1    00000001111FEF60 0000000025290D80 0000000025290D80 00000000 20061215  CEL  POSIX  XPLINK  EBCDIC  HFP
2    00000001111FF2C0 00000000250005A8 0000000000000000 ***** 20070117  C/C++  POSIX  XPLINK  EBCDIC  IEEE
3    00000001111FF3C0 000000002526E6D0 000000002526E6D0 00000DEA 20061214  CEL  POSIX  XPLINK  EBCDIC  HFP

Fully Qualified Names
DSA  Entry      Program Unit      Load Module
2    thread_func  PLPSC://'POSIX.CRTL.C(CELQSAMP)'  CELQSAMP

GPR0..... 0000000000000001  GPR1..... 00000000257669A0  GPR2..... 0000000108910290  GPR3..... 00000000257669A0
GPR4..... 00000001111FEF60  GPR5..... 0000000025292620  GPR6..... 0000000000000000  GPR7..... 0000000025292004
GPR8..... 00000000DA899660  GPR9..... 00000001114013C8  GPR10.... 0000000100003CA0  GPR11.... 0000000108358750
GPR12.... 00000001114013C8  GPR13.... 000000002576F000  GPR14.... 00000001114013C8  GPR15.... 00000001807D17D8

GPREG STORAGE:
Storage around GPR0 (0000000000000001)
-0001 0000000000000000  Inaccessible storage.
+000F 0000000000000010  Inaccessible storage.
:

[8] Control Blocks for Active Routines:
DSA for CEEOPML2: 00000001111FF760
+000000 R4..... 00000001111FFB60      R5..... 000000002576F000      R6..... 0000000025290D80
+000018 R7..... 0000000025000604      R8..... 00000000250005B4      R9..... 0000000025000658
+000030 R10.... 00000001083001B8      R11.... 00000001111FFEE8      R12.... 000000007F64A75C
+000048 R13.... 000000002576F000      R14.... 0000000111401F38      R15.... 0000000120000000
+000060 reserved. 0000000000000000      reserved. 0000000000000000      HPTRAN... 0000000000000000
+000078 reserved. 0000000000000000

DSA for thread_func: 00000001111FFAC0
+000000 R4..... 00000001111FF3C0      R5..... 0000000108300070      R6..... 00000000250005A8
+000018 R7..... 000000002526F4BC      R8..... 0000000111401FA0      R9..... 000000002576F0D0
+000030 R10.... 0000000000000080      R11.... 0000000000000000      R12.... 000000007F64A75C
+000048 R13.... 0001004000000001      R14.... 00000000251D9FD4      R15.... 000000002505CF48
+000060 reserved. 0000000108FE8C30      reserved. 0000000000000000      HPTRAN... 000000007F6C58C8
+000078 reserved. 0000000000000000

DSA for CELQPCMM: 00000001111FFBC0
+000000 R4..... 00000001111FF760      R5..... 0000000000000000      R6..... 000000002526E6D0
+000018 R7..... 000000002526FC8C      R8..... 0000000111401FA0      R9..... 000000002576F0D0
+000030 R10.... 0000000000000080      R11.... 00000001111FFEE8      R12.... 000000007F64A75C
+000048 R13.... 000000002576F000      R14.... 0000000111401F38      R15.... 0000000120000000
+000060 reserved. 0000000000000000      reserved. 0000000000000000      HPTRAN... 0000000000000000
+000078 reserved. 0000000000000000

```

Figure 142. Example dump using CEE3DMP (Part 5 of 9)

```

[9] Storage for Active Routines:
DSA frame(00000001111FEF60)
+0800 00000001111FF760 00000001 111FFB60 00000000 2576F000 |.....-.....0.|
+0810 00000001111FF770 00000000 25290D80 00000000 25000604 |.....|
+0820 00000001111FF780 00000000 250005B4 00000000 25000658 |.....|
+0830 00000001111FF790 00000001 083001B8 00000001 111FFEE8 |.....Y|
:
[10] Control Blocks Associated with the Thread:
CAA(00000001114013C8)
+0000 00000001114013C8 00000000 00000000 00000000 00000000 |.....|
+0010 00000001114013D8 - +0002AF 0000000111401677 same as above
+02B0 0000000111401678 00008000 00000000 00000000 00000000 |.....|
:
[11] Enclave Control Blocks:
EDB(0000000100005340)
+0000 0000000100005340 C3C5C5C5 C4C24040 00000000 00000000 |CEEEEDB .....|
+0010 0000000100005350 00000000 00000000 00000000 00000000 |.....|
+0020 0000000100005360 - +0000FF 000000010000543F same as above
+0100 0000000100005440 97000100 00000000 00000001 000068F8 |p.....8|
:
+01A0 00000001000054E0 00000000 00000000 00000000 00000000 |.....|
+01B0 00000001000054F0 - +0001FF 000000010000553F same as above
MEML(00000001000068F8)
+0000 00000001000068F8 00000000 00000000 00000000 00000000 |.....|
+0010 0000000100006908 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+0020 0000000100006918 00000000 00000000 00000000 00000000 |.....|
+0030 0000000100006928 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+0040 0000000100006938 00000000 00000000 00000000 00000000 |.....|
:
+0190 0000000100006A88 FFFFFFFF FFFFFFFF 00000000 00000000 |.....|
+01A0 0000000100006A98 00000000 00000000 00000000 00000000 |.....|
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 08FC7470 00000001 08910900 |.....j..|
+0030 00000001089100E8 000001F0 00001F00 00000000 00000000 |...0.....|
+0040 00000001089100F8 00000001 08FC74B0 00000000 00000000 |.....|
+0050 0000000108910108 00000000 00000000 00000000 00000000 |.....|
+0060 0000000108910118 - +00014F 0000000108910207 same as above
:
+04B0 0000000108910568 - +00097F 0000000108910A37 same as above
+0980 0000000108910A38 00000001 08FC74F0 00000000 00000000 |.....0.....|
+0990 0000000108910A48 00000000 00000000 00000000 00000000 |.....|
+09A0 0000000108910A58 - +000A2F 0000000108910AE7 same as above
+0A30 0000000108910AE8 00000001 08FC73F0 00000000 00000000 |.....0.....|
+0A40 0000000108910AF8 00000001 08FC7430 00000000 00000000 |.....|
Thread Synchronization Enclave Latch Table (EPALT)(00000001089100B0)
+0000 00000001089100B0 00000000 00000000 00000000 00000000 |.....|
+0010 00000001089100B10 - +00015F 0000000108910C5F same as above
+0160 0000000108910C60 00000000 00000000 DA8ADF60 00000000 |.....-.....|
+0170 0000000108910C70 00000000 257520A0 00000001 08911118 |.....j..|
+0180 0000000108910C80 00000000 00000000 00000000 00000000 |.....|
+0190 0000000108910C90 - +00022F 0000000108910D2F same as above
:
+0B00 0000000108911600 - +000C6F 000000010891176F same as above
+0C70 0000000108911770 00000000 2538B4E0 00000000 00000000 |.....|
+0C80 0000000108911780 00000000 00000000 00000000 00000000 |.....|
+0C90 0000000108911790 - +0013FF 0000000108911EFF same as above

DLL Information:
WSA Addr      Module Addr      Thread ID      Use Count  Name
0000000108300050
0000000108371090 000000002575B000 253E019000000000 00000001 main
0000000108390510 0000000025777000 253E019000000000 00000002 CDAEQED
0000000108396110 00000000257F5000 253E019000000000 00000001 CDAEQDPI
00000001083A0430 00000000258C4000 253E019000000000 00000001 CELQDSNF

HEAPCHK Option Control Block (HCOP)(00000001089234D0)
+0000 00000001089234D0 C8C3D6D7 00000048 00000001 00000000 |HCOP.....|
+0010 00000001089234E0 00000000 0000000A 0000000A 00000000 |.....|
+0020 00000001089234F0 00000001 08FC0090 00000001 08923518 |.....k..|
+0030 0000000108923500 00000001 08A00050 00000000 00000000 |.....&.....|
+0040 0000000108923510 00000000 00000000 C8C3C6E3 00004000 |.....HCFT..|

```

Figure 142. Example dump using CEE3DMP (Part 6 of 9)

```

HEAPCHK Element Table (HCEL) for Heapid 000000100100138 :
Header(000000108FC0090)
+0000 0000000108FC0090 C8C3C5D3 00000000 00000000 00000000 |HCEL.....|
+0010 0000000108FC00A0 00000000 00000000 00000001 00100138 |.....|
+0020 0000000108FC00B0 000001F4 0000000F 00000010 00000000 |...4.....|
Address          Seg Address      Length

Table(000000108FC00C0)
+0000 0000000108FC00C0 00000001 08300040 00000001 08300000 00000000 00000180 00000001 08FC3F50
+0020 0000000108FC00E0 00000001 083001C0 00000001 08300000 00000000 00019660 00000001 08FC5B30
+0040 0000000108FC0100 00000001 08319820 00000001 08300000 00000000 00025B40 00000001 08FC5C90
+0060 0000000108FC0120 00000001 0833F360 00000001 08300000 00000000 00019340 00000001 08FC5EB0
+0080 0000000108FC0140 00000001 083586A0 00000001 08300000 00000000 000189E0 00000001 08FC6010
+00A0 0000000108FC0160 00000001 08371080 00000001 08300000 00000000 00000060 00000001 08FC7050
+00C0 0000000108FC0180 00000001 083710E0 00000001 08300000 00000000 0001F420 00000001 08FC7530
+00E0 0000000108FC01A0 00000001 08390500 00000001 08300000 00000000 00005C00 00000001 08FEC4F0
+0100 0000000108FC01C0 00000001 08396100 00000001 08300000 00000000 0000A320 00000001 08FEE1D0
+0120 0000000108FC01E0 00000001 083A0420 00000001 08300000 00000000 00000180 00000001 08FEE4F0
+0140 0000000108FC0200 00000001 083A05A0 00000001 08300000 00000000 0001F400 00000001 08FEE6F0
+0160 0000000108FC0220 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
+0180 0000000108FC0240 00000001 083B79A0 00000001 08300000 00000000 00017720 00000001 08FEEA50
+01A0 0000000108FC0260 00000001 19D00040 00000001 19D00000 00000000 00032360 00000001 08FEEC10
+01C0 0000000108FC0280 00000001 19D323A0 00000001 19D00000 00000000 000280C0 00000001 08FEEDB0
+01E0 0000000108FC02A0 00000001 19D5A460 00000001 19D00000 00000000 000321A0 00000001 08FEEF70

```

Language Environment Trace Table:

Most recent trace entry is at displacement: 000680

Displacement	Trace Entry in Hexadecimal	Trace Entry in EBCDIC
+000000	Time 21.19.55.717535 Date 2007.01.17 Thread ID... 253E019000000000	
+000010	Member ID... 01 Flags..... 000000 Entry Type..... 00001000	
+000018	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000038	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000058	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000078	40404040 40404040	
+000080	Time 21.19.55.717715 Date 2007.01.17 Thread ID... 253E019000000000	
+000090	Member ID... 01 Flags..... 000000 Entry Type..... 00001010	
+000098	C3C5C5D7 C2D24040 00000070 00000000 00000000 257669A0 00001660 00000000	CEE3PBK
+0000B8	00000001 11400000 00000000 000024D0 00000001 11400360 00000001 114013C8H
+0000D8	00000001 11401F38 00000001 11402010 00000001 08300060 00000000 25000658
+0000F8	24000000 00000000
+000100	Time 21.19.55.717821 Date 2007.01.17 Thread ID... 253E019000000000	
+000110	Member ID... 01 Flags..... 000000 Entry Type..... 00001011	
+000118	253E10A0 00000001 00000000 7F7547D8 00000000 257669A0 00001660 00000000 ".Q.....
+000138	00000001 11400000 00000000 000024D0 00000001 11400360 00000001 114013C8H
+000158	00000001 11401F38 00000001 11402010 00000001 08300060 00000000 25000658
+000178	24000000 00000000
+000180	Time 21.19.55.717823 Date 2007.01.17 Thread ID... 253E019000000000	
+000190	Member ID... 01 Flags..... 000000 Entry Type..... 000010F0	
+000198	00000000 00000000 00000000 00000000 00000000 257669A0 00001660 00000000
+0001B8	00000001 11400000 00000000 000024D0 00000001 11400360 00000001 114013C8H
+0001D8	00000001 11401F38 00000001 11402010 00000001 08300060 00000000 25000658
+0001F8	24000000 00000000
+000200	Time 21.19.55.717845 Date 2007.01.17 Thread ID... 253E019000000000	
+000210	Member ID... 01 Flags..... 000000 Entry Type..... 00001000	
+000218	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000238	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000258	40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040	
+000278	40404040 40404040	
+000280	Time 21.19.55.717974 Date 2007.01.17 Thread ID... 253E019000000000	
+000290	Member ID... 01 Flags..... 000000 Entry Type..... 00001010	
+000298	C3C5C5D7 C2D24040 00000070 00000000 00000000 257689A0 00001660 00000000	CEE3PBKi.....
+0002B8	00000001 19900000 00000000 000024D0 00000001 19900360 00000001 199013C8H
+0002D8	00000001 19901F38 00000001 19902010 00000001 08300060 00000000 25000664
+0002F8	24000000 00000000

Figure 142. Example dump using CEE3DMP (Part 7 of 9)

```

+000300 Time 21.19.55.718015 Date 2007.01.17 Thread ID... 253E019000000000
+000310 Member ID... 01 Flags..... 000000 Entry Type.... 00001011
+000318 253E1FB0 00000002 00000000 7F7547D8 00000000 257689A0 00001660 00000000 |....."..Q.....i.....|
+000338 00000001 19900000 00000000 000024D0 00000001 19900360 00000001 199013C8 |.....-.....H|
+000358 00000001 19901F38 00000001 19902010 00000001 08300060 00000000 25000664 |.....-.....|
+000378 24000000 00000000 |.....|

+000380 Time 21.19.55.718016 Date 2007.01.17 Thread ID... 253E019000000000
+000390 Member ID... 01 Flags..... 000000 Entry Type.... 000010F0
+000398 00000000 00000000 00000000 00000000 00000000 257689A0 00001660 00000000 |.....i.....|
+0003B8 00000001 19900000 00000000 000024D0 00000001 19900360 00000001 199013C8 |.....-.....H|
+0003D8 00000001 19901F38 00000001 19902010 00000001 08300060 00000000 25000664 |.....-.....|
+0003F8 24000000 00000000 |.....|

+000400 Time 21.19.55.719149 Date 2007.01.17 Thread ID... 253E1FB000000002
+000410 Member ID... 01 Flags..... 000000 Entry Type.... 00001910
+000418 253E1FB0 00000002 00000001 19901FA0 00000001 19901F38 80000000 00000000 |.....|
+000438 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000458 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000478 00000000 00000000 |.....|

+000480 Time 21.19.55.719199 Date 2007.01.17 Thread ID... 253E1FB000000002
+000490 Member ID... 01 Flags..... 000000 Entry Type.... 00001930
+000498 253E1FB0 00000002 00000001 19901FA0 00000001 19901F38 80000000 00000000 |.....|
+0004B8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0004D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0004F8 00000000 00000000 |.....|

+000500 Time 21.19.55.719713 Date 2007.01.17 Thread ID... 253E10A0000000001
+000510 Member ID... 01 Flags..... 000000 Entry Type.... 00001910
+000518 253E10A0 00000001 00000001 11401FA0 00000001 11401F38 80000000 00000000 |.....|
+000538 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000558 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000578 00000000 00000000 |.....|

+000580 Time 21.19.55.719742 Date 2007.01.17 Thread ID... 253E10A000000001
+000590 Member ID... 01 Flags..... 000000 Entry Type.... 00001930
+000598 253E10A0 00000001 00000001 11401FA0 00000001 11401F38 80000000 00000000 |.....|
+0005B8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0005D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0005F8 00000000 00000000 |.....|

+000600 Time 21.19.55.719935 Date 2007.01.17 Thread ID... 253E0190000000000
+000610 Member ID... 01 Flags..... 000000 Entry Type.... 00000700
+000618 00000000 00000000 00000000 00000000 00000000 00000000 00000001 00005340 |.....|
+000638 00000008 00400579 00000000 00000001 00000001 082FE020 00000000 253D3012 |.....|
+000658 001F2000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+000678 00000000 00000000 |.....|

+000680 Time 21.19.55.719939 Date 2007.01.17 Thread ID... 253E0190000000000
+000690 Member ID... 01 Flags..... 000000 Entry Type.... 00000701
+000698 00000001 80000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0006B8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0006D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|
+0006F8 00000000 00000000 |.....|

```

```

Heap Storage Diagnostics
All storage has been freed.
:

```

Figure 142. Example dump using CEE3DMP (Part 8 of 9)

[12] Run-Time Options Report:

LAST WHERE SET	OPTION
DD:CEEOPTS	CEEDUMP(0,SYSOUT=*,FREE=END,SPIN=UNALLOC)
Installation default	DYNDUMP(*USERID,NODYNAMIC,TDUMP)
Installation default	ENVAR("")
Installation default	FILETAG(NOAUTOCVT,NOAUTOTAG)
DD:CEEOPTS	HEAPCHK(ON,1,0,10,10)
DD:CEEOPTS	HEAPPOLLS(ON,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,10,0,10,0,10,0,10)
DD:CEEOPTS	HEAPPOLLS64(ON,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)
Programmer default	POSIX(ON)
Installation default	PROFILE(OFF,"")
DD:CEEOPTS	RPTOPTS(ON)
DD:CEEOPTS	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	THREADSTACK64(OFF,1M,1M,128M)
DD:CEEOPTS	TRACE(ON,1048576,NODUMP,LE=8)
Installation default	TRAP(ON,SPIE)

[13] Process Control Blocks:

```
PCB(0000000100003CA0)
+0000 0000000100003CA0 C3C5C5D7 C3C24040 00000000 00000000 | CEEPCB ..... |
+0010 0000000100003CB0 00000000 00000000 00000000 00000000 | ..... |
+0020 0000000100003CC0 - +0000FF 0000000100003D9F | same as above |
+0100 0000000100003DA0 03030208 00000000 00000000 00000000 | ..... |
+0110 0000000100003DB0 00000001 00004048 00000000 00000000 | ..... |
+0120 0000000100003DC0 00000000 00000000 00000000 00000000 | ..... |
+0130 0000000100003DD0 00000000 00000000 00000001 00003A10 | ..... |
+0140 0000000100003DE0 7FC00000 00000000 00000000 00000000 | " ..... |
+0150 0000000100003DF0 00000000 00000000 00000000 00000000 | ..... |
+0160 0000000100003E00 00000000 252A3F48 00000000 00000000 | ..... |
+0170 0000000100003E10 00000000 00000000 00000000 00000000 | ..... |
+0180 0000000100003E20 - +0001BF 0000000100003E5F | same as above |
MEML(0000000100004048)
+0000 0000000100004048 00000000 00000000 00000000 00000000 | ..... |
+0010 0000000100004058 - +00005F 00000001000040A7 | same as above |
+0060 00000001000040A8 00000001 00008688 00000000 00000000 | .....fh..... |
+0070 00000001000040B8 00000000 00000000 00000000 00000000 | ..... |
+0080 00000001000040C8 - +0001AF 00000001000041F7 | same as above |
Thread Synchronization Process Latch Table (PPALT)(0000000108911F00)
+0000 0000000108911F00 DA8ADF60 00000000 00000000 257520A0 | ..... |
+0010 0000000108911F10 00000001 08911050 00000000 00000000 | .....j.&..... |
+0020 0000000108911F20 00000000 00000000 00000000 00000000 | ..... |
+0030 0000000108911F30 - +00009F 0000000108911F9F | same as above |
+00A0 0000000108911FA0 DA8ADF60 00000000 00000000 257520A0 | ..... |
+00B0 0000000108911FB0 00000001 08911500 00000000 00000000 | .....j..... |
+00C0 0000000108911FC0 00000000 00000000 00000000 00000000 | ..... |
+00D0 0000000108911FD0 - +0013FF 000000010891132FF | same as above |
```

[14] CEE3846I CEEDUMP Processing completed.

Figure 142. Example dump using CEE3DMP (Part 9 of 9)

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

For CEEDUMPs produced under a batch environment, the following items are displayed:

- ASID
Describes the address space ID.
- Job ID
Describes the JES Job ID.
- Job name
Describes the job name.
- Step name:
Describes the job's step name in which the CEEDUMP was produced.
- UserID:
Describes the TSO userid who issued the job.

For jobs running with POSIX(ON), the following additional items are displayed:

- PID
Displays the associated process ID.
- Parent PID
Displays the associated parent PID.

For CEEDUMPs produced under the z/OS UNIX shell, the following items are displayed:

- ASID
Describes the address space ID.
- PID
Displays the associated process ID.
- Parent PID
Displays the associated parent PID.
- User name
Contains the userid associated to the CEEDUMP.

[2] CEE3845I CEEDUMP Processing started.

Message CEE3845I identifies the start of the Language Environment dump processing. Similarly, message CEE3846I identifies the end of the dump processing, Message number CEE3845I can be used to locate the start of the next CEEDUMP report when scanning forward in a data set that contains several CEEDUMP reports.

[3] Enclave Information

These sections show information that is specific to an enclave.

[3] Enclave Identifier

This statement names the enclave for which information in the dump is provided.

[4] - [10] Thread Information

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

[4] Information for thread

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

[5] Traceback

For all active routines in a particular thread, the traceback section shows routine information in three parts. The first part contains:

- DSA number
A number assigned to the information for this active routine by dump processing. The number is used to associate information from the first part of the traceback with information in the second and third parts 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
- Statement number
Refers to the line number in the source code (program unit) in which a call was made or an exception took place. The statement number appears only if your routine was compiled with the options required to generate statement numbers. These options are described under “XL C and XL C++ compiler options for AMODE 64 applications” on page 351.
- Load module
The load module name displayed can be a partitioned data set member or an UNIX executable file. The load module name is also displayed in the third part of the traceback (see below for details).
- 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.
If your routine was compiled with the compile options to generate statement numbers then the program unit name displayed under this column will appear as follows:
 - If your compiled routine is in a partitioned data set then only the member will be output.
 - If your compiled routine is in a sequential data set then only the last qualifier will be shown.
 - If your compiled routine is in an UNIX filename then only what fits of the filename will be displayed in a line.

Look for the complete name of the program unit in the Fully Qualified Names section of the traceback, if your routine was compiled using compile options to generate statement numbers.

- 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 used to associate information from the first part of the traceback with information in the second and third parts 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.

The third part, which is also referred to as 'Fully Qualified Names' section, contains the following:

- DSA number
- Entry
- Program unit
Similar to the Program Unit column in part 1 except that the server name and the complete program unit (PU) name will be displayed. A PU name will appear here only if it was compiled using compile options to produce statement numbers.
- Load Module
The complete pathname of a load module name residing in an UNIX filename will be displayed here if available. The load module's full pathname will be displayed if the PATH environment variable is set such that the pathname of the load module's directory appears before the current directory (.). For load modules found in data sets, the same output shown in the traceback part 1 will also be displayed here.

[6] 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. 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)
 - Storage pointed to by General Purpose Registers
 These values are the current values at the time the condition was raised.

[7] 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.

[8] 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

[9] 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.

[10] 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.

[11] 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.

[12] Run-Time Options Report

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

[13] 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.

[14] CEE3846I CEEDUMP Processing completed.

Message CEE3846I identifies the end of the Language Environment dump processing. Similarly, message CEE3845I identifies the start of the dump processing. Message number CEE3846I can be used to locate the end of the previous CEEDUMP report when scanning backward in a data set that contains several CEEDUMP reports.

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:

DYNDUMP(*hlq*,DYNAMIC,TDUMP)

You can use the DYNDUMP run-time option to

obtain IPCS readable dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement.

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 367.

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.

See 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 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 367.
2. Decide whether to include a SYSMDUMP DD card or use the DYNDUMP run-time option.
 - Include a SYSMDUMP DD card with the desired data set name and DCB information:
LRECL=4160, BLKSIZE=4160, and RECFM=FBS.

- Specify the DYNDUMP run-time option with the following information:
DYNDUMP (*hlq*,DYNAMIC,TDUMP)
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 a z/OS UNIX shell

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

- Using `_BPXK_MDUMP`

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=hlq.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 367.

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*.

- Using `DYNDUMP`

1. Specify Language Environment run-time options:

```
export _CEE_RUNOPTS="termthdact(suboption),DYNDUMP(hlq,DYNAMIC,TDUMP)"
```

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 367.
- *hlq* is the high level qualifier for the dump data set to be created.

2. Rerun the program.

When you are done, the system dump is written to the name generated by the DYNDUMP run-time option.

For additional DYNDUMP information see *z/OS Language Environment Programming 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, see *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**]
 - [**COMP**((*value*))]
 - [**PTBL**(*value*)]
 - [**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.

PTBL(*value*)

Requests that Preinit tables be formatted according to the following values:

CURRENT

If current is specified, the Preinit table associated with the current or specified TCB is displayed.

address

If an address is specified, the PreInit table at that address is specified.

- * All active and dormant PreInit tables within the current address space are displayed; this option is time-consuming.

ACTIVE

The PreInit tables for all TCBs in the address space are displayed.

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. If only
 | the HPT keyword is specified with no value, the trace behaves similar to when
 | the value is 0.

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.

COMP(value)

Requests component control blocks to be formatted according to the following values:

C Requests a report on C/C++ Run-Time control blocks.

CIO

Requests a report on C/C++ I/O control blocks.

COBOL

Requests a report on COBOL-specific control blocks.

ALL

Requests a report on all the previous control blocks.

If the value specified in COMP is not one of the values (C, CIO, COBOL, or ALL), a message is displayed and it continues executing as if COMP(ALL) was specified.

Note: The ALL parameter for LEDATA also generates a report that includes all the component control blocks.

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 409.

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.

Usage Note: Only hexadecimal characters can be specified as addresses provided in LEDATA parameters. Special characters cause the formatter to fail. Therefore, in order to specify a 64 bit address as a parameter, it must be in the form like 123456789 instead of 1_23456789.

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 143 on page 394 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 140 on page 371. “Sections of the Language Environment LEDATA Verbexit formatted output” on page 403 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 403.

```

ALL
*****
64 BIT LANGUAGE ENVIRONMENT DATA
*****

Language Environment Product 04 V01 R09.00

[1] Information for enclave main

[2] Information for thread 253E019000000000
TCB Address: 007FF050
CAA Address: 00000001_00007B18
PCB Address: 00000001_00003CA0

[3] Registers and PSW:
GPR0..... 0000000084000000 GPR1..... 0000000084000FC7 GPR2..... 00000001082FBE00 GPR3..... 3C10000000000000
GPR4..... 00000001082FA900 GPR5..... 00000000253C45F8 GPR6..... 00000000253C4500 GPR7..... 00000000251B9B1A
GPR8..... 00000001082FBE00 GPR9..... 00000000253C459A GPR10.... 00000001082FBABF GPR11.... 00000001082FBE00
GPR12.... 00000001082FBB08 GPR13.... 00000001082FE680 GPR14.... 0000000100005DC8 GPR15.... 0000000000000000
PSW..... 07851401 80000000 00000000 253C459A

[4] Traceback:
DSA      Entry      E Offset  Statement  Load Mod      Program Unit      Service  Status
1        CEEHSDMP  +0000009A  CELQLIB   CELQLIB       CEEHSDMP         D1908   Call
2        CEEHSDSP  +00003A88  CELQLIB   CELQLIB       CEEHSDSP         D1908   Call
3        CEEOSIGJ  +0000094E  CELQLIB   CELQLIB       CEEOSIGJ         D1908   Call
4        CELQHROD  +0000024E  CELQLIB   CELQHROD      CELQHROD         D1908   Call
5        CEEOSIGG  +00000000  CELQLIB   CELQHROD      CEEOSIGG         D1908   Call
6        CELQHROD  +0000024E  CELQLIB   CELQHROD      CELQHROD         D1908   Call
7        div_zero +00000040  CELQDLL   CELQDLL       CELQDLL          1.4.f   Exception
8        main     +00000468  CELQSAMP  CELQSAMP      CELQSAMP          1.2.d   Call
9        CELQINIT +0000134C  CELQLIB   CELQINIT      CELQINIT         D1908   Call

DSA      DSA Addr      E Addr      PU Addr      PU Offset  Comp Date  Compile Attributes
1        00000001_082FA900 00000000_253C4500 00000000_253C4500 +0000009A0 20061215  CEL POSIX XPLINK EBCDIC HFP
2        00000001_082FAAC0 00000000_251B6060 00000000_251B6060 +00003A880 20061215  CEL POSIX XPLINK EBCDIC HFP
3        00000001_082FD3E0 00000000_2504AAB0 00000000_2504AAB0 +0000094E0 20070109  CEL POSIX XPLINK EBCDIC HFP
4        00000001_082FDDE0 00000000_251C9480 00000000_251C9480 +0000024E0 20061215  CEL POSIX XPLINK EBCDIC HFP
5        00000001_082FE020 00000000_253D11F8 00000000_253D11F8 +000000000 20061215  CEL POSIX XPLINK EBCDIC HFP
6        00000001_082FEE40 00000000_251C9480 00000000_251C9480 +0000024E0 20061215  CEL POSIX XPLINK EBCDIC HFP
7        00000001_082FF080 00000000_2575B5A0 00000000_00000000 +2575B5E00 20070116  C/C++ POSIX XPLINK EBCDIC IEEE
8        00000001_082FF180 00000000_250000D8 00000000_00000000 +250005400 20070116  C/C++ POSIX XPLINK EBCDIC IEEE
9        00000001_082FF280 00000000_25005010 00000000_25005010 +0000134C0 20061215  CEL POSIX XPLINK EBCDIC HFP

[5] Control Blocks Associated with the Thread:
Thread Synchronization Queue Element (SQL): 00000000_257520A0
+000000 00000000_257520A0 00000000 00000000 00000000 00000000 .....
+000010 00000000_257520B0 00000000 00000000 00000001 08358750 .....g&
+000020 00000000_257520C0 00000000 00000000 00000000 00000000 .....
+000030 00000000_257520D0 00000000 00000000 00000001 00007B18 .....#.
+000040 00000000_257520E0 00000000 00000000 00000000 00000000 .....
+000050 00000000_257520F0 - +000000 00000000_2575210F same as above

[6] Enclave Control Blocks:
Mutex and Condition Variable Blocks (MCVB+MHT+CHT): 00000001_089100B8
+000000 00000001_089100B8 00000000 00011E78 00000001 08910100 .....j..
+000010 00000001_089100C8 000007F0 00007F00 00000000 00000000 .....0..
+000020 00000001_089100D8 00000001 08FC7490 00000001 08910900 .....j..
+000030 00000001_089100E8 000001F0 00001F00 00000000 00000000 .....0..
+000040 00000001_089100F8 00000001 08FC74D0 00000000 00000000 .....
+000050 00000001_08910108 00000000 00000000 00000000 00000000 .....
+000060 00000001_08910118 - +000000 00000001_08910207 same as above
+000150 00000001_08910208 00000001 08358A20 00000000 00000000 .....
+000160 00000001_08910218 00000001 08358900 00000000 00000000 .....i.....
+000170 00000001_08910228 00000001 08358900 00000000 00000000 .....i.....
+000180 00000001_08910238 00000000 00000000 00000000 00000000 .....
+000190 00000001_08910248 - +000000 00000001_08910297 same as above
+0001E0 00000001_08910298 00000001 08358750 00000000 00000000 .....g&.....
+0001F0 00000001_089102A8 00000000 00000000 00000000 00000000 .....
+000200 00000001_089102B8 - +000000 00000001_08910307 same as above
+000250 00000001_08910308 00000001 0831AB10 00000000 00000000 .....
+000260 00000001_08910318 00000001 0831ABD0 00000000 00000000 .....
+000270 00000001_08910328 00000001 0831A990 00000000 00000000 .....z.....
+000280 00000001_08910338 00000001 0831AA50 00000000 00000000 .....&.....

```

Figure 143. Example of formatted output from LEDATA Verbexit (Part 1 of 10)

```

+000290 00000001_08910348 00000001 0831A810 00000000 00000000 |.....y.....|
+0002A0 00000001_08910358 00000001 0831A8D0 00000000 00000000 |.....y.....|
+0002B0 00000001_08910368 00000001 0831A690 00000000 00000000 |.....w.....|
+0002C0 00000001_08910378 00000001 0831A750 00000000 00000000 |.....x&.....|
+0002D0 00000001_08910388 00000001 0831A510 00000000 00000000 |.....v.....|
+0002E0 00000001_08910398 00000001 0831A5D0 00000000 00000000 |.....v.....|
+0002F0 00000001_089103A8 00000000 00000000 00000000 00000000 |.....|
+000300 00000001_089103B8 00000001 0831A450 00000000 00000000 |.....u&.....|
+000310 00000001_089103C8 00000000 00000000 00000000 00000000 |.....|
+000320 00000001_089103D8 - +000000 00000001_089104C7 |.....| same as above
+000410 00000001_089104C8 00000001 08358870 00000000 00000000 |.....h.....|
+000420 00000001_089104D8 00000000 00000000 00000000 00000000 |.....|
+000430 00000001_089104E8 00000001 0831AC30 00000000 00000000 |.....|
+000440 00000001_089104F8 00000001 083587E0 00000000 00000000 |.....g.....|
+000450 00000001_08910508 00000000 00000000 00000000 00000000 |.....|
+000460 00000001_08910518 - +000000 00000001_08910A37 |.....| same as above
+000980 00000001_08910A38 00000001 08FC7510 00000000 00000000 |.....|
+000990 00000001_08910A48 00000000 00000000 00000000 00000000 |.....|
+0009A0 00000001_08910A58 - +000000 00000001_08910AC7 |.....| same as above
+000A10 00000001_08910AC8 00000001 08FC7410 00000000 00000000 |.....|
+000A20 00000001_08910AD8 00000001 08FC7450 00000000 00000000 |.....&.....|
+000A30 00000001_08910AE8 00000000 00000000 00000000 00000000 |.....|
+000A40 00000001_08910AF8 - +000000 00000001_08910B07 |.....| same as above

Thread Synchronization Enclave Latch Table (EPALT): 00000001_08910B00
+000000 00000001_08910B00 00000000 00000000 00000000 00000000 |.....|
+000010 00000001_08910B10 - +000000 00000001_089115DF |.....| same as above
+000AE0 00000001_089115E0 00000000 2524F9C0 00000000 00000000 |.....9.....|
+000AF0 00000001_089115F0 00000000 00000000 00000000 00000000 |.....|
+000B00 00000001_08911600 - +000000 00000001_0891176F |.....| same as above
+000C70 00000001_08911770 00000000 2538B4E0 00000000 00000000 |.....|
+000C80 00000001_08911780 00000000 00000000 00000000 00000000 |.....|
+000C90 00000001_08911790 - +000000 00000001_08911EFF |.....| same as above

HEAPCHK Option Control Block (HCOP): 00000001_089234D0
+000000 00000001_089234D0 C8C3D6D7 00000048 00000001 00000000 |HCOP.....|
+000010 00000001_089234E0 00000000 0000000A 0000000A 00000000 |.....|
+000020 00000001_089234F0 00000001 08FC0090 00000001 08923518 |.....k.....|
+000030 00000001_08923500 00000001 08A00050 00000000 00000000 |.....&.....|
+000040 00000001_08923510 00000000 00000000 C8C3C6E3 00004000 |.....HCFT..|

HEAPCHK Element Table (HCEL) for Heapid 00000001 :
Header: 00000001_08FC0090
+000000 00000001_08FC0090 C8C3C5D3 00000000 00000000 00000000 |HCEL.....|
+000010 00000001_08FC00A0 00000000 00000000 00000001 00100138 |.....|
+000020 00000001_08FC00B0 000001F4 0000000C 00000000 00000000 |...4.....|

Address      Seg Addr      Length
Table: 00000001_08FC00C0
+000000 00000001_08300040 00000001_08300000 00000000 00000180 00000001 08FC3F50 |.....&|
+000020 00000001_083001C0 00000001_08300000 00000000 00019660 00000001 08FC5B30 |.....o-...$.|
+000040 00000001_08319820 00000001_08300000 00000000 00025B40 00000001 08FC5C90 |.....q.....$ ..|.
+000060 00000001_0833F360 00000001_08300000 00000000 00019340 00000001 08FC5EB0 |.....3-.....l ..|.
+000080 00000001_083586A0 00000001_08300000 00000000 000189E0 00000001 08FC6010 |.....f.....i ..|.
+0000A0 00000001_08371080 00000001_08300000 00000000 00000060 00000001 08FC7050 |.....&.....|
+0000C0 00000001_083710E0 00000001_08300000 00000000 0001F420 00000001 08FE9790 |.....4.....p.|
+0000E0 00000001_08390500 00000001_08300000 00000000 00005C00 00000001 08FEC530 |.....*.....E.|
+000100 00000001_08396100 00000001_08300000 00000000 0000A320 00000001 08FEE210 |...../.....t.....S.|
+000120 00000001_083A0420 00000001_08300000 00000000 00000180 00000001 08FEE530 |.....V.....|
+000140 00000001_083A05A0 00000001_08300000 00000000 00017400 00000001 08FEF730 |.....X.....|
+000160 00000000_00000000 00000000_00000000 00000000 00000000 00000000 00000000 |.....|
+000180 00000001_083B79A0 00000001_08300000 00000000 00017720 00000001 08FEFA90 |.....|

[7] Language Environment Trace Table:
Most recent trace entry is at displacement: 005A80
Displacement      Trace Entry in Hexadecimal      Trace Entry in EBCDIC
-----
+000000 Time 21.22.38.487562 Date 2007.01.16 Thread ID... 253E019000000000
+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

```

Figure 143. Example of formatted output from LEDATA Verbexit (Part 2 of 10)

```

+000080 Time 21.22.38.497576 Date 2007.01.16 Thread ID... 253E019000000000
+000090 Member ID... 03 Flags..... 000000 Entry Type..... 00000006
+000098 4C60604D F0F0F6C6 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F0F0 F0F9C4C6 F040D9F2 <--(006F) R1=0000000100009DF0 R2
+0000B8 7EF0F0F0 F0F0F0F0 F0F2F5F5 F7F4F6C4 F040D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0 =00000000255746D0 R3=00000000000
+0000D8 F0F0F0F0 C340C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0 0000C ERRNO=00000000 ERRNO2=0000
+0000F8 F0F0F0F0 00000000 0000....

+000100 Time 21.22.38.497582 Date 2007.01.16 Thread ID... 253E01900000000000
+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
+000178 40404040 40404040

+000180 Time 21.22.38.526042 Date 2007.01.16 Thread ID... 253E01900000000000
+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 F0F2F5F5 F7F4F6C4 F040D9F3 7EF0F0F0 F0F0F0F0 F1F0F8C6 =00000000255746D0 R3=0000000108F
+0001D8 C3F5C5F3 F040C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0 C5E30 ERRNO=00000000 ERRNO2=0000
+0001F8 F0F0F0F0 00000000 0000....

+000200 Time 21.22.38.526049 Date 2007.01.16 Thread ID... 253E01900000000000
+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
+000278 40404040 40404040

+000280 Time 21.22.38.526081 Date 2007.01.16 Thread ID... 253E01900000000000
+000290 Member ID... 03 Flags..... 000000 Entry Type..... 00000006
+000298 4C60604D F0F0F6C6 5D40D9F1 7EF0F0F0 F0F0F0F0 F1F0F8F2 C6C6F4F6 F040D9F2 <--(006F) R1=0000000100009DF0 R2
+0002B8 7EF0F0F0 F0F0F0F0 F0F2F5F5 F7F4F6C4 F040D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0 =00000000255746D0 R3=00000000000
+0002D8 F0F0F0F0 C440C5D9 D9D5D67E F0F0F0F0 F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0 0000D ERRNO=00000000 ERRNO2=0000
+0002F8 F0F0F0F0 00000000 0000....

+000300 Time 21.22.38.526083 Date 2007.01.16 Thread ID... 253E01900000000000
+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
+000378 40404040 40404040

:

+005900 Time 21.22.38.942077 Date 2007.01.16 Thread ID... 253E01900000000000
+005910 Member ID... 03 Flags..... 000000 Entry Type..... 00000005
+005918 6D979696 936D8699 85856D81 93936D96 866D9695 856D8481 A3817CC1 C6F1F26D _pool_free_all_of_one_data@AF12_
+005938 60606E4D F0F0F5F9 5D408699 85854DF0 A7F0F0F0 F0F0F0F0 F1F0F8F3 F5F8C1C2 -->(0059) free(0x0000000108358AB
+005958 F05D4040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 0)
+005978 40404040 40404040

+005980 Time 21.22.38.942079 Date 2007.01.16 Thread ID... 253E01900000000000
+005990 Member ID... 03 Flags..... 000000 Entry Type..... 00000006
+005998 4C60604D F0F0F5F9 5D40D9F1 7EF0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F1 F240D9F2 <--(0059) R1=0000000000000012 R2
+0059B8 7EF0F0F0 F0F0F0F0 F0F2F5F5 F7F4F6C4 F040D9F3 7EF0F0F0 F0F0F0F0 F1F0F8F3 =00000000255746D0 R3=00000001083
+0059D8 F5F8C2C3 F040C5D9 D9D5D67E F0F0F0F0 F0F0F7F9 40C5D9D9 D5D6F27E C3F2F5C6 58BC0 ERRNO=00000079 ERRNO2=C25F
+0059F8 F0F0F0F1 00000000 0001....

+005A00 Time 21.22.38.942080 Date 2007.01.16 Thread ID... 253E01900000000000
+005A10 Member ID... 03 Flags..... 000000 Entry Type..... 00000005
+005A18 6D848497 896D8699 85856D81 93936D96 866D9695 856D8995 86967CC1 C6F1F36D _ddpi_free_all_of_one_info@AF13_
+005A38 60606E4D F0F0F5F9 5D408699 85854DF0 A7F0F0F0 F0F0F0F0 F1F0F8F3 C1F0F5C3 -->(0059) free(0x00000001083A05C
+005A58 F05D4040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 0)
+005A78 40404040 40404040

+005A80 Time 21.22.38.942084 Date 2007.01.16 Thread ID... 253E01900000000000
+005A90 Member ID... 03 Flags..... 000000 Entry Type..... 00000006
+005A98 4C60604D F0F0F5F9 5D40D9F1 7EF0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0 F240D9F2 <--(0059) R1=0000000000000002 R2
+005AB8 7EF0F0F0 F0F0F0F0 F0F2F5F5 F7F4F6C4 F040D9F3 7EF0F0F0 F0F0F0F0 F0F0F0F0 =00000000255746D0 R3=00000000000
+005AD8 F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 F0F0F7F9 40C5D9D9 D5D6F27E C3F2F5C6 00000 ERRNO=00000079 ERRNO2=C25F
+005AF8 F0F0F0F1 00000000 0001....

```

[8] Process Control Blocks:

```

Thread Synchronization Process Latch Table (PPALT): 00000001_08911F00
+000000 00000001_08911F00 00000000 00000000 00000000 00000000 |.....|
+000010 00000001_08911F10 - +000000 00000001_089132FF same as above

```

Figure 143. Example of formatted output from LEDATA Verbexit (Part 3 of 10)


```

[9] TCB: 007FF050          LE Level: 15          ASID: 0028

[10] Active Members: C/C++

[11] CEELAA: 00000000_7F7547D8
+000000 EYE_CATCHER:LAA VERSION:00000001 PREVIOUS:00000000
+00000C NEXT:00000000 STCB:7F7543A0 ASID:0028
+000018 FLOOR31:FFBAD000 OVERFLOW31:00000000 GTAB31:00000000
+000024 LCA31:00000000 TRT31:00000000 FLOOR64:00000001_08200000
+000048 OVERFLOW64:00000000_25250098 GTAB64:00000001_0871CA00
+000058 LCA64:00000001_00000000 TRT64:00000001_00000030
+0000C8 FLAG1:A1 FLAG2:00 CB_STG64:00000001_00000000
+0000F8 CB_STG31:00000000_257520A0 SvcVec31:00000000
+000104 SANC31:00000000 CurKey31:00 SvcVec64:00000000_00000000
+000120 SANC64:00000001_00100000 CurKey64:80 ENSQ64:00000001_00100108
+000138 LHEAP64ID:00000001_001002A8 LHEAP31ID:00000000_00000000
+000148 IPTLAA:7F7547D8 SYSTEM_RETURN_CODE:00000000
+000150 SYSTEM_REASON_CODE:00000000 SLE_REASON_CODE:00000000
+000158 MSTRLAA:7F7547D8 SLEPC_ROUTER:829E9CE0 ALEI:00000000_00000000

[12] CEELCA: 00000001_00000000
+000000 EYE_CATCHER:LCA VERSION:00000001 CAA:00000001_00007B18
+000010 DIA:25752320 LAA:7F7547D8 SAVSTACK:00000000_00000000
+000028 QINIT:00000000_25005010 TLC:00000000_255D39E0

[13] CEECAA: 00000001_00007B18
+000288 DLLF:00000000_00000000 INVAR:8000 FLAG0:00
+000304 TORC:00000000 FLAG2:30 LEVEL:15 PM:04
+000368 DMC:00000000_00000000 CD:00000000 RS:00000000
+000378 ERR:00000001_082FB000 DDSA:00000001_082FF760
+000388 EDB:00000001_00005340 PCB:00000001_00003CA0 EYEPR:00000001_00007B00
+0003A0 CAA:00000001_00007B18 SHAB:00000000_00000000
+0003B0 PRGCK:00000000_00000000 URC:00000000 PICICB:00000000_00000000
+0003C8 SIGSTR:00000000 SIGSFLG:08000000 THDID:253E0190_00000000
+0003D8 RCB:00000001_00003A10 MEMBR:00000001_000084D0
+0003E8 SIGNAL_STATUS:00000000_00000008 HCOM_REG14:00000000_00000000
+0003F8 EDCHPXV:00000000_2546C78 THREADHEAPID:00000000_00000000
+000408 SYS_RTNODE:00000000 SYS_RSNCODE:00000000 SIGNGPTR:00000001_00007F30
+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 SQLADDR:00000000_257520A0
+000510 VBA:00000001_08913350 TCS:00000001_08FEC450
+000520 CONDWAITDSAREG:00000000_00000000 THDSTATUS:00000000_00000000
+000570 FBTK:00000000_00000000_00000000_00000000 PTLXPTR:00000000_00000000
+000588 TICB_PTR:00000001_00006AB0 FWD_CHAIN:00000001_114013C8
+0005A0 BKWD_CHAIN:00000001_199013C8 TCB:007FF050
+0007EC DIA:25752320 DLLFFLAG:00 MCBPTR:00000000_25773DAB
+000838 MAD:00000000_25773048 MFD:00000000_25752998

[14] CEEPCB: 00000001_00003CA0
+000000 EYE:CEEPCB SYSTEM:03 HRDWR:03 SBSYS:02 FLAG2:08
+000108 DBGHE:00000000_00000000 DMEMBR:00000001_00004048
+000118 ZL0D:00000000_00000000 ZDEL:00000000_00000000
+000128 ZGETST:00000000_00000000 ZFREEST:00000000_00000000
+000138 RCB:00000001_00003A10 OMVS_LEVEL:7FC00000 CHAIN:00000000_00000000
+000150 PRFEH:00000000_00000000 FLAG6:00 HABD_CLEANUP:0000
+000160 QFEXIT:00000000_252A3F48 ABENDCODE:00000000
+0001C4 REASON:00000000 F3456:00000040 MEML:00000001_00004020
+0001D8 MEMBR:00000001_00004048 LEHL:00000001_00001560
+0001E8 CMXB:00000001_00001738 STV:02 PM_BYTE:00 FLAG1:00
+0001F8 ISA:00000001_00000000 ISA_SIZE:0001CA00
+000204 SRV_COUNT:00000000 SRV_UWORD:00000000_00000000
+000210 WORKAR:00000000_00000000 LOAD:00000000_2571C250
+000220 DELETE:00000000_2571C240 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_2571C230 RLAT:00000000_2571C220
+000270 ELAT:00000000_2571C210 IPTQ:00000000_2571C200
+000280 IENV:00000000_2571C1F0 LODTYP:FFFFFFFF LEVEL:FFFFFFFF
+000290 LLTPTR:00000001_00000738 RC:00000000 REASON:00000000
+0002A0 RC_MOD:00000000 FBTK:00000000_00000000_00000000_00000000
+0002B8 PPALTADDR:00000001_08911F00 PPALTSIZE:00001400
+0002C8 PICB:00000001_00003C80 XPLINKFLAGS:80 QICB:00000001_00010688
+0002E0 CALLERS_SAVE:00000000_00006F58

CEEMEML: 00000001_00004048
+000000 MEMLDEF:..... EXIT:00000000_00000000 LLVTL:00000000_00000000

```

Figure 143. Example of formatted output from LEDATA Verbexit (Part 4 of 10)

```

[15] CEERCB: 00000001_00003A10
+000000 EYE:CEERCB SYSTEM:03 HARDWARE:03 SUBSYS:02
+000043 FLAGS:00 DMEMBR:00000001_00004048 VERSION_ID:04010900
+000058 PCB_CHAIN:00000001_00003CA0

[16] CEEEDB: 00000001_00005340
+000000 EYE:CEEEDB FLAG:97 BIPM:00 CREATOR_ID:01
+000108 BMEMBR:00000001_000068F8 OPTCB:00000001_00005DC8
+000118 USER_RC:00000000 RSN_CODE:00000000 PCB:00000001_00003CA0
+000128 APPL_STR:00000000_00000000 ENVAR:00000001_00004810
+000140 ENV_ANCHOR:00000001_00005478 BOTRB:00000001_08910090
+000150 CAACHAIN:00000001_00007B18 FLAGS1:82000000 THDSACT:00000000_00000000
+000168 DCB:00000000 INPUT_R1:00000000_00006FF0
+000178 LAST_RB:00000000 LAST_RBCT:00000000
+000180 ENV_LGTH:00000000_00000200 ENVAR_A:00000001_00004A18
+000190 ENV_ANCHOR_A:00000001_000054C8

CEEMEML: 00000001_000068F8
+000000 MEMLDEF:..... EXIT:00000000_00000000 LLVTL:FFFFFFFF_FFFFFFFF

```

[17] Language Environment Run-Time Options in effect.

```

LAST WHERE SET      Override  OPTIONS
*****
DD:CEEOPTS          OVR        CEEDUMP(00000000,SYSOUT=*,
                        FREE=END,SPIN=UNALLOC)
INSTALLATION DEFAULT OVR        DYNDUMP(*USERID,
                        NODYNAMIC,TDUMP)
INSTALLATION DEFAULT OVR        ENVAR("")
INSTALLATION DEFAULT OVR        FILETAG(NOAUTOCVT,NOAUTOTAG)
DD:CEEOPTS          OVR        HEAPCHK(ON,00000001,00000000,00000010,
                        00000010)
PROGRAMMER DEFAULT  OVR        HEAPPOLS(ON,
                        00000008,00000010,
                        00000032,00000010,
                        00000128,00000010,
                        00000256,00000010,
                        00001024,00000010,
                        00002048,00000010,
                        00000000,00000010,
                        00000000,00000010,
                        00000000,00000010,
                        00000000,00000010,
                        00000000,00000010,
                        00000000,00000010,
                        00000000,00000010,
                        00000000,00000010)
DD:CEEOPTS          OVR        HEAPPOLS64(ON,
                        00000008,00004000,
                        00000032,00002000,
                        00000128,00000700,
                        00000256,00000350,
                        00001024,00000100,
                        00002048,00000050)
                        00003072,00000050)
                        00004096,00000050)
                        00008192,00000025)
                        00016384,00000010)
                        00032768,00000005)
                        00065536,00000005)
INSTALLATION DEFAULT OVR        HEAP64(0000000000000001,0000000000000001,
                        KEEP,00032768,00032768,KEEP,
                        00004096,00004096,FREE)
INSTALLATION DEFAULT OVR        INFOMSGFILTER(OFF)
INSTALLATION DEFAULT OVR        IOHEAP64(0000000000000001,0000000000000001,
                        FREE,00012288,00000081,FREE,
                        00004096,00004096,FREE)
INSTALLATION DEFAULT OVR        LIBHEAP64(0000000000000001,0000000000000001,
                        FREE,00016384,00008192,FREE,
                        00008192,00004096,FREE)
INSTALLATION DEFAULT OVR        NATLANG(ENU)
PROGRAMMER DEFAULT  OVR        POSIX(ON)
INSTALLATION DEFAULT OVR        PROFILE(OFF,"")
DD:CEEOPTS          OVR        RPTOPTS(ON)
DD:CEEOPTS          OVR        RPTSTG(ON)
INSTALLATION DEFAULT OVR        STACK64(0000000000000001,0000000000000001,
                        0000000000000128)
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,0000000000000001,
                        0000000000000001,
                        0000000000000128)
PROGRAMMER DEFAULT  OVR        TRACE(ON,01048576,NODUMP,LE=00000001)
INSTALLATION DEFAULT OVR        TRAP(ON,SPIE)

```

Figure 143. Example of formatted output from LEDATA Verbexit (Part 5 of 10)

```

[18] Heap Storage Control Blocks
      Heappools trace available. To display: IP VERBX LEDATA 'HPT(*)'

      ENSQ: 00000001_00100108
+000000 EYE_CATCHER:ENSQ HEAPALLOC_VAL:00000000
+000008 HEAPFREE_VAL:00000000 DSAALLOC_VAL:00000000 FLAGS1:80000000
+000014 IPT_TOKEN:000000A0 00000002 00000016 007FF050
+000024 HEAPLOCKWORD:00000000 RPT_STOR:00000001_00100410
+000030 UHEAP64:C8D7C3D8 00000000 00000001 001005B0 00000001 001007F0 00000000 00000001 00
+000060 LHEAP64:C8D7C3D8 00000000 00000001 001005E0 00000001 00100670 00000000 00000001 00
+000090 UHEAP31:C8D7C3D8 00000000 00000001 00100198 00000001 00100198 00000000 00008000 00
+0000C0 LHEAP31:C8D7C3D8 00000000 00000001 001007C0 00000001 001007C0 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:007FF050 HEAPCHK:00000000_00000000
+000188 STSB:00000000_00000000 SASB:00000000_00000000
+000198 TOKEN:7F7547D8 00000000
+0001A0 THDLHEAP64:C8D7C3D8 00000000 00000001 00100790 00000001 00100790 00000000 00000001 00
+0001D0 IOHEAP64:C8D7C3D8 00000000 00000001 00100760 00000001 00100760 00000000 00000001 00
+000200 IOHEAP31:C8D7C3D8 00000000 00000001 001006A0 00000001 001006A0 00000000 00003000 00
+000230 IOHEAP24:C8D7C3D8 00000000 00000001 001006D0 00000001 001006D0 00000000 00001000 00
+000260 SM_CELL_BLOCK:00000001_00100490 SPDE:00000001_00100730

      User Heap64 Control Blocks

      HPCQ: 00000001_00100138
+000000 EYE_CATCHER:HPCQ FIRST:00000001_001005B0 LAST:00000001_001007F0
+000018 INITSIZE:00000000 00000001 INCRSIZE:00000000 00000001
+00002C OPTIONS:80000000

      HPSQ: 00000001_00005058
+000000 BYTES_ALLOC:00000000 001C3320
+000008 CURR_ALLOC:00000000 000CF080 GET_REQ:00000000 00000000
+000018 FREE_REQ:00000000 00000001 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000

      THNQ: 00000001_001005B0
+000000 EYE_CATCHER:THNQ FLAGS:80000000 NEXT:00000001_001007F0
+000010 PREV:00000001_00100138 HEAPID:00000001_00100138
+000020 SEGMENT:00000001_08300000 SEG_LEN:00000000 00100000

      HANQ: 00000001_08300000
+000000 EYE_CATCHER:HANQ FLAGS:80000000 HEAPID:00000001_00100138
+000020 SEGMENT:00000001_08300000 ROOT:00000001_083CF0C0
+000030 SEG_LEN:00000000 00100000 ROOT_LEN:00000000 00030F40

      Free Storage Tree for Heap Segment 00000001

      Depth      Node      Node      Parent      Left      Right      Left      Right
      Address    Length    Node      Node      Node      Node      Length    Length
0 00000001083CF0C0 0000000000030F40 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000

      Map of Heap Segment 00000001

      To display entire segment: IP LIST 0000000108300000 LEN(X'000000000100000') ASID(X'0028')

      0000000108300040: Allocated storage element, length=000000000000180.
      To display: IP LIST 0000000108300040 LEN(X'000000000000180') ASID(X'0028')
      0000000108300050: C36DE6E2 C1F6F440 40404040 40404040 00000001 08300070 00000000 250005A8 |C_WSA64 .....y|

      00000001083001C0: Allocated storage element, length=000000000019660.
      To display: IP LIST 00000001083001C0 LEN(X'000000000019660') ASID(X'0028')
      00000001083001D0: 00000000 00000005 00000000 00000005 00000001 08319840 00000001 08319A28 |.....q .....|
      0000000108319820: Allocated storage element, length=000000000025B40.
      To display: IP LIST 0000000108319820 LEN(X'000000000025B40') ASID(X'0028')
      0000000108319830: 00000000 00000007 00000000 00000007 00000000 00000000 00000000 00000000 |.....|

      000000010833F360: Allocated storage element, length=000000000019340.
      To display: IP LIST 000000010833F360 LEN(X'000000000019340') ASID(X'0028')
      000000010833F370: 00000000 00000006 00000000 00000006 00000000 00000000 00000000 00000000 |.....|

      00000001083586A0: Allocated storage element, length=0000000000189E0.
      To display: IP LIST 00000001083586A0 LEN(X'0000000000189E0') ASID(X'0028')
      00000001083586B0: 00000000 00000003 00000000 00000003 C3C4D3D3 00000000 00000000 00000000 |.....CDLL.....|

      0000000108371080: Allocated storage element, length=000000000000060.
      To display: IP LIST 0000000108371080 LEN(X'000000000000060') ASID(X'0028')
      0000000108371090: C36DE6E2 C1F6F440 40404040 40404040 00000000 000006F0 00000000 25574500 |C_WSA64 .....0.....|

```

Figure 143. Example of formatted output from LEDATA Verbx (Part 6 of 10)

```

00000001083710E0: Allocated storage element, length=00000000001F420.
To display: IP LIST 00000001083710E0 LEN(X'00000000001F420') ASID(X'0028')
00000001083710F0: 00000000 00000001 00000000 00000001 00000001 08358990 00000000 00000000 |.....i.....|

0000000108390500: Allocated storage element, length=000000000005C00.
To display: IP LIST 0000000108390500 LEN(X'000000000005C00') ASID(X'0028')
0000000108390510: C36DE6E2 C1F6F440 40404040 40404040 000000FF 00000000 00000001 08394360 |C_WSA64 .....-|
0000000108396100: Allocated storage element, length=000000000000A320.
To display: IP LIST 0000000108396100 LEN(X'000000000000A320') ASID(X'0028')
0000000108396110: C36DE6E2 C1F6F440 40404040 40404040 00000001 08399D80 00000000 258492C0 |C_WSA64 .....dk.|

00000001083A0420: Allocated storage element, length=000000000000180.
To display: IP LIST 00000001083A0420 LEN(X'000000000000180') ASID(X'0028')
00000001083A0430: C36DE6E2 C1F6F440 40404040 40404040 00000000 00000000 00000001 08FEC410 |C_WSA64 .....D.|

00000001083A05A0: Allocated storage element, length=0000000000017400.
To display: IP LIST 00000001083A05A0 LEN(X'0000000000017400') ASID(X'0028')
00000001083A05B0: 00000000 00000004 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

00000001083B79A0: Allocated storage element, length=000000000017720.
To display: IP LIST 00000001083B79A0 LEN(X'000000000017720') ASID(X'0028')
00000001083B79B0: 00000000 00000002 00000000 00000000 00000000 00000000 00000000 D3D4D9C5 |.....LMRE|

00000001083CF0C0: Free storage element, length=000000000030F40.
To display: IP LIST 00000001083CF0C0 LEN(X'000000000030F40') ASID(X'0028')

Summary of analysis for Heap Segment 0000000108300000:
Amounts of identified storage: Free:00030F40 Allocated:000CF080 Total:000FFFC0
Number of identified areas : Free: 1 Allocated: 12 Total: 13
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.
:
[19] Stack Storage Control Blocks

SANC: 00000001_00100000
+000000 EYE_CATCHER:SANC VERSION:0001 LENGTH:0100
+000008 SEGMENT_SIZE:00000000 00000081 ACTIVE_STACK:00000001_00200000
+000018 BOS:00000001_082FFFE0 INIT_SIZE:00000000 00000001
+000028 INCR_SIZE:00000000 00000001 USER_STACK:00000001_00200000
+000038 USER_BOS:00000001_082FFFE0 USER_FLOOR:00000001_08200000
+000048 RESERVE_STACK:00000001_08500000 SDCB:00000000_00000000
+000060 PTDATA:00000000_00000000 OCB_INCRSZ:00000000 00000001
+000070 CURR_ALLOC:00000000 00000001 FLAGS1:00000000
+00007C FLAGS2:00000000 USER_ORIGIN:00000001_00200000

DSA backchain
DSA: 00000001_082FA900
+000800 HPR4:00000001_082FAAC0 HPR5:00000000 253C45F8
+000810 HPR6:00000000 253C4500 HPR7:00000000 251B9B1A
+000820 HPR8:00000001_082FBB08 HPR9:00000001 00000004
+000830 HPR10:00000001_082FBABF HPR11:00000001_082FBE00
+000840 HPR12:00000001_082FCABE HPR13:00000001_082FE680
+000850 HPR14:00000001_00005DC8 HPR15:00000001_00006554
+000860 HPHKSAV:00000001_082FBA80 HPTRAN:00000001_082FB208
+000878 HPRENT:00000000 251181E8

Contents of DSA at Location : 00000001_082FB100

+000000 00000001_082FB100 00000001 082FAAC0 00000000 253C45F8 |.....8
+000010 00000001_082FB110 00000000 253C4500 00000000 251B9B1A |.....
+000020 00000001_082FB120 00000001 082FBB08 00000001 00000004 |.....
+000030 00000001_082FB130 00000001 082FBABF 00000001 082FBE00 |.....
+000040 00000001_082FB140 00000001 082FCABE 00000001 082FE680 |.....W.
+000050 00000001_082FB150 00000001 00005DC8 00000001 00006554 |.....)H.....
+000060 00000001_082FB160 00000001 082FBA80 00000001 082FC6E0 |.....F.
+000070 00000001_082FB170 00000001 082FB208 00000000 251181E8 |.....aY
+000080 00000001_082FB180 00000001 082FAAC0 00000000 25743020 |.....
+000090 00000001_082FB190 00000000 2558CB50 00000000 251B78DA |.....&.....
+0000A0 00000001_082FB1A0 00000001 082FBE00 00000001 082FBF60 |.....-
+0000B0 00000001_082FB1B0 00000001 082FBABF 00000001 082FBA08 |.....
+0000C0 00000001_082FB1C0 00000001 082FCABE 00000001 082FE680 |.....W.
+0000D0 00000001_082FB1D0 00000001 082FB208 00000000 251181D8 |.....aQ
+0000E0 00000001_082FB1E0 00000001 082FC6E0 89848540 85A78385 |.....F.ide exce
+0000F0 00000001_082FB1F0 00000000 00000000 A8A2A385 9440C396 |.....system Co

```

Figure 143. Example of formatted output from LEDATA Verbexit (Part 7 of 10)

```

DSA: 00000001_082FAAC0
+000800 HPR4:00000001 082FD3E0 HPR5:00000000 251BABA0
+000810 HPR6:00000000 251B6060 HPR7:00000000 2504B400
+000820 HPR8:00000001 089135B0 HPR9:00000000 00000005
+000830 HPR10:00000001 082FE3DF HPR11:00000001 082FE0C0
+000840 HPR12:00000001 089135B0 HPR13:00000001 082FE680
+000850 HPR14:00000000 2504B300 HPR15:00000000 2530A300
+000860 HPHKSAV:00000000 00000000 HPTRAN:00000000 00000000
+000878 HPRENT:00000000 00000000

```

Contents of DSA at Location : 00000001_082FB2C0

```

+000000 00000001_082FB2C0 00000001 082FD3E0 00000000 251BABA0 .....L.....
+000010 00000001_082FB2D0 00000000 251B6060 00000000 2504B400 .....-.....
+000020 00000001_082FB2E0 00000001 089135B0 00000000 00000005 .....j.....
+000030 00000001_082FB2F0 00000001 082FE3DF 00000001 082FE0C0 .....T.....
+000040 00000001_082FB300 00000001 089135B0 00000001 082FE680 .....j.....W.
+000050 00000001_082FB310 00000000 2504B300 00000000 2530A300 .....t.....
+000060 00000001_082FB320 00000000 00000000 00000000 00000000 .....
+000070 00000001_082FB330 - +000000 00000001_082FB33F ..... same as above
+000080 00000001_082FB340 00000001 082FBB08 00000001 00000003 .....
+000090 00000001_082FB350 00000001 00000003 00000001 00007208 .....
+0000A0 00000001_082FB360 00000001 082FC190 00000001 082FBF90 .....A.....
+0000B0 00000001_082FB370 00000000 25773048 00000000 00000000 .....
+0000C0 00000001_082FB380 00000000 00000000 00000000 00000000 .....
+0000D0 00000001_082FB390 - +000000 00000001_082FB3BF ..... same as above

```

```

DSA: 00000001_082FD3E0
+000800 HPR4:00000001 082FDDE0 HPR5:00000000 2504C3EC
+000810 HPR6:00000000 2504AAB0 HPR7:00000000 251C96D0
+000820 HPR8:00000000 25754AD8 HPR9:00000000 25754BD0
+000830 HPR10:00000000 25754A30 HPR11:00000000 00000020
+000840 HPR12:00000001 00007B18 HPR13:00000001 082FE680
+000850 HPR14:00000000 253D6504 HPR15:00000000 00000003
+000860 HPHKSAV:00000001 00000000 HPTRAN:082FDB18 00000000
+000878 HPRENT:00000060 00000000

```

Contents of DSA at Location : 00000001_082FDBE0

```

+000000 00000001_082FDBE0 00000001 082FDDE0 00000000 2504C3EC .....C.
+000010 00000001_082FDBF0 00000000 2504AAB0 00000000 251C96D0 .....o.
+000020 00000001_082FDC00 00000000 25754AD8 00000000 25754BD0 .....Q.....
+000030 00000001_082FDC10 00000000 25754A30 00000000 00000020 .....
+000040 00000001_082FDC20 00000001 00007B18 00000001 082FE680 .....#.....W.
+000050 00000001_082FDC30 00000000 253D6504 00000000 00000003 .....

```

```

:
[20] Condition Management Control Blocks
HCOM: 00000000_25753700
+000000 PICA_AREA:00000000 00000000 EYES:HCOM SIZE:28E0 LEVEL:0001
+000010 CAA_PTR1:00000001 00007B18 CVTDCB:9B FLAG1:60104000
+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_082FCFE0
+000050 SPIE_TOKEN:00000000 DMCP:00000000_00000000
+0000F0 4083_DSA:00000000 00000000 SHUNT_VALIDFLAG:00000000
+000704 SHUNT_COUNTER:00000000 SHUNT_ADDR:00000000_00000000
+000710 SHUNT_PSW:00000000 00000000 00000000 00000000
+000720 SHUNT_REG0:00000000 00000000
+000728 SHUNT_REG1:00000000 00000000
+000730 SHUNT_REG2:00000000 00000000
+000738 SHUNT_REG3:00000000 00000000
+000740 SHUNT_REG4:00000000 00000000
+000748 SHUNT_REG5:00000000 00000000
+000750 SHUNT_REG6:00000000 00000000
+000758 SHUNT_REG7:00000000 00000000
+000760 SHUNT_REG8:00000000 00000000
+000768 SHUNT_REG9:00000000 00000000
+000770 SHUNT_REG10:00000000 00000000
+000778 SHUNT_REG11:00000000 00000000
+000780 SHUNT_REG12:00000000 00000000
+000788 SHUNT_REG13:00000000 00000000
+000790 SHUNT_REG14:00000000 00000000
+000798 SHUNT_REG15:00000000 00000000 SHUNT_CODE1:00000000_00000000
+0007A8 SHUNT_CODE2:00000000_00000000

```

Figure 143. Example of formatted output from LEDATA Verbexit (Part 8 of 10)

```

CIBH: 0000001_082FCFE0
+000000 EYE:CIBH BACK:00000001_00006AB8 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
+0000C0 OLD_MIB:00000000_00000000 COND64:00000000_00000000
+0000D0 MIB:00000000_00000000 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:..... BBRANCH_OFFSET:00000000
+00031C BBRANCH_STMTID:..... BBRANCH_STMTLEN:0000

Machine State
+000348 MCH_EYE:....
+000350 GPR00:00000000_00000000 GPR01:00000000_00000000
+000360 GPR02:00000000_00000000 GPR03:00000000_00000000
+000370 GPR04:00000000_00000000 GPR05:00000000_00000000
+000380 GPR06:00000000_00000000 GPR07:00000000_00000000
+000390 GPR08:00000000_00000000 GPR09:00000000_00000000
+0003A0 GPR10:00000000_00000000 GPR11:00000000_00000000
+0003B0 GPR12:00000000_00000000 GPR13:00000000_00000000
+0003C0 GPR14:00000000_00000000 GPR15:00000000_00000000
+0003D0 PSW:00000000_00000000_00000000_00000000
+0003E0 ILC:0000 IC1:00 IC2:00 PFT:00000000_00000000
+0003F0 FLT_0:00000000_00000000 FLT_2:00000000_00000000
+000400 FLT_4:00000000_00000000 FLT_6:00000000_00000000
+000410 FLT_1:00000000_00000000
+000418 FLT_3:00000000_00000000
+000420 FLT_5:00000000_00000000
+000428 FLT_7:00000000_00000000
+000430 FLT_8:00000000_00000000 FLT_9:00000000_00000000
+000440 FLT_10:00000000_00000000 FLT_11:00000000_00000000
+000450 FLT_12:00000000_00000000 FLT_13:00000000_00000000
+000460 FLT_14:00000000_00000000 FLT_15:00000000_00000000
+000470 FPC:00000000 AFP_FLAGS:00 EXT:00000000_00000000
+0004A0 BEA:00000000_00000000

CIBH: 0000001_00006AB8
+000000 EYE:CIBH BACK:00000000_00000000 FWD:00000001_082FCFE0
+000018 LEVEL:0002 SIZE:0C00 PTR_CIB:00000001_082FBE00
+000028 FLAG1:C12B0000 HDLQ:00000000_00000000 STATE:00000000
+000040 PRM_DESC:00000000_00000000 PRM_PREFIX:00000000_00000000
+000050 PRM_LIST:00000001_082FBE28_00000001_082FBF50_00000001_082FBF60_00000001_00007208
+000070 PARM_DESC:00000000_00000000 PARM_PREFIX:00000000_00000000
+000080 PARM_LIST:00000001_082FBF48_00000001_082FBE00_00000001_082FBF60_00000001_00007208
+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 (CEE3209S)
+0000C0 OLD_MIB:00000000_00000000 COND64:00030C89_59C3C5C5 (CEE3209S)
+0000D0 MIB:00000000_00000000 PL:00000000_250008C0 SV2:00000001_082FF080
+0000E8 SV1:00000001_082FF080 INT:00000000_2575B5E0 MID:00000003
+000100 HDL_SF:00000001_082FF280 HDL_EPT:00000000_251BB1D0
+000110 HDL_RST:00000000_00000000 RSM_SF:00000001_082FF080
+000120 RSM_PT:00000000_2575B5E2 RSM_MACH:00000001_00007000
+000130 SUSPEND_EH:00000000_00000000 COND_DFT:00000003
+000140 Q_DATA_TOKEN:00000000_00000000 FDBK:00000000_00000000
+000150 ABNAME:..... BBRANCH_OFFSET:00000000
+00031C BBRANCH_STMTID:..... BBRANCH_STMTLEN:0000

```

Figure 143. Example of formatted output from LEDATA Verbexit (Part 9 of 10)

```

Machine State
+000348 MCH_EYE:ZMCH
+000350 GPR00:00000000 00000000 GPR01:00000001 00009DF0
+000360 GPR02:00000001 082FEF8 GPR03:00000000 00000012
+000370 GPR04:00000001 082FF080 GPR05:00000000 000006F0
+000380 GPR06:00000000 00000000 GPR07:00000000 00000001
+000390 GPR08:00000000 250000E4 GPR09:00000000 2575B630
+0003A0 GPR10:00000000 250014E0 GPR11:00000001 08FC5E70
+0003B0 GPR12:00000001 00005340 GPR13:00000000 00006F58
+0003C0 GPR14:00000000 25250098 GPR15:00000000 0000001F
+0003D0 PSW:07852401 80000000 00000000 2575B5E2
+0003E0 ILC:0002 IC1:00 IC2:09 PFT:00000000_00000000
+0003F0 FLT_0:4328007E 7DC78A9C FLT_2:00000000 00000000
+000400 FLT_4:3C100000 00000000 FLT_6:34100000 00000000
+000410 FLT_1:00000000 00000000
+000418 FLT_3:00000000 00000000
+000420 FLT_5:00000000 00000000
+000428 FLT_7:00000000 00000000
+000430 FLT_8:00000000 00000000 FLT_9:00000000 00000000
+000440 FLT_10:00000000 00000000 FLT_11:00000000 00000000
+000450 FLT_12:00000000 00000000 FLT_13:00000000 00000000
+000460 FLT_14:00000000 00000000 FLT_15:00000000 00000000
+000470 FPC:00000000 AFP_FLAGS:00 EXT:00000000_00000000
+0004A0 BEA:00000000_00000001

CIB: 00000001_082FBE00
+000000 EYE:CIB BACK:00000000_00000000 FWD:00000000_00000000
+000018 SIZE:0190 VERSION:0004 PLAT_ID:00000000
+000028 COND:000300C6 59C3C5C5 (CEE0198S) MIB:00000000_00000000
+000038 MACHINE:00000001 082FBF90 OLD_COND_64:00030C89 59C3C5C5 (CEE3209S)
+000048 OLD_MIB:00000000_00000000 FLG_1:00 FLG_2:00 FLG_3:00
+000053 FLG_4:04 HDL_SF:00000001_082FF180 HDL_EPT:00000000_251BB1D0
+000068 HDL_RST:00000000_00000000 RSM_SF:00000001_082FF080
+000078 RSM_PT:00000000_2575B5E2 RSM_MACH:00000001_00007000
+000088 COND_DEFAULT:00000003 PH_CALLEE_SF:00000001_082FF080
+000098 HDL_SF_FMT:01 PH_CALLEE_FMT:01 BBRANCH_STMTLEN:0000
+00009C BBRANCH_OFFSET:00000000 BBRANCH_STMTID:.....
+0000D0 VSR:00000000 00000000 VSTOR:00000000_00000000
+0000E0 VRPSA:00000000_00000000 MCB:00000000_00000000
+0000F0 MRN:..... MFLAG:00 FLG_5:48 FLG_6:22 FLG_7:04
+000103 FLG_8:00 ABCD:940C9000 ABRC:00000009 ABNAME:.....
+000118 PL:00000000_250008C0 SV2:00000001_082FF080 SV1:00000001_082FF080
+000130 INT:00000000_2575B5E0 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 SIGNO:00000008 PPSD:00000001_00007220

```

[21] Message Processing Control Blocks

```

CMXB: 00000001_00001738
+000000 EYE:CMXB SIZE:0160 FLAGS:8000 DHEAD1:00000000_00000000
+000010 DHEAD2:00000000_00000000

TMXB: 00000001_00007858
+000000 EYE:TMXB MIB_CHAIN_PTR:00000001_000078A0

MGF: 00000001_000078A0
+000000 EYE:CMIB PREV:00000000_00000001 NEXT:000078A0_00000001

```

[22] No PIPICB associated with CAA at address : 00000001_00007B18

:

Exiting Language Environment Data

Figure 143. Example of formatted output from LEDATA Verbexit (Part 10 of 10)

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 includes, 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 390.

[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.

[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.
- 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). See *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). See *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). See *z/OS Language Environment Vendor Interfaces* for a description of the fields in the CAA. If there is any, DLL failure data is also formatted.

[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” on page 409.

[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.

[22] Preinitialization Information

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

This section formats information related to preinitialization. See section PTBL LEDATA output for more information. If the preinitialization service CELQPIPI was not used to initialize this environment, the message: No PIPICB associated with CAA is displayed instead.

PTBL LEDATA output

The Language Environment IPCS Verbexit LEDATA command generates formatted output of Preinit tables when the PTBL or ALL parameter are specified. If ALL is specified, PTBL defaults to CURRENT value. Figure 144 on page 408 illustrates the output produced when the Verbexit LEDATA command is invoked with the PTBL parameter.

```

PTBL(CURRENT)
*****
          64 BIT LANGUAGE ENVIRONMENT DATA
*****

Language Environment Product 04 V01 R09.00

PreInitialization Programming Interface Trace Data
CELQPIPI Environment Table Entry and Trace Entry :
Active CELQPIPI Environment ( Address 00000001_00010B80 )
Eyecatcher : CELQIPTB
TCB address : 008D6E88

CELQPIPI Environment :
Environment Type : MAIN
Sequence of Calls not active
Exits not established
Signal Interrupt Routines not registered
Service Routines are not active

CELQPIPI Environment Enclave Initialized
Number of CELQPIPI Table Entries = 3

CELQPIPI Table Entry Information :
CELQPIPI Table Index 0 ( Entry 1 )
Routine Name = ISJPPCA3
Routine Type = C/C++
Routine Entry Point = 00000000_21053000
Routine Function Pointer = 00000000_210530C0
Routine was loaded by Language Environment
Routine Address was resolved
Routine Function Descriptor was valid
Routine was valid
Routine Return Code = 0
Routine Reason Code = 0

Entry of routine in CELQPIPI Table for Index 0 ( 00000001_00010D38 )

+000000 00000001_00010D38 00000000 00000000 00000000 210530C0 |.....|
+000010 00000001_00010D48 00000000 2105A808 80000000 00000000 |.....y.....|
+000020 00000001_00010D58 00000000 00000000 00000000 00000000 |.....|
+000030 00000001_00010D68 - +000000 00000001_00010D77 |.....|
+000040 00000001_00010D78 00000000 00000000 00000000 21053000 |.....|
+000050 00000001_00010D88 00000000 00000003 00000000 2105A908 |.....Z.....|
+000060 00000001_00010D98 00000000 00000000 00000003 00000000 |.....|
+000070 00000001_00010DA8 00000000 21053000 00000000 21053000 |.....|
+000080 00000001_00010DB8 00009000 00000000 C9E2D1D7 D7C3C1F3 |.....ISJPPCA3|
+000090 00000001_00010DC8 40404040 40404040 00000000 00000000 |.....|

CELQPIPI Table Index 1 ( Entry 2 ) not in use.
CELQPIPI Table Index 2 ( Entry 3 ) not in use.

```

Figure 144. Example of formatted PTBL output from LEDATA Verbexit (Part 1 of 2)

```

CELQPIPI Trace Table Entries :
Call Type = INIT_MAIN
  PIPi Driver Address = 00000000_2090082A
  Load Service Return Code = 0
  Load Service Reason Code = 0
  Most Recent Return Code = 0
  Most Recent Reason Code = 0
  An ABEND will be issued if storage can not be obtained
  Service RC = 0 :A new environment was initialized.

Call Type = ADD_ENTRY
  Routine Table Index = 1
  Routine Name = ISJPPCA1
  Routine Address = 00000000_2105E000
  Load Service Return Code = 0
  Load Service Reason Code = 3
  Service RC = 0 :The routine was added to the PreInit table.

Call Type = IDENTIFY_ENTRY
  Routine Table Index = 1
  Routine Programming Language = C/C++
  Service RC = 0 :The programming language has been returned.

Call Type = CALL_MAIN
  Routine Table Index = 1
  Enclave Return Code = 0
  Enclave Reason Code = 0
  Routine Feedback Code = 0000000000000000
  Service RC = 0 :The environment was activated and the routine called.

Call Type = DELETE_ENTRY
  Routine Table Index = 1
  Routine Name = ISJPPCA1
  Routine Address = 00000000_2105E000
  Service RC = 0 :The routine was deleted from the PreInit table.

Call Type = CALL_MAIN
  Routine Table Index = 0
  Enclave Return Code = 0
  Enclave Reason Code = 0
  Routine Feedback Code = 0000000000000000
  Service RC = 0 :The environment was activated and the routine called.

Exiting Language Environment Data

```

Figure 144. Example of formatted PTBL output from LEDATA Verbexit (Part 2 of 2)

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 145 on page 410 illustrates the output produced by specifying the HEAP option. “Heap report sections of the LEDATA output” on page 418 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.

HEAP

64 BIT LANGUAGE ENVIRONMENT DATA

Language Environment Product 04 V01 R09.00

Heap Storage Control Blocks

Heappools trace available. To display: IP VERBX LEDATA 'HPT(*)'

ENSQ: 00000001_00100108
+000000 EYE_CATCHER:ENSQ HEAPALLOC_VAL:00000000
+000008 HEAPFREE_VAL:00000000 DSAALLOC_VAL:00000000 FLAGS1:80000000
+000014 IPT_TOKEN:000000A0 00000002 00000016 007FF050
+000024 HEAPLOCKWORD:00000000 RPT_STOR:00000001_00100410
+000030 UHEAP64:C8D7C3D8 00000000 00000001 001005B0 00000001 001007F0 00000000 00000001 00
+000060 LHEAP64:C8D7C3D8 00000000 00000001 001005E0 00000001 00100670 00000000 00000001 00
+000090 UHEAP31:C8D7C3D8 00000000 00000001 00100198 00000001 00100198 00000000 00008000 00
+0000C0 LHEAP31:C8D7C3D8 00000000 00000001 001007C0 00000001 001007C0 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:007FF050 HEAPCHK:00000000_00000000
+000188 STSB:00000000_00000000 SASB:00000000_00000000
+000198 TOKEN:7F7547D8 00000000
+0001A0 THDLHEAP64:C8D7C3D8 00000000 00000001 00100790 00000001 00100790 00000000 00000001 00
+0001D0 IOHEAP64:C8D7C3D8 00000000 00000001 00100760 00000001 00100760 00000000 00000001 00
+000200 IOHEAP31:C8D7C3D8 00000000 00000001 001006A0 00000001 001006A0 00000000 00003000 00
+000230 IOHEAP24:C8D7C3D8 00000000 00000001 001006D0 00000001 001006D0 00000000 00001000 00
+000260 SM_CELL_BLOCK:00000001_00100490 SPDE:00000001_00100730

User Heap64 Control Blocks

HPCQ: 00000001_00100138
+000000 EYE_CATCHER:HPCQ FIRST:00000001_001005B0 LAST:00000001_001007F0
+000018 INITSIZE:00000000 00000001 INCRSIZE:00000000 00000001
+00002C OPTIONS:80000000
HPSQ: 00000001_00005058
+000000 BYTES_ALLOC:00000000 001C3320
+000008 CURR_ALLOC:00000000 000CF080 GET_REQ:00000000 00000000
+000018 FREE_REQ:00000000 00000001 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000
THNQ: 00000001_001005B0
+000000 EYE_CATCHER:THNQ FLAGS:80000000 NEXT:00000001_001007F0
+000010 PREV:00000001_00100138 HEAPID:00000001_00100138
+000020 SEGMENT:00000001_08300000 SEG_LEN:00000000 00100000
HANQ: 00000001_08300000
+000000 EYE_CATCHER:HANQ FLAGS:80000000 HEAPID:00000001_00100138
+000020 SEGMENT:00000001_08300000 ROOT:00000001_083CF0C0
+000030 SEG_LEN:00000000 00100000 ROOT_LEN:00000000 00030F40

[1] Free Storage Tree for Heap Segment 00000001

Table with 8 columns: Depth, Node Address, Node Length, Parent Node, Left Node, Right Node, Left Length, Right Length. Row 0 shows values for Node Address (00000001083CF0C0), Node Length (0000000000030F40), Parent Node (0000000000000000), Left Node (0000000000000000), Right Node (0000000000000000), Left Length (0000000000000000), and Right Length (0000000000000000).

Figure 145. Example formatted detailed heap segment report from LEDATA Verbx (Part 1 of 9)

```

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000108300000 LEN(X'0000000000100000') ASID(X'0028')

0000000108300040: Allocated storage element, length=000000000000180.
To display: IP LIST 0000000108300040 LEN(X'000000000000180') ASID(X'0028')
0000000108300050: C36DE6E2 C1F6F440 40404040 40404040 00000001 08300070 00000000 250005A8 |C_WSA64 .....y|

00000001083001C0: Allocated storage element, length=0000000000019660.
To display: IP LIST 00000001083001C0 LEN(X'0000000000019660') ASID(X'0028')
00000001083001D0: 00000000 00000005 00000000 00000005 00000001 08319840 00000001 08319A28 |.....q .....|
:
00000001083B79A0: Allocated storage element, length=0000000000017720.
To display: IP LIST 00000001083B79A0 LEN(X'0000000000017720') ASID(X'0028')
00000001083B79B0: 00000000 00000002 00000000 00000000 00000000 00000000 00000000 00000000 D3D4D9C5 |.....LMRE|

00000001083CF0C0: Free storage element, length=0000000000030F40.
To display: IP LIST 00000001083CF0C0 LEN(X'0000000000030F40') ASID(X'0028')

Summary of analysis for Heap Segment 0000000108300000:
Amounts of identified storage: Free:00030F40 Allocated:000CF080 Total:000FFFC0
Number of identified areas : Free: 1 Allocated: 12 Total: 13
00000000 bytes of storage were not accounted for.
No errors were found while processing this heap segment.

THNQ: 00000001_001007F0
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001_00100138
+000010 PREV:00000001_001005B0 HEAPID:00000001_00100138
+000020 SEGMENT:00000001_19C00000 SEG_LEN:00000000_00100000

HANQ: 00000001_19C00000
+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001_00100138
+000020 SEGMENT:00000001_19C00000 ROOT:00000001_19C00040
+000030 SEG_LEN:00000000_00100000 ROOT_LEN:00000000_000FFFC0

This is the last heap segment in the current heap

[1] Free Storage Tree for Heap Segment 00000001

Depth Node Node Parent Left Right Left Right
Address Length Node Node Node Length Length
0 0000000119C00040 000000000000FFFC0 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000119C00000 LEN(X'0000000000100000') ASID(X'0028')

0000000119C00040: Free storage element, length=00000000000FFFC0.
To display: IP LIST 0000000119C00040 LEN(X'00000000000FFFC0') ASID(X'0028')

Summary of analysis for Heap Segment 0000000119C00000:
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.
This is the last heap segment in the current heap.

Library Heap64 Control Blocks

HPCQ: 00000001_00100168
+000000 EYE_CATCHER:HPCQ FIRST:00000001_001005E0 LAST:00000001_00100670
+000018 INITSIZE:00000000 00000001 INCRSIZE:00000000 00000001
+00002C OPTIONS:90000000

HPSQ: 00000001_000050E8
+000000 BYTES_ALLOC:00000000 0067A940
+000008 CURR_ALLOC:00000000 0067A840 GET_REQ:00000000 00000042
+000018 FREE_REQ:00000000 00000003 GETMAINS:00000000 00000003
+000028 FREEMAINS:00000000 00000000

THNQ: 00000001_001005E0
+000000 EYE_CATCHER:THNQ FLAGS:80000000 NEXT:00000001_00100610
+000010 PREV:00000001_00100168 HEAPID:00000001_00100168
+000020 SEGMENT:00000001_08400000 SEG_LEN:00000000_00100000

HANQ: 00000001_08400000
+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

```

Figure 145. Example formatted detailed heap segment report from LEDATA Verbexit (Part 2 of 9)

```

[1] Free Storage Tree for Heap Segment 00000001

      Node      Node      Parent      Left      Right      Left      Right
Depth  Address  Length  Node      Node      Node      Length  Length
  0  0000000108400040  000000000000FFFC0  0000000000000000  0000000000000000  0000000000000000  0000000000000000

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000108400000 LEN(X'000000000100000') ASID(X'0028')

0000000108400040: Free storage element, length=000000000000FFFC0.
To display: IP LIST 0000000108400040 LEN(X'0000000000FFFC0') ASID(X'0028')

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.

THNQ: 00000001_00100610
+000000 EYE_CATCHER:THNQ  FLAGS:00000000  NEXT:00000001_00100640
+000010 PREV:00000001_001005E0  HEAPID:00000001_00100168
+000020 SEGMENT:00000001_08800000  SEG_LEN:00000000 00200000

HANQ: 00000001_08800000
+000000 EYE_CATCHER:HANQ  FLAGS:00000000  HEAPID:00000001_00100168
+000020 SEGMENT:00000001_08800000  ROOT:00000001_08943AC0
+000030 SEG_LEN:00000000 00200000  ROOT_LEN:00000000 000BC540

[1] Free Storage Tree for Heap Segment 00000001

      Node      Node      Parent      Left      Right      Left      Right
Depth  Address  Length  Node      Node      Node      Length  Length
  0  0000000108943AC0  000000000000BC540  0000000000000000  0000000000000000  0000000000000000  0000000000000000

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000108800000 LEN(X'000000000200000') ASID(X'0028')

0000000108800040: Allocated storage element, length=000000000100040.
To display: IP LIST 0000000108800040 LEN(X'000000000100040') ASID(X'0028')
0000000108800050: C003F3EE 2540A23A 253E0190 00000000 03000000 00000005 94818995 40404040 |... s.....main |

0000000108900080: Allocated storage element, length=0000000000132A0.
To display: IP LIST 0000000108900080 LEN(X'0000000000132A0') ASID(X'0028')
0000000108900090: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

:

0000000108943600: Allocated storage element, length=0000000000004C0.
To display: IP LIST 0000000108943600 LEN(X'0000000000004C0') ASID(X'0028')
0000000108943610: D8D7C3C2 000004B0 0000000C 00010000 00000001 00000000 00000000 08E00050 |QPCB.....&|

0000000108943AC0: Free storage element, length=00000000000BC540.
To display: IP LIST 0000000108943AC0 LEN(X'00000000000BC540') ASID(X'0028')

Summary of analysis for Heap Segment 0000000108800000:
Amounts of identified storage: Free:000BC540      Allocated:00143A80      Total:001FFFC0
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.

THNQ: 00000001_00100640
+000000 EYE_CATCHER:THNQ  FLAGS:00000000  NEXT:00000001_00100670
+000010 PREV:00000001_00100610  HEAPID:00000001_00100168
+000020 SEGMENT:00000001_08A00000  SEG_LEN:00000000 00400000
HANQ: 00000001_08A00000

+000000 EYE_CATCHER:HANQ  FLAGS:00000000  HEAPID:00000001_00100168
+000020 SEGMENT:00000001_08A00000  ROOT:00000001_08D48340
+000030 SEG_LEN:00000000 00400000  ROOT_LEN:00000000 000B7CC0

```

Figure 145. Example formatted detailed heap segment report from LEDATA Verbexit (Part 3 of 9)


```

[1] Free Storage Tree for Heap Segment 00000001

      Node      Node      Parent      Left      Right      Left      Right
Depth  Address    Length    Node      Node      Node      Length    Length
  0  0000000108D48340  000000000000B7CC0  0000000000000000  0000000000000000  0000000000000000  0000000000000000

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000108A00000 LEN('000000000400000') ASID('0028')

0000000108A00040: Allocated storage element, length=000000000348300.
To display: IP LIST 0000000108A00040 LEN('000000000348300') ASID('0028')
0000000108A00050: C8D7E3C1 00000000 00046030 000000F0 00000001 08A000C8 00000001 08A460F8 |HPTA.....-....0.....H.....u-8|

0000000108D48340: Free storage element, length=0000000000B7CC0.
To display: IP LIST 0000000108D48340 LEN('0000000000B7CC0') ASID('0028')

Summary of analysis for Heap Segment 0000000108A00000:
Amounts of identified storage: Free:000B7CC0      Allocated:00348300      Total:003FFFC0
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.

THNQ: 00000001_00100670
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001_00100168
+000010 PREV:00000001_00100640 HEAPID:00000001_00100168
+000020 SEGMENT:00000001_08E00000 SEG_LEN:00000000 00200000

HANQ: 00000001_08E00000
+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001_00100168
+000020 SEGMENT:00000001_08E00000 ROOT:00000001_08FEEC20
+000030 SEG_LEN:00000000 00200000 ROOT_LEN:00000000 000113E0

This is the last heap segment in the current heap

[1] Free Storage Tree for Heap Segment 00000001

      Node      Node      Parent      Left      Right      Left      Right
Depth  Address    Length    Node      Node      Node      Length    Length
  0  0000000108FEEC20  000000000000113E0  0000000000000000  0000000108FEE960  0000000000000000  0000000000000120  0000000000000000
  1  0000000108FEE960  0000000000000120  0000000108FEEC20  0000000000000000  0000000000000000  0000000000000000  0000000000000000

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000108E00000 LEN('000000000200000') ASID('0028')

0000000108E00040: Allocated storage element, length=0000000001C0040.
To display: IP LIST 0000000108E00040 LEN('0000000001C0040') ASID('0028')
0000000108E00050: 00000000 00000002 00000000 00000000 00000000 00000001 00000000 00000000 |.....|

0000000108FC0080: Allocated storage element, length=000000000003EC0.
To display: IP LIST 0000000108FC0080 LEN('000000000003EC0') ASID('0028')
0000000108FC0090: C8C3C5D3 00000000 00000000 00000000 00000000 00000001 00100138 |HCEL.....|

:

0000000108FEEA80: Allocated storage element, length=000000000001A0.
To display: IP LIST 0000000108FEEA80 LEN('000000000001A0') ASID('0028')
0000000108FEEA90: 00000000 25462598 00000000 00000000 C3C5D3D8 D3C9C240 00000008 C3C5D3D8 |.....q.....CELQLIB ....CELQ|

0000000108FEEC20: Free storage element, length=0000000000113E0.
To display: IP LIST 0000000108FEEC20 LEN('0000000000113E0') ASID('0028')

Summary of analysis for Heap Segment 0000000108E00000:
Amounts of identified storage: Free:00011500      Allocated:001EEAC0      Total:001FFFC0
Number of identified areas : Free:      2 Allocated:      54 Total:      56
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 145. Example formatted detailed heap segment report from LEDATA Verbexit (Part 4 of 9)

User Heap31 Control Blocks

```
HPCQ: 00000001_00100198
+000000 EYE_CATCHER:HPCQ FIRST:00000001_00100198 LAST:00000001_00100198
+000018 INITSIZE:00000000 00008000 INCRSIZE:00000000 00008000
+00002C OPTIONS:40000000
```

** NO SEGMENTS ALLOCATED **

```
HPSQ: 00000001_00005088
+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: 00000001_001001C8
+000000 EYE_CATCHER:HPCQ FIRST:00000001_001007C0 LAST:00000001_001007C0
+000018 INITSIZE:00000000 00004000 INCRSIZE:00000000 00002000
+00002C OPTIONS:50000000
```

```
HPSQ: 00000001_00005118
+000000 BYTES_ALLOC:00000000 00001128
+000008 CURR_ALLOC:00000000 00001128 GET_REQ:00000000 00000001
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000
```

```
THNQ: 00000001_001007C0
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001_001001C8
+000010 PREV:00000001_001001C8 HEAPID:00000001_001001C8
+000020 SEGMENT:00000000_25773000 SEG_LEN:00000000 00004000
```

```
HANQ: 00000000_25773000
+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001_001001C8
+000020 SEGMENT:25773000 ROOT:25774168 SEG_LEN:00004000
+00002C ROOT_LEN:00002E98
```

This is the last heap segment in the current heap

[1] Free Storage Tree for Heap Segment 00000000

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	25774168	00002E98	00000000	00000000	00000000	00000000	00000000

[2] Map of Heap Segment 00000000

To display entire segment: IP LIST 00000000 LEN(X'00004000') ASID(X'0028')

25773040: Allocated storage element, length=00001128. To display: IP LIST 25773040 LEN(X'00001128') ASID(X'0028')

25774168: Free storage element, length=00002E98. To display: IP LIST 25774168 LEN(X'00002E98') ASID(X'0028')

Summary of analysis for Heap Segment 00000000:

Amounts of identified storage: Free:00002E98 Allocated:00001128 Total:00003FC0
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.

User Heap24 Control Blocks

```
HPCQ: 00000001_001001F8
+000000 EYE_CATCHER:HPCQ FIRST:00000001_001001F8 LAST:00000001_001001F8
+000018 INITSIZE:00000000 00001000 INCRSIZE:00000000 00001000
+00002C OPTIONS:30000000
```

** NO SEGMENTS ALLOCATED **

```
HPSQ: 00000001_000050B8
+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 145. Example formatted detailed heap segment report from LEDATA Verbexit (Part 5 of 9)

Library Heap24 Control Blocks

```
HPCQ: 00000001_00100228
+000000 EYE_CATCHER:HPCQ FIRST:00000001_00100228 LAST:00000001_00100228
+000018 INITSIZE:00000000 00002000 INCRSIZE:00000000 00001000
+00002C OPTIONS:30000000
```

** NO SEGMENTS ALLOCATED **

```
HPSQ: 00000001_00005148
+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: 00000001_001002A8
+000000 EYE_CATCHER:HPCQ FIRST:00000001_00100790 LAST:00000001_00100790
+000018 INITSIZE:00000000 00000001 INCRSIZE:00000000 00000001
+00002C OPTIONS:90000000
```

```
HPSQ: 00000001_00005208
+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
```

```
THNQ: 00000001_00100790
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001_001002A8
+000010 PREV:00000001_001002A8 HEAPID:00000001_001002A8
+000020 SEGMENT:00000001_19B00000 SEG_LEN:00000000 00100000
```

```
HANQ: 00000001_19B00000
+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001_001002A8
+000020 SEGMENT:00000001_19B00000 ROOT:00000001_19B00380
+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	0000000119B00380	000000000000FFC80	00000000000000000	00000000000000000	00000000000000000	00000000000000000	00000000000000000

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000119B00000 LEN(X'000000000100000') ASID(X'0028')

0000000119B00040: Allocated storage element, length=0000000000000340.

To display: IP LIST 0000000119B00040 LEN(X'0000000000000340') ASID(X'0028')

0000000119B00050: D7C3C9C2 00000000 00000000 00000000 00000000 00010310 00000000 |PCIB.....|

0000000119B00380: Free storage element, length=000000000000FFC80.

To display: IP LIST 0000000119B00380 LEN(X'000000000000FFC80') ASID(X'0028')

Summary of analysis for Heap Segment 0000000119B00000:

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.

I/O Heap64 Control Blocks

```
HPCQ: 00000001_001002D8
+000000 EYE_CATCHER:HPCQ FIRST:00000001_00100760 LAST:00000001_00100760
+000018 INITSIZE:00000000 00000001 INCRSIZE:00000000 00000001
+00002C OPTIONS:90000000
```

```
HPSQ: 00000001_00005178
+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
```

Figure 145. Example formatted detailed heap segment report from LEDATA Verbexit (Part 6 of 9)

```

THNQ: 00000001_00100760
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001_001002D8
+000010 PREV:00000001_001002D8 HEAPID:00000001_001002D8
+000020 SEGMENT:00000001_19A00000 SEG_LEN:00000000 00100000

HANQ: 00000001_19A00000
+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001_001002D8
+000020 SEGMENT:00000001_19A00000 ROOT:00000001_19A103C0
+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	0000000119A103C0	00000000000EFC40	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

[2] Map of Heap Segment 00000001

To display entire segment: IP LIST 0000000119A00000 LEN(X'000000000100000') ASID(X'0028')

```

0000000119A00040: Allocated storage element, length=0000000000000300.
To display: IP LIST 0000000119A00040 LEN(X'0000000000000300') ASID(X'0028')
0000000119A00050: 00000001 19A00120 00000000 00000001 00000000 00000000 00000000 00000000 |.....|

0000000119A00340: Allocated storage element, length=0000000000000060.
To display: IP LIST 0000000119A00340 LEN(X'0000000000000060') ASID(X'0028')
0000000119A00350: 94859496 99A84B84 81A38100 00000000 00000000 00000000 00000000 00000000 |memory.data.....|

0000000119A003A0: Allocated storage element, length=0000000000010020.
To display: IP LIST 0000000119A003A0 LEN(X'0000000000010020') ASID(X'0028')
0000000119A003B0: A2969485 408481A3 81A29694 85409496 99854084 81A38185 A5859540 94969985 |some datasome more dataeven more|

0000000119A103C0: Free storage element, length=00000000000EFC40.
To display: IP LIST 0000000119A103C0 LEN(X'00000000000EFC40') ASID(X'0028')

```

```

Summary of analysis for Heap Segment 0000000119A00000:
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: 00000001_00100308
+000000 EYE_CATCHER:HPCQ FIRST:00000001_001006A0 LAST:00000001_001006A0
+000018 INITSIZE:00000000 00003000 INCRSIZE:00000000 00002000
+00002C OPTIONS:50000000

HPSQ: 00000001_000051A8
+000000 BYTES_ALLOC:00000000 00002490
+000008 CURR_ALLOC:00000000 00002490 GET_REQ:00000000 0000000E
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000

THNQ: 00000001_001006A0
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001_00100308
+000010 PREV:00000001_00100308 HEAPID:00000001_00100308
+000020 SEGMENT:00000000_25757000 SEG_LEN:00000000 00003000

HANQ: 00000000_25757000
+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001_00100308
+000020 SEGMENT:25757000 ROOT:257594D0 SEG_LEN:00003000
+00002C ROOT_LEN:00000B30

```

This is the last heap segment in the current heap

[1] Free Storage Tree for Heap Segment 00000000

Depth	Node Address	Node Length	Parent Node	Left Node	Right Node	Left Length	Right Length
0	257594D0	00000B30	00000000	00000000	00000000	00000000	00000000

Figure 145. Example formatted detailed heap segment report from LEDATA Verbexit (Part 7 of 9)

```

[2] Map of Heap Segment 00000000

To display entire segment: IP LIST 00000000 LEN(X'00003000') ASID(X'0028')

25757040: Allocated storage element, length=00000388. To display: IP LIST 25757040 LEN(X'00000388') ASID(X'0028')
25757048: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

257573C8: Allocated storage element, length=00000070. To display: IP LIST 257573C8 LEN(X'00000070') ASID(X'0028')
257573D0: D6E2C9D6 00007048 00000000 000077F0 000078A8 00000001 00000001 FFFFFFFF |OSIO.....0...y.....|

25757438: Allocated storage element, length=00000040. To display: IP LIST 25757438 LEN(X'00000040') ASID(X'0028')
25757440: C4C3C2C5 00380000 00007048 00000000 C0880000 80000000 00000000 00000000 |DCBE.....h.....|

:

257590C8: Allocated storage element, length=00000370. To display: IP LIST 257590C8 LEN(X'00000370') ASID(X'0028')
257590D0: C1C6C3C2 00000000 00000000 257590F8 AFCBAFCB 00000000 00000001 08371120 |AFCB.....8.....|

25759438: Allocated storage element, length=00000098. To display: IP LIST 25759438 LEN(X'00000098') ASID(X'0028')
25759440: 007D0000 40404040 40404040 4040F0F0 F0F0F0F0 F0F0F2F5 F7F5C2F5 C5F04B84 |.'.. 000000002575B5E0.d|

257594D0: Free storage element, length=00000B30. To display: IP LIST 257594D0 LEN(X'00000B30') ASID(X'0028')

Summary of analysis for Heap Segment 00000000:
Amounts of identified storage: Free:00000B30 Allocated:00002490 Total:00002FC0
Number of identified areas : Free: 1 Allocated: 14 Total: 15
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: 00000001_00100338
+000000 EYE_CATCHER:HPCQ FIRST:00000001_001006D0 LAST:00000001_001006D0
+000018 INITSIZE:00000000 00001000 INCRSIZE:00000000 00001000
+00002C OPTIONS:30000000

HPSQ: 00000001_000051D8
+000000 BYTES_ALLOC:00000000 00000B10
+000008 CURR_ALLOC:00000000 00000B10 GET_REQ:00000000 0000000A
+000018 FREE_REQ:00000000 00000000 GETMAINS:00000000 00000001
+000028 FREEMAINS:00000000 00000000

THNQ: 00000001_001006D0
+000000 EYE_CATCHER:THNQ FLAGS:00000000 NEXT:00000001_00100338
+000010 PREV:00000001_00100338 HEAPID:00000001_00100338
+000020 SEGMENT:00000000_00007000 SEG_LEN:00000000 00001000

HANQ: 00000000_00007000
+000000 EYE_CATCHER:HANQ FLAGS:00000000 HEAPID:00000001_00100338
+000020 SEGMENT:00007000 ROOT:00007B50 SEG_LEN:00001000
+00002C ROOT_LEN:000004B0

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 00007B50 000004B0 00000000 00000000 00000000 00000000 00000000

[2] Map of Heap Segment 00000000

To display entire segment: IP LIST 00000000 LEN(X'00001000') ASID(X'0028')

00007040: Allocated storage element, length=00000088. To display: IP LIST 00007040 LEN(X'00000088') ASID(X'0028')
00007048: 25757440 00000000 00000000 0B000000 00020000 01008FF0 03724000 00006DD8 |... ..0..._Q|

000070C8: Allocated storage element, length=00000248. To display: IP LIST 000070C8 LEN(X'00000248') ASID(X'0028')
000070D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

00007310: Allocated storage element, length=000004D8. To display: IP LIST 00007310 LEN(X'000004D8') ASID(X'0028')
00007318: EBEC0008 0024B909 00FFB904 00B1E3E0 10000004 E3E0E000 00040530 89E00002 |.....T.....T.....i...|

:

00007B18: Allocated storage element, length=00000038. To display: IP LIST 00007B18 LEN(X'00000038') ASID(X'0028')
00007B20: 00007B20 25759440 00000000 0000007D 00000000 00000000 00000000 00000000 |..#...m .....|

00007B50: Free storage element, length=000004B0. To display: IP LIST 00007B50 LEN(X'000004B0') ASID(X'0028')

```

Figure 145. Example formatted detailed heap segment report from LEDATA Verbexit (Part 8 of 9)

```
Summary of analysis for Heap Segment 00000000:
Amounts of identified storage: Free:000004B0 Allocated:00000B10 Total:00000FC0
Number of identified areas : Free: 1 Allocated: 10 Total: 11
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 145. Example formatted detailed heap segment report from LEDATA Verbexit (Part 9 of 9)

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 heap pools 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.

HPT(3)

64 BIT LANGUAGE ENVIRONMENT DATA

Language Environment Product 04 V01 R0A.00

[1] HEAPP00LS64 Trace Table

[2] POOLID: 3 ASID: 001F AVAILABLE ENTRIES: 12 OF 12

[3] Timestamp: 2008/03/14 18:20:40.239878
Type: FREE Cell Address: 00000001086588E0 CpuId: 01 Tcb: 008D7820

CALL NAME	CALL ADDRESS	CALL OFFSET
GetStorage::~GetStorage()	0000000025B001B0	00000056
foo8()	0000000025B00348	0000006A
foo7()	0000000025B003D8	00000010
foo6()	0000000025B00410	00000010
foo5()	0000000025B00448	00000010
foo4()	0000000025B00480	00000010
foo3()	0000000025B004B8	00000010
foo2()	0000000025B004F0	00000010
foo1()	0000000025B00528	00000010
thread	0000000025B005C8	00000000

Timestamp: 2008/03/14 18:20:40.239875
Type: FREE Cell Address: 0000000108658970 CpuId: 01 Tcb: 008D7820

CALL NAME	CALL ADDRESS	CALL OFFSET
GetStorage::~GetStorage()	0000000025B001B0	00000056
foo9()	0000000025B00260	0000006E
foo8()	0000000025B00348	00000046
foo7()	0000000025B003D8	00000010
foo6()	0000000025B00410	00000010
foo5()	0000000025B00448	00000010
foo4()	0000000025B00480	00000010
foo3()	0000000025B004B8	00000010
foo2()	0000000025B004F0	00000010
foo1()	0000000025B00528	00000000

Timestamp: 2008/03/14 18:20:40.239873
Type: GET Cell Address: 0000000108658970 CpuId: 01 Tcb: 008D7820

CALL NAME	CALL ADDRESS	CALL OFFSET
GetStorage::GetStorage(int)	0000000025B00118	00000058
foo9()	0000000025B00260	0000003A
foo8()	0000000025B00348	00000046
foo7()	0000000025B003D8	00000010
foo6()	0000000025B00410	00000010
foo5()	0000000025B00448	00000010
foo4()	0000000025B00480	00000010
foo3()	0000000025B004B8	00000010
foo2()	0000000025B004F0	00000010
foo1()	0000000025B00528	00000000

Figure 146. Example formatted detailed HEAPP00LS trace report from LEDATA Verbexit (Part 1 of 5)

```

Timestamp: 2008/03/14          18:20:40.239870
Type: GET Cell Address: 00000001086588E0 CpuId: 01 Tcb: 008D7820
CALL NAME CALL ADDRESS CALL OFFSET
GetStorage::~GetStorage(int) 0000000025B00118 00000058
foo8() 0000000025B00348 00000036
foo7() 0000000025B003D8 00000010
foo6() 0000000025B00410 00000010
foo5() 0000000025B00448 00000010
foo4() 0000000025B00480 00000010
foo3() 0000000025B004B8 00000010
foo2() 0000000025B004F0 00000010
foo1() 0000000025B00528 00000010
thread 0000000025B005C8 00000000

Timestamp: 2008/03/14          18:20:40.238024
Type: FREE Cell Address: 00000001086588E0 CpuId: 01 Tcb: 008D7AD0
CALL NAME CALL ADDRESS CALL OFFSET
GetStorage::~~GetStorage() 0000000025B001B0 00000056
foo8() 0000000025B00348 0000006A
foo7() 0000000025B003D8 00000010
foo6() 0000000025B00410 00000010
foo5() 0000000025B00448 00000010
foo4() 0000000025B00480 00000010
foo3() 0000000025B004B8 00000010
foo2() 0000000025B004F0 00000010
foo1() 0000000025B00528 00000010
thread 0000000025B005C8 00000000

Timestamp: 2008/03/14          18:20:40.238021
Type: FREE Cell Address: 0000000108658970 CpuId: 01 Tcb: 008D7AD0
CALL NAME CALL ADDRESS CALL OFFSET
GetStorage::~~GetStorage() 0000000025B001B0 00000056
foo9() 0000000025B00260 0000006E
foo8() 0000000025B00348 00000046
foo7() 0000000025B003D8 00000010
foo6() 0000000025B00410 00000010
foo5() 0000000025B00448 00000010
foo4() 0000000025B00480 00000010
foo3() 0000000025B004B8 00000010
foo2() 0000000025B004F0 00000010
foo1() 0000000025B00528 00000000

Timestamp: 2008/03/14          18:20:40.238016
Type: GET Cell Address: 0000000108658970 CpuId: 01 Tcb: 008D7AD0
CALL NAME CALL ADDRESS CALL OFFSET
GetStorage::~GetStorage(int) 0000000025B00118 00000058
foo9() 0000000025B00260 0000003A
foo8() 0000000025B00348 00000046
foo7() 0000000025B003D8 00000010
foo6() 0000000025B00410 00000010
foo5() 0000000025B00448 00000010
foo4() 0000000025B00480 00000010
foo3() 0000000025B004B8 00000010
foo2() 0000000025B004F0 00000010
foo1() 0000000025B00528 00000000

Timestamp: 2008/03/14          18:20:40.238013
Type: GET Cell Address: 00000001086588E0 CpuId: 01 Tcb: 008D7AD0
CALL NAME CALL ADDRESS CALL OFFSET
GetStorage::~GetStorage(int) 0000000025B00118 00000058
foo8() 0000000025B00348 00000036
foo7() 0000000025B003D8 00000010

```

Figure 146. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 2 of 5)

```

foo6()          0000000025B00410  00000010
foo5()          0000000025B00448  00000010
foo4()          0000000025B00480  00000010
foo3()          0000000025B004B8  00000010
foo2()          0000000025B004F0  00000010
foo1()          0000000025B00528  00000010
thread         0000000025B005C8  00000000

Timestamp: 2008/03/14      18:20:40.158670
Type: GET Cell Address: 0000000108658850 CpuId: 01 Tcb: 008FF038
CALL NAME CALL ADDRESS CALL OFFSET
CEEOPMI   0000000025D95900  00000C8C
CEEOPC    0000000025D588C0  00001672
pthread_create  0000000025D6F7E8  00000628
main      0000000025B00640  000000AE
CELQINIT  0000000025B04010  00000000

Timestamp: 2008/03/14      18:20:40.140382
Type: GET Cell Address: 00000001086587C0 CpuId: 01 Tcb: 008FF038
CALL NAME CALL ADDRESS CALL OFFSET
CEEOPMI   0000000025D95900  00000C8C
pthread_mutex_init  000000002604E6E0  0000005C
pthread_create  0000000025D6F7E8  0000033A
main      0000000025B00640  000000AE
CELQINIT  0000000025B04010  00000000

Timestamp: 2008/03/14      18:20:40.140373
Type: GET Cell Address: 0000000108658730 CpuId: 01 Tcb: 008FF038
CALL NAME CALL ADDRESS CALL OFFSET
CEEOPMI   0000000025D95900  00000C8C
pthread_mutex_init  000000002604E6E0  0000005C
pthread_create  0000000025D6F7E8  000002A2
main      0000000025B00640  000000AE
CELQINIT  0000000025B04010  00000000

Timestamp: 2008/03/14      18:20:40.023500
Type: GET Cell Address: 00000001086586A0 CpuId: 01 Tcb: 008FF038
CALL NAME CALL ADDRESS CALL OFFSET
dllinit   0000000025B767B8  00000090
CEEZIDT   0000000025CE0738  00000ADC
CELQINMN  0000000025CEA830  00000F52
CELQINIT  0000000025B04010  00000000

```

HEAPPOOLS Trace Table

POOLID: 3 ASID: 001F AVAILABLE ENTRIES: 8 OF 8

```

Timestamp: 2008/03/14      18:20:40.239877
Type: FREE Cell Address: 000000002635E058 CpuId: 01 Tcb: 008D7820
CALL NAME CALL ADDRESS CALL OFFSET
GetStorage::~GetStorage()  0000000025B001B0  0000003A
foo8()          0000000025B00348  0000006A
foo7()          0000000025B003D8  00000010
foo6()          0000000025B00410  00000010
foo5()          0000000025B00448  00000010
foo4()          0000000025B00480  00000010
foo3()          0000000025B004B8  00000010
foo2()          0000000025B004F0  00000010
foo1()          0000000025B00528  00000010
thread         0000000025B005C8  00000000

```

Figure 146. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 3 of 5)

```

Timestamp: 2008/03/14          18:20:40.239874
Type: FREE Cell Address: 000000002635E0E0 CpuId: 01 Tcb: 008D7820
CALL NAME                      CALL ADDRESS      CALL OFFSET
GetStorage::~GetStorage()      0000000025B001B0 0000003A
foo9()                         0000000025B00260 0000006E
foo8()                         0000000025B00348 00000046
foo7()                         0000000025B003D8 00000010
foo6()                         0000000025B00410 00000010
foo5()                         0000000025B00448 00000010
foo4()                         0000000025B00480 00000010
foo3()                         0000000025B004B8 00000010
foo2()                         0000000025B004F0 00000010
foo1()                         0000000025B00528 00000000

Timestamp: 2008/03/14          18:20:40.239872
Type: GET Cell Address: 000000002635E0E0 CpuId: 01 Tcb: 008D7820
CALL NAME                      CALL ADDRESS      CALL OFFSET
GetStorage::~GetStorage(int)   0000000025B00118 0000002E
foo9()                         0000000025B00260 0000003A
foo8()                         0000000025B00348 00000046
foo7()                         0000000025B003D8 00000010
foo6()                         0000000025B00410 00000010
foo5()                         0000000025B00448 00000010
foo4()                         0000000025B00480 00000010
foo3()                         0000000025B004B8 00000010
foo2()                         0000000025B004F0 00000010
foo1()                         0000000025B00528 00000000

Timestamp: 2008/03/14          18:20:40.239869
Type: GET Cell Address: 000000002635E058 CpuId: 01 Tcb: 008D7820
CALL NAME                      CALL ADDRESS      CALL OFFSET
GetStorage::~GetStorage(int)   0000000025B00118 0000002E
foo8()                         0000000025B00348 00000036
foo7()                         0000000025B003D8 00000010
foo6()                         0000000025B00410 00000010
foo5()                         0000000025B00448 00000010
foo4()                         0000000025B00480 00000010
foo3()                         0000000025B004B8 00000010
foo2()                         0000000025B004F0 00000010
foo1()                         0000000025B00528 00000010
thread                         0000000025B005C8 00000000

Timestamp: 2008/03/14          18:20:40.238023
Type: FREE Cell Address: 000000002635E058 CpuId: 01 Tcb: 008D7AD0
CALL NAME                      CALL ADDRESS      CALL OFFSET
GetStorage::~GetStorage()      0000000025B001B0 0000003A
foo8()                         0000000025B00348 0000006A
foo7()                         0000000025B003D8 00000010
foo6()                         0000000025B00410 00000010
foo5()                         0000000025B00448 00000010
foo4()                         0000000025B00480 00000010
foo3()                         0000000025B004B8 00000010
foo2()                         0000000025B004F0 00000010
foo1()                         0000000025B00528 00000010
thread                         0000000025B005C8 00000000

```

Figure 146. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 4 of 5)

```

Timestamp: 2008/03/14      18:20:40.238018
Type: FREE Cell Address: 000000002635E0E0 CpuId: 01 Tcb: 008D7AD0
CALL NAME      CALL ADDRESS  CALL OFFSET
GetStorage::~~GetStorage() 0000000025B001B0 0000003A
foo9()         0000000025B00260 0000006E
foo8()         0000000025B00348 00000046
foo7()         0000000025B003D8 00000010
foo6()         0000000025B00410 00000010
foo5()         0000000025B00448 00000010
foo4()         0000000025B00480 00000010
foo3()         0000000025B004B8 00000010
foo2()         0000000025B004F0 00000010
foo1()         0000000025B00528 00000000

Timestamp: 2008/03/14      18:20:40.238015
Type: GET Cell Address: 000000002635E0E0 CpuId: 01 Tcb: 008D7AD0
CALL NAME      CALL ADDRESS  CALL OFFSET
GetStorage::~GetStorage(int) 0000000025B00118 0000002E
foo9()         0000000025B00260 0000003A
foo8()         0000000025B00348 00000046
foo7()         0000000025B003D8 00000010
foo6()         0000000025B00410 00000010
foo5()         0000000025B00448 00000010
foo4()         0000000025B00480 00000010
foo3()         0000000025B004B8 00000010
foo2()         0000000025B004F0 00000010
foo1()         0000000025B00528 00000000

Timestamp: 2008/03/14      18:20:40.238005
Type: GET Cell Address: 000000002635E058 CpuId: 01 Tcb: 008D7AD0
CALL NAME      CALL ADDRESS  CALL OFFSET
GetStorage::~GetStorage(int) 0000000025B00118 0000002E
foo8()         0000000025B00348 00000036
foo7()         0000000025B003D8 00000010
foo6()         0000000025B00410 00000010
foo5()         0000000025B00448 00000010
foo4()         0000000025B00480 00000010
foo3()         0000000025B004B8 00000010
foo2()         0000000025B004F0 00000010
foo1()         0000000025B00528 00000010
thread        0000000025B005C8 00000000
Exiting Language Environment Data

```

Figure 146. Example formatted detailed HEAPPOOLS trace report from LEDATA Verbexit (Part 5 of 5)

[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 147 on page 427 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 438 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: 00000001_00007B18
+000310 CGENE:00000001_0000BAE0 CREN:00000000_00000000
+000320 CPRMS:00000001_00005598 TRACE:00000001 CTHD:00000001_00008F08
+000338 CURR_FECB:00000001_0000B400 CGEN_CPCB:00000001_00008688
+000348 CGEN_CEDB:00000001_0000A620 CFLG3:00 CIO:00000001_000088D8

[2] CGENE: 00000001_0000BAE0
+000000 CGENEYE:.-./ CGENESIZE:00C200C4 CGENEPTR:007C00C1_00C300C5
+0000D4 CERRNO:006000A3 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:08FC4E90_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_00000000_00000000_00000000_00000000_00000000
+000574 IMSPCLIST:0000C048_00000000 ADDRTBL:00000000_00000001
+000710 ABND_CODE:00000000 RSN_CODE:00000000

[3] CEDB: 00000001_0000A620
+000000 EYE:CEDB SIZE:00000C48 PTR:00000001_0000A620
+000010 CLLST:00000000_20C02DB8 CEELANG:0003 CASWITCH:0000
+000020 CLWA:00000001_0000C088 CALTLWA:00000000_00000000
+000030 CCADDR:00000000_20C000D8 CFLGS:00000080 CANCHOR:00000000_00000000
+000050 RPLEN:00000000_00000000 ACBLEN:00000000_00000000
+000060 LC:00000001_0000B270 VALID_HIGH:00000000_20C53BF0
+000070 _LOW:00000000_20C53690 HEAD_FECB:00000000_00000000
+000080 ATEXTIT_CHAIN:00000000_00000000 EMPTY_CHAIN:00000001_0000A770
+000090 MAINPRMS:00000001_08FC5E70 STDINFILE:00000001_0000A2B8
+0000A0 STDOUTFILE:00000001_00009BE8 STDERRFILE:00000001_00009F50
+0000B0 CTYPE:00000000_21324876 TZDFLT:00000000_00003840
+0000C0 CINFO:00000001_0000B370 CMS_WRITE_DISK:4040
+0000CC _DISK_SET:00000000
+0000D0 MIN_FLT:00100000_00000000_00000000_00000000
+0000E0 MAX_FLT:7FFFFFFF_FFFFFFFF_71FFFFFF_FFFFFFFF
+0000F0 FLT_EPS:3C100000 DBL_EPS:34100000_00000000
+000100 LDBL_EPS:26100000_00000000_18000000_00000000_00000000_00000000_00000000_00000000

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 1 of 12)

```

+000118 MTF_MAINTASK_BLK:00000000_00000000      MSG_SETTING:00
+000124 DEPTH:00000000      SCREEN_WIDTH:00000000  USERID:HEALY .
+000138 HEAP24_ANCHOR:00000000_00000000      TCIC:00000000_00000000
+000148 TKCLI:00000000_00000000
+000150 ATEXTIT_FUNC01:00000001 0000A798 00000000 00000000 00000000 00000000 00000000 00000000 00
+000178 ATEXTIT_FUNC02:00000001 0000A7C0 00000000 00000000 00000000 00000000 00000000 00000000 00
+0001A0 ATEXTIT_FUNC03:00000001 0000A7E8 00000000 00000000 00000000 00000000 00000000 00000000 00
+0001C8 ATEXTIT_FUNC04:00000001 0000A810 00000000 00000000 00000000 00000000 00000000 00000000 00
+0001F0 ATEXTIT_FUNC05:00000001 0000A838 00000000 00000000 00000000 00000000 00000000 00000000 00
+000218 ATEXTIT_FUNC06:00000001 0000A860 00000000 00000000 00000000 00000000 00000000 00000000 00
+000240 ATEXTIT_FUNC07:00000001 0000A888 00000000 00000000 00000000 00000000 00000000 00000000 00
+000268 ATEXTIT_FUNC08:00000001 0000A8B0 00000000 00000000 00000000 00000000 00000000 00000000 00
+000290 ATEXTIT_FUNC09:00000001 0000A8D8 00000000 00000000 00000000 00000000 00000000 00000000 00
+0002B8 ATEXTIT_FUNC10:00000001 0000A900 00000000 00000000 00000000 00000000 00000000 00000000 00
+0002E0 ATEXTIT_FUNC11:00000001 0000A928 00000000 00000000 00000000 00000000 00000000 00000000 00
+000308 ATEXTIT_FUNC12:00000001 0000A950 00000000 00000000 00000000 00000000 00000000 00000000 00
+000330 ATEXTIT_FUNC13:00000001 0000A978 00000000 00000000 00000000 00000000 00000000 00000000 00
+000358 ATEXTIT_FUNC14:00000001 0000A9A0 00000000 00000000 00000000 00000000 00000000 00000000 00
+000380 ATEXTIT_FUNC15:00000001 0000A9C8 00000000 00000000 00000000 00000000 00000000 00000000 00
+0003A8 ATEXTIT_FUNC16:00000001 0000A9F0 00000000 00000000 00000000 00000000 00000000 00000000 00
+0003D0 ATEXTIT_FUNC17:00000001 0000AA18 00000000 00000000 00000000 00000000 00000000 00000000 00
+0003F8 ATEXTIT_FUNC18:00000001 0000AA40 00000000 00000000 00000000 00000000 00000000 00000000 00
+000420 ATEXTIT_FUNC19:00000001 0000AA68 00000000 00000000 00000000 00000000 00000000 00000000 00
+000448 ATEXTIT_FUNC20:00000001 0000AA90 00000000 00000000 00000000 00000000 00000000 00000000 00
+000470 ATEXTIT_FUNC21:00000001 0000AAB8 00000000 00000000 00000000 00000000 00000000 00000000 00
+000498 ATEXTIT_FUNC22:00000001 0000AAE0 00000000 00000000 00000000 00000000 00000000 00000000 00
+0004C0 ATEXTIT_FUNC23:00000001 0000AB08 00000000 00000000 00000000 00000000 00000000 00000000 00
+0004E8 ATEXTIT_FUNC24:00000001 0000AB30 00000000 00000000 00000000 00000000 00000000 00000000 00
+000510 ATEXTIT_FUNC25:00000001 0000AB58 00000000 00000000 00000000 00000000 00000000 00000000 00
+000538 ATEXTIT_FUNC26:00000001 0000AB80 00000000 00000000 00000000 00000000 00000000 00000000 00
+000560 ATEXTIT_FUNC27:00000001 0000ABA8 00000000 00000000 00000000 00000000 00000000 00000000 00
+000588 ATEXTIT_FUNC28:00000001 0000ABD0 00000000 00000000 00000000 00000000 00000000 00000000 00
+0005B0 ATEXTIT_FUNC29:00000001 0000ABF8 00000000 00000000 00000000 00000000 00000000 00000000 00
+0005D8 ATEXTIT_FUNC30:00000001 0000AC20 00000000 00000000 00000000 00000000 00000000 00000000 00
+000600 ATEXTIT_FUNC31:00000001 0000AC48 00000000 00000000 00000000 00000000 00000000 00000000 00
+000628 ATEXTIT_FUNC32:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00
+000650 HEAD_FOREIGN_FECB:00000000_00000000
+000658 SNAP_DUMP_COUNT:00000000      ENVIRON:00000000_00000000
+000668 GETENV_BUF:00000000_00000000      _BUF_LEN:00000000_00000000
+000678 CDLL:00000001_08FEE770      INSPECT_GLOBALS:00000000_00000000
+000688 _JMP_BUFF:00000000_00000000      _BACK_END:00000000_00000000
+000698 _FLAGS:00000000      _TAB:00000000_00000000
+0006A8 INTOFFLIST:00000000_00000000      CGEN_CRENT:00000000_00000000
+0006B8 _CPRMS:00000001_00005598      _CEDCXV:00000000_00000000
+0006C8 _CEDCOV:00000000_00000000      _EPCBLIST:00000000_00000000
+0006D8 CAA_ADDR:00000001_00007B18      USERIDLENGTH:00000000_00000005
+0006F0 TZSHR:00000001_0000B520 00000001_0000B528
+000700 MAXUNGETCOUNT:0004      RWSTATIC:00000001_08300050
+000710 RWLEN:00000000_00000170      CSGDLLI:00000000_00000000
+000720 DLLISIZE:00000000_00000000      IOGET_ANY:00000000_2131EC80
+000730 _BELOW:00000000_2131EC70      IOFREE_ANY:00000000_2131EC60
+000740 _BELOW:00000000_2131EC50      CSGSTINIT:00000000_00000000
+000750 STINITSIZE:00000000_00000000      MTFMAINTASKBLK:00000000_00000000
+000760 SIGTABLE:00000001_0000B668      INIT_STDIN:00000001_0000A2B8

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 2 of 12)


```

+000770 _STDOUT:00000001_00009BE8 _STDERR:00000001_00009F50
+000788 TABNUM:00000000_00000008 REDIR:00 AVAIL13:00000000_00000000
+0007A0 OPENMVS_FLAGS:00 MRPSTDR:00000000_2131EC10 MWPSTDR:00000000_2131EC00
+0007B8 MRPSTDC:00000000_00000000 MWPSTDC:00000000_00000000
+0007C8 OWRP1:00000000_00000000 OWRP3:00000000_00000000
+0007D8 STATIC_EDCOV:00000000_00000000 GETENV_BUF2:00000000_00000000
+0007E8 _BUF2_LEN:00000000_00000000 DLCB_MUTEX:00000001_08FC7328
+0007F8 _CONDV:00000001_08FC7330 EDCOV:00000000_00000000
+000808 LCX:00000001_0000B520 MUTEX_ATTR:00000001_08FC7250
+000818 STOR_INIT_B:00001000 _INCR_B:00001000
+000820 STOR_INIT:00003000 _INCR:00002000 DEMANGLE:00000000_00000000
+000838 TEMPR15:00000000_00000000 TERMINATE:00000000_2131EC40
+000848 CXX_INV:00000000_00000000 D4_JOIN_MUTEX_ATTR:00000001_08FC7308
+000860 _MUTEX:00000001_08FC7310 _CONDV_ATTR:00000001_08FC7318
+000870 _CONDV:00000001_08FC7320 DLLANCHOR:00000000_00000000
+000880 DLLLAST:00000000_00000000 MEM24P:00000000_00000000
+000890 RTLMUTEX_ARRAYPTR:00000001_08FC7258 MSGCATLIST:00000000_00000000
+0008A0 SRCHP:00000000_00000000 ETOAP:00000000_00000000
+0008B0 ATOEP:00000000_00000000 NDMGMTP:00000000_00000000
+0008C0 POPENP:00000000_00000000 RND48P:00000000_00000000
+0008D0 BRK_HEAPID:00000000_00000000 START:00000000_00000000
+0008E0 _CURRENT:00000000_00000000 _END:00000000_00000000
+0008F8 RESTARTTABLE:00000000_00000000 SYSLOGP:00000000_00000000
+000908 LOGIN_NAME:..... PREV_UMASK_VAL:00000000
+000918 HFP_LDBL_LMS:00100000_00000000_00000000_00000000_7FFFFFFF_FFFFFFFF_71FFFFFF_FFFFFFFF_26
+000948 BFP_LDBL_LMS:00010000_00000000_00000000_00000000_7FEFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFF_3F
+0009D8 HFP_DBL_LMS:00100000_00000000_00000000_7FFFFFFF_FFFFFFFF_34100000_00000000
+0009F0 BFP_DBL_LMS:00100000_00000000_7FEFFFFFFF_FFFFFFFF_3CB00000_00000000_7FF00000_00000000_FF
+000A38 HFP_MHDC:412B7E15_1628AED3_41171547_652B82FE_406F2DEC_549B9439_40B17217_F7D1CF7A_41
+000AA8 BFP_MHDC:4005BF0A_8B145769_3FF71547_652B82FE_3FDBC7B_1526E50E_3FE62E42_FEFA39EF_40
+000B18 HFP_FLT_LMS:00100000_7FFFFFFF_3C100000
+000B24 BFP_FLT_LMS:00800000_7F7FFFFFFF_34000000_7F800000_FF800000_7FA00000_FFA00000_7FC00000_FF
+000B48 HFP_FHDC:00000010_0006000E_001C0006_000F0020_FFC0FFC0_FFC0FFB2_FFB2FFB2_003F003F_00
+000B70 BFP_FHDC:00010002_00180035_00710006_000F0021_FF83FC03_C003FFDB_FECDECBD_00800400_40
+000B98 ASCII_CCID:0333 EBCDIC_CCID:0417 LOGIN_NAME_A:.....
+000BA8 CINFO_A:00000001_0000BD88 LC_C:00000001_0000BC40
+000BB8 _A:00000001_0000BCA0 LCX_A:00000001_0000BE18
+000BC8 CORRESTABLE:00000000_214CFD58
+000BD0 TZSHR_A:00000001_0000BE18_00000001_0000BE20 CGENVEC3:00000000_00000000
+000BE8 CEDBVEC3:00000000_00000000 FORKQ_HEAD:00000000_00000000
+000BF8 FORKQ_TAIL:00000000_00000000 PTHREAD_YIELD1:0001D100
+000C04 PTHREAD_YIELD2:00002E80 ICONV_MODE:.. _TECH:.....
+000C12 _USERSET:..

```

```

[4] CTHD: 00000001_00008F08
+000000 CTHDEYE:CTHD SIZE:00000528 CTHDPTR:00000001_00008F08
+000010 STORPTR:00000000_00000000 TOKPTR:00000000_251DEF20
+000020 ASCTIME_RESULT:.....
+00003A SNAP_DUMP_FLAG:00 FP_MODE:C4 GMTIME_BKDN:00000001_00009638
+000048 TIMECALLED:00000000 DATECALLED:00000000
+000050 DTCALLED:00000000 LOC_CALLED:00000000
+000058 DOFMTO_DISCARDS:00000000_00000000 CERRNO:00000079
+000068 AMRC:00000000_25757278 AMRC2:00000000_25757380
+000078 GDATE:00000000_00000000 OPTARGV:00000000_00000000
+000088 OPTERRV:00000001 OPTINDV:00000001
+000090 OPTOPTV:00000000 OPTSIND:00000000 DLGHTV:00000000
+0000A0 TZONEV:00000000_00000000 GTDTERrv:00000000
+0000B0 OPTARGP:00000001_00008F88 OPTERRP:00000001_00008F90
+0000C0 OPTINDP:00000001_00008F94 OPTOPTP:00000001_00008F98
+0000D0 DLGHTP:00000001_00008FA0 TZONEP:00000001_00008FA8
+0000E0 GTDTERrv:00000001_00008FB0 RNDSTGP:00000000_00000000
+0000F0 LOCNAM:00000000_00000000 ENCRYPTP:00000000_00000000
+000100 CRYPTP:00000000_00000000 RND48P:00000000_00000000

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 3 of 12)

```

+000110 L64AP:00000000_00000000 WCSTOKP:00000000_00000000
+000120 CUSERP:00000000_00000000 GPASSP:00000000_00000000
+000130 UTMPXP:00000000_00000000 NDMGMT:00000000_00000000
+000140 RECOMP:00000000_00000000 STACKPTR:00000000_00000000
+000150 STACKSIZE:00000000_00000000 STACKFLAGS:40 MCVTP:00000000
+000160 H_ERRNO:00000000 SD:FFFFFFFF HOSTENT_DATA_P:00000000_00000000
+000170 HOSTENT_P:00000000_00000000 NETENT_DATA_P:00000000_00000000
+000180 NETENT_P:00000000_00000000 PROTOENT_DATA_P:00000000_00000000
+000190 PROTOENT_P:00000000_00000000 SERVENT_DATA_P:00000000_00000000
+0001A0 SERVENT_P:00000000_00000000 NTOA_BUF:.....
+0001C0 _LOC1V:00000000_00000000 _LOCSV:00000000_00000000
+0001D0 HERRNOP:00000001_00009068 _LOC1P:00000000_00000000
+0001E0 REXECP:00000000_00000000 CXSEXCEPTION:00000000_00000000
+0001F0 TEMPDCBE:00000000_2135A068 T_ERRNOV:00000000
+000200 T_ERRNOP:00000001_00009100 _LOC1P:00000000_00000000
+000210 _loc2p:00000000_00000000 _locsp:00000000_00000000
+000220 _loc1v:00000000_00000000 _loc2v:00000000_00000000
+000230 THD_STORAGE:00000000_00000000 CONTEXT_LINK:00000000_00000000
+000240 FLAGS1:00000000 LABEL_VAR:00000001_00009768
+000250 ABND_CODE:00000000 RSN_CODE:00000000
+000258 STRFTIME_ERADTCALLED:00000000
+00025C STRFTIME_ERADATECALLED:00000000
+000260 STRFTIME_ERATIMECALLED:00000000
+000264 STRFTIME_ERAYEARCALLED:00000000 MBRLEN_STATE:0000
+00026A MBRTOWC_STATE:0000 WCRTOMB_STATE:0000
+00026E MBSRTOWCS_STATE:0000 WCSRTOBMS_STATE:0000 MBLLEN_STATE:0000
+000274 MBTOWC_STATE:0000 CURR_HEAP_ID:00000000 CURR_CAA:00000000_00000000
+000288 CURR_MOD_HANDLE:00000000_00000000 CURR_BMR:00000000_00000000
+000298 CU_LIST:00000000_00000000 CURR_STATUS:00
+0002A8 RAND_NEXT:00000000_00000001 STRErrorBUF:00000001_00008C90
+0002B8 TMPAREA:00000000_00000000 IOWORKAREA:00000000_2135A140
+0002C8 TEMPDCB:00000000_000070D0 TEMPJFCB:00000000_00007130
+0002D8 TEMPDCB:00000000_2135A0A0 NAMEBUF:00000000_00000000
+0002E8 ERRNO_JR:C25F0001 RET_STRUCT:00000000_00000000
+0002F8 BKDN_IS_LOCALTIME:00000000 SWPRINTF_SIZE:00008000
+000300 SWPRINTF_BUF:00000000_00000000 S99P:2135A048
+000310 MUTEXCTARRAY:00000001_000099C0
+000318 STRFTIME_ERANAMECALLED:00000000 DLL_LOADLEVEL:00000000
+000340 _CONSTLIST:00000000_00000000 FCB_MUTEX:00000000_00000000
+000350 HSPABHWA:00000000_00000000 MUTEX_SAVE:00000001_00009A78
+000368 INITIAL_CPU_TIME:40C065E0_00000000 FCB_MUTEX_OK:00000001
+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_00008DBC CTHD_OURFDSET:00000000_00000000
+000470 IEDECWAP:00000000_00000000 RETVAL_P:00000000_00000000
+000490 CTHD_SETENV_FB:..... LIBASCIIWAP:00000000
+0004AC SETLOCALE_A_CALLED:00 ASCIINAMEBUFL:0000
+0004B8 ASCIINAMEBUF:00000000_00000000 LOCNAME_A:00000000_00000000
+0004C8 EBCDICENTRY:00000001_08358B40 ASCIIENTRY:00000001_08358B58
+0004D8 CTHD_NASCIIWAP:00000000_00000000 CTHD_PROCRESP:00000000_00000000
+0004E8 CTHD_OPTN_P:00000000_00000000 CTHD_GAI_STRError_P:00000000_00000000
+0004F8 CTHD_CKPATHDD_PARAMS:00000000 CTHD_HEXDEC_PTR:00000000
+000504 CTHD_DECHEX_PTR:00000000

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 4 of 12)

```

[5]      CPCB: 00000001_00008688
+000000 CPCB EYE:CPCB      CPCB SIZE:000000C0      CPCB_PTR:00000000_00000000
+000010 FLAGS1:40000000      TTKNHDR:00000000_00000000      TTKN:00000000_00000000
+000024 FOOTPRINT:00000000_00000001      CODE370:0000A620_00000000
+000034 CIO:00000000_00000001      Reuse:000088D8      RSAbove:00000000_00000001
+00004C  _RSAboveLen:00008688      _RSBelow:00000000_00000000
+00005C  _RSBelowLen:00000000

```

```

[6]      CIO: 00000001_000088D8
+000000 EYE:CIO      SIZE:00000108      PTR:00000000_00000000      FLG1:01
+000011 FLG2:80      FLG3:00      FLG4:00      DUMMYF:00000001_000089E0
+000020 EDCZ24:00000000_00000000      FCBSTART:00000000_2135C0F8
+000030 DUMMYFCB:00000001_00008A08      MFCBSTART:00000001_19A00050
+000040 IOANYLIST:00000000_00000000      IOBELOWLIST:00000000_00000000
+000050 FCBDLIST:00000001_00009C08      PERRORBUF:00000001_000087A0
+000060 TMPCCOUNTER:00000000_00000000      TEMPMEM:00000000
+000070 PROMPTBUF:00000000_00000000      IO24:00000000_00007318
+000080 IOEXITS:00000000_000077D0      TERMINALCHAIN:00000000_00000000
+000090 VANCHOR:00000000_00000000      XTI:00000000_00000000
+0000A0 ENOWP24:00000000_2131F200      MAXNUMDESCRPS:00000000_00000000
+0000B0 DESCARRAY:00000000_00000000      TEMPFILENUM:00000000
+0000C8 CSS:00000000_00000000      DUMMY_NAME:.....
+0000D8 HOSTNAME_CACHE:00000000_00000000      HOSTADDR_CACHE:00000000_00000000
+0000E8 IO31:00000000_2131FC10      LAST_FD_CLOSE:00000000_00000000
+0000F8 IOGET64:00000000_2131EC30      IOFREE64:00000000_2131EC20

```

Exiting CRTL Environment Data

```

*****
64 BIT CRTL I/O CONTROL BLOCKS
*****

```

```

      CIO: 00000001_000088D8
+000000 EYE:CIO      SIZE:00000108      PTR:00000000_00000000      FLG1:01
+000011 FLG2:80      FLG3:00      FLG4:00      DUMMYF:00000001_000089E0
+000020 EDCZ24:00000000_00000000      FCBSTART:00000000_2135C0F8
+000030 DUMMYFCB:00000001_00008A08      MFCBSTART:00000001_19A00050
+000040 IOANYLIST:00000000_00000000      IOBELOWLIST:00000000_00000000
+000050 FCBDLIST:00000001_00009C08      PERRORBUF:00000001_000087A0
+000060 TMPCCOUNTER:00000000_00000000      TEMPMEM:00000000
+000070 PROMPTBUF:00000000_00000000      IO24:00000000_00007318
+000080 IOEXITS:00000000_000077D0      TERMINALCHAIN:00000000_00000000
+000090 VANCHOR:00000000_00000000      XTI:00000000_00000000
+0000A0 ENOWP24:00000000_2131F200      MAXNUMDESCRPS:00000000_00000000
+0000B0 DESCARRAY:00000000_00000000      TEMPFILENUM:00000000
+0000C8 CSS:00000000_00000000      DUMMY_NAME:.....
+0000D8 HOSTNAME_CACHE:00000000_00000000      HOSTADDR_CACHE:00000000_00000000
+0000E8 IO31:00000000_2131FC10      LAST_FD_CLOSE:00000000_00000000
+0000F8 IOGET64:00000000_2131EC30      IOFREE64:00000000_2131EC20

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 5 of 12)

```

[7]   FFIL: 00000000_2135C0D0
+000000 MARKER1:AFCB      __FP:00000000_2135C0F8
+000010 MARKER2:AFCAFCB   FCBMUTEX:00000001_08371120
+000020 THREADID:2109B260 00000000

   File name: memory.data
   FCB: 00000000_2135C0F8
+000000 BUFPTR:00000001_19A003D5   COUNTIN:00000000 00000000
+000010 COUNTOUT:00000000 0000FFDB   READFUNC:00000000_2131F210
+000020 WRITEFUNC:00000000_2131FC00   FLAGS1:1000   DEPTH:0000
+000030 NAME:00000000_2135C3F0   _LENGTH:00000000 0000000B
+000040 _BUFSIZE:00000000 00000048   MEMBER:.....   NEXT:00000000_2135A6A0
+000058 PREV:00000000_00000000   PARENT:00000000_2135C0F8
+000068 CHILD:00000000_00000000   DDNAME:.....   FD:FFFFFFFF
+00007D DEVTYPE:08   FCBTYPE:0055   FSCE:00000000_2135C2E0
+000088 UNGETBUF:00000000_2135C2E0   REPOS:00000000_2131FBF0
+000098 GETPOS:00000000_2131FB80   CLOSE:00000000_2131FB70
+0000A8 FLUSH:00000000_2131FBE0   UTILITY:00000000_2131FB60
+0000B8 USERBUF:00000000_00000000   LRECL:00000000 00000400
+0000C8 BLKSIZE:00000000_00010000   REALBUFPTR:00000000_00000000
+0000D8 UNGETCOUNT:00000000 00000000   BUFSIZE:00000000 00010000
+0000E8 BUF:00000001_19A003B0   CURSOR:00000001_19A003B0
+0000F8 ENDODATA: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_2131F780
+000150 DBCSTART:00000000_00000000   UTILITYAREA:00000000_00000000
+000160 INTERCEPT:00000000_00000000   FLAGS2:43020008 40001100
+000170 DBCSTATE:0000   FCB_CPCB:00000001_00000868
+0001C0 LLPOSMAJOR:00000000 00000000
+0001C8 LLSAVEMAJOR:00000000 00000000
+0001D0 LLPOSMINOR:00000000 00000000
+0001D8 LLSAVEMINOR:00000000 00000000

   MEMO: 00000000_2135C2E0
+000000 MEMO_EYE:MEMO   MFCB:00000001_19A00050   NEBULA:00000001_19A00120
+000018 NEBINDEX:0001   CURRDS:00000000 00000000   READ:00000000_2131F210
+000030 WRITE:00000000_2131FC00   REPOS:00000000_2131FBF0
+000040 FLUSH:00000000_2131FBE0

   FFIL: 00000000_2135A678
+000000 MARKER1:AFCB      __FP:00000000_2135A6A0
+000010 MARKER2:AFCAFCB   FCBMUTEX:00000001_08371100
+000020 THREADID:2109B260 00000000

   File name: myfile.data

   FCB: 00000000_2135A6A0
+000000 BUFPTR:00000000_2135B0E3   COUNTIN:00000000 00000000
+000010 COUNTOUT:00000000 00000FE5   READFUNC:00000000_2131F210
+000020 WRITEFUNC:00000000_2131F7A0   FLAGS1:1000   DEPTH:0000
+000030 NAME:00000000_2135A998   _LENGTH:00000000 0000000B
+000040 _BUFSIZE:00000000 00000048   MEMBER:.....   NEXT:00000001_0000A2D8
+000058 PREV:00000000_2135C0F8   PARENT:00000000_2135A6A0
+000068 CHILD:00000000_00000000   DDNAME:.....   FD:00000000
+00007D DEVTYPE:09   FCBTYPE:007C   FSCE:00000000_2135A888
+000088 UNGETBUF:00000000_2135A888   REPOS:00000000_2131F870
+000098 GETPOS:00000000_2131F860   CLOSE:00000000_2131F840
+0000A8 FLUSH:00000000_2131F850   UTILITY:00000000_2131F800
+0000B8 USERBUF:00000000_00000000   LRECL:00000000 00000000
+0000C8 BLKSIZE:00000000_00000000   REALBUFPTR:00000000_2135B0C8
+0000D8 UNGETCOUNT:00000000 00000000   BUFSIZE:00000000 00001000

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 6 of 12)

```

+0000E8 BUF:00000000_2135B0C8 CURSOR:00000000_00000000
+0000F8 ENDOFDATA:00000000_00000000 SAVEDBUF:00000000_00000000
+000108 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_2131F780
+000150 DBCSTART:00000000_00000000 UTILITYAREA:00000000_00000000
+000160 INTERCEPT:00000000_00000000 FLAGS2:40120040_00001300
+000170 DBCSSTATE:0000 FCB_CPCB:00000001_00008688
+0001C0 LLPOSMAJOR:00000000_00000000
+0001C8 LLSAVEMAJOR:00000000_00000000
+0001D0 LLPOSMINOR:00000000_00000000
+0001D8 LLSAVEMINOR:00000000_00000000

HFSF: 00000000_2135A888
+000000 HFSF_EYE:HFSF READ:00000000_2131F210 WRITE:00000000_2131F7A0
+000018 REPOS:00000000_2131F870 GETPOS:00000000_2131F860
+000028 FLUSH:00000000_2131F850 READBUFLLEN:00000000_00000000
+000038 OPENFLAG:00000491 FLAG1:00000000 HFSF_ST_MODE:030001A4
+000048 HFSF_LAST_FSTAT:43281E34 A142E3FC

FFIL: 00000001_0000A2B0
+000000 MARKER1:AFCB __FP:00000001_0000A2D8
+000010 MARKER2:AFCB_AFCB FCBMUTEX:00000001_08FC7290
+000020 THREADID:2109B260_00000000

File name: DD:SYSIN

FCB: 00000001_0000A2D8
+000000 BUFPTR:00000000_00000000 COUNTIN:00000000_00000000
+000010 COUNTOUT:00000000_00000000 READFUNC:00000000_2131FEC0
+000020 WRITEFUNC:00000000_2131F200 FLAGS1:8000 DEPTH:0000
+000030 NAME:00000001_0000A5D0 _LENGTH:00000000_0000000B
+000040 _BUFSIZE:00000000_00000048 MEMBER:..... NEXT:00000001_00009F70
+000058 PREV:00000000_2135A6A0 PARENT:00000001_0000A2D8
+000068 CHILID:00000000_00000000 DDNAME:SYSIN FD:FFFFFFFF
+00007D DEVTYPE:06 FCBTYP:0041 FSCE:00000001_0000A4C0
+000088 UNGETBUF:00000001_0000A4C0 REPOS:00000000_213206C0
+000098 GETPOS:00000000_2131FC60 CLOSE:00000000_2131FD80
+0000A8 FLUSH:00000000_2131FEB0 UTILITY:00000000_2131FEA0
+0000B8 USERBUF:00000000_00000000 LRECL:00000000_00000000
+0000C8 BLKSIZE:00000000_00001800 REALBUFPTR:00000000_00000000
+0000D8 UNGETCOUNT:00000000_00000000 BUFSIZE:00000000_00001801
+0000E8 BUF:00000000_00000000 CURSOR:00000000_00000000
+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_2131FC20 EXITUNGETC:00000000_00000000
+000150 DBCSTART:00000000_00000000 UTILITYAREA:00000000_00000000
+000160 INTERCEPT:00000000_00000000 FLAGS2:00110000_60088040
+000170 DBCSSTATE:0000 FCB_CPCB:00000001_00008688
+0001C0 LLPOSMAJOR:00000000_00000000
+0001C8 LLSAVEMAJOR:00000000_00000000
+0001D0 LLPOSMINOR:00000000_00000000
+0001D8 LLSAVEMINOR:00000000_00000000

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 7 of 12)

```

OSNS: 00000001_0000A4C0
+000000 OSNS_EYE:OSNS READ:00000000_2131FD70 WRITE:00000000_2131F200
+000018 REPOS:00000000_213206C0 GETPOS:00000000_2131FC60
+000028 CLOSE:00000000_2131FD80 FLUSH:00000000_2131FD50
+000038 UTILITY:00000000_2131FD40 EXITFTELL:00000000_2131FC20
+000048 EXITUNGETC:00000000_00000000 OSIOBLK:00000000_2135A5C8
+000058 NEWLINEPTR:00000000_00000000 RECLENGTH:00000000_00001800
+000068 FLAGS:01800000

OSIO: 00000000_2135A5C8
+000000 OSIO_EYE:OSIO DCBW:00000000 DCBRU:00007A00
+00000C JFCB:00007A68 CURRMBUF:00000000 MBUFCOUNT:00000001
+000018 READMAX:00000000 CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF BLKSPERTRK:0000 OSIO_ACCESS_METHOD:02
+000027 OSIO_NOSEEK_TO_SEEK:00 FIRSTPOS:00000000
+00002C LASTPOS:00000000 NEWPOS:00000000 READFUNCNUM:00000003
+000038 WRITEFUNCNUM:00000004 FCB:00000001_0000A2D8 PARENT:2135A5C8
+00004C FLAGS1:90000000 DCBERU:2135A638 DCBEW:00000000
+000058 OSIO_VOLSEQ:0000 OSIO_NEWVOLSEQ:0000 OSIO_EXT:00000000
+000060 OSIO_HIGHVOL:0000 APPENDEDLASTVOLSEQ:0000
+000064 OSIO_JFCBX:00000000

DCB: 00000000_00007A00
+000000 DCBRELAD:2135A638 DCBFDAD:00000000_00000000
+000014 DCBBUFNO:05 DCBSRG1:40 DCBEODAD:000000 DCBRECFCM:C0
+000025 DCBEXLSA:0077D8 DCBDDNAM:.u..... DCBMACR1:CD
+000033 DCBMACR2:9A DCBSYNAD:000000 DCBBLKSI:1800 DCBNCP:00
+000052 DCBLRECL:0000

DCBE: 00000000_2135A638
+000000 DCBEID:DCBE DCBELEN:0038 RESERVED0:0000
+000008 DCBEDCB:00007A00 DCBERELA:00000000 DCBEFLG1:C0
+000011 DCBEFLG2:88 DCBENSTR:0000 DCBEFLG3:80
+000024 DCBESIZE:00000000 DCBEEODA:20E0FE54
+00002C DCBESYNA:20E0FDD8 MULTSDN:00

JFCB: 00000000_00007A68
+000000 JFCBDSNM:NULLFILE
+00002C JFCBELNM: JFCBTSDM:00 JFCBDSCB:000000
+000046 JFCBVLSQ:0000 JFCBIND1:00 JFCBIND2:C1
+000058 JFCBUFNO:00 JFCDSRG1:00 JFCDSRG2:00
+000064 JFCRECFM:00 JFCBLKSI:0000 JFCLRECL:0000 JFCNCP:00
+000075 JFCBNVOL:00 JFCBNVOLS:
+000094 JFCBEXTL:00 JFCBEXAD:000000 JFCFLG1:00
+0000AE JFCBVLCT:01

FFIL: 00000001_00009F48
+000000 MARKER1:AFCB __FP:00000001_00009F70
+000010 MARKER2:AFCAFCB FCBMUTEX:00000001_08FC7268
+000020 THREADID:2109B260 00000000

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 8 of 12)

File name: DD:SYSOUT

```
FCB: 00000001_00009F70
+000000 BUFPTR:00000000_2135C45F   COUNTIN:00000000_00000000
+000010 COUNTOUT:00000000_0000006A   READFUNC:00000000_2131F210
+000020 WRITEFUNC:00000000_2131FDC0   FLAGS1:9000   DEPTH:0000
+000030 NAME:00000001_0000A268   _LENGTH:00000000_0000000B
+000040 _BUFSIZE:00000000_00000048   MEMBER:.....   NEXT:00000001_00009C08
+000058 PREV:00000001_0000A2D8   PARENT:00000001_00009F70
+000068 CHILD:00000000_00000000   DDNAME:SYSOUT   FD:FFFFFFFF
+00007D DEVTYPE:02   FCCTYPE:0043   FSCE:00000001_0000A158
+000088 UNGETBUF:00000001_0000A158   REPOS:00000000_213206C0
+000098 GETPOS:00000000_2131FC60   CLOSE:00000000_2131FDE0
+0000A8 FLUSH:00000000_2131FDB0   UTILITY:00000000_2131FDA0
+0000B8 USERBUF:00000000_00000000   LRECL:00000000_00000089
+0000C8 BLKSIZE:00000000_00000372   REALBUFPTR:00000000_00000000
+0000D8 UNGETCOUNT:00000000_00000000   BUFSIZE:00000000_0000008A
+0000E8 BUF:00000000_2135C440   CURSOR:00000000_2135C444
+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:0004   SAVESTATE:0000
+000140 EXITFTELL:00000000_2131FC20   EXITUNGETC:00000000_2131F780
+000150 DBCSTART:00000000_00000000   UTILITYAREA:00000000_00000000
+000160 INTERCEPT:00000000_00000000   FLAGS2:42228020_2A188040
+000170 DBCSTATE:0000   FCB_CPCB:00000001_00008688
+0001C0 LLPOSMAJOR:00000000_00000000
+0001C8 LLSAVEMAJOR:00000000_00000000
+0001D0 LLPOSMINOR:00000000_00000000
+0001D8 LLSAVEMINOR:00000000_00000000

OSNS: 00000001_0000A158
+000000 OSNS_EYE:OSNS   READ:00000000_2131F210   WRITE:00000000_2131FDC0
+000018 REPOS:00000000_213206C0   GETPOS:00000000_2131FC60
+000028 CLOSE:00000000_2131FDE0   FLUSH:00000000_2131FDB0
+000038 UTILITY:00000000_2131FDA0   EXITFTELL:00000000_2131FC20
+000048 EXITUNGETC:00000000_2131F780   OSIOBLK:00000000_2135A518
+000058 NEWLINEPTR:00000000_2135C4C9   RECLENGTH:00000000_00000085
+000068 FLAGS:84800000

OSIO: 00000000_2135A518
+000000 OSIO_EYE:OSIO   DCBW:000078E0   DCBRU:00000000
+00000C JFCB:00007948   CURRMBUF:00007B20   MBUFCOUNT:00000001
+000018 READMAX:00000001   CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF   BLKSPERTRK:0000   OSIO_ACCESS_METHOD:02
+000027 OSIO_NOSEEK_TO_SEEK:00   FIRSTPOS:00000000
+00002C LASTPOS:00000000   NEWPOS:00000000   READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005   FCB:00000001_00009F70   PARENT:2135A518
+00004C FLAGS1:80000000   DCBERU:00000000   DCBEW:2135A588
+000058 OSIO_VOLSEQ:0000   OSIO_NEWVOLSEQ:0000   OSIO_EXT:00000000
+000060 OSIO_HIGHVOL:0000   APPENDEDLASTVOLSEQ:0000
+000064 OSIO_JFCBX:00000000

DCB: 00000000_000078E0
+000000 DCBRELAD:2135A588   DCBFDAD:00000000_00000002
+000014 DCBBUFNO:01   DCBSRG1:40   DCBEODAD:000000   DCBREFCM:54
+000025 DCBEXLSA:0077D8   DCBDDNAM:...&...   DCBMACR1:56
+000033 DCBMACR2:00   DCBSYNAD:000000   DCBBLKSI:0372   DCBNCP:21
+000052 DCBLRECL:007D
```

Figure 147. Example formatted C/C++ output from LEDATA VerbeXit (Part 9 of 12)

```

DCBE: 00000000_2135A588
+000000 DCBEID:DCBE      DCBELEN:0038      RESERVED0:0000
+000008 DCBEDCB:000078E0      DCBERELA:00000000      DCBEFLG1:C0
+000011 DCBEFLG2:88      DCBENSTR:0000      DCBEFLG3:80
+000024 DCBESIZE:00000000      DCBEEODA:20E0FE54
+00002C DCBESYNA:20E0FDD8      MULTSDN:00

JFCB: 00000000_00007948
+000000 JFCBDSNM:HEALY.CELQSAMP.JOB24799.D0000104.?
+00002C JFCBELNM:      JFCBTSDM:20      JFCBDSCB: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      JFCBNVOLS:
+000094 JFCBEXTL:00      JFCBEXAD:000000      JFCFLGS1:00
+0000AE JFCBVLCT:01

MBUF: 00000000_00007B20
+000000 NEXTMBUF:00007B20_      BUFFER:2135C440_      CHECKRESULT:00000000
+00000C BLKSIZE:0000007D
+000010 DECB:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

FFIL: 00000001_00009BE0
+000000 MARKER1:AFCB      __FP:00000001_00009C08
+000010 MARKER2:AFCBFAFCB      FCBMUTEX:00000001_08FC7268
+000020 THREADID:2109B260 00000000

File name: DD:SYSPRINT

FCB: 00000001_00009C08
+000000 BUFPTR:00000000_2135A485      COUNTIN:00000000 00000000
+000010 COUNTOUT:00000000 00000084      READFUNC:00000000_2131F210
+000020 WRITEFUNC:00000000_2131FDC0      FLAGS1:9000      DEPTH:0000
+000030 NAME:00000001_00009F00      _LENGTH:00000000 0000000B
+000040 _BUFSIZE:00000000 00000048      MEMBER:.....      NEXT:00000001_00008A08
+000058 PREV:00000001_00009F70      PARENT:00000001_00009C08
+000068 CHILD:00000000_00000000      DDNAME:SYSPRINT      FD:FFFFFFFF
+00007D DEVTYPE:02      FCBTYP:0043      FSCE:00000001_00009DF0
+000088 UNGETBUF:00000001_00009DF0      REPOS:00000000_213206C0
+000098 GETPOS:00000000_2131FC60      CLOSE:00000000_2131FDE0
+0000A8 FLUSH:00000000_2131FDB0      UTILITY:00000000_2131FDA0
+0000B8 USERBUF:00000000_00000000      LRECL:00000000 00000089
+0000C8 BLKSIZE:00000000_00000372      REALBUFPTR:00000000_00000000
+0000D8 UNGETCOUNT:00000000 00000000      BUFSIZE:00000000 0000008A
+0000E8 BUF:00000000_2135A480      CURSOR:00000000_2135A484
+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:0002      SAVESTATE:0000
+000140 EXITFTELL:00000000_2131FC20      EXITUNGETC:00000000_2131F780
+000150 DBCSTART:00000000_00000000      UTILITYAREA:00000000_00000000

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 10 of 12)


```

+000160 INTERCEPT:00000000 00000000  FLAGS2:43128020 2A188040
+000170 DBCSSTATE:0000  FCB_CPCB:00000001_00008688
+0001C0 LLPOSMAJOR:00000000 00000000
+0001C8 LLSAVEMAJOR:00000000 00000000
+0001D0 LLPOSMINOR:00000000 00000000
+0001D8 LLSAVEMINOR:00000000 00000000

  OSNS: 00000001_00009DF0
+000000 OSNS_EYE:OSNS  READ:00000000_2131F210  WRITE:00000000_2131FDC0
+000018 REPOS:00000000_213206C0  GETPOS:00000000_2131FC60
+000028 CLOSE:00000000_2131FDE0  FLUSH:00000000_2131FDB0
+000038 UTILITY:00000000_2131FDA0  EXITFTELL:00000000_2131FC20
+000048 EXITUNGETC:00000000_2131F780  OSIOBLK:00000000_2135A3D0
+000058 NEWLINEPTR:00000000_2135A509  RECLENGTH:00000000 00000085
+000068 FLAGS:84800000

  OSIO: 00000000_2135A3D0
+000000 OSIO_EYE:OSIO  DCBW:00007048  DCBRU:00000000
+00000C JFCB:000077F0  CURRMBUF:000078A8  MBUFCOUNT:00000001
+000018 READMAX:00000001  CURBLKNUM:FFFFFFFF
+000020 LASTBLKNUM:FFFFFFFF  BLKSPERTRK:0000  OSIO_ACCESS_METHOD:02
+000027 OSIO_NOSEEK_TO_SEEK:00  FIRSTPOS:00000000
+00002C LASTPOS:00000000  NEWPOS:00000000  READFUNCNUM:00000002
+000038 WRITEFUNCNUM:00000005  FCB:00000001_00009C08  PARENT:2135A3D0
+00004C FLAGS1:80000000  DCBERU:00000000  DCBEW:2135A440
+000058 OSIO_VOLSEQ:0000  OSIO_NEWVOLSEQ:0000  OSIO_EXT:00000000
+000060 OSIO_HIGHVOL:0000  APPENDEDLASTVOLSEQ:0000
+000064 OSIO_JFCBX:00000000

  DCB: 00000000_00007048
+000000 DCBRELAD:2135A440  DCBFDAD:00000000 00000000
+000014 DCBBUFNO:01  DCBSRG1:40  DCBEODAD:000000  DCBREFCM:54
+000025 DCBEXLSA:0077D8  DCBDDNAM:.@.&...  DCBMACR1:56
+000033 DCBMACR2:00  DCBSYNAD:000000  DCBBLKSI:0372  DCBNCP:21
+000052 DCBLRECL:0016

  DCBE: 00000000_2135A440
+000000 DCBEID:DCBE  DCBELEN:0038  RESERVED0:0000
+000008 DCBEDCB:00007048  DCBERELA:00000000  DCBEFLG1:C0
+000011 DCBEFLG2:88  DCBENSTR:0000  DCBEFLG3:80
+000024 DCBESIZE:00000000  DCBEEODA:20E0FE54
+00002C DCBESYNA:20E0FDD8  MULTSDN:00

  JFCB: 00000000_000077F0
+000000 JFCBDSNM:HEALY.CELQSAMP.JOB24799.D0000102.?
+00002C JFCBELNM:  JFCBTSDM:20  JFCBDSCB:000000
+000046 JFCBVLSQ:0000  JFCBIND1:00  JFCBIND2:81
+000058 JFCBUFNO:00  JFCDSRG1:00  JFCDSRG2:00
+000064 JFCREFCM:00  JFCBLKSI:0000  JFCLRECL:0000  JFCNCP:00
+000075 JFCBNVOL:00  JFCBNVOLS:
+000094 JFCBEXTL:00  JFCBEXAD:000000  JFCFLGS1:00
+0000AE JFCBVLCT:01

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 11 of 12)

```

MBUF: 00000000_000078A8
+000000 NEXTMBUF:000078A8_ BUFFER:2135A480_ CHECKRESULT:00000000
+00000C BLKSIZE:00000016
+000010 DECB:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Dummy FCB encountered at location 0000000100008A08

AMRC: 00000000_2135A278
+000000 CODE:00000000 RBA:00000000 LAST_OP:00000098
+00000C FILL_LEN:00000000 MSG_LEN:00000000
+000014 STR1:.....
+000050 STR1_CONT:.....
+00008C PARM0R0:00000000 PARMR1:00000000
+00009C STR2:.....
+0000DC RPLFDBWD:00000000 XRBA:00000000 00000000
+0000E8 AMRC_NOSEEK_TO_SEEK:00

AMRC2: 00000000_2135A380
+000000 __ERROR2:00000000 00000000 __FILEPTR:00000000_

File name: memory.data

[8] MFCB: 00000001_19A00050
+000000 FIRSTNEBULA:00000001_19A00120 REFCNT:00000000 00000001
+000010 RESERVED:00000000 00000000 NEXTMFCB:00000000_00000000
+000020 WRITEFCB:00000000_2135C0F8 FLAG1:0001 DEPTH:0000
+000030 NAME:00000001_19A00350 NAMELENGTH:00000000 0000000B
+000040 NAMEBUFSIZE:00000000 00000048 MEMBER:.....
+000050 NEXTFCB:00000000_00000000 PREVFCB:00000000_00000000
+000060 PARENTFCB:00000000_2135C0F8 CHILDFCB:00000000_00000000
+000070 PREVFCB:00000000_00000000 PARENTMFCB:00000001_19A00050
+000080 CHILDMFCB:00000000_00000000 HIPERKEY:00000000 00000000
+000090 CURHSPBYTES:00000000 00000000 LASTBYTE:0000
+00009A CREATELEVEL:0000 FLAG2:00000000
+0000A0 LASTNEBULA:00000001_19A00120 LASTNEBINDEX:0001
+0000B0 LASTDS:00000000 00000000 MAXHSPBYTES:00000000 00000000
+0000C0 LASTBLKOFFSET:00000000 00000000 MFCB_CPCB:00000001_00008688

Exiting CRTL I/O Control Blocks
Exiting Language Environment Data

```

Figure 147. Example formatted C/C++ output from LEDATA Verbexit (Part 12 of 12)

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

FFIL This section formats the header of the C/C++ file control block (FCB).

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.

HFSF — UNIX file system file

HSPF — Hiper-Space file

INTC — Intercept file

MEMF — Memory file

OSNS — OS no seek

OSFS — OS fixed text

OSVF — OS variable text

OSUT — OS undefined format text

TDQF — CICS Transient Data Queue file

TERM — Terminal file

VSAM — VSAM file

Other FSCEs will be displayed using a generic overlay.

OSIO The OS IO interface control block.

OSIOE

The OS IO extended interface control block.

DCB The data control block. For more information about the DCB, see *z/OS DFSMS Macro Instructions for Data Sets*.

DCBE The data control block extension. For more information about the DCBE, see *z/OS DFSMS Macro Instructions for Data Sets*.

JFCB The job file control block (JFCB). For more information about the JFCB, see *z/OS MVS Data Areas, Vol 3 (IVT-RCWK)*.

JFCBX

The job file control block extension (JFCBX).

| **MBUF**

| The message buffer control block (MBUF).

| **[8] Memory File Control Blocks**

| This section formats the C/C++ memory file control block (MFCB).

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 Figure 148 on page 441 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: 00000000_7F6F7000
      +000000 ID:ALEC   Ascbl:00FBBE00   Flags1:40000000
      +00000C UseCount:00000001   ASATable@1:7F6E8414
      +000014 ASATable@2:7F6E8754   ASATable@3:00000000
      +00001C ASATable@4:00000000   MCallRtn:82991A80
      +000024 UCallRtn:82999DB8   LatchSetTok:7F6C0B40 00000074
      +000030 ALei:00000001_001053A0   ALes:00000001_00107CD0
      +000040 StackCPID:7F6C3F00   AROTCB:008FF028 EnvTypeNum:00000000
      +00004C WorkECB:808E6F10
      +000050 AROTToken:0000006C 00000003 00000003 008FF028
      +000060 WTPPE:00000053 023B8240 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:008D8E88
      +0000B0 SystemOCB@:00000000_00000000
```

[2] Load Module Control Blocks

Queue #: 0000000000000000

```
ALMI: 00000001_00100F40
+000000 ID:ALMI   ModuleSize:00000600   ModuleName:CELQDSNF
+000010 UseCount:00000000 00000001   LoadPoint:00000000 264E3000
+000020 EntryPoint:00000000 264E3001
```

```
ALMI: 00000001_00100B40
+000000 ID:ALMI   ModuleSize:0000F000   ModuleName:CDAEQED
+000010 UseCount:00000000 00000001   LoadPoint:00000000 26396000
+000020 EntryPoint:00000000 26396001
```

```
ALMI: 00000001_00100540
+000000 ID:ALMI   ModuleSize:000017AD   ModuleName:CEEMENU3
+000010 UseCount:00000000 00000000   LoadPoint:00000000 26386298
+000020 EntryPoint:00000000 26386298
```

Queue #: 0000000000000002

```
ALMI: 00000001_00101340
+000000 ID:ALMI   ModuleSize:00000450   ModuleName:EDCUCSNM
+000010 UseCount:00000000 00000001   LoadPoint:00000000 05F15640
+000020 EntryPoint:00000000 05F15640
```

Queue #: 0000000000000003

```
ALMI: 00000001_00100D40
+000000 ID:ALMI   ModuleSize:00018800   ModuleName:CDAEQDPI
+000010 UseCount:00000000 00000001   LoadPoint:00000000 26414000
+000020 EntryPoint:00000000 26414001
```

```
ALMI: 00000001_00100340
+000000 ID:ALMI   ModuleSize:00000200   ModuleName:ALEM001
+000010 UseCount:00000000 00000003   LoadPoint:00000000 26384000
+000020 EntryPoint:00000000 26384001
```

Queue #: 0000000000000006

```
ALMI: 00000001_00100740
+000000 ID:ALMI   ModuleSize:00000400   ModuleName:CDIVZERO
+000010 UseCount:00000000 00000001   LoadPoint:00000000 26393000
+000020 EntryPoint:00000000 26393001
```

Queue #: 0000000000000007

```
ALMI: 00000001_00101140
+000000 ID:ALMI   ModuleSize:00000655   ModuleName:CEL4CTBL
+000010 UseCount:00000000 00000001   LoadPoint:00000000 264E7D58
+000020 EntryPoint:00000000 264E7D58
```

[3] User Managed Control Blocks

Figure 148. Example of formatted AUTH output from LEDATA Verbexit (Part 1 of 4)

```

[4]  ALEI: 00000001_001053A0
+000000 ID:ALEI      Flags1:80000000   InstanceNum:00000000 00000025
+000010 Next:00000000_00000000         Prev:00000000 00000000
+000020 Flags2:40000000   EnclaveSeq#:00000001   LAA:01ED6498
+00002C SavedLAA:7F710E80   Alec:00000000 7F6F7000
+000038 CallerPSWKey:00000000 00000070   ASASStack:00000000 7F6C4A28
+000048 ParmListPtr:00000000_7F705D58     ParmListPtrK0:00000000_7F6C42F8
+000058 RTOPtr:00000001_0050042C     RTOLen:00000000 00000012
+000068 RTBLE:00000001_001055B8     CallAlri:00000001 00100940
+000078 EnvAlris:00000000_00000000     SystemRtnCode:00000000
+000084 SystemRsnCode:00000000     SystemRtnCodeJr:00000000
+00011C SystemRsnCodeJr:00000000     ALES:00000000_00000000

[5] Routine Control Blocks

Queue #: 0000000000000010

Routine: CDIVZERO
  ALRI: 00000001_00100940
+000000 ID:ALRI      Flags:80000000   InstanceNum:00000000 00000026
+000010 NEXT:00000000_00000000         ALEC:00000000 7F6F7000
+000020 AleiAddress:00000001_001053A0   AleiInstanceNum:00000000_00000025
+000030 DllName:..... RoutineNamePtr:00000001_00100B20
+000040 RoutineNameLen:00000000 00000008   RoutineAddr:00000000 26393178
+000050 QSTRAddr:00000000_26393000   DllKeyPtr:00000000_00000000
+000060 DllKeyLen:00000000 00000000   ParmLen:00000000
+00006C EnvFlags:40000000   FuncEnv:00000000_00000010
+000078 FuncEntry:00000000_26393178   MasterAlri:00000000_00000000
+0000E8 Ales:00000000_00000000   LUAlri:00000000 00000000
+0000F8 EnvType:00000000   EnclaveSeq#:00000001
+000100 NextEnvAlri:00000000_00000000

Queue #: 0000000000000015

Routine: CDIVZERO
  ALRI: 00000001_00100940
+000000 ID:ALRI      Flags:80000000   InstanceNum:00000000 00000026
+000010 NEXT:00000000_00000000         ALEC:00000000 7F6F7000
+000020 AleiAddress:00000001_001053A0   AleiInstanceNum:00000000_00000025
+000030 DllName:..... RoutineNamePtr:00000001_00100B20
+000040 RoutineNameLen:00000000 00000008   RoutineAddr:00000000 26393178
+000050 QSTRAddr:00000000_26393000   DllKeyPtr:00000000_00000000
+000060 DllKeyLen:00000000 00000000   ParmLen:00000000
+00006C EnvFlags:40000000   FuncEnv:00000000_00000010
+000078 FuncEntry:00000000_26393178   MasterAlri:00000000_00000000
+0000E8 Ales:00000000_00000000   LUAlri:00000000 00000000
+0000F8 EnvType:00000000   EnclaveSeq#:00000001
+000100 NextEnvAlri:00000000_00000000

[6] System Managed Control Blocks

[7]  ALES: 00000001_00107CD0
+000000 ID:ALES      UseCount:00000000
+000008 InstanceNum:00000000 00000000   Next:00000000 00000000
+000018 ENVID:RTOTCB5A   Flags1:00000000 00000000   Alec:00000000 7F6F7000
+000030 NumEnvType:00000000 00000003   CPPtr:00000001_001082C8
+000040 AllocSize:00000000 00000B48   SystemRtnCode:00000000
+00004C SystemRsnCode:00000000   SystemRtnCodeJr:00000000
+000054 SystemRsnCodeJr:00000000   DiagRtn:00000000_00000000
+000060 DiagTkn:00000000 00000000

[8] ETINDEX: 00000001

ALESETE: 00000001_00107E18
+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 148. Example of formatted AUTH output from LEDATA Verbexit (Part 2 of 4)

[9] Routine Control Blocks

Queue #: 000000000000017

```
Routine: ALEM001
  ALRI: 00000001_00200140
+000000 ID:ALRI   Flags:88000000   InstanceNum:00000000 00000022
+000010 NEXT:00000001_00200340   ALEC:00000000_7F6F7000
+000020 AleiAddress:00000000_00000000   AleiInstanceNum:00000000_00000000
+000030 DllName:..... RoutineNamePtr:00000001_00200320
+000040 RoutineNameLen:00000000 00000008   RoutineAddr:00000000_263840C0
+000050 QSTRAddr:00000000_26384000   DllKeyPtr:00000000_00000000
+000060 DllKeyLen:00000000 00000000   ParmLen:00000000
+00006C EnvFlags:00000000   FuncEnv:00000000_00000000
+000078 FuncEntry:00000000_00000000   MasterAlri:00000000_00000000
+0000E8 Ales:00000001_00107CD0   LUAlri:00000001 00200340
+0000F8 EnvType:00000001   EnclaveSeq#:00000000
+000100 NextEnvAlri:00000000_00000000
```

```
Routine: ALEM001
  ALRI: 00000001_00200140
+000000 ID:ALRI   Flags:88000000   InstanceNum:00000000 00000022
+000010 NEXT:00000001_00200340   ALEC:00000000_7F6F7000
+000020 AleiAddress:00000000_00000000   AleiInstanceNum:00000000_00000000
+000030 DllName:..... RoutineNamePtr:00000001_00200320
+000040 RoutineNameLen:00000000 00000008   RoutineAddr:00000000_263840C0
+000050 QSTRAddr:00000000_26384000   DllKeyPtr:00000000_00000000
+000060 DllKeyLen:00000000 00000000   ParmLen:00000000
+00006C EnvFlags:00000000   FuncEnv:00000000_00000000
+000078 FuncEntry:00000000_00000000   MasterAlri:00000000_00000000
+0000E8 Ales:00000001_00107CD0   LUAlri:00000001 00200340
+0000F8 EnvType:00000001   EnclaveSeq#:00000000
+000100 NextEnvAlri:00000000_00000000
```

[10]

```
ALEI: 00000001_0010E0D8
+000000 ID:ALEI   Flags1:00000000   InstanceNum:00000000 0000000A
+000010 Next:00000001_0010DECO   Prev:00000000_00000000
+000020 Flags2:00000000   EnclaveSeq#:00000002   LAA:01ED8E18
+00002C SavedLAA:7F710E80   Alec:00000000_7F6F7000
+000038 CallerPSWKey:00000000 00000070   ASASStack:00000000_7F6C4AC0
+000048 ParmListPtr:00000000_7F704AA8   ParmListPtrK0:00000000_7F6C4348
+000058 RTOPtr:00000001_0000042C   RTOLen:00000000 00000401
+000068 RTBLE:00000000_00000000   CallAlri:00000001_00200340
+000078 EnvAlris:00000001_00200340   SystemRtnCode:00000000
+000084 SystemRsnCode:00000000   SystemRtnCodeJr:00000000
+00011C SystemRsnCodeJr:00000000   ALES:00000001_00107CD0
```

[11]

```
Routine: ALEM001
  ALRI: 00000001_00200340
+000000 ID:ALRI   Flags:88000000   InstanceNum:00000000 00000022
+000010 NEXT:00000000_00000000   ALEC:00000000_7F6F7000
+000020 AleiAddress:00000001_0010E0D8   AleiInstanceNum:00000000_00000000
+000030 DllName:..... RoutineNamePtr:00000001_00200320
+000040 RoutineNameLen:00000000 00000008   RoutineAddr:00000000_263840C0
+000050 QSTRAddr:00000000_26384000   DllKeyPtr:00000000_00000000
+000060 DllKeyLen:00000000 00000000   ParmLen:00000010
+00006C EnvFlags:40000000   FuncEnv:00000000_00000010
+000078 FuncEntry:00000000_263840C0   MasterAlri:00000001_00200140
+0000E8 Ales:00000001_00107CD0   LUAlri:00000000 00000000
+0000F8 EnvType:00000001   EnclaveSeq#:00000001
+000100 NextEnvAlri:00000000_00000000
```

[10]

```
ALEI: 00000001_0010DECO
+000000 ID:ALEI   Flags1:00000000   InstanceNum:00000000 00000009
+000010 Next:00000001_0010DCAB   Prev:00000000_00000000
+000020 Flags2:80000000   EnclaveSeq#:00000000   LAA:01ED8C98
+00002C SavedLAA:00000000   Alec:00000000_7F6F7000
+000038 CallerPSWKey:00000000 00000070   ASASStack:00000000_00000000
+000048 ParmListPtr:00000000_00000000   ParmListPtrK0:00000000_00000000
+000058 RTOPtr:00000001_00207CD0   RTOLen:00000000 00000400
+000068 RTBLE:00000000_00000000   CallAlri:00000000_00000000
+000078 EnvAlris:00000000_00000000   SystemRtnCode:00000000
+000084 SystemRsnCode:00000000   SystemRtnCodeJr:00000000
+00011C SystemRsnCodeJr:00000000   ALES:00000001_00107CD0
```

Figure 148. Example of formatted AUTH output from LEDATA Verbexit (Part 3 of 4)

```

[10]  ALEI: 00000001_0010DCA8
+000000 ID:ALEI  Flags1:00000000  InstanceNum:00000000 00000008
+000010 Next:00000001_0010DA90  Prev:00000000 00000000
+000020 Flags2:80000000  EnclaveSeq#:00000000  LAA:01ED8B18
+00002C SavedLAA:00000000  Alec:00000000 7F6F7000
+000038 CallerPSWKey:00000000 00000070  ASASStack:00000000 00000000
+000048 ParmListPtr:00000000_00000000  ParmListPtrK0:00000000_00000000
+000058 RTOPtr:00000001_00207CD0  RTOLen:00000000 00000400
+000068 RTBLE:00000000_00000000  CallAlri:00000000 00000000
+000078 EnvAlris:00000000_00000000  SystemRtnCode:00000000
+000084 SystemRsnCode:00000000  SystemRtnCodeJr:00000000
+00011C SystemRsnCodeJr:00000000  ALES:00000001_00107CD0
:
:
[8]  ETINDEX: 00000002

ALESETE: 00000001_00107FA8
+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

[9] Routine Control Blocks

Queue #: 0000000000000017

Routine: ALEM001
ALRI: 00000001_00200540
+000000 ID:ALRI  Flags:88000000  InstanceNum:00000000 00000023
+000010 NEXT:00000001_00200740  ALEC:00000000 7F6F7000
+000020 AleiAddress:00000000_00000000  AleiInstanceNum:00000000_00000000
+000030 DllName:..... RoutineNamePtr:00000001_00200720
+000040 RoutineNameLen:00000000 00000008  RoutineAddr:00000000_263840C0
+000050 QSTRAddr:00000000_26384000  DllKeyPtr:00000000_00000000
+000060 DllKeyLen:00000000 00000000  ParmLen:00000000
+00006C EnvFlags:00000000  FuncEnv:00000000_00000000
+000078 FuncEntry:00000000_00000000  MasterAlri:00000000_00000000
+0000E8 Ales:00000001_00107CD0  LUAlri:00000001_00200740
+0000F8 EnvType:00000002  EnclaveSeq#:00000000
+000100 NextEnvAlri:00000000_00000000

Routine: ALEM001
ALRI: 00000001_00200540
+000000 ID:ALRI  Flags:88000000  InstanceNum:00000000 00000023
+000010 NEXT:00000001_00200740  ALEC:00000000 7F6F7000
+000020 AleiAddress:00000000_00000000  AleiInstanceNum:00000000_00000000
+000030 DllName:..... RoutineNamePtr:00000001_00200720
+000040 RoutineNameLen:00000000 00000008  RoutineAddr:00000000_263840C0
+000050 QSTRAddr:00000000_26384000  DllKeyPtr:00000000_00000000
+000060 DllKeyLen:00000000 00000000  ParmLen:00000000
+00006C EnvFlags:00000000  FuncEnv:00000000_00000000
+000078 FuncEntry:00000000_00000000  MasterAlri:00000000_00000000
+0000E8 Ales:00000001_00107CD0  LUAlri:00000001_00200740
+0000F8 EnvType:00000002  EnclaveSeq#:00000000
+000100 NextEnvAlri:00000000_00000000

[10]  ALEI: 00000001_001102F0
+000000 ID:ALEI  Flags1:00000000  InstanceNum:00000000 00000015
+000010 Next:00000001_001100D8  Prev:00000000 00000000
+000020 Flags2:00000000  EnclaveSeq#:00000002  LAA:01ED7018
+00002C SavedLAA:7F710E80  Alec:00000000 7F6F7000
+000038 CallerPSWKey:00000000 00000070  ASASStack:00000000 7F6C4AC0
+000048 ParmListPtr:00000000_7F705AA8  ParmListPtrK0:00000000_7F6C4348
+000058 RTOPtr:00000001_0030042C  RTOLen:00000000 00000401
+000068 RTBLE:00000000_00000000  CallAlri:00000001_00200740
+000078 EnvAlris:00000001_00200740  SystemRtnCode:00000000
+000084 SystemRsnCode:00000000  SystemRtnCodeJr:00000000
+00011C SystemRsnCodeJr:00000000  ALES:00000001_00107CD0
:
:
[8]  ETINDEX: 00000003

ALESETE: 00000001_00108138
+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 148. Example of formatted AUTH output from LEDATA Verbexit (Part 4 of 4)

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 28 on page 447. 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 149 :
CBF 15890 struct(CELCAA)

```

CEECAA: 00000001_00007B18
+000288 DLLF:00000000_00000000      INVAR:8000  FLAG0:00
+000304 TORC:00000000      FLAG2:30  LEVEL:15   PM:04
+000368 DMC:00000000_00000000      CD:00000000      RS:00000000
+000378 ERR:00000001_082FBE00      DDSA:00000001_082FF760
+000388 EDB:00000001_00005340      PCB:00000001_00003CA0  EYEPTR:00000001_00007B00
+0003A0 CAA:00000001_00007B18      SHAB:00000000_00000000
+0003B0 PRGCK:00000000_00000000      URC:00000000      PICICB:00000000_00000000
+0003C8 SIGSCTR:00000000      SIGSFLG:08000000      THDID:253E0190_00000000
+0003D8 RCB:00000001_00003A10      MEMBR:00000001_000084D0
+0003E8 SIGNAL_STATUS:00000000_00000008      HCOM_REG14:00000000_00000000
+0003F8 EDCHPXV:00000000_25546C78      THREADHEAPID:00000000_00000000
+000408 SYS_RTNCODE:00000000      SYS_RSNCODE:00000000      SIGNGPTR:00000001_00007F30
+000418 SIGNG:00000001      AB_STATUS:00      STACKDIRECTION:00
+000420 AB_GR0:00000000_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      DE VH:00000000_00000000
+0004F8 PtatPtr:00000000_00000000      SQELADDR:00000000_257520A0
+000510 VBA:00000001_08913350      TCS:00000001_08FEC450
+000520 CONDWAITDSAREG:00000000_00000000      THDSTATUS:00000000_00000000
+000570 FBTK:00000000_00000000_00000000_00000000      PTXLPTR:00000000_00000000
+000588 TICB_PTR:00000001_00006AB0      FWD_CHAIN:00000001_114013C8
+0005A0 BKWD_CHAIN:00000001_199013C8      TCB:007FF050
+0007EC DIA:25752320      DLLFFLAG:00      MCBPTR:00000000_25773DA8
+000838 MAD:00000000_25773048      MFD:00000000_25752998

```

Figure 149. The CAA formatted by the CBFORMAT IPCS command

For more information on using the IPCS CBF command, see the "CBFORMAT subcommand" section in *z/OS MVS IPCS Commands, SA22-7594*.

Table 28. Language Environment control blocks which can be individually formatted

Control Block	Description
CELCIB	Condition Information Block
CELCIBH	Condition Information Block Header
CELDLLF	DLL Failure Control Block
CELD SA	Dynamic Storage Area
CELD SATR	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

Table 28. Language Environment control blocks which can be individually formatted (continued)

Control Block	Description
CELSANC	Storage Management Control Block
CELSTSB	Storage Report Statistics Block

Table 29. 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 (`/`), the directory is writable, and the CEEDUMP path name does not exceed 1024 characters
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, see 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 150. 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 30. `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 483.

When `LE=2` is specified: The following Language Environment records may be generated.

Table 31. `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

Table 31. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
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
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

Table 31. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
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	EUI	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)

Table 31. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
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
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

Table 31. LE=2 entry records (continued)

Member ID	Record Type	Class	Event	Description
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)
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 32. 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 31 on page 451.

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 33. 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 483.

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 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	=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 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 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 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	=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 40404040	main
+000338	60606E4D F0F1F6C4	5D408493 9398A485	99A88695 4D5D4040	40404040 40404040	-->(016D) d11queryfn()
+000358	40404040 40404040	40404040 40404040	40404040 40404040	40404040 40404040	
+000378	40404040 40404040				

Figure 151. Trace table in dump output

Chapter 13. 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, see *z/OS XL C/C++ Programming Guide* to ensure that you are creating the fetchable module correctly.
- If you are using DLLs, see *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++ programs

You can use C/C++ conventions such as `__amrc` and `perror()` when you debug C/C++ programs.

Using the `__amrc` and `__amrc2` structures to debug input/output

`__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 152 on page 461 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;
        #if __EDC_TARGET >= 0x22080000
[9]     unsigned char      __rp1fdbwd[4]; /* rp1 feedback word */
        #endif /* __EDC_TARGET >= 0x22080000 */
        #if __EDC_TARGET >= 0x41080000
[10]    #ifdef __LP64
            unsigned long   __XRBA; /* 8 byte RBA      */
        #elif defined(__LL)
            unsigned long long __XRBA; /* 8 byte RBA      */
        #else
            unsigned int     __XRBA1; /* high half of 8 byte RBA */
            unsigned int     __XRBA2; /* low half of 8 byte RBA  */
        #endif
[11]    unsigned char      __amrc_noseek_to_seek;
        /* padding to make amrc 256 bytes */
        char             __amrc_pad[23];
    } __amrc_type;

```

Figure 152. `__amrc` structure

Figure 153 shows the `__amrc2` structure as it appears in `stdio.h`.

```

    struct {
[12]     int      __error2;
        char     __pad__error2[4];
[13]     FILE    *__fileptr;
[14]     int      __reserved{6};
    }

```

Figure 153. `__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 34 on page 463.
- [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] **__rplfdbwd** This field contains feedback information related to a VSAM RLS failure. This is the feedback code from the IFGRPL control block.
- [10] **__XRBA** This is the 8 byte relative byte address 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 may be used in subsequent calls to `flocate()`.
- [11] **__amrc_noseek_to_seek**
This field contains the reason for the switch from QSAM (noseek) to BSAM with `NOTE` and `POINT` macros requested (seek) by the XL C/C++ Run-Time Library. This field is set when system-level I/O macro processing triggers an ABEND condition. The macro name values (defined in `stdio.h`) for this field are as follows:

Macro	Definition
<code>__AM_BSAM_NOSWITCH</code>	No switch was made.
<code>__AM_BSAM_UPDATE</code>	The data set is open for update

Macro	Definition
__AM_BSAM_BSAMWRITE	The data set is already open for write (or update) in the same C process.
__AM_BSAM_FBS_APPEND	The data set is recfm=FBS and open for append
__AM_BSAM_LRECLX	The data set is recfm=LRECLX (used for VBS data sets where records span the largest blocksize allowed on the device)
__AM_BSAM_PARTITIONED_DIRECTORY	The data set is the directory for a regular or extended partitioned data set
__AM_BSAM_PARTITIONED_INDIRECT	The data set is a member of a partitioned data set, and the member name was not specified at allocation

[12] **__error2** A secondary error code. For example, an unsuccessful rename or remove operation places its reason code here.

[13] **__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.

[14] **__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 34 lists `__last_op` values you could receive and where to look for further information.

Table 34. `__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 <code>__error</code> with return code from OS OPEN macro.
__BSAM_CLOSE	Sets <code>__error</code> with return code from OS CLOSE macro.
__BSAM_READ	No return code (either <code>__abend</code> (<code>errno == 92</code>) or <code>__msg</code> (<code>errno == 66</code>) filled in).
__BSAM_NOTE	NOTE returned 0 unexpectedly, no return code.
__BSAM_POINT	This will not appear as an error lastop.
__BSAM_WRITE	No return code (either <code>__abend</code> (<code>errno == 92</code>) or <code>__msg</code> (<code>errno == 65</code>) filled in).
__BSAM_CLOSE_T	Sets <code>__error</code> with return code from OS CLOSE TYPE=T.
__BSAM_BLDL	Sets <code>__error</code> with return code from OS BLDL macro.
__BSAM_STOW	Sets <code>__error</code> with return code from OS STOW macro.
__TGET_READ	Sets <code>__error</code> with return code from TSO TGET macro.
__TPUT_WRITE	Sets <code>__error</code> with return code from TSO TPUT macro.
__IO_DEVTYPE	Sets <code>__error</code> with return code from I/O DEVTYPE macro.
__IO_RDJFCB	Sets <code>__error</code> with return code from I/O RDJFCB macro.
__IO_TRKCALC	Sets <code>__error</code> with return code from I/O TRKCALC macro.

Table 34. `__last_op` values and diagnosis information (continued)

Value	Further Information
<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.
<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 <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.

Table 34. `__last_op` values and diagnosis information (continued)

Value	Further Information
<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_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.

Table 34. `__last_op` values and diagnosis information (continued)

Value	Further Information
<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> .
<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`). By default, the `errno2` value will be appended to the end of the `perror()` string.

If you do not want the `errno2` value appended to the `perror()` string, set the `__EDC_ADD_ERRNO2` environment variable to 0.

Figure 154 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 154. Example of a routine using `perror()`

Using `__errno2()` to diagnose application problems

Use the `__errno2()` function when diagnosing problems in an application program. This function enables z/OS XL C/C++ application programs to access additional diagnostic information, `errno2` (`errnojr`), associated with `errno`. The `__errno2` may be set by the z/OS XL C/C++ run-time library, z/OS UNIX callable services, or other callable services. The `errno2` is intended for diagnostic display purposes only and is not a programming interface.

Note: Not all functions set `errno2` when `errno` is set. In the cases where `errno2` is not set, the `__errno2()` function may return a residual value. You may use the `__err2ad()` function to clear `errno2` to reduce the possibility of a residual value being returned.

Figure 155 is an example of a routine using `__errno2()` and Figure 156 on page 468 shows the sample output from that routine.

```
#pragma runopts(posix(on))
#define _EXT
#include <stdio.h>
#include <errno.h>

int main(void) {
    FILE *f;
    f = fopen("testfile.dat", "r")
    if (f==NULL) {
        perror("fopen() failed");
        printf("__errno2 = %08x\n", __errno2());
    }
    return 0;
}
```

Figure 155. Example of a routine using `__errno2()`

```
fopen() failed: EDC5129I No such file or directory. (errno2=0x05620062)
__errno2 = 05620062
```

Figure 156. Sample output of routine using __errno2()

Figure 157 is an example of a routine using the environment variable `_EDC_ADD_ERRNO2` and Figure 158 shows the sample output from that routine.

```
#pragma runopts(posix(on))
#define _EXT
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int main(void) {
    FILE *fp;
    /* do NOT add errno2 to perror message */
    setenv("_EDC_ADD_ERRNO2", "0", 1);
    fp = fopen("testfile.dat", "r");
    if (fp == NULL)
        perror("fopen() failed");
    return 0;
}
```

Figure 157. Example of a routine using _EDC_ADD_ERRNO2

```
fopen() failed: EDC5129I No such file or directory.
```

Figure 158. Sample output of a routine using _EDC_ADD_ERRNO2

Figure 159 on page 469 is an example of a routine using `__err2ad()` in combination with `__errno2()` and Figure 160 on page 469 shows the sample output from that routine.

```

#pragma runopts(posix(on))
#define _EXT
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int main(void) {
    FILE *f;
    setenv("_EDC_ADD_ERRNO2", "0", 1);
    f = fopen("testfile.dat", "r");
    if (f == NULL) {
        perror("fopen() failed");
        printf("__errno2 = %08x\n", __errno2());
    }
    /* reset errno2 to zero */
    *__err2ad() = 0x0;
    printf("__errno2 = %08x\n", __errno2());
    f = fopen(*testfile.dat, "r");

    if (fp == NULL) {
        perror("fopen() failed");
        printf(*__errno2 = %08x\n", __errno2());
    }
    return 0;
}

```

Figure 159. Example of a routine using `__err2ad()` in combination with `__errno2()`

```

fopen() failed: EDC5129I No such file or directory.
__errno2 = 05620062
__errno2 = 00000000
fopen() failed: EDC5129I No such file or directory.
__errno2 = 05620062

```

Figure 160. Sample output of routine using `__err2ad()` in combination with `__errno2()`

For more information about `_EDC_ADD_ERRNO2`, see *z/OS XL C/C++ Programming Guide*.

For more information about `__errno2()` and `__err2ad()`, see *z/OS XL C/C++ Run-Time Library Reference*.

Using C/C++ listings

For a detailed description of available listings, see *z/OS XL C/C++ User's Guide*.

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 58 on page 206.

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:

```
pp          5828-0:15      Class = parameter,      Location = 2432(r4),      Length = 8
```

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
00000003	funcb	+0000002E	CDIVZERO			Exception

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Attributes
00000003	00000001082FF080	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 161 on page 472 shows an example of an aggregate.

```

typedef struct {
    int asid;
    void *addr;
    asfAmodeType amode;
} asfTargetRef;
asfTargetRef tempTargetRef;

```

Figure 161. Example code for structure variables

Figure 162 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 162. 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 162. 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.

See 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 163 on page 474 shows a sample C routine that uses the `cdump` function to generate a dump.

Figure 168 on page 478 shows the dump output.

```

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

void hsigfpe(int);
void hsigterm(int);
void atf1(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(atf1);
    if (rc) {
        fprintf(stderr, "Failed on registration of atexit function atf1\n");
        exit(103);
    }
}

```

Figure 163. 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 163. Example C routine using `cdump()` to generate a dump (Part 2 of 2)

Figure 164 shows a fetched C module:

```

#include <ctest.h>

#pragma linkage(func1, fetchable)
int func1(void) {
    __cdump("This is a sample dump");
    return(0);
}

```

Figure 164. Fetched module for C routine

Sample C++ routine that generates a Language Environment dump

Figure 165 on page 476 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 165. Example C++ routine with protection exception generating a dump

Figure 166 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 166. Template file STACK.C

Figure 167 on page 477 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 167. Header file STACK.H

Sample Language Environment dump with C/C++-specific information

This sample dump was produced by compiling the routines shown in Figure 163 on page 474 and Figure 164 on page 475. They were both compiled using options LP64 and GONUM to produce statement numbers in the CEEDUMP. Notice the sequence of calls in the traceback section - CELQINIT is the Language Environment module that invokes the main entry. main calls fetchPtr() at statement number 60, which in turn, through @@FECBMODULE1 fetches the user-defined function func1 shown in Figure 164 on page 475. func1 calls the library routine __cdump() in statement number 5. The complete program unit names for main and func1 are shown in the Fully Qualified Names section along with its load module name.

Information for enclave main

Information for thread 25AC52800000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
00000001	_cdump	+00000000		CELQLIB		HLE7730	Call
00000002	func1	+00000020	5	MODULE1	func1.c		Call
00000003	@@FECBMODULE1						
		-005F601C			** NoName **		Call
00000004	main	+00000284	60	CSAMPLE	FIG142		Call
00000005	CELQINIT	+0000134C		CELQLIB	CELQINIT	HLE7730	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Attributes
00000001	0000001082FEDC0	000000025C1FFE0	0000000000000000	*****	20060111	XPLINK EBCDIC POSIX IEEE
00000002	0000001082FEF00	000000025D8A0D0	0000000000000000	*****	20060222	XPLINK EBCDIC POSIX IEEE
00000003	0000001082FF000	000000025D8D058	000000025D8D048	005F600C		XPLINK EBCDIC POSIX Floating Point
00000004	0000001082FF180	0000000257000D0	0000000000000000	*****	20060222	XPLINK EBCDIC POSIX IEEE
00000005	0000001082FF280	000000025703010	000000025703010	0000134C	20060111	XPLINK EBCDIC POSIX Floating Point

Fully Qualified Names

DSA	Entry	Program Unit	Load Module
00000002	func1	AQMVSQE:/u/alfcar/tools/func1.c	MODULE1
00000004	main	PLPSC:/'POSIX.CRTL.C(FIG142)'	CSAMPLE

Control Blocks for Active Routines:

DSA for _cdump: 0000001082FF5C0					
+000000	R4.....	0000001082FEF00	R5.....	000000025D77E10	R6..... 000000025C1FFE0
+000018	R7.....	000000025D8A0F2	R8.....	000000025796F40	R9..... 000000025D8A110
+000030	R10.....	0000001082FF880	R11.....	0000000000000000	R12..... 000000100007AC0
+000048	R13.....	0000001082FF180	R14.....	00000010000B238	R15..... 000000025D8D048
+000060	reserved.	000000025AA5FD6	reserved.	0000000000000000	HPTRAN... 000000025D71580
+000078	reserved.	000000100007AC0			
DSA for func1: 0000001082FF700					
+000000	R4.....	0000001082FF000	R5.....	0000001083012E0	R6..... 000000025D8A0D0
+000018	R7.....	00000002579703E	R8.....	000000025796F40	R9..... 0000001082FF800
+000030	R10.....	0000001082FF880	R11.....	0000000000000000	R12..... 000000100007AC0
+000048	R13.....	0000001082FF180	R14.....	00000010000B238	R15..... 000000025D8D048
+000060	reserved.	0000000180F58FF	reserved.	001007FF00000000	HPTRAN... 0000001082FF920
+000078	reserved.	0000000000000000			
DSA for @@FECBMODULE1 : 0000001082FF800					
+000000	R4.....	0000001082FF180	R5.....	C0F0FFFE7AC07FF	R6..... 000000025D8D058
+000018	R7.....	000000025700356	R8.....	0000000000006FE8	R9..... 000000025700410
+000030	R10.....	000000025700620	R11.....	000000108415310	R12..... 000000100005300
+000048	R13.....	0000000000006F50	R14.....	000000025753360	R15..... 00000000000001F
+000060	reserved.	0000000700006FE8	reserved.	000000025700410	HPTRAN... 000000100002000
+000078	reserved.	000000108415310			
DSA for main: 0000001082FF980					
+000000	R4.....	0000001082FF280	R5.....	000000108300090	R6..... 0000000257000D0
+000018	R7.....	00000002570435E	R8.....	0000000000006FE8	R9..... 000000025701548
+000030	R10.....	000000025700620	R11.....	000000108415310	R12..... 000000100005300
+000048	R13.....	0000000000006F50	R14.....	000000025753360	R15..... 00000000000001F
+000060	reserved.	0000000000000000	reserved.	0000000000000000	HPTRAN... 000000000000000
+000078	reserved.	0000000000000000			
DSA for CELQINIT: 0000001082FFA80					
+000000	R4.....	0000001082FF760	R5.....	0000000000000000	R6..... 000000025703010
+000018	R7.....	0000000257044F8	R8.....	0000000000000000	R9..... 000000000000000
+000030	R10.....	0000000000000000	R11.....	0000000000000000	R12..... 000000000000000
+000048	R13.....	0000000000000000	R14.....	0000000000000000	R15..... 000000000000000
+000060	reserved.	0000000000000000	reserved.	0000000000000000	HPTRAN... 000000000000000
+000078	reserved.	0000000000000000			

Storage for Active Routines:

DSA frame(0000001082FEDC0)					
+0800	0000001082FF5C0	00000001	082FEF00	00000000	25D77E10 P=.
+0810	0000001082FF5D0	00000000	25C1FFE0	00000000	25D8A0F2 A.....Q.2
+0820	0000001082FF5E0	00000000	25796F40	00000000	25D8A110 ?Q..
+0830	0000001082FF5F0	00000001	082FF880	00000000	00000000 8.....
+0840	0000001082FF600	00000001	00007AC0	00000001	082FF180 :.....1.
+0850	0000001082FF610	00000001	0000B238	00000000	25D8D048 :.....Q..
+0860	0000001082FF620	00000000	25AA5FD6	00000000	00000000 ^0.....
+0870	0000001082FF630	00000000	25D71580	00000001	00007AC0 P.....:
+0880	0000001082FF640	00000001	082FF6A0	00000000	25C20719 6.....B..
+0890	0000001082FF650	00000001	082FF690	00000000	00000000 6.....

Figure 168. Example dump from sample C routine (Part 1 of 4)

```

+08A0 00000001082FF660 00000000 00000000 00000000 00000000 |.....|
+08B0 00000001082FF670 00000000 00000000 00000001 082FF75C |.....7*|
+08C0 00000001082FF680 00000000 25D8D0E0 00000000 25D8D0C8 |....Q.....Q.H|
+08D0 00000001082FF690 00010C7B 49C3C5C5 00000000 00000000 |...#.CEE.....|
+08E0 00000001082FF6A0 E38889A2 4089A240 8140A281 94979385 |This is a sample|
+08F0 00000001082FF6B0 4084A494 97404040 40404040 40404040 |dump|
+0900 00000001082FF6C0 40404040 40404040 40404040 40404040 |same as above|
+0910 00000001082FF6D0 - +00092F 00000001082FF6EF |.....|
+0930 00000001082FF6F0 00000001 083012D0 00000000 00000020 |.....|

DSA frame(00000001082FEF00)
+0800 00000001082FF700 00000001 082FF000 00000001 083012E0 |.....0.....|
+0810 00000001082FF710 00000000 25D8A0D0 00000000 2579703E |.....Q.....|
+0820 00000001082FF720 00000000 25796F40 00000001 082FF800 |.....?.....8.|
+0830 00000001082FF730 00000001 082FF880 00000000 00000000 |.....8.....|
+0840 00000001082FF740 00000001 00007AC0 00000001 082FF180 |.....:.....1.|
+0850 00000001082FF750 00000001 0000B238 00000000 25D8D048 |.....:.....Q..|
+0860 00000001082FF760 00000000 180F58FF 001007FF 00000000 |.....:.....|
+0870 00000001082FF770 00000001 082FF920 00000000 00000000 |.....9.....|
+0880 00000001082FF780 00000000 25D8A110 00000001 00007AC0 |.....Q.....:..|
+0890 00000001082FF790 00000001 00007AC0 00000001 082FF85C |.....:.....8*|
+08A0 00000001082FF7A0 00000001 082FF8E0 00000001 082FF8E8 |.....8.....8Y|
+08B0 00000001082FF7B0 00000001 082FF8F0 00000000 25D67E10 |.....80.....0=|
+08C0 00000001082FF7C0 00000001 08300070 00000001 0000A5E8 |.....:.....vY|
+08D0 00000001082FF7D0 00000000 25753360 00000000 0000001F |.....-.....|
+08E0 00000001082FF7E0 40404040 40404040 00000000 00000001 |.....:.....|
+08F0 00000001082FF7F0 00000000 00000000 00000001 00000000 |.....:.....|

DSA frame(00000001082FF000)
+0800 00000001082FF800 00000001 082FF180 C0F0FFF E7AC07FF |.....1..0..X...|
+0810 00000001082FF810 00000000 25D8D058 00000000 25700356 |.....Q.....|
+0820 00000001082FF820 00000000 00006FE8 00000000 25700410 |.....?Y.....|
+0830 00000001082FF830 00000000 25700620 00000001 08415310 |.....:.....|
+0840 00000001082FF840 00000001 00005300 00000000 00006F50 |.....:.....?&|
+0850 00000001082FF850 00000000 25753360 00000000 0000001F |.....-.....|
+0860 00000001082FF860 00000007 00006FE8 00000000 25700410 |.....:.....?Y.....|
+0870 00000001082FF870 00000001 00002000 00000001 08415310 |.....:.....|
+0880 00000001082FF880 00000001 082FF180 00000000 25D67E10 |.....1.....0=|
+0890 00000001082FF890 00000000 25940F18 00000000 00000019 |.....m.....|
+08A0 00000001082FF8A0 00000000 00006FE8 00000000 25700410 |.....:.....?Y.....|
+08B0 00000001082FF8B0 00000000 25700620 00000000 25D8A000 |.....:.....Q..|
+08C0 00000001082FF8C0 00000001 08415378 00000000 25D8A000 |.....:.....Q..|
+08D0 00000001082FF8D0 00000000 25D8A000 00000001 082FF89C |.....Q.....8..|
+08E0 00000001082FF8E0 00000001 083012D0 00000000 25D8D048 |.....:.....Q..|
+08F0 00000001082FF8F0 00000000 00000000 00000000 00000000 |.....:.....|
+0900 00000001082FF900 00000000 257003F0 00000000 00000000 |.....0.....|
+0910 00000001082FF910 00000000 00000000 58F0C2B8 58F0FFA4 |.....:.....0B..0.u|
+0920 00000001082FF920 00000001 08300080 00000000 00000000 |.....:.....|
+0930 00000001082FF930 D4D6C4E4 D3C5F140 00000000 00002000 |MODULE1|
+0940 00000001082FF940 00000000 00000000 00000000 00000000 |.....:.....|
+0950 00000001082FF950 - +00095F 00000001082FF95F |same as above|
+0960 00000001082FF960 00000000 00000008 00000000 00000000 |.....:.....|
+0970 00000001082FF970 00000000 00000000 00000001 00000000 |.....:.....|

DSA frame(00000001082FF180)
+0800 00000001082FF980 00000001 082FF280 00000001 08300090 |.....2.....|
+0810 00000001082FF990 00000000 257000D0 00000000 2570435E |.....:.....;|
+0820 00000001082FF9A0 00000000 00006FE8 00000000 25701548 |.....:.....?Y.....|
+0830 00000001082FF9B0 00000000 25700620 00000001 08415310 |.....:.....|
+0840 00000001082FF9C0 00000001 00005300 00000000 00006F50 |.....:.....?&|
+0850 00000001082FF9D0 00000000 25753360 00000000 0000001F |.....-.....|
+0860 00000001082FF9E0 00000000 00000000 00000000 00000000 |.....:.....|
+0870 00000001082FF9F0 - +00087F 00000001082FF9FF |same as above|
+0880 00000001082FFA00 00000000 25D8D170 00000000 00000020 |.....QJ.....|
+0890 00000001082FFA10 00000000 25D8D178 00000000 00000000 |.....:.....QJ.....|
+08A0 00000001082FFA20 00000000 00000000 00000000 00000000 |.....:.....|
+08B0 00000001082FFA30 - +0008BF 00000001082FFA3F |same as above|
+08C0 00000001082FFA40 00000000 25D8D178 00000000 25D873D8 |.....QJ.....Q.Q|
+08D0 00000001082FFA50 00000000 25D88750 00000000 00000000 |.....:.....Qg&.....|
+08E0 00000001082FFA60 00000000 00000000 00000000 00000000 |.....:.....|
+08F0 00000001082FFA70 - +0008FF 00000001082FFA7F |same as above|

```

Figure 168. Example dump from sample C routine (Part 2 of 4)

```

DSA frame(0000001082FF280)
+0800 00000001082FFA80 00000001 082FF760 00000000 00000000 |.....7-.....|
+0810 00000001082FFA90 00000000 25703010 00000000 257044F8 |.....8|
+0820 00000001082FFAA0 00000000 00000000 00000000 00000000 |.....|
+0830 00000001082FFAB0 - +00087F 00000001082FFAFF |same as above|
+0880 00000001082FFB00 00000001 082FFD08 00000001 082FF760 |.....7-|
+0890 00000001082FFB10 00000001 00007AC0 00000000 00000000 |.....:.....|
+08A0 00000001082FFB20 00000000 00000000 00000000 00000000 |.....|
+08B0 00000001082FFB30 - +0009FF 00000001082FFC7F |same as above|
+0A00 00000001082FFC80 00000000 00000000 00000001 082FFD08 |.....|
+0A10 00000001082FFC90 00000000 00000000 00000000 00000000 |.....|
+0A20 00000001082FFCA0 - +000A7F 00000001082FFCFF |same as above|
+0A80 00000001082FFD00 00000000 00000000 00000001 00000000 |.....|
+0A90 00000001082FFD10 00000000 00000000 00000000 00000000 |.....|
+0AA0 00000001082FFD20 - +000ADF 00000001082FFD5F |same as above|
+0AE0 00000001082FFD60 00000001 00003C60 00000001 00005300 |.....-.....|
+0AF0 00000001082FFD70 00000001 00007AC0 00000000 00000000 |.....|
+0B00 00000001082FFD80 00000000 00000000 00000000 00000000 |.....|
+0B10 00000001082FFD90 - +000B4F 00000001082FFDCF |same as above|
+0B50 00000001082FFDD0 00000000 00000000 00000001 08300090 |.....|
+0B60 00000001082FFDE0 00000000 00000000 00000000 00000000 |.....|
+0B70 00000001082FFDF0 - +000BAF 00000001082FFE2F |same as above|
+0BB0 00000001082FFE30 00000000 00000000 00000001 08300090 |.....|
+0BC0 00000001082FFE40 00000000 00000001 00000001 08415320 |.....|
+0BD0 00000001082FFE50 00000001 08413430 00000001 082FF280 |.....2.|
+0BE0 00000001082FFE60 00000000 25704444 00000000 257000D0 |.....|
+0BF0 00000001082FFE70 00000000 00000098 00000000 00000000 |.....q.....|
+0C00 00000001082FFE80 00000000 00000000 00000000 00000000 |.....|
+0C10 00000001082FFE90 - +000CDF 00000001082FFF5F |same as above|

Control Blocks Associated with the Thread:
CAA(0000000100007AC0)
+0000 0000000100007AC0 00000000 00000000 00000000 00000000 |.....|
+0010 0000000100007AD0 - +0002AF 0000000100007D6F |same as above|
+02B0 0000000100007D70 00000000 00000000 00000000 00000000 |.....|
+02C0 0000000100007D80 00000000 00000000 00000000 00000000 |.....|
+02D0 0000000100007D90 - +00030F 0000000100007DCF |same as above|
+0310 0000000100007DD0 00000001 0000B918 00000000 00000000 |.....|
+0320 0000000100007DE0 00000001 00005558 00000000 00000000 |.....|
+0330 0000000100007DF0 00000001 00008E58 00000000 25D8D048 |.....Q..|
+0340 0000000100007E00 00000001 00008630 00000001 0000A498 |.....f.....uq|
+0350 0000000100007E10 00000000 00000000 00000001 00008828 |.....h..|
+0360 0000000100007E20 03030210 15040000 00000000 257D2FC6 |.....'F|
+0370 0000000100007E30 00000000 00000000 00000001 00007668 |.....|
+0380 0000000100007E40 00000001 082FF760 00000001 00005300 |.....7-.....|
+0390 0000000100007E50 00000001 00003C60 00000001 00007AA8 |.....-.....:y|
+03A0 0000000100007E60 00000001 00007AC0 00000000 00000000 |.....|
+03B0 0000000100007E70 00000000 00000004 00000000 00000000 |.....|
+03C0 0000000100007E80 00000000 00000000 00000000 00000000 |.....|
+03D0 0000000100007E90 25AC5280 00000000 00000001 000039D0 |.....|
+03E0 0000000100007EA0 00000001 00008478 00000000 00000000 |.....d.....|
+03F0 0000000100007EB0 00000000 00000000 00000000 25770FE0 |.....|
+0400 0000000100007EC0 00000000 00000000 00000000 00000000 |.....|
+0410 0000000100007ED0 00000001 00007ED8 00000001 00000000 |.....=Q.....|
+0420 0000000100007EE0 00000000 00000000 00000000 00000000 |.....|
+0430 0000000100007EF0 - +0004BF 0000000100007F7F |same as above|

MATH PARAMETERS:(0000000025D82AD0)
+0000 0000000025D82AD0 00000000 00000000 00000000 00000000 |.....|
+0010 0000000025D82AE0 - +00001F 0000000025D82AEF |same as above|

Thread Synchronization Queue Element (SQL)(0000000025D820A0)
+0000 0000000025D820A0 00000000 00000000 00000000 00000000 |.....|
+0010 0000000025D820B0 - +00002F 0000000025D820CF |same as above|
+0030 0000000025D820D0 00000000 00000000 00000001 00007AC0 |.....|
+0040 0000000025D820E0 00000000 00000000 00000000 00000000 |.....|
+0050 0000000025D820F0 - +00006F 0000000025D8210F |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

CEE3DMP called by program unit (entry point __cdump) at offset +00000000.

Registers on Entry to CEE3DMP:
PM..... 0100
GPR0.... ***** GPR1.... ***** GPR2.... ***** GPR3.... *****
GPR4.... 00000001082FEDC0 GPR5.... 00000000257D07CC GPR6.... 00000000257CCB80 GPR7.... 0000000025C20124
GPR8.... 0000000025D77E10 GPR9.... 00000001082FF6A0 GPR10.... 0000000025D8A110 GPR11.... 0000000000000001
GPR12.... 0000000100007AC0 GPR13.... 00000001082FF180 GPR14.... 000000010000B238 GPR15.... 0000000025D8D048
GPREG STORAGE:
Storage around GPR0 is invalid.

```

Figure 168. Example dump from sample C routine (Part 3 of 4)


```

Storage around GPR1 is invalid.
Storage around GPR2 is invalid.
Storage around GPR3 is invalid.

Storage around GPR4 (0000001082FEDC0)
+0800 0000001082FF5C0 00000001 082FEF00 00000000 25D77E10 |.....P=|
+0810 0000001082FF5D0 00000000 25C1FFE0 00000000 25D8A0F2 |....A.....Q.2|
+0820 0000001082FF5E0 00000000 25796F40 00000000 25D8A110 |.....?.....Q..|
+0830 0000001082FF5F0 00000001 082FF880 00000000 00000000 |.....8.....|
+0840 0000001082FF600 00000001 00007AC0 00000001 082FF180 |.....:.....1..|
+0850 0000001082FF610 00000001 0000B238 00000000 25D8D048 |.....:.....Q..|
Storage around GPR5 (0000000257D07CC)
-0020 0000000257D07AC 00000000 00000000 00000000 00000000 |.....|
-0010 0000000257D07BC - +FFFFFF 0000000257D07CB |.....|
+0000 0000000257D07CC 257CCB80 00000F38 00000F38 00000000 |.e.....|
+0010 0000000257D07DC 257D0880 00000000 257D0874 00000000 |.'.....'|
+0020 0000000257D07EC 257D0850 00000000 257D0850 00000000 |.'&.....'&....|
+0030 0000000257D07FC - +00003F 0000000257D080B |.....|
same as above
Storage around GPR6 (0000000257CCB80)
-0020 0000000257CCB60 F0F9F0F0 0005C4F1 F9F0F200 00000000 |0900..D1902....|
-0010 0000000257CCB70 00C300C5 00C500F1 00003DE8 00000740 |.C.E.E.1...Y...|
+0000 0000000257CCB80 EB134880 0024B904 0004A74B F8C0EB5F |.....x.8.^|
+0010 0000000257CCB90 48080024 E3040800 00244190 4FFFEB13 |....T.....|...|
+0020 0000000257CCBA0 49800024 E38004B8 0017E380 80580004 |....T.....T....|
+0030 0000000257CCBB0 E3C08008 0004A788 00504080 4C6C1F88 |T.....xh.& .%h|
Storage around GPR7 (000000025C20124)
-0020 000000025C20104 C0600000 024E4470 6060C070 00000249 |.-...+...-.....|
-0010 000000025C20114 EB568000 00044130 48D04120 71790D76 |.....|
+0000 000000025C20124 07004800 48D0B914 0000A70E 0003A784 |.....x...x..xd|
+0010 000000025C20134 FF75A70E 0005A784 FF7112BB A774FF6E |..x...xd...x..>|
+0020 000000025C20144 A7390000 EB4B4800 000447F0 70020000 |x.....0.....|
+0030 000000025C20154 00000000 00C300C5 00C500F1 00000838 |.....C.E.E.1...|
Storage around GPR8 (000000025D77E10)
-0020 000000025D77DF0 00000000 25D78EB0 00000000 25C2ED28 |.....P.....B..|
-0010 000000025D77E00 00000000 25D76E20 00000000 25BBADC8 |.....P>.....H|
+0000 000000025D77E10 00000000 257D07CC 00000000 257CCB80 |.....'.....e..|
+0010 000000025D77E20 00000000 25BB7E98 00000000 25BB7E98 |.....=q.....=q|
+0020 000000025D77E30 00000000 257D69BC 00000000 257D6560 |.....'.....'|
+0030 000000025D77E40 0001018F 49C3C5C5 00000000 00000000 |.....CEE.....|
Storage around GPR9 (0000001082FF6A0)
-0020 0000001082FF680 00000000 25D8D0E0 00000000 25D8D0C8 |.....Q.....Q.H|
-0010 0000001082FF690 00010C7B 49C3C5C5 00000000 00000000 |.....#.CEE.....|
+0000 0000001082FF6A0 E38889A2 4089A240 8140A281 94979385 |This is a sample|
+0010 0000001082FF6B0 4084A494 97404040 40404040 40404040 |dump|
+0020 0000001082FF6C0 40404040 40404040 40404040 40404040 |.....|
+0030 0000001082FF6D0 - +00003F 0000001082FF6DF |.....|
same as above
Storage around GPR10(000000025D8A110)
-0020 000000025D8A0F0 0D760700 A7390000 E3704818 0004EB89 |....x...T.....i|
-0010 000000025D8A100 48200004 41404100 47F07002 00000000 |.....0.....|
+0000 000000025D8A110 E38889A2 4089A240 8140A281 94979385 |This is a sample|
+0010 000000025D8A120 4084A494 97000000 02CE07C0 00000068 |dump.....|
+0020 000000025D8A130 80800281 00000503 0000003C 01000000 |.....a.....|
+0030 000000025D8A140 000586A4 9583F140 FFFFFFF98 00000000 |..funcl ..q....|
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(000000100007AC0)
-0020 000000100007AA0 00000000 00000000 C3C5C5C3 C1C14040 |.....CEECAA |
-0010 000000100007AB0 00000000 00000000 00000000 00000000 |.....|
+0000 000000100007AC0 - +00003F 000000100007AFF |.....|
same as above
Storage around GPR13(0000001082FF180)
-0020 0000001082FF160 00000001 00000000 00000000 25853F8C |.....e..|
-0010 0000001082FF170 00000000 25D88AC0 00000000 25C20719 |.....Q.....B..|
+0000 0000001082FF180 084134F0 25D82118 082FE740 25853F44 |...0.Q....X.e..|
+0010 0000001082FF190 08413550 00000020 000044E8 25D8218C |..&.....Y.Q..|
+0020 0000001082FF1A0 25D8A000 00007AC0 082FF8C8 00000000 |.Q.....8H.....|
+0030 0000001082FF1B0 00000000 00000000 00000000 00000D90 |.....|
Storage around GPR14(00000010000B238)
-0020 00000010000B218 00000000 00000000 00000000 00000000 |.....|
-0010 00000010000B228 - +00003F 00000010000B277 |.....|
same as above
Storage around GPR15(000000025D8D048)
-0020 000000025D8D028 00008000 00007E78 00000000 00000000 |.....=.....|
-0010 000000025D8D038 00000000 00000000 25D8D000 00000128 |.....:.....Q.....|
+0000 000000025D8D048 00C300C5 00C500F1 00000038 00000184 |.C.E.E.1.....d|
+0010 000000025D8D058 EB134880 0024B904 0014EB4F 46800024 |.....|
+0020 000000025D8D068 A74BFE80 41940800 B90400F6 A7FBFFF0 |x...m...6x..0|
+0030 000000025D8D078 E3806100 001707F8 02CE00FF 00000024 |T./...8.....|

```

Figure 168. 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  
ERRNO=xxxxxxxxx ERRNO2=xxxxxxxxx
```

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 169 on page 484 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 F1F0F8F7 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 5D4086A6 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 F1F0F8F7 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 169. 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 5D406D6D 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 169. 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 450.

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 170 on page 486 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, GONUM (to produce statement numbers) 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 170. C routine with a divide-by-zero error

To debug this routine, use the following steps:

1. Locate the Original 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'52' 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 171 on page 487 shows the generated traceback from the dump.

Information for enclave main

Information for thread 25AC70A000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
00000001	CEEHDSP	+00000000		CELQLIB	CEEHDSP	HLE7730	Call
00000002	CEEOSIGJ	+0000094E		CELQLIB	CEEOSIGJ	HLE7730	Call
00000003	CELQHRD	+0000024E		CELQLIB	CELQHRD	HLE7730	Call
00000004	CEEOSIGG	-1D58AE60		CELQLIB	CEEOSIGG	HLE7730	Call
00000005	CELQHRD	+0000024E		CELQLIB	CELQHRD	HLE7730	Call
00000006	funcb	+00000052	18	CDIVZERO	CDIVZERO		Exception
00000007	main	+00000026	12	CDIVZERO	CDIVZERO		Call
00000008	CELQINIT	+00001340		CELQLIB	CELQINIT	HLE7730	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Attributes	Service	Status
00000001	0000001082FAAC0	0000000025793E88	0000000025793E88	00000000	20060109	XPLINK EBCDIC	POSIX	Floating Point
00000002	0000001082FD3E0	000000002588FE28	000000002588FE28	0000094E	20051214	XPLINK EBCDIC	POSIX	Floating Point
00000003	0000001082FDDE0	00000000257B50D8	00000000257B50D8	0000024E	20051214	XPLINK EBCDIC	POSIX	Floating Point
00000004	0000001082FE020	00000000258894E0	00000000258894E0	1D58AE60	20051214	XPLINK EBCDIC	POSIX	Floating Point
00000005	0000001082FEE40	00000000257B50D8	00000000257B50D8	0000024E	20051214	XPLINK EBCDIC	POSIX	Floating Point
00000006	0000001082FF080	00000000257080D0	0000000000000000	*****	20060221	XPLINK EBCDIC	POSIX	IEEE
00000007	0000001082FF180	0000000025708170	0000000000000000	*****	20060221	XPLINK EBCDIC	POSIX	IEEE
00000008	0000001082FF280	000000002570B010	000000002570B010	00001340	20051214	XPLINK EBCDIC	POSIX	Floating Point

Fully Qualified Names

DSA	Entry	Program Unit	Load Module
00000006	funcb	PLPSC:/'POSIX.CRTL.C(CDIVZERO)'	/u/alfcar/tools/CDIVZERO
00000007	main	PLPSC:/'POSIX.CRTL.C(CDIVZERO)'	/u/alfcar/tools/CDIVZERO

Condition Information for Active Routines

Condition Information for PLPSC:/'POSIX.CRTL.C(CDIVZERO)'

(DSA address 0000001082FF080)

CIB Address: 0000001082FBE00

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: PLPSC:/'POSIX.CRTL.C(CDIVZERO)'

Entry: funcb Statement: 18 Offset: +00000052

Machine State:

ILC..... 0002 Interruption Code..... 0009

PSW..... 0785240180000000 0000000025708124

GPR0..... 0000000000000000 GPR1..... 0000001082FFA40 GPR2..... 000000108414340 GPR3..... 000000108414340

GPR4..... 0000001082FF080 GPR5..... 000000108300060 GPR6..... 0000000000000000 GPR7..... 000000100000000

GPR8..... 000000002570CC8 GPR9..... 00000000257081B8 GPR10.... 0000000025708278 GPR11.... 000000108414330

GPR12.... 000000100005300 GPR13.... 000000000006F60 GPR14.... 00000000257BF218 GPR15.... 00000000000001F

FPC..... 00000000

FPR0..... 4327CCCA 93BCFEF1 FPR1..... 00000000 00000000

FPR2..... 00000000 00000000 FPR3..... 00000000 00000000

FPR4..... 3C100000 00000000 FPR5..... 00000000 00000000

FPR6..... 34100000 00000000 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(0000000025708112)

+0000 0000000025708112 0004E300 70000014 A70BFFB7 8E600020 |..T.....x....-..|

+0010 0000000025708122 1D60B904 00075000 48C0E320 48C00014 |.-.....&...T.....|

Figure 171. Sections of the dump from example C/C++ routine (Part 1 of 2)

Chapter 13. Debugging AMODE 64 C/C++ routines 487

```

Parameters, Registers, and Variables for Active Routines:
funcb (DSA address 00000001082FF080):
  DOWNSTACK DSA
  Saved Registers:
    GPR0..... *****
    GPR1..... *****
    GPR2..... *****
    GPR3..... *****
    GPR4..... 00000001082FF080  GPR5..... BBBB
    GPR6..... 00000000257B50D8  GPR7..... 0000000100000001
    GPR8..... 0000000025700CC8  GPR9..... 00000000257081B8  GPR10.... 0000000025708278  GPR11.... 0000000108414330
    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 08300060 | .....1.....-|
    +0810 00000001082FF890 00000000 25708170 00000000 2570819A | .....a.....a.|
    +0820 00000001082FF8A0 00000000 25700CC8 00000000 25709138 | .....H.....j.|
    +0830 00000001082FF8B0 00000000 25708278 00000001 08414330 | .....b.....-|
    +0840 00000001082FF8C0 00000001 00005300 00000000 00006F60 | .....?-|
    +0850 00000001082FF8D0 00000000 257BF218 00000000 0000001F | .....#2.....-|

```

Figure 171. Sections of the dump from example C/C++ routine (Part 2 of 2)

2. In the traceback, statement number 12, corresponding to DSA 7, refers to line: `aa = funcb(&aa);` in the listing. This is where entry `funcb` is called. Similarly, statement number 18, corresponding to DSA 6, points to line: `result = fa/(statint-73);` in the listing. This line is where the divide by zero exception takes place.
3. Locate the instruction with the divide-by-zero error in the Pseudo Assembly Listing in Figure 172 on page 489.
The offset (within `funcb`) of the exception from the traceback (X'52') reveals the divide instruction: `DR R6,R0` at that location. Instructions at offsets X'32' through X'58' 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

Timestamp and Version Information
000010 F2F0 F0F6                      =C'2006'           Compiled Year
000014 F0F2 F2F1                      =C'0221'           Compiled Date MMDD
000018 F1F5 F3F3 F1F5                  =C'153315'         Compiled Time HHMMSS
00001E F0F1 F0F8 F0F0                  =C'010800'         Compiler Version

Timestamp and Version End

15694A01 V1.8 z/OS XL C                      'POSIX.CRTL.C(CDIVZERO)': funcb                      02/21/06 15:33:15 Page 148

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

000015 | * int funcb(int *pp) {

000028                      @2L0   DS    0D
000028 00C300C5                      =F'12779717'       XPLink entrypoint marker
00002C 00C500F1                      =F'12910833'
000030 00000108                      =F'264'
000034 00000100                      =F'256'
000000                      000015 | funcb   DS    0D
000000 EB59 4708 0024 000015 |         STMG  r5,r9,1800(r4)
000006 A74B FF00 000015 |         AGHI  r4,H'-256'
00000A                      End of Prolog

00000A C090 0000 006F 000000 |         LARL  r9,F'111'
000010 E310 4980 0024 000015 |         STG   r1,pp(,r4,2432)
000016                      000016 | * int result;
000017                      000017 | * fa = *pp;
000016 E360 4980 0004 000017 |         LG   r6,pp(,r4,2432)
00001C E300 6000 0014 000017 |         LGF  r0,(*)int(,r6,0)
000022 E360 4808 0004 000017 |         LG   r6,#Save_ADA_Ptr_2(,r4,2056)
000028 E360 6000 0004 000017 |         LG   r6,A(fa)(,r6,0)
00002E 5000 6000                      000017 |         ST   r0,fa(,r6,0)
000018                      000018 | * result = fa/(statint-73);
000032 E360 6000 0014 000018 |         LGF  r6,fa(,r6,0)
000038 E370 4808 0004 000018 |         LG   r7,#Save_ADA_Ptr_2(,r4,2056)
00003E E370 7008 0004 000018 |         LG   r7,A(statint)(,r7,8)
000044 E300 7000 0014 000018 |         LGF  r0,statint(,r7,0)
00004A A70B FFB7 000018 |         AGHI  r0,H'-73'
00004E 8E60 0020 000018 |         SRDA  r6,32
000052 1D60                      000018 |         DR   r6,r0
000054 B904 0007 000018 |         LGR  r0,r7
000058 5000 48C0 000018 |         ST   r0,result(,r4,2240)
000019                      000019 | * printf("Result = %d\n",result);
00005C E320 48C0 0014 000019 |         LGF  r2,result(,r4,2240)
000062 E360 4808 0004 000019 |         LG   r6,#Save_ADA_Ptr_2(,r4,2056)
000068 EB56 6010 0004 000019 |         LMG  r5,r6,A(printf)(r6,16)
00006E B904 0019 000019 |         LGR  r1,r9
000072 0D76 000019 |         BASR r7,r6
000074 0700 000019 |         NOPR 0
000020                      000020 | * return result;
000076 E330 48C0 0014 000020 |         LGF  r3,result(,r4,2240)
000021                      000021 | * }
00007C                      000021 | @2L3   DS    0H

00007C                      Start of Epilog
00007C E370 4818 0004 000021 |         LG   r7,2072(,r4)
000082 EB89 4820 0004 000021 |         LMG  r8,r9,2080(r4)
000088 4140 4100 000021 |         LA   r4,256(,r4)
00008C 47F0 7002 000021 |         B   2(,r7)

```

Figure 172. Pseudo assembly listing (Part 1 of 3)

```

*** General purpose registers used: 1111011101000000
*** Floating point registers used: 1111111100000000
*** Size of register spill area: 128(max) 0(used)
*** Size of dynamic storage: 0
*** Size of executable code: 144

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) {

0000C8      @1L0    DS    0D
0000C8 00C300C5      =F'12779717'      XPLink entrypoint marker
0000CC 00C500F1      =F'12910833'
0000D0 00000090      =F'144'
0000D4 00000100      =F'256'
000000      000009      main    DS    0D
000000 EB57 4708 0024 000009      STMG  r5,r7,1800(r4)
000006 A74B FF00      000009      AGHI  r4,H'-256'
00000A      End of Prolog

000010      * int aa, bb=1;
00000A A709 0001      000010      LGHI  r0,H'1'
00000E 5000 48C4      000010      ST    r0,bb(,r4,2244)
000011      * aa = bb;
000011      000011      LGF   r0,bb(,r4,2244)
000012 E300 48C4 0014 000011      ST    r0,aa(,r4,2240)
000018 5000 48C0      000011      * aa = funcb(ā);
000012      000012      LA    r1,aa(,r4,2240)
000020 E350 4808 0004 000012      LG    r5,#Save_ADA_Ptr_1(,r4,2056)
000026 A775 FF9D      000012      BRAS  r7,funcb
00002A 0701      000012      NOPR  1
00002C B904 0003      000012      LGR   r0,r3
000030 5000 48C0      000012      ST    r0,aa(,r4,2240)
000031      000013      * return(aa);
000034 E330 48C0 0014 000013      LGF   r3,aa(,r4,2240)
00003A      000014      * }
00003A      000014      @1L2    DS    0H

00003A      Start of Epilog
00003A E370 4818 0004 000014      LG    r7,2072(,r4)
000040 4140 4100      000014      LA    r4,256(,r4)
000044 47F0 7002      000014      B    2(,r7)

*** General purpose registers used: 1111011100000000
*** Floating point registers used: 1111111100000000
*** Size of register spill area: 128(max) 0(used)
*** Size of dynamic storage: 0
*** Size of executable code: 72

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 07C0     =H'1984'     GPR save mask
000004 00000090 =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 00000090 =F'144'     Code length
000014 01000000 =F'1677216' Interface mapping flags
000018 0005 **** =AL2(5),C'funcb'
000020 FFFFFFF8     =F'-264'    Offset to Entry Point Marker

PPA1 End

```

Figure 172. Pseudo assembly listing (Part 2 of 3)

```

PPA1: Entry Point Constants
000000 02                =A1(2)                Version
000001 CE            =A1(206)             CEL signature
000002 0700          =H'1792'             GPR save mask
000004 00000068      =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 00000048      =F'72'              Code length
000014 01000000      =F'16777216'        Interface mapping flags
000018 0004 ****      AL2(4),C'main'
000020 FFFFFFF70        =F'-144'            Offset to Entry Point Marker

PPA1 End

PPA4: Compile Unit Debug Block
000000 80000000          =F'-2147483648'     Additional Flags
000004 84060238      =F'-2079981000'     Flags
000008 0000000000000000 =D'0'              R/O static Offset
000010 0000000000000000 =D'0'              R/W static Offset
000018 0000000000000000 =D'0'              Symbol Offset Table Offset
000020 FFFFFFFF80      =D'-384'            CSECT Start Offset
000028 0000000000000000 =D'0'              Code CSECT Size
000030 FFFFFFFF88      =D'-376'            No program region
000038 00000000          =F'0'              DWARF File Name
00003C ****          C''

PPA4 End

PPA2: Compile Unit Block
000000 0300 2204          =F'50340356'        Flags
000004 FFFF FE40      =A(CELSQSTR-PPA2)
000008 FFFF FFC0      =A(PPA4-PPA2)
00000C FFFF FE50      =A(TIMESTAMP-PPA2)
000010 0000 0000      =F'0'              No primary
000014 9140 0000      =F'-1858076672'     Flags

PPA2 End

```

Figure 172. Pseudo assembly listing (Part 3 of 3)

- 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 173 shows the WSA address.

```

Enclave Control Blocks:
.
.
.
DLL Information:
WSA Addr      Module Addr      Thread ID      Use Count  Name
0000000108300050

```

Figure 173. 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 174 on page 492. In this example, the offset is X'30'.

```

-----
CLASS C_WSA64          LENGTH =      38  ATTRIBUTES = MRG, DEFER , RMODE= 64
                      OFFSET =      0  IN SEGMENT 002          ALIGN = QDWORD
-----

CLASS
OFFSET  NAME          TYPE    LENGTH  SECTION
   0    $PRIV000012  PART     10
   10   CDIVZERO#S   PART     20   CDIVZERO#C
   30   statint      PART      4   statint
   34   fa           PART      4   fa

```

Figure 174. Writable static map

6. 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.
7. Use IPCS to display the writeable static area in the system dump. The value at location X'108300080' 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 175. IPCS storage display of the writeable static area

Calling a nonexistent function

Figure 176 on page 493 demonstrates the error of calling a nonexistent function. This routine was compiled with the compiler options LP64, GONUM, LIST, OFFSET, 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 176. C/C++ example of calling a nonexistent subroutine

To debug this routine, use the following steps:

1. Locate the Original Condition message in the Condition Information for Active Routines section of the dump, shown in Figure 177 on page 494. 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 Location section of the dump indicates that the exception occurred at offset X'-209000D0' within function funca and that there may have been a bad branch from statement 17 offset X'+00000036' 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'00000002' in the instruction address of the PSW shown in the Condition Information section. This address indicates that an instruction in the routine branched outside the bounds of the routine.

In the traceback, the statement number displayed for entry 'main' points to line 12 in the source code shown in Figure 176. This line contains the statement "funca(&aa); " in which entry 'funca' is called. As message CEE3841I explains, for entry 'funca' no statement number could be displayed. In this example, this problem is caused because 'funca' has an invalid offset. For further information about this message see *z/OS Language Environment Run-Time Messages*.

CEE3845I CEEDUMP Processing started.

Information for enclave main

Information for thread 8000000000000000

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHDSP	+00000000		CELQLIB	CEEHDSP	D1908	Call
2	CELQHR0D	+0000024E		CELQLIB	CELQHR0D	D1908	Call
3	funca	-209000D0		EXIST			Exception
CEE3841I A statement number is not available for this DSA. An internal routine failed with return code 08 and reason code 1C							
4	main	+00000034	12	EXIST	EXIST		Call
5	CELQINIT	+0000134C		CELQLIB	CELQINIT	D1908	Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile	Attributes
1	00000001082FC520	0000000020AB3680	0000000020AB3680	00000000	20061215	CEL	XPLINK EBCDIC HFP
2	00000001082FEE40	0000000020AC6AA0	0000000020AC6AA0	0000024E	20061215	CEL	XPLINK EBCDIC HFP
3	00000001082FF080	00000000209000D0	0000000000000000	*****	20070122	C/C++	XPLINK EBCDIC IEEF
4	00000001082FF180	0000000020900138	0000000000000000	*****	20070122	C/C++	XPLINK EBCDIC IEEF
5	00000001082FF280	0000000020903010	0000000020903010	0000134C	20061215	CEL	XPLINK EBCDIC HFP

Fully Qualified Names

DSA	Entry	Program Unit	Load Module
4	main	PLPSC://'POSIX.CRTL.C(EXIST)'	EXIST

Condition Information for Active Routines

Condition Information for (DSA address 00000001082FF080)

CIB Address: 00000001082FD860

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: -209000D0

Possible Bad Branch: Statement: 17 Offset: +00000036

Machine State:

ILC..... 0002 Interruption Code.... 0001

PSW..... 0785240180000000 0000000000000002

GPR0..... 0000000108300060 GPR1..... 00000001082FFA40 GPR2..... 0000000108401F60 GPR3..... 0000000108400070

GPR4..... 00000001082FF080 GPR5..... 000A0000000130E1 GPR6..... 0000000000000000 GPR7..... 0000000020900108

GPR8..... 00000000209000DC GPR9..... 00000000209001A0 GPR10..... 00000000209002A8 GPR11..... 0000000108401F50

GPR12..... 0000000100005340 GPR13..... 0000000000006F58 GPR14..... 0000000020B4E0A0 GPR15..... 000000000000001F

Storage dump near condition, beginning at location(0000000000000000)

+0000 0000000000000000 Inaccessible storage.

+0010 0000000000000010 Inaccessible storage.

GPREG STORAGE:

Storage around GPR0 (0000000108300060)

-0020	0000000108300040	00000001	08300000	00000000	00000060-
-0010	0000000108300050	C36DE6E2	C1F6F440	40404040	40404040	C_WSA64
+0000	0000000108300060	94818995	0086A495	83810000	00000000	main.funca.....
+0010	0000000108300070	00000001	08300090	00000000	209000D0
+0020	0000000108300080	00000000	000000C0	00000000	20E71FF8X.8
+0030	0000000108300090	00000000	00000000	00000000	00000000

Figure 177. Sections of the dump from example C routine (Part 1 of 3)

```

Storage around GPR1 (0000001082FFA40)
-0020 0000001082FFA20 00000000 00000000 00000000 00000000 |.....|
-0010 0000001082FFA30 - +FFFFFF 0000001082FFA3F |same as above|
+0000 0000001082FFA40 00000000 00000000 00000001 08300060 |.....-|
+0010 0000001082FFA50 00000000 00000000 00000000 00000000 |.....|
+0020 0000001082FFA60 - +00003F 0000001082FFA7F |same as above|
Storage around GPR2 (000000108401F60)
-0020 000000108401F40 00000001 08400000 00000000 00000040 |.....|
-0010 000000108401F50 00000001 00000020 00000001 08401F60 |.....-|
+0000 000000108401F60 00000001 08401BF0 00000000 00000000 |.....0.....|
+0010 000000108401F70 00000000 00000000 00000000 00000000 |.....|
+0020 000000108401F80 00000001 08400000 00000000 000014E0 |.....|
+0030 000000108401F90 C3C4D3C7 6DC8C4D9 00000001 08401FBC |CDLG_HDR.....|
Storage around GPR3 (000000108400070)
-0020 000000108400050 6DC9C3D6 D5E56DD4 D6C4C57E D3000000 |_ICONV_MODE=L...|
-0010 000000108400060 00000001 08400000 00000000 000000C0 |.....|
+0000 000000108400070 00000001 084042B0 00000000 00000000 |.....|
+0010 000000108400080 00000000 00000000 00000000 20900000 |.....|
+0020 000000108400090 00000000 00000000 00000001 08300050 |.....&|
+0030 0000001084000A0 00000000 00000000 00000001 80C00000 |.....|
Storage around GPR4 (0000001082FF080)
+0800 0000001082FF880 00000001 082FF180 00000001 08300060 |.....1.....-|
+0810 0000001082FF890 00000000 20900000 00000000 2090016E |.....>|
+0820 0000001082FF8A0 00000000 20900144 00000000 209001A0 |.....|
+0830 0000001082FF8B0 00000000 209002A8 00000001 08401F50 |.....y.....&|
+0840 0000001082FF8C0 00000001 00005340 00000000 00006F58 |.....?.|
+0850 0000001082FF8D0 00000000 20B4E0A0 00000000 0000001F |.....|
Storage around GPR5 (000A0000000130E1)
-0020 000A0000000130C1 Inaccessible storage.
-0010 000A0000000130D1 Inaccessible storage.
+0000 000A0000000130E1 Inaccessible storage.
+0010 000A0000000130F1 Inaccessible storage.
+0020 000A000000013101 Inaccessible storage.
+0030 000A000000013111 Inaccessible storage.
Storage around GPR6 (0000000000000000)
+0000 0000000000000000 Inaccessible storage.
+0010 0000000000000010 Inaccessible storage.
+0020 0000000000000020 Inaccessible storage.
+0030 0000000000000030 Inaccessible storage.
+0040 0000000000000040 Inaccessible storage.
+0050 0000000000000050 Inaccessible storage.
Storage around GPR7 (0000000020900108)
-0020 00000000209000E8 E3104980 0024E360 48080004 E3606010 |T....T-...T--|
-0010 00000000209000F8 0004E360 60000004 EB566000 00040D76 |..T-.....|
+0000 0000000020900108 0700B904 0003E360 49800004 50006000 |.....T-...&-..|
+0010 0000000020900118 47F08040 EB484800 000447F0 70020000 |.0. ....0....|
+0020 0000000020900128 00C300C5 00C500F1 000000B0 00000100 |.C.E.E.1.....|
+0030 0000000020900138 EB494700 0024A74B FF000D80 C0900000 |.....x.....|
:
:
:
Enclave Control Blocks:
:
:
:
DLL Information:
WSA Addr      Module Addr      Thread ID      Use Count      Name
000000108300050 00000002105B000 800000000000000 00000001      main
000000108301210 00000002105B000 800000000000000 00000002      CDAEQED
000000108306E10 0000000210D0000 800000000000000 00000001      CDAEQDPI
00000010830FE90 000000021194000 800000000000000 00000001      CELQDSNF
:
:
:

```

Figure 177. Sections of the dump from example C routine (Part 2 of 3)

Run-Time Options Report:

LAST WHERE SET	OPTION
DD:CEEOPPTS	CEEDUMP(0, SYSOUT=*, FREE=END, SPIN=UNALLOC)
DD:CEEOPPTS	DYNDUMP(POSIX.HEALY.ZOS19, DYNAMIC, TDUMP)
Installation default	ENVAR("")
DD:CEEOPPTS	ENVAR(" I CONV_MODE=L")
Installation default	FILETAG(NOAUTOCVT, NOAUTOTAG)
Installation default	HEAPCHK(OFF, 1, 0, 0, 0)
Installation default	HEAPPOLS64(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)
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,)
DD:CEEOPPTS	TERMTHDACT(UADUMP, , 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)

Process Control Blocks:

PCB(0000000100003CA0)					
+0000 0000000100003CA0	C3C5C5D7	C3C24040	00000000	00000000	CEEPCB
+0010 0000000100003CB0	00000000	00000000	00000000	00000000
+0020 0000000100003CC0	- +0000FF	0000000100003D9F			same as above
+0100 0000000100003DA0	03030208	00000000	00000000	00000000
+0110 0000000100003DB0	00000001	00004048	00000000	00000000
+0120 0000000100003DC0	00000000	00000000	00000000	00000000
+0130 0000000100003DD0	00000000	00000000	00000001	00003A10
+0140 0000000100003DE0	7F800000	00000000	00000000	00000000	".....
+0150 0000000100003DF0	00000000	00000000	00000000	00000000
+0160 0000000100003E00	- +0001BF	0000000100003E5F			same as above
MEML(0000000100004048)					
+0000 0000000100004048	00000000	00000000	00000000	00000000
+0010 0000000100004058	- +00005F	00000001000040A7			same as above
+0060 00000001000040A8	00000001	00008688	00000000	00000000fh.....
+0070 00000001000040B8	00000000	00000000	00000000	00000000
+0080 00000001000040C8	- +0001AF	00000001000041F7			same as above

CEE3846I CEEDUMP Processing completed.

Figure 177. Sections of the dump from example C routine (Part 3 of 3)

- Find the branch instructions for funca in the listing in Figure 178 on page 497. Notice the BASR r7, r6 instruction at offset X'000036'. This branch is part of the instruction aa=func_ptr(); in statement 17 in Figure 176 on page 493.

```

OFFSET OBJECT CODE      LINE# FILE#   P S E U D O   A S S E M B L Y   L I S T I N G

Timestamp and Version Information
000010 F2F0 F0F7                =C'2007'           Compiled Year
000014 F0F1 F2F2                =C'0122'          Compiled Date MMDD
000018 F1F6 F2F5 F4F6          =C'162546'       Compiled Time HHMMSS
00001E F0F1 F0F9 F0F0          =C'010900'       Compiler Version

Timestamp and Version End

OFFSET OBJECT CODE      LINE# FILE#   P S E U D O   A S S E M B L Y   L I S T I N G

000016 |      * void funca(int* aa) {

000028                @2L0   DS      0D
000028 00C300C5                =F'12779717'      XPLink entrypoint marker
00002C 00C500F1                =F'12910833'
000030 000000F8                =F'248'
000034 00000100                =F'256'
000000                000016 |      funca   DS      0D
000000 EB48 4700 0024 000016 |      STMG    r4,r8,1792(r4)
000006 A74B FF00                000016 |      AGHI    r4,H'-256'
00000A 0D80                000016 |      BASR    r8,0
00000C                End of Prolog

00000C E350 4808 0024 000016 |      STG     r5,#Save_ADA_Ptr_2(,r4,2056)
000012 E350 48C0 0024 000016 |      STG     r5,#Save_WSA_Ptr_2(,r4,2240)
000018 E310 4980 0024 000016 |      STG     r1,aa(,r4,2432)
000017                *      *aa = func_ptr();
00001E E360 4808 0004 000017 |      LG     r6,#Save_ADA_Ptr_2(,r4,2056)
000024 E360 6010 0004 000017 |      LG     r6,A(func_ptr)(,r6,16)
00002A E360 6000 0004 000017 |      LG     r6,func_ptr(,r6,0)
000030 EB56 6000 0004 000017 |      LMG    r5,r6,&ADA_&EPA(r6,0)
000036 0D76                000017 |      BASR    r7,r6
000038 0700                000017 |      NOPR    0
00003A B904 0003                000017 |      LGR    r0,r3
00003E E360 4980 0004 000017 |      LG     r6,aa(,r4,2432)
000044 5000 6000                000017 |      ST     r0,(*)int(,r6,0)
000048 47F0 8040                000018 |      *      return;
000048                B      @2L3
00004C                *      }
00004C                @2L3   DS      0H

00004C                Start of Epilog
00004C EB48 4800 0004 000019 |      LMG    r4,r8,2048(r4)
000052 47F0 7002                000019 |      B      2(,r7)

*** General purpose registers used: 111111110000000
*** Floating point registers used: 111111110000000
*** Size of register spill area: 256(max) 0(used)
*** Size of dynamic storage: 0
*** Size of executable code: 86

```

Figure 178. Pseudo assembly listing (Part 1 of 2)

```

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
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) {
000090 |          @1L0   DS      0D
000090 00C300C5          =F'12779717'      XPLink entrypoint marker
000094 00C500F1          =F'12910833'
000098 000000B0          =F'176'
00009C 00000100          =F'256'
000000 |          main   DS      0D
000000 EB49 4700 0024 000010 |          STMG   r4,r9,1792(r4)
000006 A74B FF00 000010 |          AGHI   r4,H'-256'
00000A 0D80 000010 |          BASR   r8,0
00000C |          End of Prolog
00000C C090 0000 002E 000000 |          LARL   r9,F'46'
000012 E350 4808 0024 000010 |          STG    r5,#Save_ADA_Ptr_1(,r4,2056)
000018 E350 48C8 0024 000010 |          STG    r5,#Save_WSA_Ptr_1(,r4,2248)
000011 |          *   int aa;
000012 |          *   funca(&aa);
000012 |          LA     r1,aa(,r4,2240)
000022 E350 4808 0004 000012 |          LG     r5,#Save_ADA_Ptr_1(,r4,2056)
000028 E360 4808 0004 000012 |          LG     r6,#Save_ADA_Ptr_1(,r4,2056)
00002E E360 6018 0004 000012 |          LG     r6,=V(funca)(,r6,24)
000034 0D76 000012 |          BASR   r7,r6
000036 0700 000012 |          NOPR   0
000033 |          *   printf("result of funca = %d\n",aa);
000038 E320 48C0 0014 000013 |          LGF   r2,aa(,r4,2240)
00003E E360 4808 0004 000013 |          LG     r6,#Save_ADA_Ptr_1(,r4,2056)
000044 EB56 6020 0004 000013 |          LMG   r5,r6,=A(printf)(r6,32)
00004A B904 0019 000013 |          LGR   r1,r9
00004E 0D76 000013 |          BASR   r7,r6
000050 0700 000013 |          NOPR   0
000052 47F0 804A 000014 |          *   return;
000052 |          B     @1L2
000055 |          * }
000056 |          @1L2   DS      0H
000056 |          Start of Epilog
000056 EB49 4800 0004 000015 |          LMG   r4,r9,2048(r4)
00005C B909 0033 000015 |          SGR   r3,r3
000060 47F0 7002 000015 |          B     2(,r7)
000000 |          *** General purpose registers used: 111111111000000
000000 |          *** Floating point registers used: 111111100000000
000000 |          *** Size of register spill area: 256(max) 0(used)
000000 |          *** Size of dynamic storage: 0
000000 |          *** Size of executable code: 100
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
000104 0000 0000
000000 |          Constant Area
000010 9985A2A4 93A34096 864086A4 95838140 | result of funca
000010 7E406C84 1500 | = %d..

```

Figure 178. Pseudo assembly listing (Part 2 of 2)

- Find the offset of `func_ptr` in the Writable Static Map, shown in Figure 179 on page 499.

```

-----
CLASS  C_WSA64          LENGTH =      48  ATTRIBUTES = MRG, DEFER , RMODE= 64
                        OFFSET =      0  IN SEGMENT 002      ALIGN = QDWORD
-----

```

```

CLASS
OFFSET  NAME          TYPE   LENGTH  SECTION
   0    $PRIV000012   PART    10
  10    EXIST#S       PART    30    EXIST#C
  40    func_ptr      PART     8    func_ptr

```

Figure 179. Writable static map

4. Add the offset of func_ptr (X'40') to the address of WSA (X'108300050') (the WSA address was obtained from the dump report in Figure 178 on page 497). The result (X'108300090') 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 180 shows the sections of the dump.

```

LIST 01 08300050. ASID(X'00CC') LENGTH(X'0100') AREA
_8300050. C36DE6E2 C1F6F440 40404040 40404040 C_WSA64
_8300060. 94818995 0086A495 83810000 00000000 main.funca.....
_8300070. 00000001 08300090 00000000 209000D0 .....}
_8300080. 00000000 000000C0 00000000 20E71FF8 .....{.....X.8
_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 180. 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 388.

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                               Current Allocation
Management class . . : STANDARD           Allocated cylinders : 30
Storage class . . . : OS390              Allocated extents . : 1
Volume serial . . . : DPXDU1
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 181. 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. See “Formatting and analyzing system dumps” on page 389 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 heap pools statistics

To request a Language Environment storage report set RPTSTG(ON). If the C/C++ application specified the HEAPPOOLS(ON) or HEAPPOOLS64(ON) run-time option, the storage report displays HeapPools statistics. For a sample storage report showing HeapPools statistics for a multithreaded C/C++ application, see Figure 138 on page 354.

The following describes the C/C++ specific heap pool information.

HEAPPOOLS64 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.

HEAPPOOLS64 statistics:

- Pool *p* size: *ssss* Get requests: *gggg*
 - *p* — the number of the pool. When there are multiple pools for a cell size, the pools are numbered using the format *aa.bbb*.
 - *aa* — the number for the cell size.
 - *bbb* — the number for the pool within the cell size.
 - *ssss* — the cell size specified for the pool.
 - *gggg* - the number of storage requests that were satisfied from this 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 HEAPPOOLS64 Statistics report are not serialized when collected, therefore the values are not necessarily exact.

HEAPPOOLS64 summary: The HEAPPOOLS64 Summary displays a report of the HEAPPOOLS64 Statistics and provides suggested cell sizes.

- Specified Cell Size — the size of the cell specified in the HEAPPOOLS64 run-time option
- Element Size — the size of the cell plus any additional storage needed for control information or to maintain alignment
- Cells Per Extent — the cell pool count specified by the HEAPPOOLS64 run-time option. When there is more than one pool for a cell size, the count is divided by the number of pools.
- Extents Allocated — the number of times that each pool allocated an extent in order to optimize storage usage. The extents allocated needs to 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*.

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. HEAPPOOLS run-time option can be used by AMODE 64 applications to manage user heap storage above the 16MB line and below the 2GB bar.

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* Get requests: *gggg*
 - *p* — the number of the pool. When there are multiple pools for a cell size, the pools are numbered using the format *aa.bbb*.
 - *aa* — the number for the cell size.
 - *bbb* — the number for the pool within the cell size.
 - *ssss* — the cell size specified for the pool.
 - *gggg* — the number of storage requests that were satisfied from this 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.

- Specified Cell Size — the size of the cell specified in the HEAPPOOLS run-time option
- Element Size — the size of the cell plus any additional storage needed for control information or to maintain alignment
- 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:

Initial Heap Size * (Extent Percent/100))/(Element Size)

with a minimum of four cells.

Note: Having a small number of cells per extent is not suggested because the pool can allocate many extents, which causes 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 need to be either one or two. If the number of extents allocated is too high, 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 can 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:

$(\text{Maximum Cells Used} * (\text{Element Size}) * 100) / \text{Initial Heap Size}$
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 is used for the last suggested cell size.

For more information about stack and heap storage, 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.

User-created 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 range
 - *yyy* — the high side of the range
 - *n* — the number of requests in the 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 on the `__ucreate()` call
- Cells Per Extent — the cell pool count specified on the `__ucreate()` call
- Extents Allocated — the number of times that each pool allocated an extent in order to optimize storage usage.
- 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 `__umalloc/__ufree` 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 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 182 on page 505.

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 Extent	Extents Allocated	Maximum Cells Used	Cells In 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 182. 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 information.
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 12, “Using Language Environment AMODE 64 debugging facilities,” on page 367.
 - 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, see 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, see 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 510.

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 183 on page 510 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 183. 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 184 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 184. 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)

See <http://techsupport.services.ibm.com/guides/handbook.html> for detailed information.

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 35 describes how to produce documentation required for submission with the APAR.

Table 35. 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: <ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> • 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:

<http://www.ibm.com/systems/z/os/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.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS™, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

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	IMS	System/370
C/370	IMS/ESA	System z
CICS	Language Environment	TCS
COBOL/370	MVS	VisualAge
DB2	MVS/ESA	WebSphere
DFSMS	Open Class	z/OS
DFSMS/MVS	OS/390	zSeries
DFSORT	Resource Link	z/VM
IBM	SAA	
IBMLink	SXM	

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.

Adobe, Acrobat, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated 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
- *IBM Open Class Library Transition Guide*, SC09-4948

z/OS Run-Time Library Extensions

- *C/C++ Legacy Class Libraries Reference*, SC09-7652

z/OS Metal C Runtime Library

- *z/OS Metal C Programming Guide and Reference*, SA23-2225, SA23-2225

Enterprise COBOL for z/OS, V4R1

- *Enterprise COBOL for z/OS Licensed Program Specifications*, GI11-7871
- *Enterprise COBOL for z/OS Customization*, SC23-8526
- *Enterprise COBOL for z/OS Language Reference*, SC23-8528
- *Enterprise COBOL for z/OS Programming Guide*, SC23-8529
- *Enterprise COBOL for z/OS Migration Guide*, GC23-8527

Enterprise COBOL for z/OS, V3R4

- *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

Debug Tool

- Debug Tool documentation is available at: <http://www.ibm.com/software/awdtools/debugtool/>

VS FORTRAN Version 2

- *Language Environment for MVS & VM 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 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

z/OS

- *z/OS Introduction and Release Guide, GA22-7502*
- *z/OS Program Directory, G110-0670*
- *z/OS 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

- *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

- *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

- *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*

Index

Special characters

__abend 160, 460
__alloc 160, 460
__amrc 160, 460
__cabend() function 359, 366, 387
__code 160, 460
__error 160, 460
__feedback 160, 460
__last_op 160, 460
__le_cib_get() function 359
__le_message_get_and_and_write() function 352, 361
__le_message_get() function 361
__le_msg_write() function 352, 361
__msg 161, 462
__reset_exception_handler() function 359
__set_exception_handler() function 359, 360
_BPXK_MDUMP 82, 388
_EDC_ADD_ERRNO2 166, 468

Numerics

64-bit applications 351, 363, 367, 459

A

abend codes
 >= 4000 30, 364
 < 4000 30, 364
 4093 363
 passing to operating system 23
 system, example of 32, 365
 user-specified, example of 32, 365
 user, example of 32, 365, 366
 using 32, 365
abends
 internal, table of output 345
 Language Environment 32, 341, 365
 requested by assembler user exit 23
 system 32, 366
 under CICS 341
 user 32, 366
ABPERC run-time option
 function 8
 generating a system dump and 80
 modifying condition handling behavior and 21
ABTERMENC run-time option 8, 24
 using 24
accessibility 515
AGGREGATE compiler option 4, 6
AMODE 64 applications
 classifying errors 363
 debugging C/C++ 459
 preparing for debugging 351
 using debugging 367
anywhere heap
 statistics 18
APAR (Authorized Program Analysis Report) 511

APAR (Authorized Program Analysis Report)
 (*continued*)
 documentation 512
application program interfaces (API) 359, 361
application programs
 debugging
 handling a storage dump written to a BFS
 file 216, 499
 handling a storage dump written to an HFS
 file 216, 499
argument
 in dump 62
arguments, registers, and variables for active
 routines 60, 385
assembler language
 user exit 23, 24
 for CICS 345
 generating a system dump with 80, 387
 modifying condition handling behavior and 23
 using 23, 24
atexit
 information in dump 189
Authorized Program Analysis Report (APAR) 511
automatic variables
 locating in dump 289, 322

B

base locator
 for working storage 238
 in dump 238
below heap
 statistics 18
BLOCKS option of CEE3DMP callable service 37

C

C library function
 trace table entries for 483
C return codes to CICS 342
C-CAA (C-specific common anchor area)
 See C/C++, C-specific common anchor area (C-CAA)
C/C++
 __amrc
 example of structure 160, 460
 information in dump 191
 __msg 161, 462
 atexit
 information in dump 189
 C-specific common anchor area (C-CAA) 189
 cdump() function 177, 472
 compiler options 3
 debugging examples 206, 213, 485
 dump
 information in 190
 parameter in 173
 signal information in 189

C/C++ (*continued*)

- structure variables, locating in 175
- system, structures in 175
- file
 - control block information 190
 - status and attributes in dump 190
- functions
 - calling dump, example 177, 472
 - cdump() 43, 177, 369, 472
 - csnap() 43, 177, 178, 369, 472, 473
 - ctrace() 43, 177, 178, 369, 472, 473
 - fetch() 159, 459
 - fopen() 161, 462
 - perror() 159, 166, 459, 466
 - printf() 160, 460
 - to produce dump output 43, 369
- memory file control block 190
- stdio.h 160, 460
- timestamp 177

CAA (common anchor area) 65, 386, 390, 391, 392, 406, 438

call chain 62

CALL statement

- CDUMP/CPDUMP 259
- DUMP/PDUMP 258
- SDUMP 260

callable services 20

- 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 26, 361

case 2 condition token 26, 361

cdump() function 177, 364, 472

CEE prefix 29, 31, 363, 365

CEE3ABD—terminate enclave with an abend 20, 32, 80

- generating a dump and 80
- handling user abends and 20, 32
- modifying condition handling behavior and 20

CEE3DMP—generate dump 35, 62

- See also Language Environment dump
- generating a Language Environment dump with 35
- options 36
- relationship to PLIDUMP 281, 317
- syntax 36

CEE3GRO—returns location offset 20

CEE3SRP—set resume point 20

CEEBXITA assembler user exit 23

CEECXITA assembler user exit 345

CEEDCOD—decompose a condition token 26

CEEDUMP—Language Environment Dump Service

- See Language Environment dump
- control blocks 102, 403
- locating 146

CEEHDLR—register user condition handler 22

CEEHDLR—register user exception handler 360

CEEMGET—get a message 26

CEEMOUT—dispatch a message 24

CEEMRCE—move resume cursor to designated label 20

CEEMRCR—move resume cursor relative to handle cursor 20

CEEMSG—get, format, and dispatch a message 26

CEESGL—signal a condition 26

CEESTART 159

CEL prefix 363

CELQSTRT 459

character

- data dump 259

CHECK run-time option

- function 8
- modifying condition handling behavior and 21

CHECKOUT compiler option 4

CICS

- abends 341
 - application, from an EXEC CICS command 346
- debugging for 339
- debugging information, table of locations 339
- destination control table (DCT) 339
- example traceback in CESE transient data queue 339
- examples of output 339
- nonzero reason code returned, table of output 345
- reason codes 341
- register and program status word contents 341
- return codes
 - Language Environment 342
- run-time messages 339
- transaction
 - dump 340
 - rollback 345

class test 230

classifying errors table 29, 363

CLLE (COBOL load list entry) 237

COBCOM control block 242

COBOL
 base locator for working storage 238
 compiler options 5
 debugging examples 242, 255
 dump
 external data in 238
 file information in 238
 linkage section in 238
 local variables in 235
 routine information in 235
 run unit storage in 240
 stack frames for active routines in 235
 working storage in 238
 errors 229
 listings 233
 memory file control block 189
 program class storage 238
 return codes to CICS 342
 routine
 calling Language Environment dump service 233
COBVEC control block 242
command
 syntax diagrams xix
COMMAREA (Communication Area) 341
compiler options
 C 3
 COBOL 5
 Fortran 5
 LP64 351, 459
 PL/I 6
condition
 information
 for active routines 60, 384
 in dump 60, 385
 POSIX 26
 unhandled 27, 362
condition handling
 behavior, modifying 20
 user-written condition handler 20, 22
condition information block (CIB) 62, 74
condition manager 27, 362
CONDITION option of CEE3DMP callable service 38, 62
condition token 25, 26, 360
 case 1 26, 361
 case 2 26, 361
 example of 27, 361
conditions, nested 27, 362
control block
 for active routines 61, 385
csnap() function 178, 473
ctrace() function 178, 364, 473

D
data
 map listing 233
 values 62
DCB (data control block) 237
DCT (destination control table) 339
DEBUG run-time option 8

debugging
 C, examples 206, 213, 485
 COBOL, examples 242, 255
 for CICS 339
 Fortran, examples 265, 271
 PL/I, examples 293, 299, 326, 334
 tool 35, 367
DEPTHCONDLMT run-time option
 function 8, 352
 modifying condition handling behavior and 21
 wait/loop error and 30
diagnosis checklist 507
disability 515
display
 errnojr_value 33
DISPLAY statement 25, 229
displays
 errnojr_value 33
divide-by-zero error 206, 248, 267, 299, 304, 334, 338, 485
DSA (dynamic save area) 63
 See stack, frame
dummy DSA 63
dump
 an area of storage 259
 date in 56, 382
 dynamically allocated storage in 62
 storage
 written to an HFS file 216, 499
 symbolic 260
DUMP suboption of TERMTHDACT run-time option 39, 367
DUMP/PDUMP routine 258
 format specifications 258
 output 258
 usage considerations 258

E
ECB (enclave control block) 61, 386
EDB (enclave data block) 61, 386
EDC prefix 29, 31, 363, 365
edcmtext shell command 33
EIB (exec interface block) 341
enclave
 identifier in dump 58, 383
 member list 61, 62, 386
 storage 62
 termination
 behavior, establishing 24
entry information 56, 381
ENTRY option of CEE3DMP callable service 38
entry point
 name of active routines in dump 383
ERRCOUNT run-time option
 function 8
 modifying condition handling behavior and 21
 wait/loop error and 30
errno 189
errnojr_value
 displaying 33

- error
 - determining source of 507
 - message while Language Environment was handling
 - another error 27, 362
 - unanticipated 29, 363
- ESD compiler option 6
- examples
 - application abends from 346
 - C routines 206, 213, 485
 - calling a nonexistent subroutine 245, 297, 330
 - COBOL routines 242, 255
 - divide-by-zero error 206, 248, 267, 299, 304, 334, 338, 485
 - Fortran routines 265, 271
 - output under CICS 339
 - PL/I routines 293, 299, 326, 334
 - SUBSCRIPTRANGE error 242, 293, 326
- exception handling
 - behavior, modifying 359
 - user-written exception handler 359
- EXEC CICS DUMP statements 341
- external data
 - for COBOL programs in dump 238

F

- fetch
 - fetch information in dump 190
- fetch() function 159, 459
- fetchable module 159, 459
- file
 - for COBOL, in dump 238
 - status key 230
- file control block (FCB) 190
- FILES option of CEE3DMP callable service 37
- FLAG compiler option 4
- floating point registers
 - in dump 57
- fopen() function 161, 462
- FOR prefix 29, 31
- Fortran
 - compiler options 5
 - debugging examples 265, 271
 - dump services 257
 - errors, determining the source of 255
 - listings 257

G

- general purpose registers 57
- GMAREA 237
- GONUMBER compiler option 4

H

- HANDLE ABEND EXEC CICS command 339
- header files, C
 - ctest.h 177, 472
 - errno.h 206, 485
 - stdio.h 160, 460
 - stdlib.h 206, 485

- heap storage
 - created by CEECRHP callable service 19
 - in LEDATA Output 106, 409
 - reports 110, 418
 - storage in dump 62
 - user 17
- HEAP64 run-time option 353
- HEAPCHK run-time option
 - function 8, 111, 352, 419
- HEAPPOOLS
 - storage statistics 218, 219, 502, 503
 - user-created, __uheapreport() 503
 - user-created, _uheapreport 219
 - trace report 86, 112, 391, 420
- HEAPPOOLS run-time option 353
- HEAPPOOLS64
 - storage statistics 501
- HEAPPOOLS64 run-time option 353

I

- I/O
 - conventions 159, 459
- IBM prefix 29, 31
- IGZ prefix 29, 31
- INFMSGFILTER run-time option
 - function 8, 352
- INITIALIZE statement 230
- instance specific information (ISI) 361
- instruction length counter (ILC) in dump 60, 385
- Inter-procedural Analysis (IPA) 351
- interactive problem control system (IPCS)
 - analyzing a storage dump 217, 499
 - cbf command 446
 - Verbexit 389, 390, 393, 426
- INTERRUPT compiler option
 - function 7
- INTERRUPT run-time option 8
- interruption code in dump 60, 385
- IOHEAP64 run-time option 353
- ITBLK in dump 242

K

- keyboard 515

L

- LAA (library anchor area) 447
- language constructs 229
- Language Environment
 - return codes to CICS 342
 - symbolic feedback code 25, 360
- Language Environment dump 367, 369
 - C information in 188
 - CEEDUMP 8, 35, 351
 - COBOL information in 237
 - default options 38
 - example traceback in 62, 386
 - Fortran information in 263
 - multiple enclaves and 79

Language Environment dump (*continued*)

- options
 - BLOCKS 37
 - CONDITION 38, 62
 - ENCLAVE 36
 - ENTRY 38
 - FILES 37
 - FNAME 38
 - NOBLOCKS 37
 - NOCONDITION 38
 - NOENTRY 38
 - NOFILES 37
 - NOSTORAGE 37
 - NOTRACEBACK 37
 - NOVARIABLES 37
 - PAGESIZE(n) 38
 - STACKFRAME 38
 - STORAGE 37
 - THREAD 37
 - TRACEBACK 37, 62
 - VARIABLES 37, 62
- output
 - for C routines 182
 - for COBOL program 233
 - for Fortran routines 263
 - for PL/I routines 281, 316
 - information for multiple enclaves 43, 369
- PL/I information in 283, 319
- section descriptions 56, 381
- TERMTHDACT suboptions 40, 368
- title 56, 381
- traceback with condition information
 - C routine 182, 477
 - COBOL program 235
 - Fortran routine 256, 263, 264
 - Language Environment routine 56, 381
 - PL/I routine 281, 316
- using C functions 43, 369
- using CDUMP/CPDUMP subroutine 257
- using CEE3DMP callable service 35, 56, 381
- using DUMP/PDUMP subroutine 257
- using PLIDUMP subroutine 43, 281, 316
- using SDUMP subroutine 257
- using TERMTHDACT run-time option 39, 367

LCA (library communication area) 447

LEDATA

- IPCS Verbexit 84, 389
 - C/C++ Output 116, 426
 - COBOL Output 140
 - Format 85, 390
 - Parameters 85, 390
 - Understanding Output 87, 393

LIBHEAP64 run-time option 353

linkage section

- for COBOL programs in dump 238

LIST compiler option 4, 5, 7

listings generated by compiler

- COBOL 233
- Fortran 257
- PL/I 274, 310

LMESSAGE compiler option 7

- local
 - variables 60
- LP64
 - compiler option 351, 459

M

- machine state information
 - in dump 60, 385
- MAP compiler option 5, 7
- MEMLIMIT storage parameter 363
- memory file control block (MFCB) 189, 190
- message
 - classifying errors and 30, 364
 - run-time, CICS 339
 - user-created 24
 - using in your routine 24
- module
 - fetchable 159, 459
- module name prefixes, Language Environment 29, 363
- MSG suboption
 - of TERMTHDACT 39, 367
- MSGFILE run-time option
 - function 8
 - run-time messages and 30, 364
- MSGQ run-time option 8

N

- nested condition 27, 362
- no response (wait/loop) 30, 364
- 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 517
- NOTRACEBACK option of CEE3DMP callable service 37
- NOVARIABLES option of CEE3DMP callable service 37

O

- OFFSET compiler option 4, 5, 7
- optimizing
 - C 3, 63
 - COBOL 5
 - Fortran 260
 - PL/I 6
- options
 - C compiler 3
 - COBOL compiler 5
 - defaults for dump 40, 368
 - determining run-time in effect 9, 11, 352
 - Fortran compiler 5
 - Language Environment run-time 8, 351
 - PL/I compiler 6, 7

- out-of-storage condition
 - virtual storage 363
- OUTDD compiler option 5
- output
 - incorrect 30, 364
 - missing 30, 364

P

- page number
 - in dump 56, 382
- PAGESIZE(n) option of CEE3DMP callable service 38
- parameter
 - checking value of 23
- perror() function 166, 466
- PL/I
 - address of interrupt, finding in dump 286, 321
 - CAA address, finding in dump 292, 325
 - common anchor area (CAA) 292, 325
 - compiler listings
 - object code listing 278
 - static storage map 277
 - variable storage map 278
 - compiler options
 - generating listings with 274, 310
 - list of 6
 - CSECT 277
 - debugging examples 293, 299, 326, 334
 - dump
 - error type, finding in 286, 320
 - parameter list, finding contents in 290, 323
 - PL/I information, finding in 283, 290, 319, 323
 - PLIDUMP subroutine and 281, 316
 - statement number, finding in 286, 321
 - timestamp, finding in 290
 - variables, finding in 289, 322
 - ERROR ON-unit 272, 286, 308, 320
 - errors 271, 274, 307, 310
 - floating-point register 272, 308
 - object code listing 278
 - ON statement control block 278
 - static storage listing 277
 - SUBSCRIPTRANGE condition 273, 293, 296, 297, 308, 309, 326, 330
- PLIDUMP subroutine 281, 317
- PMR (problem management record) 511
- pointer
 - variable 159, 459
- PPA 509
- preventive service planning (PSP) bucket 508
- printf() function 25, 160, 460
- problem management record (PMR) 511
- procedure division listings 233
- process
 - control block 62, 386
 - member list 62, 386
- process control block (PCB) 62, 386
- PROFILE run-time option
 - function 9, 352
- program
 - class storage 238

- program prolog area 509
- program status word (PSW) 60, 385
- PSP (preventive service planning) bucket 508

Q

- QUIET suboption of TERMTHDACT run-time option 39, 367

R

- reason code
 - nonzero returned to CICS 345
 - under CICS 341
- registers 0–15
 - in dump 57
- release number
 - in dump 56, 382
- request parameter list (RPL) 190
- return code
 - bad or nonzero 30, 364
- RPTOPTS run-time option 9
- RPTSTG run-time option 11, 353
- run unit
 - COBOL 240
 - level control block 240
 - storage in dump 62, 240
- run-time
 - messages
 - under CICS 339
- run-time options 20, 359
 - determining those in effect 9, 11, 352
 - sample options report 9
 - specifying 23

S

- scope
 - terminator 229
- SDUMP routine
 - description 260
 - format specifications 260
 - output 260
 - usage considerations 260
- service routines
 - CDUMP/CPDUMP 259
 - DUMP/PDUMP 258
 - SDUMP 260
- SET statement 230
- shortcut keys 515
- signal information in dump 189
- sorted cross-reference listing 233
- SOURCE compiler option 4, 5, 7
- stack
 - frame 63
 - frame format 64, 65
- STACK64 run-time option 353
- STACKFRAME option of CEE3DMP callable service 38
- statement numbers
 - in dump 57, 382

- static
 - variables in dump 289, 322
 - writable map 171, 172, 176
- status
 - of routines in dump 384
- stderr 25, 364, 367, 368
- stdio.h 160, 460
- stdout 25
- storage
 - evaluating use of 11, 353
 - for active routines 61, 385
 - leak detecting 62, 111, 386, 419
 - report 11, 353
 - statistics 17, 18
- STORAGE compiler option 7
- storage dump
 - written to an HFS file 216, 499
- STORAGE option of CEE3DMP callable service 37
- STORAGE run-time option 9, 352
- structure
 - map 175
 - variable example code 175
- symbolic
 - feedback code 25, 360
- symbolic dumps 260
 - how to call under Fortran 260
- syntax diagrams
 - how to read xix
- system abend
 - with TRAP(OFF) 30, 364
 - with TRAP(ON) 30, 364
- system dump
 - generating 80, 386
 - in z/OS UNIX shell 82

T

- task global table (TGT) 237
- TERMINAL compiler option 4, 7
- TERMTHDACT run-time option
 - function 8, 39, 297, 330, 351, 367
 - generating a dump and 21, 359
 - modifying condition handling behavior and 9, 352
 - suboptions 39, 367
- TEST compiler option 4, 5, 6
- TEST run-time option 9, 352
- text file name prefixes, Language Environment 29, 363
- THDCOM in dump 242
- THREAD option of CEE3DMP callable service 37
- THREADSTACK64 run-time option 353
- time
 - in dump 56, 382
- TRACE run-time option
 - function 9, 352
 - trace table 147, 450
- TRACE suboption of TERMTHDACT run-time option 39, 367
- TRACEBACK option of CEE3DMP callable service 37, 62
- transaction
 - dump 341

- transaction (*continued*)
 - rollback 345
 - rollback effects of assembler user exit on 345
 - work area 341
- TRAP run-time option
 - function 9, 352
 - Language Environment condition handling and 22, 80, 387
 - Language Environment exception handling and 359

U

- UADUMP suboption of TERMTHDACT run-time option 21, 40, 359, 368
- UAIMM suboption of TERMTHDACT run-time option 21, 40, 359, 368
- UAONLY suboption of TERMTHDACT run-time option 21, 39, 359, 368
- UATRACE suboption of TERMTHDACT run-time option 21, 39, 359, 368
- unhandled conditions 24, 27, 362
 - establishing enclave termination behavior for 24
- USE EXCEPTION/ERROR declaratives 230
- USE FOR DEBUGGING declarative 231
- user
 - abend 32, 365, 366
 - code 30, 364
 - exit 23, 24
 - heap
 - statistics 18
 - stack
 - statistics 17
 - user-specified abends 32, 365
- USRHDLR run-time option 8, 22, 351
- utility and service subroutines
 - CDUMP/CPDUMP 259
 - DUMP/PDUMP 258
 - SDUMP 260

V

- variables
 - in Language Environment dump 62
 - structure example code 175
- VARIABLES option of CEE3DMP callable service 37, 62
- VBREF compiler option 5
- verb cross-reference 233
- Verbexit
 - LEDATA 84, 389
- version number
 - in dump 56, 382

W

- working storage
 - in dump 61, 238, 385

X

XPLINK

- downward-growing stack 64
- finding XPLINK information in a dump 198
- stack frame format 64, 65
- storage statistics 17
- trace table entries for 199

XREF compiler option 5, 7

XUFLOW run-time option

- function 9
- modifying condition handling behavior and 22

Z

z/OS UNIX System Services

- C application program and 216, 499
- generating a system dump 82

Readers' Comments — We'd Like to Hear from You

**z/OS
Language Environment
Debugging Guide**

Publication No. GA22-7560-09

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

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. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: mhvrcfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address



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
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5694-A01

Printed in USA

GA22-7560-09

