z/OS

# Language Environment Debugging Guide

GA22-7560-04

z/OS

# Language Environment Debugging Guide

GA22-7560-04

**Note**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 279.

**Fifth Edition, March 2004**

This is a major revision of GA22-7560-03.

This edition applies to Language Environment in z/OS Version 1 Release 5 (5694-A01), Version 1, Release 5 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

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

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

- Title and order number of this document
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

**© Copyright International Business Machines Corporation 1991, 2004. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

**Figures** . . . . . ix

**Tables** . . . . . xiii

**About this document** . . . . . xv
Using your documentation. . . . . xvi
How to read syntax diagrams . . . . . xvii
Symbols . . . . . xvii
Syntax items . . . . . xvii
Syntax examples. . . . . xviii
This Debugging Guide . . . . . xix
Where to find more information . . . . . xix
Accessing z/OS licensed documents on the Internet . . . . . xix
Using LookAt to look up message explanations . . . . . xx
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
C and C++ compiler options . . . . . 3
COBOL compiler options. . . . . 6
Fortran compiler options . . . . . 7
PL/I compiler options . . . . . 7
VisualAge PL/I compiler options . . . . . 8
Using Language Environment run-time options . . . . . 9
Determining run-time options in effect . . . . . 10
Using the CLER CICS transaction to display and set run-time options. . . . 12
Controlling storage allocation. . . . . 12
Stack storage statistics . . . . . 18
Heap storage statistics . . . . . 20
HeapPools storage statistics . . . . . 21
Modifying condition handling behavior . . . . . 21
Language Environment callable services . . . . . 21
Language Environment run-time options . . . . . 22
Customizing condition handlers . . . . . 23
Invoking the assembler user exit . . . . . 24
Establishing enclave termination behavior for unhandled conditions . . . . 25
Using messages in your routine. . . . . 25
C/C++ . . . . . 26
COBOL . . . . . 26
Fortran . . . . . 26
PL/I . . . . . 26
Using condition information . . . . . 26
Using the feedback code parameter . . . . . 27
Using the symbolic feedback code. . . . . 28

**Chapter 2. Classifying errors** . . . . . 31
Identifying problems in routines . . . . . 31
Language Environment module names . . . . . 31
Common errors in routines . . . . . 31

© Copyright IBM Corp. 1991, 2004

Interpreting run-time messages . . . . . 32
Message prefix . . . . . 33
Message number . . . . . 33
Severity code . . . . . 33
Message text . . . . . 34
Understanding abend codes . . . . . 34
User abends . . . . . 34
System abends . . . . . 34

**Chapter 3. Using Language Environment debugging facilities** . . . . . 37
Debug tool . . . . . 37
Language Environment dump service, CEE3DMP . . . . . 37
Generating a Language Environment dump with CEE3DMP . . . . . 37
Generating a Language Environment dump with TERMTHDACT . . . . . 41
Generating a Language Environment dump with language-specific functions 45
Understanding the Language Environment dump . . . . . 45
Debugging with specific sections of the Language Environment dump. . . . 62
Multiple enclave dumps. . . . . 75
Generating a system dump . . . . . 77
Steps for generating a system dump in a batch run-time environment. . . . 77
Steps for generating a system dump in an IMS run-time environment . . . . 78
Steps for generating a system dump in a CICS run-time environment . . . . 78
Steps for generating a Language Environment U4039 abend . . . . . 79
Steps for Generating a system dump in a z/OS UNIX shell. . . . . 79
Formatting and analyzing system dumps . . . . . 80
Preparing to use the Language Environment support for IPCS . . . . . 80
Language Environment IPCS Verbexit – LEDATA . . . . . 80
Format . . . . . 81
Parameters . . . . . 81
Understanding the HEAPPOOLS trace output . . . . . 83
Understanding the Language Environment IPCS verbexit LEDATA output 84
Understanding the HEAP LEDATA output . . . . . 98
Diagnosing heap fragmentation problems. . . . . 104
Understanding the C/C++-specific LEDATA output . . . . . 104
C/C++-specific sections of the LEDATA output . . . . . 109
Understanding the COBOL-specific LEDATA output . . . . . 110
COBOL-specific sections of the LEDATA Output . . . . . 114
Formatting individual control blocks . . . . . 114
Requesting a Language Environment trace for debugging. . . . . 116
Locating the trace dump . . . . . 117
Using the Language Environment trace table format in a dump report . . . 118
Understanding the trace table entry (TTE) . . . . . 118
Sample dump for the trace table entry . . . . . 123

**Part 2. Debugging Language-Specific Routines**. . . . . 125

**Chapter 4. Debugging C/C++ Routines** . . . . . 127
Debugging C/C++ Input/Output Programs . . . . . 127
Using the __amrc and __amrc2 Structures . . . . . 127
__last_op Values. . . . . 129
Displaying an Error Message with the perror() Function . . . . . 133
Using __errno2() to Diagnose Application Problems . . . . . 134
Using C/C++ Listings . . . . . 135
Generating C/C++ Listings and Maps . . . . . 135
Finding variables. . . . . 137
Generating a Language Environment Dump of a C/C++ Routine . . . . . 145

cdump() . . . . . 145
csnap() . . . . . 146
ctrace() . . . . . 146
Sample C Routine that Calls cdump . . . . . 146
Sample C++ Routine that Generates a Language Environment Dump . . . 148
Sample Language Environment Dump with C/C++-Specific Information 150
Finding C/C++ Information in a Language Environment Dump . . . . . 156
Sample Language Environment Dump with XPLINK-Specific Information 161
Finding XPLINK Information in a Language Environment Dump . . . . . 166
C/C++ Contents of the Language Environment Trace Tables. . . . . 167
Debugging Examples of C/C++ Routines . . . . . 172
Divide-by-Zero Error . . . . . 172
Calling a Nonexistent Non-XPLINK Function . . . . . 176
Calling a Nonexistent XPLINK Function . . . . . 179
Handling Dumps Written to the z/OS UNIX File System . . . . . 183
Multithreading Consideration . . . . . 184
Understanding C/C++ Heap Information in Storage Reports . . . . . 184
Language Environment Storage Report with HeapPools Statistics. . . . . 185
C Function, __uheapreport, Storage Report . . . . . 186

**Chapter 5. Debugging COBOL programs** . . . . . 189
Determining the source of error . . . . . 189
Tracing program logic . . . . . 189
Finding input/output errors . . . . . 189
Handling input/output errors. . . . . 190
Validating data (class test) . . . . . 190
Assessing switch problems . . . . . 190
Generating information about procedures. . . . . 190
Using COBOL listings . . . . . 193
Generating a Language Environment dump of a COBOL program. . . . . 193
COBOL program that calls another COBOL program . . . . . 193
COBOL program that calls the Language Environment CEE3DMP callable service . . . . . 194
Sample Language Environment dump with COBOL-specific information 195
Finding COBOL information in a dump. . . . . 197
Debugging example COBOL programs. . . . . 201
Subscript range error . . . . . 201
Calling a nonexistent subroutine . . . . . 204
Divide-by-zero error. . . . . 207

**Chapter 6. Debugging FORTRAN routines** . . . . . 213
Determining the source of errors in FORTRAN routines . . . . . 213
Identifying run-time errors . . . . . 213
Using FORTRAN compiler listings . . . . . 215
Generating a Language Environment dump of a FORTRAN routine . . . . . 215
DUMP/PDUMP subroutines. . . . . 216
CDUMP/CPDUMP subroutines . . . . . 217
SDUMP subroutine . . . . . 218
Finding FORTRAN information in a Language Environment dump. . . . . 221
Understanding the Language Environment traceback table . . . . . 222
Examples of debugging FORTRAN routines. . . . . 223
Calling a nonexistent routine . . . . . 223
Divide-by-zero error. . . . . 225

**Chapter 7. Debugging PL/I routines** . . . . . 229
Determining the source of errors in PL/I routines . . . . . 229

Logic errors in the source routine. . . . . 229
Invalid use of PL/I . . . . . 229
Unforeseen errors . . . . . 230
Invalid input data. . . . . 230
Compiler or run-time routine malfunction . . . . . 230
System malfunction. . . . . 230
Unidentified routine malfunction . . . . . 230
Storage overlay problems . . . . . 231
Using PL/I compiler listings . . . . . 232
Generating PL/I listings and maps . . . . . 232
Finding information in PL/I listings . . . . . 233
Generating a Language Environment dump of a PL/I routine . . . . . 239
PLIDUMP syntax and options . . . . . 239
PLIDUMP usage notes . . . . . 240
Finding PL/I information in a dump . . . . . 241
Traceback . . . . . 241
Control blocks for active routines. . . . . 243
Control blocks associated with the thread. . . . . 245
PL/I contents of the Language Environment trace table . . . . . 247
Debugging example of PL/I routines . . . . . 247
Subscript range error . . . . . 247
Calling a nonexistent subroutine . . . . . 250
Divide-by-Zero error . . . . . 252

**Chapter 8. Debugging under CICS** . . . . . 257
Accessing debugging information. . . . . 257
Locating Language Environment run-time messages . . . . . 257
Locating the Language Environment traceback. . . . . 257
Locating the Language Environment dump . . . . . 258
Using CICS transaction dump . . . . . 258
Using CICS register and program status word contents . . . . . 259
Using Language Environment abend and reason codes . . . . . 259
Using Language Environment return codes to CICS. . . . . 259
Activating Language Environment feature trace records under CICS. . . . . 260
Ensuring transaction rollback . . . . . 262
Finding data when Language Environment returns a nonzero return code 262
Finding data when Language Environment abends internally . . . . . 262
Finding data when Language Environment Abends from an EXEC CICS command . . . . . 263
Displaying and modifying run-time options with the CLER transaction . . . . 263

**Part 3. Appendixes** . . . . . 265

**Appendix A. Diagnosing Problems with Language Environment** . . . . . 267
Diagnosis Checklist. . . . . 267
Locating the Name of the Failing Routine for a Non-XPLINK Application 268
Searching the IBM Software Support Database . . . . . 273
Preparing Documentation for an Authorized Program Analysis Report (APAR) 274

**Appendix B. Accessibility** . . . . . 277
Using assistive technologies . . . . . 277
Keyboard navigation of the user interface. . . . . 277
z/OS Information. . . . . 277

**Notices** . . . . . 279
Programming Interface Information . . . . . 281

Trademarks. . . . . 281

**Bibliography** . . . . . 283
Language Products Publications . . . . . 283
Related Publications . . . . . 284
Softcopy Publications . . . . . 285

**Index** . . . . . 287

# Figures


1. Options Report Example Produced by Run-Time Option RPTOPTS(ON) . . . 11
2. Storage Report Produced by Run-Time Option RPTSTG(ON) . . . 13
3. Storage Report Produced by RPTSTG(ON) with XPLINK . . . 15
4. Language Environment Condition Token . . . 28
5. The C program CELSAMP . . . 46
6. The C DLL CELDLL. . . . 49
7. Example Dump Using CEE3DMP. . . . 50
8. Upward-Growing (Non-XPLINK) Stack Frame Format . . . 63
9. Downward-Growing (XPLINK) Stack Frame Format . . . 64
10. Common Anchor Area . . . 65
11. Condition Information Block . . . 72
12. Machine State Information Block . . . 74
13. Language Environment Dump of Multiple Enclaves . . . 76
14. Example of Formatted Output from LEDATA Verbexit . . . 85
15. Example Formatted Detailed Heap Segment Report from LEDATA Verbexit . . . 99
16. Example Formatted C/C++ Output from LEDATA Verbexit . . . 105
17. Example Formatted COBOL Output from LEDATA Verbexit . . . 112
18. The CAA Formatted By The CBFORMAT IPCS Command . . . 115
19. Format of the Trace Table Entry . . . 118
20. Trace Table in Dump Output . . . 124
21. __amrc Structure . . . 128
22. __amrc2 Structure . . . 128
23. Example of a Routine Using perror() . . . 133
24. Example of a Routine Using __errno2() . . . 134
25. Example of a Routine Using _EDC_ADD_ERRNO2 . . . 134
26. Sample Output of a Routine Using _EDC_ADD_ERRNO2 . . . 135
27. Writable Static Map Produced by Prelinker . . . 139
28. Location of RENT Static Variable in Storage . . . 140
29. Writable Static Map Produced by Prelinker . . . 140
30. Location of NORENT Static Variable in Storage . . . 141
31. Example Code for Parameter Variable . . . 142
32. Example Code for Parameter Variable . . . 142
33. Partial Storage Offset Listing . . . 143
34. Example Code for Structure Variable . . . 143
35. Example of Aggregate Map . . . 144
36. Writable Static Map Produced by Prelinker . . . 144
37. Example C Routine Using cdump to Generate a Dump . . . 147
38. Fetched module for C routine. . . . 148
39. Example C++ Routine with Protection Exception Generating a Dump . . . 149
40. Template file STACK.C . . . 149
41. Header file STACK.H . . . 150
42. Example Dump from Sample C Routine . . . 151
43. Memory File Control Block . . . 159
44. Registers on Entry to CEE3DMP . . . 160
45. Parameters, Registers, and Variables for Active Routines . . . 160
46. Condition Information for Active Routines . . . 161
47. Sample XPLINK-compiled Program (tranmain) Which Calls a NOXPLINK-compiled Program 162
48. Sample NOXPLINK-compiled Program (trandll) Which Calls an XPLINK-compiled Program 163
49. Example Dump of Calling Between XPLINK and non-XPLINK Programs . . . 164
50. Trace Table with C/C++ Trace Table Entry Types 1 thru 4 . . . 169
51. Trace Table with XPLINK Trace Table Entries 5 and 6. . . . 171
52. C Routine with a Divide-by-Zero Error . . . 172
53. Sections of the Dump from Example C/C++ Routine . . . 173


© Copyright IBM Corp. 1991, 2004

54. Pseudo Assembly Listing . . . 174
55. C/C++ CAA Information in Dump . . . 175
56. Writable Static Map . . . 175
57. Enclave Storage Section of Dump . . . 176
58. C/C++ Example of Calling a Nonexistent Subroutine . . . 176
59. Sections of the Dump from Example C Routine . . . 177
60. Pseudo Assembly Listing . . . 178
61. Writable Static Map . . . 178
62. Enclave Control Blocks and Storage sections in Dump . . . 179
63. C/C++ Example of Calling a Nonexistent Subroutine . . . 179
64. Sections of the Dump from Example C Routine . . . 180
65. Pseudo Assembly Listing . . . 182
66. Writable Static Map . . . 182
67. Enclave Control Blocks and Storage sections in Dump . . . 183
68. IPCS Panel for Entering Data Set Information . . . 184
69. Storage Report Generated by __uheapreport() . . . 187
70. Example of Using the WITH DEBUGGING MODE Clause . . . 192
71. COBOL Program COBDUMP1 Calling COBDUMP2 . . . 194
72. COBOL Program COBDUMP2 Calling the Language Environment Dump Service CEE3DMP 195
73. Sections of the Language Environment Dump Called from COBDUMP2 . . . 196
74. Control Block Information for Active COBOL Routines . . . 198
75. Storage for Active COBOL Programs . . . 199
76. Enclave-Level Data for COBOL Programs . . . 200
77. Process-Level Control Blocks for COBOL Programs . . . 201
78. COBOL Example of Moving a Value Outside an Array Range . . . 202
79. Sections of Language Environment Dump for COBOLX . . . 202
80. COBOL Listing for COBOLX . . . 204
81. COBOL Example of Calling a Nonexistent Subroutine . . . 204
82. Sections of Language Environment Dump for COBOLY . . . 205
83. COBOL Listing for COBOLY . . . 206
84. Parameters, Registers, and Variables for Active Routines Section of Dump for COBOLY 207
85. Main COBOL Program, COBOL Subroutine, and Assembler Routine . . . 208
86. Sections of Language Environment Dump for Program COBOLZ1 . . . 209
87. COBOL Listing for COBOLZ2 . . . 210
88. Listing for ASSEMZ3 . . . 210
89. Variables Section of Language Environment Dump for COBOLZ2 . . . 211
90. Listing for COBOLZ2 . . . 211
91. Variables Section of Language Environment Dump for COBOLZ1 . . . 211
92. Example Program That Calls SDUMP . . . 220
93. Language Environment Dump Generated Using SDUMP . . . 221
94. Sections of the Language Environment Dump . . . 222
95. Example of Calling a Nonexistent Routine . . . 223
96. Sections of the Language Environment Dump Resulting from a Call to a Nonexistent Routine 224
97. FORTRAN Routine with a Divide-by-Zero Error . . . 225
98. Language Environment Dump from Divide-By-Zero FORTRAN Example . . . 226
99. PL/I Routine Compiled with LIST and MAP . . . 233
100. Compiler-Generated Listings from Example PL/I Routine . . . 234
101. Traceback Section of Dump . . . 242
102. Task Traceback Section . . . 243
103. Control Blocks for Active Routines Section of the Dump . . . 244
104. Control Blocks Associated with the Thread Section of the Dump . . . 246
105. Example of Moving a Value Outside an Array Range . . . 248
106. Sections of the Language Environment Dump . . . 249
107. Example of Calling a Nonexistent Subroutine . . . 250
108. Sections of the Language Environment Dump . . . 251
109. PL/I Routine with a Divide-by-Zero Error . . . 252

110. Variables from Routine SAMPLE . . . . . 252
111. Object Code Listing from Example PL/I Routine . . . . . 253
112. Language Environment Dump from Example PL/I Routine . . . . . 254
113. Language Environment Traceback Written to the Transient Data Queue . . . . . 258
114. CICS trace output in the ABBREV format. . . . . 260
115. CICS trace output in the FULL format. . . . . 261
116. Language Environment Non-XPLINK PPA1 and PPA2. . . . . 270
117. Language Environment PPA1 for XPLINK . . . . . 271
118. PPA2: Compile Unit Block (Non-XPLINK) . . . . . 271
119. PPA2 Compile Unit Block for XPLINK. . . . . 272
120. PPA2 Timestamp and Version Information for XPLINK. . . . . 272
121. C PPA1. . . . . 273
122. Nonconforming Entry Point Type with Sample Dump . . . . . 273

# Tables


1. How to Use z/OS Language Environment Publications . . . . . xvi
2. Syntax examples . . . . . xviii
3. Common Error Symptoms, Possible Causes, and Programmer Responses . . . . . 32
4. CEE3DMP Options . . . . . 38
5. TERMTHDACT Suboptions, Level of Information, and Destinations . . . . . 41
6. Condition Handling of 0Cx ABENDS . . . . . 44
7. List of CAA Fields . . . . . 66
8. Language Environment Control Blocks Which Can Be Individually Formatted . . . . . 115
9. LE=1 Entry Records . . . . . 119
10. LE=2 Entry Records . . . . . 119
11. Format of the Mutex/CV/Latch Records . . . . . 122
12. LE=8 Entry Records . . . . . 122
13. __last_op Values and Diagnosis Information . . . . . 130
14. Contents of Listing and Associated Compiler Options . . . . . 135
15. C Compiler Listings . . . . . 136
16. C++ Compiler Listings . . . . . 137
17. C/C++ IPA Link Step Listings . . . . . 137
18. Finding the WSA base address . . . . . 138
19. Compiler-generated COBOL Listings and Their Contents. . . . . 193
20. Compiler-generated FORTRAN Listings and Their Contents . . . . . 215
21. Compiler-generated PL/I Listings and Their Contents . . . . . 232
22. Typical Comments in a PL/I Static Storage Listing . . . . . 235
23. Comments in a PL/I Object Code Listing . . . . . 237
24. PL/I Mnemonics. . . . . 238
25. Finding Data When Language Environment Returns a Nonzero Return Code . . . . . 262
26. Finding Data When Language Environment Abends Internally . . . . . 262
27. Finding Data When Language Environment Abends from an EXEC CICS Command . . . . . 263
28. Problem Resolution Documentation Requirements . . . . . 275


© Copyright IBM Corp. 1991, 2004

# About this document

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

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

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

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

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

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

Language Environment supports, but is not required for, an interactive debug tool for debugging applications in your native z/OS environment. The IBM interactive

© Copyright IBM Corp. 1991, 2004

Debug Tool is available with z/OS, or with the latest releases of the C/C++, COBOL, PL/I and VisualAge for Java compiler products.

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 283.

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

| To ... | Use ... |
|---|---|
| Evaluate Language Environment | *z/OS Language Environment Concepts Guide* |
| Plan for Language Environment | *z/OS Language Environment Concepts Guide*<br>*z/OS Language Environment Run-Time Application Migration Guide* |
| Install Language Environment | *z/OS Program Directory* |
| Customize Language Environment | *z/OS Language Environment Customization* |
| Understand Language Environment program models and concepts | *z/OS Language Environment Concepts Guide*<br>*z/OS Language Environment Programming Guide* |
| Find syntax for Language Environment run-time options and callable services | *z/OS Language Environment Programming Reference* |
| Develop applications that run with Language Environment | *z/OS Language Environment Programming Guide* and your language programming guide |
| Debug applications that run with Language Environment, diagnose problems with Language Environment | *z/OS Language Environment Debugging Guide* |
| Get details on run-time messages | *z/OS Language Environment Run-Time Messages* |

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

| To ... | Use ... |
|---|---|
| Develop interlanguage communication (ILC) applications | *z/OS Language Environment Writing Interlanguage Communication Applications* and your language programming guide |
| Migrate applications to Language Environment | *z/OS Language Environment Run-Time Application Migration Guide* 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*

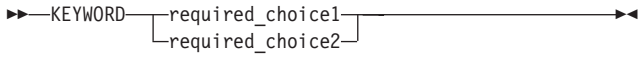
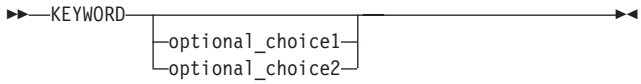
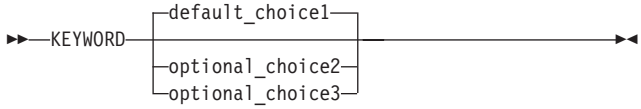
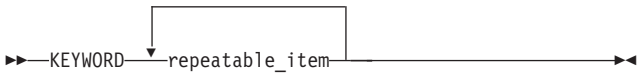
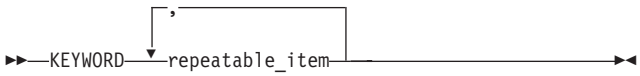
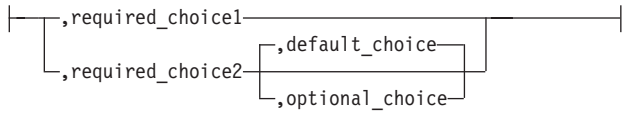
| Item | Syntax example |
|---|---|
| Required item.<br><br>Required items appear on the main path of the horizontal line. You must specify these items. | |
| Required choice.<br><br>A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack. |  |
| Optional item.<br><br>Optional items appear below the main path of the horizontal line. | |
| Optional choice.<br><br>An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack. |  |
| Default.<br><br>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. |  |

*Table 2. Syntax examples (continued)*

| Item | Syntax example |
|---|---|
| Variable.<br><br>Variables appear in lowercase italics. They represent names or values. | |
| Repeatable item.<br><br>An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.<br><br>A character within the arrow means you must separate repeated items with that character.<br><br>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. |  <br>  |
| Fragment.<br><br>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. | <br>**fragment:**<br>  |

## 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. At the end of this book is a list of all Language Environment and HLL messages.

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

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

## Where to find more information

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

## Accessing z/OS licensed documents on the Internet

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

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

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code. [1]

To obtain your IBM Resource Link user ID and password, log on to:

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

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

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

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

Printed licensed documents are not available from IBM.

You can use the PDF format on either **z/OS Licensed Product Library CD-ROM** or IBM Resource Link to print licensed documents.

## Using LookAt to look up message explanations

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

You can use LookAt from the following locations to find IBM message explanations for z/OS elements and features, z/VM®, and VSE:

- The Internet. You can access IBM message explanations directly from the LookAt Web site at http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e systems to access IBM message explanations, using LookAt from a TSO/E command line (for example, TSO/E prompt, ISPF, or z/OS UNIX® System Services running OMVS).
- Your Windows® workstation. You can install code to access IBM message explanations on the *z/OS Collection* (SK3T-4269), using LookAt from a Windows DOS command line.
- Your wireless handheld device. You can use the LookAt Mobile Edition with a handheld device that has wireless access and an Internet browser (for example, Internet Explorer for Pocket PCs, Blazer, or Eudora for Palm OS, or Opera for Linux handheld devices). Link to the LookAt Mobile Edition from the LookAt Web site.

You can obtain code to install LookAt on your host system or Windows workstation from a disk on your *z/OS Collection* (SK3T-4269), or from the LookAt Web site (click **Download**, and select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

1. z/OS.e™ customers received a Memo to Licensees, (GI10-0684) that includes this key code.

## Information updates on the web

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

`http://www.s390.ibm.com:80/bookmgr-cgi/bookmgr.cmd/BOOKS/ZIDOCMST/CCONTENTS`

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

# Summary of Changes

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

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

The following summarizes the changes to that information.

**New Information**

- The heap pools algorithm now allows for the definition of one to twelve heap pools. For more information, see "Understanding the HEAPPOOLS trace output" on page 83, and "Language Environment Storage Report with HeapPools Statistics" on page 185.

**Changed Information**

- The information about allocating stack storage has been updated. For more information, see "Allocating stack storage" on page 20.

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

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

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

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

The following summarizes the changes to that information.

**New Information**

- Information is added to indicate this document supports z/OS.e.
- Additonal Common Anchor Area (CAA) fields have been added. For more information, see Table 7 on page 66.

**Changed Information**

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

© Copyright IBM Corp. 1991, 2004

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

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

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

The following summarizes the changes to that information.

**New Information**

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

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

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

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

The following summarizes the changes to that information.

**New Information**

- A new trace function is available which allows the user to format individual control blocks in a dump, rather than having to create the full LEDATA output which contains many formatted control blocks. See “Formatting individual control blocks” on page 114.
- “Understanding the trace table entry (TTE)” on page 118 has been enhanced to provide significant additional information about the content and meaning of trace table entries.
- The HEAPCHK run-time option has a new suboption (*call-level*) which will aid in identifying the cause of heap storage leaks. See “Diagnosing storage leak problems” on page 103.

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.

© Copyright IBM Corp. 1991, 2004

# Chapter 1. Preparing your routine for debugging

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

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

## Setting compiler options

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

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

## C and C++ compiler options

When using C, set the TEST(ALL) suboption, which is equivalent to TEST(LINE,BLOCK,PATH,SYM,HOOK). For 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 *z/OS C/C++ User's Guide*.

**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.

© Copyright IBM Corp. 1991, 2004

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 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 C, or z/OS 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 C compile, cross reference listing with reference, definition, and modification information.<br><br>For 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 C/C++ User's Guide*.

## IPA compile step sub-options

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

### ATTRIBUTE | NOATTRIBUTE

Indicates whether the compiler generates information in the IPA object file that will be used in the IPA Link step if you specify the ATTR or XREF option on that step.

The difference between specifying IPA(ATTR) and specifying ATTR, or XREF, is that IPA(ATTR) does not cause the compiler to generate a Cross Reference listing section after IPA Compile step source analysis is complete. It also does not cause the compiler to generate a Storage Offset or External Symbol Cross Reference listing section during IPA Compile step code generation.

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

**GONUMBER | NOGONUMBER**

Indicates whether the compiler saves information about source file line numbers in the IPA object file. The difference between specifying IPA(GONUMBER) and GONUMBER is that IPA(GONUMBER) does not cause GONUMBER tables to be built during IPA Compile step code generation. If the compiler does not build GONUMBER tables, the size of the object module is smaller.

The default is IPA(NOGONUMBER). The abbreviations are IPA(GONUM|NOGONUM). If you specify the GONUMBER or LIST option, it overrides the IPA(NOGONUMBER) option.

**LIST | NOLIST**

Indicates whether the compiler saves information about source line numbers in the IPA object file. The difference between specifying IPA(LIST) and LIST is that IPA(LIST) does not cause a Pseudo-Assembly listing to be generated during IPA Compile step code generation.

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

**XREF | NOXREF**

Indicates whether the compiler generates information in the IPA object file that will be used in the IPA Link step if you specify ATTR or XREF on that step.

The difference between specifying IPA(XREF) and specifying ATTR or XREF is that IPA(XREF) does not cause the compiler to generate a Cross Reference listing section after IPA Compile step source analysis is complete. It also does not cause the compiler to generate a Storage Offset or External Symbol Cross Reference listing section during IPA Compile step code generation.

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

## IPA link step sub-options

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

**DUP | NODUP**

Indicates whether the IPA Link step writes a message and a list of duplicate symbols to the console.

The default is IPA(DUP).

**ER | NOER**

Indicates whether the IPA Link step writes a message and a list of unresolved symbols to the console.

The default is IPA(NOER).

**MAP | NOMAP**

Specifies that the IPA Link step will produce a listing. The listing contains a Prolog and the following sections:
- Object File Map
- Source File Map
- Compiler Options Map
- Global Symbols Map
- Partition Map for each partition

The default is IPA(NOMAP).

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

## COBOL compiler options

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

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

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

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

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

## Fortran compiler options

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

**FIPS** Specifies whether standard language flagging is to be performed. This is valuable if you want to write a program conforming to FORTRAN 77.

**FLAG** Specifies the level of diagnostic messages to be written. **I** (Information), **E** (Error), **W** (Warning), or **S** (Severe). You can also use FLAG to suppress messages that are below a specified level. This is useful if you want to suppress information messages, for example.

**GOSTMT** Specifies that statement numbers are included in the run-time messages and in the Language Environment dump.

**ICA** Specifies whether intercompilation analysis is to be performed, specifies the files containing intercompilation analysis information to be used or updated, and controls output from intercompilation analysis. Specify ICA when you have a group of programs and subprograms that you want to process together and you need to know if there are any conflicting external references, mismatched commons, and so on.

**LIST** Specifies whether the object module list is to be written. The LIST option lets you see the pseudo-assembly language code that is similar to what is actually generated.

**MAP** Specifies whether a table of source program variable names, named constants, and statement labels and their displacements is to be produced.

**OPTIMIZE** Specifies the optimizing level to be used during compilation. If you are debugging your program, it is advisable to use NOOPTIMIZE.

**SDUMP** Specifies whether dump information is to be generated.

**SOURCE** Specifies whether a source listing is to be produced.

**SRCFLG** Controls insertion of error messages in the source listing. SRCFLG allows you to view error messages after the initial line of each source statement that caused the error, rather than at the end of the listing.

**SXM™** Formats SREF or MAP listing output to a 72-character width.

**SYM** Invokes the production of SYM cards in the object text file. SYM cards contain location information for variables within a Fortran program.

**TERMINAL** Specifies whether error messages and compiler diagnostics are to be written on the SYSTERM data set and whether a summary of messages for all the compilations is to be written at the end of the listing.

**TEST** Specifies whether to override any optimization level above OPTIMIZE(0). This option adds run-time overhead.

**TRMFLG** Specifies whether to display the initial line of source statements in error and their associated error messages at the terminal.

**XREF** Creates a cross-reference listing.

**VECTOR** Specifies whether to invoke the vectorization process. A vectorization report provides detailed information about the vectorization process.

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

## PL/I compiler options

When using PL/I, specify the TEST compiler option to control the level of testing capability that are generated as part of the object code. Suboptions of the TEST option such as SYM, BLOCK, STMT, and PATH control the location of test hooks

and specify whether or not a symbol table is generated. For more information about TEST, its suboptions, and the placement of test hooks, see *PL/I for MVS & VM Programming Guide*.

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

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

**AGGREGATE** Specifies that a layout for arrays and major structures appears in the listing.

**ESD** Includes the external symbol dictionary in the listing.

**GONUMBER / GOSTMT** Tells the compiler to produce additional information specifying that line numbers from the source routine can be included in run-time messages and in the Language Environment dump.

**INTERRUPT** Specifies that users can establish an ATTENTION ON-unit that gains control when an attention interrupt occurs.

**LIST** Produces a listing of the assembler expansion of source code and global tables, literal pools, information about working storage, and size of routine's working storage.

**LMESSAGE** Tells the compiler to produce run-time messages in a long form. If the cause of a run-time malfunction is a programmer's understanding of language semantics, specifying LMESSAGE could better explain warnings or other information generated by the compiler.

**MAP** Tells the compiler to produce tables showing how the variables are mapped in the static internal control section and in the stack frames, thus enabling static internal and automatic variables to be found in the Language Environment dump. If LIST is also specified, the MAP option also produces tables showing constants, control blocks, and INITIAL variable values.

**OFFSET** Specifies that the compiler prints a table of statement or line numbers for each procedure with their offset addresses relative to the primary entry point of the procedure.

**SOURCE** Specifies that the compiler includes a listing of the source routine in the listing.

**STORAGE** Includes a table of the main storage requirements for the object module in the listing.

**TERMINAL** Specifies what parts of the compiler listing produced during compilation are directed to the terminal.

**TEST** Specifies the level of testing capability that is generated as part of the object code. TEST also controls the location of test hooks and whether or not the symbol table is generated. Because the TEST option increases the size of the object code and can affect performance, limit the number and placement of hooks.

**XREF and ATTRIBUTES** Creates a sorted cross-reference listing with attributes.

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

## VisualAge PL/I compiler options

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

**AGGREGATE** Specifies that a layout for arrays and major structures appears in the listing.

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

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

## Using Language Environment run-time options

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

The following Language Environment run-time options affect debugging:

| | |
|---|---|
| **ABPERC** | Specifies that the indicated abend code bypasses the condition handler. |
| **ABTERMENC** | Specifies enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater. |
| **CHECK** | Determines whether run-time checking is performed. |
| **NODEBUG** | Controls the COBOL USE FOR DEBUGGING declarative. |
| **DEPTHCONDLMT** | Specifies the limit for the depth of nested synchronous conditions in user-written condition handlers. (Asynchronous signals do not affect DEPTHCONDLMT.) |
| **ERRCOUNT** | Specifies the number of synchronous conditions of severity 2 or greater tolerated. (Asynchronous signals do not affect ERRCOUNT.) |
| **HEAPCHK** | Determines whether additional heap check tests are performed. |
| **INFOMSGFILTER** | Filters user specified informational messages from the MSGFILE. **Note:** Affects only those messages generated by Language Environment® and any routine that calls CEEMSG. Other routines that write to the message file, such as CEEMOUT, do not have a filtering option. |
| **INTERRUPT** | Causes Language Environment to recognize attention interrupts. |
| **MSGFILE** | Specifies the ddname of the Language Environment message file. |

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

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

## Determining run-time options in effect

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

Figure 1 on page 11 shows a sample options report.

```
Options Report for Enclave main 01/15/03 12:33:57 PM
Language Environment V01 R05.00

LAST WHERE SET                OPTION
------------------------------------------------------------------------------
Installation default            ABPERC(NONE)
Installation default            ABTERMENC(ABEND)
Installation default          NOAIXBLD
Programmer default              ALL31(ON)
Assembler user exit             ANYHEAP(32768,16384,ANYWHERE,FREE)
Installation default          NOAUTOTASK
Assembler user exit             BELOWHEAP(8192,8192,FREE)
Installation default            CBLOPTS(ON)
Installation default            CBLPSHPOP(ON)
Installation default            CBLQDA(OFF)
Installation default            CHECK(ON)
Installation default            COUNTRY(US)
Installation default          NODEBUG
Installation default            DEPTHCONDLMT(10)
Installation default            ENVAR("")
Installation default            ERRCOUNT(0)
Installation default            ERRUNIT(6)
Installation default            FILEHIST
Installation default            FILETAG(NOAUTOCVT,NOAUTOTAG)
Default setting               NOFLOW
Assembler user exit             HEAP(49152,16384,ANYWHERE,KEEP,8192,4096)
Installation default            HEAPCHK(OFF,1,0,0,0)
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            INFOMSGFILTER(OFF,,,,)
Installation default            INQPCOPN
Installation default            INTERRUPT(OFF)
Installation default            LIBRARY(SYSCEE)
Programmer default              LIBSTACK(4096,4096,FREE)
Installation default            MSGFILE(SYSOUT,FBA,121,0,NOENQ)
Installation default            MSGQ(15)
Installation default            NATLANG(ENU)
Mapped                          NONIPTSTACK(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)
Invocation command              RPTOPTS(ON)
Invocation command              RPTSTG(ON)
Installation default          NORTEREUS
Installation default            RTLS(OFF)
Installation default          NOSIMVRD
Programmer default              STACK(4096,4096,ANYWHERE,FREE,524288,131072)
Programmer default              STORAGE(NONE,NONE,NONE,1024)
Installation default            TERMTHDACT(TRACE,,96)
Installation default          NOTEST(ALL,"*","PROMPT","INSPPREF")
Installation default            THREADHEAP(4096,4096,ANYWHERE,KEEP)
Programmer default              THREADSTACK(ON,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            VCTRSAVE(OFF)
Installation default            VERSION()
Installation default            XPLINK(OFF)
Installation default            XUFLOW(AUTO)
```

*Figure 1. Options Report Example Produced by Run-Time Option RPTOPTS(ON)*

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

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

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

- TRAP(ON|OFF)
- TERMTHDACT(QUIET|MSG|TRACE|DUMP|UAONLY|UATRACE|UADUMP|UAIMM)
- RPTOPTS(ON|OFF)
- RPTSTG(ON|OFF)
- ALL31(ON|OFF)
- CBLPSHPOP(ON|OFF)

For more information about the CICS CLER transaction, see “Displaying and modifying run-time options with the CLER transaction” on page 263.

## Controlling storage allocation

The following run-time options control storage allocation:

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

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

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

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

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

```
Storage Report for Enclave main 01/15/03 12:33:57 PM
Language Environment V01 R05.00

    STACK statistics:
      Initial size:                                        4096
      Increment size:                                      4096
      Maximum used by all concurrent threads:              7504
      Largest used by any thread:                          7504
      Number of segments allocated:                           2
      Number of segments freed:                               0
    THREADSTACK statistics:
      Initial size:                                        4096
      Increment size:                                      4096
      Maximum used by all concurrent threads:              3336
      Largest used by any thread:                          3336
      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):       28312
      Successful Get Heap requests:                         242
      Successful Free Heap requests:                        213
      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):      105744
      Successful Get Heap requests:                          37
      Successful Free Heap requests:                         20
      Number of segments allocated:                           6
      Number of segments freed:                               5             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
           1
```

*Figure 2. Storage Report Produced by Run-Time Option RPTSTG(ON) (Part 2 of 2)*

```
Storage Report for Enclave main 01/15/03 1:16:48 PM
Language Environment V01 R05.00

    STACK statistics:
      Initial size:                                        4096
      Increment size:                                      4096
      Maximum used by all concurrent threads:              4256
      Largest used by any thread:                          4256
      Number of segments allocated:                           2
      Number of segments freed:                               0
    THREADSTACK statistics:
      Initial size:                                        4096
      Increment size:                                      4096
      Maximum used by all concurrent threads:             11688
      Largest used by any thread:                          2584
      Number of segments allocated:                           6
      Number of segments freed:                               0
    XPLINK STACK statistics:
      Initial size:                                      524288
      Increment size:                                    131072
      Largest used by any thread:                          4416
      Number of segments allocated:                           1
      Number of segments freed:                               0
    XPLINK THREADSTACK statistics:
      Initial size:                                      131072
      Increment size:                                    131072
      Largest used by any thread:                          2944
      Number of segments allocated:                           6
      Number of segments freed:                               0
    LIBSTACK statistics:
      Initial size:                                        4096
      Increment size:                                      4096
      Maximum used by all concurrent threads:                 0
      Largest used by any thread:                             0
      Number of segments allocated:                           0
      Number of segments freed:                               0
    THREADHEAP statistics:
      Initial size:                                        4096
      Increment size:                                      4096
      Maximum used by all concurrent threads:                 0
      Largest used by any thread:                             0
      Successful Get Heap requests:                           0
      Successful Free Heap requests:                          0
      Number of segments allocated:                           0
      Number of segments freed:                               0
    HEAP statistics:
      Initial size:                                       32768
      Increment size:                                     32768
      Total heap storage used (sugg. initial size):       56544
      Successful Get Heap requests:                          29
      Successful Free Heap requests:                         13
      Number of segments allocated:                           2
      Number of segments freed:                               0
```

*Figure 3. Storage Report Produced by RPTSTG(ON) with XPLINK (Part 1 of 4)*

```
HEAP24 statistics:
  Initial size:                                        8192
  Increment size:                                      4096
  Total heap storage used (sugg. initial size):           0
  Successful Get Heap requests:                           0
  Successful Free Heap requests:                          0
  Number of segments allocated:                           0
  Number of segments freed:                               0
ANYHEAP statistics:
  Initial size:                                       16384
  Increment size:                                      8192
  Total heap storage used (sugg. initial size):     1033472
  Successful Get Heap requests:                          42
  Successful Free Heap requests:                         20
  Number of segments allocated:                          10
  Number of segments freed:                               8
BELOWHEAP statistics:
  Initial size:                                        8192
  Increment size:                                      4096
  Total heap storage used (sugg. initial size):           0
  Successful Get Heap requests:                           0
  Successful Free Heap requests:                          0
  Number of segments allocated:                           0
  Number of segments freed:                               0
Additional Heap statistics:
  Successful Create Heap requests:                        1
  Successful Discard Heap requests:                       1
  Total heap storage used:                             4912
  Successful Get Heap requests:                           3
  Successful Free Heap requests:                          3
  Number of segments allocated:                           2
  Number of segments freed:                               2
HeapPools Statistics:
  Pool  1 size:     8
    Successful Get Heap requests:     1-    8              8
  Pool  2 size:    32
    Successful Get Heap requests:     9-   16              3
    Successful Get Heap requests:    17-   24              5
    Successful Get Heap requests:    25-   32              3
  Pool  3 size:   128
    Successful Get Heap requests:    33-   40              3
    Successful Get Heap requests:    41-   48              3
    Successful Get Heap requests:    49-   56              3
    Successful Get Heap requests:    57-   64              4
    Successful Get Heap requests:    65-   72              3
    Successful Get Heap requests:    73-   80              4
    Successful Get Heap requests:    81-   88              5
    Successful Get Heap requests:    89-   96              4
    Successful Get Heap requests:    97-  104              4
    Successful Get Heap requests:   113-  120              5
    Successful Get Heap requests:   121-  128              4
```

*Figure 3. Storage Report Produced by RPTSTG(ON) with XPLINK (Part 2 of 4)*

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

*Figure 3. Storage Report Produced by RPTSTG(ON) with XPLINK (Part 3 of 4)*

```
      Pool  6 size:  2048
        Successful Get Heap requests:  1169- 1176               1
        Successful Get Heap requests:  1185- 1192               1
        Successful Get Heap requests:  1217- 1224               2
        Successful Get Heap requests:  1257- 1264               1
        Successful Get Heap requests:  1377- 1384               1
        Successful Get Heap requests:  1401- 1408               1
        Successful Get Heap requests:  1521- 1528               1
        Successful Get Heap requests:  1537- 1544               1
        Successful Get Heap requests:  1545- 1552               1
        Successful Get Heap requests:  1569- 1576               1
        Successful Get Heap requests:  1665- 1672               1
        Successful Get Heap requests:  1761- 1768               1
        Successful Get Heap requests:  1785- 1792               1
        Successful Get Heap requests:  1929- 1936               1
        Successful Get Heap requests:  1937- 1944               1
        Successful Get Heap requests:  1953- 1960               1
      Requests greater than the largest cell size:            19
    HeapPools Summary:
      Cell  Extent   Cells Per  Extents    Maximum     Cells In
      Size  Percent  Extent     Allocated  Cells Used  Use
      ------------------------------------------------------------
          8   10         204           1           6           0
         32   10          81           1           4           1
        128   10          24           1          13           4
        256   10          12           2          19           4
       1024   10           4           4          13          12
       2048   10           4           1           3           2
      ------------------------------------------------------------
      Suggested Percentages for current Cell Sizes:
        HEAPP(ON,8,1,32,1,128,6,256,16,1024,41,2048,19,0)
      Suggested Cell Sizes:
        HEAPP(ON,
                88,,168,,232,,328,,416,,512,,
                648,,1264,,1792,,2232,,2712,,2992,)
    Largest number of threads concurrently active:             6
End of Storage Report
```

*Figure 3. Storage Report Produced by RPTSTG(ON) with XPLINK (Part 4 of 4)*

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

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

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

## Stack storage statistics

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

### STACK, THREADSTACK, and LIBSTACK 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 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.

### HEAP, HEAP24, THREADHEAP, 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.

### THREADHEAP statistics

- 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.

### HEAP, HEAP24, ANYHEAP, BELOWHEAP, and additional heap Statistics

- Total heap storage used—the largest total amount used by the enclave at any one time.

### HEAP, HEAP24, THREADHEAP, ANYHEAP, BELOWHEAP, and additional heap statistics

- 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.

## HeapPools storage statistics

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

# Modifying condition handling behavior

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

You can modify condition handling behavior in the following ways:

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

## Language Environment callable services

You can use these callable services to modify condition handling:

| | |
|---|---|
| **CEE3ABD** | Terminates an enclave using an abend. |
| **CEEMRCE** | Moves the resume cursor to an explicit location where resumption is to occur after a condition has been handled. |
| **CEEMRCR** | Moves the resume cursor relative to the current position of the handle cursor. |
| **CEE3CIB** | Returns a pointer to a condition information block (CIB) associated with a given condition token. The CIB contains detailed information about the condition. |

**CEE3GRO** Returns the offset of the location within the most current Language Environment-conforming routine where a condition occurred.

**CEE3SPM** Specifies the settings of the routine mask. The routine mask controls:
- Fixed overflow
- Decimal overflow
- Exponent underflow
- Significance

You can use CEE3SPM to modify Language Environment hardware conditions. Because such modifications can affect the behavior of your routine, however, you should be careful when doing so.

**CEE3SRP** Sets a resume point within user application code to resume from a Language Environment user condition handler.

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

## Language Environment run-time options

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

**ABPERC** Specifies a system- or user-specified abend code that percolates without further action while the Language Environment condition handler is enabled. Normal condition handling activities are performed for everything except the specified abend code. System abends are specified as S**hhh**, where **hhh** is a hexadecimal system abend code.

User abends are specified as U`dddd`, where `dddd` is a decimal user abend code. Any other 4-character EBCDIC string, such as NONE, that is not of the form S**hhh** can also be specified as a user-specified abend code. You can specify only one abend code with this option. This option assumes the use of TRAP(ON). ABPERC is not supported in CICS.

Language Environment ignores ABPERC(0Cx). No abend is percolated and Language Environment condition handling semantics are in effect.

**CHECK** Specifies that checking errors within an application are detected. The Language Environment-conforming languages can define error checking differently.

**DEPTHCONDLMT** Limits the extent to which synchronous conditions can be nested in a user-written condition handler. (Asynchronous signals do not affect DEPTHCONDLMT.) For example, if you specify 5, the initial condition and four nested conditions are processed. If the limit is exceeded, the application terminates with abend code 4091 and reason code 21 (X'15').

**ERRCOUNT** Specifies the number of synchronous conditions of severity 2, 3, and 4 that are tolerated before the enclave terminates abnormally. (Asynchronous signals do not affect ERRCOUNT.) If you specify 0 an unlimited number of conditions is tolerated.

**INTERRUPT** Causes attentions recognized by the host operating system to be passed to and recognized by Language Environment after the environment has been initialized.

**TERMTHDACT** Sets the level of information that is produced when a condition of severity 2 or greater remains unhandled within the enclave. There are five possible parameter settings for different levels of information:
- QUIET for no information
- MSG for message only
- TRACE for message and a traceback
- DUMP for message, traceback, and Language Environment dump
- UAONLY for message and a system dump of the user address space
- UATRACE for message, Language Environment dump with traceback information only, and a system dump of the user address space
- UADUMP for message, traceback, Language Environment dump, and system dump
- UAIMM for a system dump of the user address space of the original abend or program interrupt prior to the Language Environment condition manager processing the condition.

**TRAP(ON)** Fully enables the Language Environment condition handler. This causes the Language Environment condition handler to intercept error conditions and routine interrupts.

When TRAP(ON, NOSPIE) is specified, Language Environment handles all program interrupts and abends through an ESTAE. Use this feature when you do not want Language Environment to issue an ESPIE macro.

During normal operation, you should use TRAP(ON) when running your applications.

**TRAP(OFF)** Disables the Language Environment condition handler from handling abends and program checks/interrupts. ESPIE is not issued with TRAP(OFF), it is still possible to invoke the condition handler through the CEESGL callable service and pass conditions to registered user-written condition handlers.

Specify TRAP(OFF) when you do not want Language Environment to issue an ESTAE or an ESPIE.

When TRAP(OFF), TRAP(OFF,SPIE), or TRAP(OFF,NOSPIE) is specified and either a program interrupt or abend occurs, the user exit for termination is ignored.

TRAP(OFF) can cause several unexpected side effects. For further information, see the TRAP run-time option in *z/OS Language Environment Programming Reference*.

**USRHDLR** Specifies the behavior of two user-written condition handlers. The first handler specified will be registered at stack frame 0. The second handler specified will be registered before any other user-written condition handlers, once the handler is enabled by a condition.

When you specify USRHDLR(lastname,supername), lastname gets control at stack frame 0. Supername will get get control first, before any user-written condition handlers but after supername has gone through the enablement phase, when a condition occurs.

**XUFLOW** Specifies whether an exponent underflow causes a routine interrupt.

## Customizing condition handlers

User-written condition handlers permit you to customize condition handling for certain conditions. You can register a user-written condition handler for the current

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

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

- Resume
- Percolate
- Promote
- Fix-up and resume

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

## Invoking the assembler user exit

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

- Enclave initialization
- Enclave termination
- Process termination

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

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

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

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

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

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

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

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

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

## Establishing enclave termination behavior for unhandled conditions

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

- The assembler user exit (see "Invoking the assembler user exit" on page 24 and *z/OS Language Environment Programming Guide*)
- POSIX signal default action (see *z/OS Language Environment Programming Guide*)
- The ABTERMENC run-time option (discussed below)

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

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

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

## Using messages in your routine

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

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

When multiple Language Environment environments are running in the same address space and the same MSGFILE ddname is specified, writing contention can

occur. To avoid contention, use the MSGFILE suboption ENQ. ENQ tells Language Environment to perform serialization around writes to the MSGFILE ddname specified which eliminates writing contention. Writing contention can also be eliminated by specifying unique MSGFILE ddnames.

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

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

### C/C++

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

### COBOL

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

### Fortran

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

### PL/I

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

## Using condition information

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

You can use this condition information in two primary ways:

- To specify the feedback code parameter when calling Language Environment services (see “Using the feedback code parameter”).
- To code a symbolic feedback code in a user-written condition handler (see “Using the symbolic feedback code” on page 28).

## 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 and OS/390, 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 *z/OS Language Environment 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 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.

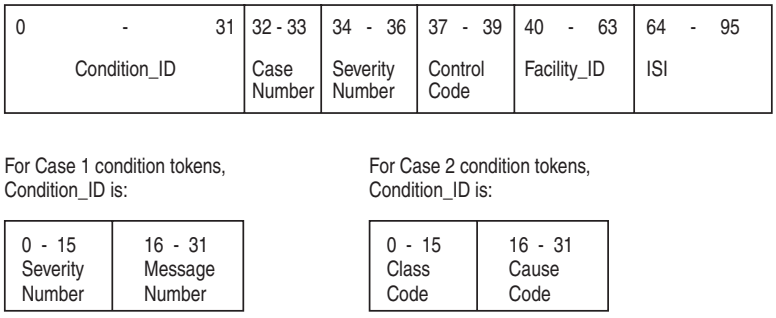



*Figure 4. Language Environment Condition Token*

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

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

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

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

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

## Using the symbolic feedback code

The symbolic feedback code represents the first 8 bytes of a 12-byte condition token. You can think of the symbolic feedback code as the nickname for a

condition. As such, the symbolic feedback code can be used in user-written condition handlers to screen for a given condition, even if it occurs at different locations in an application.

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

# Chapter 2. Classifying errors

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

## Identifying problems in routines

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

## Language Environment module names

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

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

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

## Common errors in routines

These common errors have simple solutions:

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

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

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

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

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

© Copyright IBM Corp. 1991, 2004

*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<br><br>b) The assembler user exit requested an abend for an unhandled condition of severity ≥2 | See the Language Environment abend codes in *z/OS Language Environment Run-Time Messages*. 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<br><br>b) An unhandled software-raised condition occurred and ABTERMENC(ABEND) was in effect<br><br>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 *z/OS Language Environment Run-Time Messages*. |
| System abend with TRAP(OFF) | Cause depends on type of malfunction | Respond appropriately. Refer to the messages and codes book of the operating system. |
| System abend with TRAP(ON) | System-detected error | Refer to the messages and codes book of the operating system. |
| No response (wait/loop) | Application logic failure | Check routine logic. Ensure ERRCOUNT and DEPTHCONDLMT run-time options are set to a nonzero value. |
| Unexpected message (message received was not from most recent service) | Condition caused by something related to current service | Generate a traceback using CEE3DMP. |
| Incorrect output | Incorrect file definitions, storage overlay, incorrect routine mask setting, references to uninitialized variables, data input errors, or application routine logic error | Correct the appropriate parameters. |
| No output | Incorrect ddname, file definitions, or message file setting | Correct the appropriate parameters. |
| Nonzero return code from enclave | Unhandled condition of severity 2, 3, or 4, or the return code was issued by the application routine | Check the Language Environment message file for run-time message. |
| Unexpected output | Conflicting library module names | Refer to the name conflict resolution steps outlined in *z/OS Language Environment Programming Guide*. |

## Interpreting run-time messages

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

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

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

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

In the following example Language Environment message:

```
CEE3206S The system detected a specification exception.
```

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

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

## Message prefix

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

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

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

## Message number

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

## Severity code

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

## Message text

The message text provides a brief explanation of the condition.

# Understanding abend codes

Under Language Environment, abnormal terminations generate abend codes. There are two types of abend codes: 1) user (Language Environment and user-specified) abends and 2) system abends. User abends follow the format of U*dddd*, where *dddd* is a decimal user abend code. System abends follow the format of S*hhh*, 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 25, 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.

User abends, such as Language Environment 4`xxx` abends or abends raised by a call to the CEE3ABD service, can cause the generation of a system dump. Although system dumps are sometimes required for debugging complex error situations, it is usually better to generate a Language Environment-formatted dump. To request a Language Environment dump whenever an unhandled condition is raised, specify both TRAP(ON) and TERMTHDACT(DUMP) run-time options.

Your routine can also produce a Language Environment dump at any time by calling the CEE3DMP callable service (see “Generating a Language Environment dump with CEE3DMP” on page 37). The TRAP(ON) run-time option causes the Language Environment condition handler to attempt to handle the system abend. For a detailed explanation of the run-time options and callable services discussed in this section, see *z/OS Language Environment Programming Reference*.

## System abends

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

When a system abend occurs, the operating system can generate a system dump. System dumps are written to ddname SYSMDUMP, SYSABEND, or SYSUDUMP.

System dumps show the memory state at the time of the condition. See “Generating a system dump” on page 77 for more information about system dumps.

# Chapter 3. Using Language Environment debugging facilities

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

## Debug tool

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

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

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

## Language Environment dump service, CEE3DMP

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

There are three ways to invoke the Language Environment dump service:
- CEE3DMP callable service
- TERMTHDACT run-time option
- HLL-specific functions

## Generating a Language Environment dump with CEE3DMP

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.

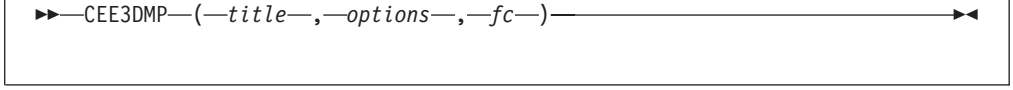
© Copyright IBM Corp. 1991, 2004

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 (/), and the directory is writable
3. the directory found in environment variable *TMPDIR* (an environment variable that indicates the location of a temporary directory if it is not /tmp)
4. the **/tmp** directory.

The syntax for CEE3DMP is:

**Syntax**




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

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

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

Following is a list of CEE3DMP options and related information:

*Table 4. CEE3DMP Options*

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

*Table 4. CEE3DMP Options (continued)*

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

*Table 4. CEE3DMP Options (continued)*

| Dump Options | Abbreviation | Action Taken |
|---|---|---|
| STACKFRAME(n) | SF(n) | Dumps a fixed number of stack frames, indicated by *n*, 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 45.

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

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

*Table 5. TERMTHDACT Suboptions, Level of Information, and Destinations*

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

*Table 5. TERMTHDACT Suboptions, Level of Information, and Destinations (continued)*

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

The TRACE and UATRACE suboptions of TERMTHDACT use these dump options:
- ENCLAVE(ALL)
- NOENTRY
- CONDITION
- TRACEBACK
- THREAD(ALL)
- NOBLOCKS
- NOSTORAGE
- VARIABLES
- FILES
- STACKFRAME(ALL)
- PAGESIZE(60)
- FNAME(CEEDUMP)
- GENOPTS
- REGSTOR(96)

The DUMP and UADUMP suboptions of TERMTHDACT use these dump options:
- ENCLAVE(ALL)
- NOENTRY
- CONDITION
- TRACEBACK
- THREAD(CURRENT)

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

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

## Considerations for setting TERMTHDACT options

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

- COBOL Considerations
  - The following TERMTHDACT suboptions for COBOL are recommended, UAONLY, UATRACE, and UADUMP. A system dump will always be generated when one of these suboptions is specified.
- PL/I Considerations
  - After a normal return from a PL/I ERROR ON-unit, or from a PL/I FINISH ON-unit, Language Environment considers the condition unhandled. If a GOTO is not performed and the resume cursor is not moved, then the thread terminates. The TERMTHDACT setting guides the amount of information that is produced, so the message is not presented twice.
- PL/I MTF Considerations
  - TERMTHDACT applies to a task that terminates abnormally due to an unhandled condition of severity 2 or higher that is percolated beyond the initial routine's stack frame. All active subtasks that were created from the incurring task will terminate abnormally, but the enclave will continue to run.
- z/OS UNIX Considerations
  - The TERMTHDACT option applies when a thread terminates abnormally. Abnormal termination of a single thread causes termination of the entire enclave. If an unhandled condition of severity 2 or higher percolates beyond the first routine's stack frame the enclave terminates abnormally.
  - If an enclave terminates due to a POSIX default signal action, then TERMTHDACT applies to conditions that result from software signals, program checks, or abends.
  - If running under a shell and Language Environment generates a system dump, then a core 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 CESE|CICSDDS suboption of the TERMTHDACT run-time option. Table 6 on page 44 shows the behavior of CESE|CICSDDS when they are used with the other suboptions of TERMTHDACT.
  - Since Language Environment does not own the ESTAE, the suboption UAIMM will be treated as UAONLY.
  - All associated Language Environment dumps will be suppressed if termination processing is the result of an EXEC CICS ABEND with NODUMP.

*Table 6. Condition Handling of 0Cx ABENDS*

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

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

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

## Generating a Language Environment dump with language-specific functions

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

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

Fortran programs can call SDUMP, DUMP/PDUMP, or CDUMP/CPDUMP to generate a Language Environment dump. CEE3DMP cannot be called directly from a Fortran program. For more information on these functions, see “Generating a Language Environment dump of a FORTRAN routine” on page 215.

PL/I routines can call PLIDUMP instead of CEE3DMP to produce a dump. PLIDUMP includes options that you can specify to obtain a variety of information in the dump. For a detailed explanation about PLIDUMP, see “Generating a Language Environment dump of a PL/I routine” on page 239.

## 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 50 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 57.

The CEE3DMP was generated by the C program CELSAMP shown in Figure 5 on page 46. CELSAMP uses the DLL CELDLL shown in Figure 6 on page 49.

```
#pragma options(SERVICE("1.1.c"),NOOPT,TEST,GONUMBER)
#pragma runopts(TERMTHDACT(UADUMP),POSIX(ON))
#define _OPEN_THREADS
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <dll.h>
#include <signal.h>
#include <leawi.h>
#include <ceeedcct.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 exitting\n",parm);
  pthread_exit(NULL);
}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  _CHAR80 title;
  _CHAR255 options;
  _FEEDBACK fc;

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

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

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

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

*Figure 6. The C DLL CELDLL*

For easy reference, the sections of the following dump are numbered to correspond with the descriptions in “Sections of the Language Environment dump” on page 57.

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

[3]Registers on Entry to CEE3DMP:

  PM....... 0100
  GPR0..... 2471CEB0  GPR1..... 00024A00  GPR2..... 2471CEB0  GPR3..... A4700B06
  GPR4..... 00024A00  GPR5..... 00000010  GPR6..... 000000F0  GPR7..... 000A2794
  GPR8..... 2471CEC0  GPR9..... 2471CFA0  GPR10.... 000A26C2  GPR11.... 2479A6E8
  GPR12.... 00015920  GPR13.... 00024800  GPR14.... 800180E2  GPR15.... A4759658
  FPR0..... 4DB035F6  D8F87B96            FPR2..... 00000000  00000000
  FPR4..... 00000000  00000000            FPR6..... 00000000  00000000
    GPREG STORAGE:
      Storage around GPR0 (2471CEB0)
        -0020 2471CE90  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
        +0000 2471CEB0  C3C5D3C4 D3D340E6 E2C10000 00000000  6E6E6E40 84A49497 6D956D97 8599837A  |CELDLL WSA......>>> dump_n_perc:|
        +0020 2471CED0  40D4A287 407B4089 A2406C84 15000000  E3C8D9C5 C1C44DC1 D3D35D40 C2D3D6C3  | Msg # is %d....THREAD(ALL) BLOC|
      Storage around GPR1 (00024A00)
        -0020 000249E0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404054  |                               .|
        +0000 00024A00  000248B0 00024900 000248A0 00024A40  00004A48 00024800 00024E70 A475A51C  |.............. ..........+.u.v.|
        +0020 00024A20  2475C480 04000000 00024A9C 259B1028  00000000 0004E544 0004EF94 2478E6D8  |..D..................V....m..WQ|

⋮

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

[5]Information for thread 24ABED8000000000

  Registers on Entry to CEE3DMP:
    PM....... 0100
    GPR0..... 2471CEB0  GPR1..... 00024A00  GPR2..... 2471CEB0  GPR3..... A4700B06
    GPR4..... 00024A00  GPR5..... 00000010  GPR6..... 000000F0  GPR7..... 000A2794
    GPR8..... 2471CEC0  GPR9..... 2471CFA0  GPR10.... 000A26C2  GPR11.... 2479A6E8
    GPR12.... 00015920  GPR13.... 00024800  GPR14.... 800180E2  GPR15.... A4759658
    FPR0..... 4DB035F6  D8F87B96            FPR2..... 00000000  00000000
    FPR4..... 00000000  00000000            FPR6..... 00000000  00000000
    GPREG STORAGE:
      Storage around GPR0 (2471CEB0)
        -0020 2471CE90  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
        +0000 2471CEB0  C3C5D3C4 D3D340E6 E2C10000 00000000  6E6E6E40 84A49497 6D956D97 8599837A  |CELDLL WSA......>>> dump_n_perc:|
        +0020 2471CED0  40D4A287 407B4089 A2406C84 15000000  E3C8D9C5 C1C44DC1 D3D35D40 C2D3D6C3  | Msg # is %d....THREAD(ALL) BLOC|

⋮

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


⋮
```

*Figure 7. Example Dump Using CEE3DMP (Part 1 of 8)*

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

      Machine State:
        ILC..... 0004    Interruption Code..... 0009
        PSW..... 078D2400 A4702A0A
        GPR0..... 000242B8  GPR1..... 000242A8  GPR2..... A4915A12  GPR3..... A47021C6
        GPR4..... 80007E04  GPR5..... 25993870  GPR6..... 259938E8  GPR7..... 25993CC8
        GPR8..... 00000000  GPR9..... 00000001  GPR10.... A4915952  GPR11.... 80007D20
        GPR12.... 00015920  GPR13.... 000241E0  GPR14.... A47029F6  GPR15.... 00000012
    Storage dump near condition, beginning at location: 247029F6
      +000000 247029F6  4400C1C4 4400C1AC 41800001 8E800020  5D80D09C 5090D09C 4400C1AC 417063F8  |..AD..A.........)...&.....A....8|
 [8]Parameters, Registers, and Variables for Active Routines:
    dump_n_perc (DSA address 00024800):
      Saved Registers:
        GPR0..... 2471CEB0  GPR1..... 00024A00  GPR2..... 2471CEB0  GPR3..... A4700B06
        GPR4..... 00024A00  GPR5..... 00000010  GPR6..... 000000F0  GPR7..... 000A2794
        GPR8..... 2471CEC0  GPR9..... 2471CFA0  GPR10.... 000A26C2  GPR11.... 2479A6E8
        GPR12.... 00015920  GPR13.... 00024800  GPR14.... 800180E2  GPR15.... A4759658
    GPREG STORAGE:
      Storage around GPR0 (2471CEB0)
        -0020 2471CE90  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|

⋮

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

⋮
      Local Variables:

⋮

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

*Figure 7. Example Dump Using CEE3DMP (Part 2 of 8)*

```
    DSA for main: 000241E0
      +000000  FLAGS.... 1000      member... 0000      BKC...... 000240C8  FWC...... 000242B8  R14...... A47029F6
      +000010  R15...... 2489F3D8  R0....... 000242B8  R1....... 000242A8  R2....... A4915A12  R3....... A47021C6
      +000024  R4....... 80007E04  R5....... 25993870  R6....... 259938E8  R7....... 25993CC8  R8....... 00000001
      +000038  R9....... 80000000  R10...... A4915952  R11...... 80007D20  R12...... 00015920  reserved. 000163D0
      +00004C  NAB...... 000242B8  PNAB..... 00017038  reserved. 00014D30  247C16F0  00015920  00000000
      +000064  reserved. 00024328  reserved. 01000000  MODE..... A470271C  reserved. 00000000  00000000
      +000078  reserved. 00000000  reserved. 00000000
    CIB for main: 000A26A8
      +000000 000A26A8  C3C9C240 00000000 00000000 010C0004  00000000 00000000 00030C89 59C3C5C5  |CIB .....................i.CEE|
      +000020 000A26C8  00000001 000A27B4 00030C89 59C3C5C5  00000001 00000000 000241E0 A47ECFC8  |..........i.CEE...........u=.H|
      +000040 000A26E8  00000000 000241E0 24702A0A 24717D50  00000003 00000000 00000000 00000000  |..............'&.............|
      +000060 000A2708  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |............................|
      +000080 000A2728 - +00009F 000A2747             same as above
      +0000A0 000A2748  00000000 00000000 00000000 00000000  40250000 940C9000 00000009 00000000  |............... ...m..........|
      +0000C0 000A2768  00000000 247031E0 000241E0 000241E0  24702A06 00000000 00000000 00000001  |............................|
      +0000E0 000A2788  00000002 00000003 00000002 00000014  00000002 00000000 00000000 40404040  |........................    |
      +000100 000A27A8  00000008 24717E9C 00000000 E9D4C3C8  00000000 000242B8 000242A8 A4915A12  |......=.....ZMCH..........yuj!.|


⋮

[10]Storage for Active Routines:
    DSA frame: 00024800
      +000000 00024800  1004FFF0 00024748 00024A10 800180E2  A4759658 2471CEB0 00024A00 2471CEB0  |...0...........Su.o.............|
      +000020 00024820  A4700B06 00024A00 00000010 000000F0  000A2794 2471CEC0 2471CFA0 000A26C2  |u..............0...m..........B|
      +000040 00024840  2479A6E8 00015920 000163D0 00024A10  2474D3FF 2474C400 00015920 000248B4  |..wY..............L...D.........|
      +000060 00024860  000248A0 24717AE4 000248DC A4700BC6  00024890 000A24C0 24717AE8 00024890  |......:U....u..F..........Y....|
      +000080 00024880  000248B4 000248DC 00000001 A4A8DCF6  000A3430 000A3421 247178D0 0000088E  |............uy.6................|
      +0000A0 000248A0  00000003 00015920 000A26A8 00000000  E2819497 93854084 A4949740 97999684  |...........y....Sample dump prod|
      +0000C0 000248C0  A4838584 4082A840 83819393 89958740  C3C5C5F3 C4D4D740 40404040 40404040  |uced by calling CEE3DMP         |
      +0000E0 000248E0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
      +000100 00024900  E3C8D9C5 C1C44DC1 D3D35D40 C2D3D6C3  D2E240E2 E3D6D9C1 C7C54040 40404040  |THREAD(ALL) BLOCKS STORAGE      |
      +000120 00024920  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
      +000140 00024940 - +0001DF 000249DF             same as above
      +0001E0 000249E0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404054  |                               .|
      +000200 00024A00  000248B0 00024900 000248A0 00024A40  00004A48 00024800 00024E70 A475A51C  |.............. ..........+.u.v.|
    DSA frame: 00024748
      +000000 00024748  10003B40 000A2098 00024928 A479A744  24700AB8 2471CEB0 247178F8 00044220  |... ...q....u.x...........8....|
      +000020 00024768  259ADB90 000A26A8 00000002 247178D0  000A3097 247499A5 25993870 247479A7  |.......y..........p..rv.r.....x|
      +000040 00024788  A474EAC0 00015920 000163D0 00024800  800118E0 00000000 00000000 259A0408  |u...........................|
      +000060 000247A8  00000009 25993C9D 259A0408 A474C756  259A03F0 24719C7C 2471A1B0 A4A81C38  |.....r......u.G....0...@....uy..|
      +000080 000247C8  000247F0 00024802 00024800 000247FC  000247F8 00024802 000247F0 00024802  |...0.............8.......0....|
      +0000A0 000247E8  00024800 000A3110 2491595E 00000000  00024018 00024018 1004FFF0 00024748  |.........j.;..... ... ....0....|
    DSA frame: 000241E0
      +000000 000241E0  10000000 000240C8 000242B8 A47029F6  2489F3D8 000242B8 000242A8 A4915A12  |...... H....u..6.i3Q.......yuj!.|
      +000020 00024200  A47021C6 80007E04 25993870 259938E8  25993CC8 00000001 80000000 A4915952  |u..F..=..r...r.Y.r.H........uj..|
      +000040 00024220  80007D20 00015920 000163D0 000242B8  00017038 00014D30 247C16F0 00015920  |..'...................(..@.0....|
      +000060 00024240  00000000 00024328 01000000 A470271C  00000000 00000000 00000000 00000000  |...........u...................|
      +000080 00024260  00000000 A47C1792 04000000 F97FC14F  00000001 F97FC14F 259AD3DE 00000000  |....u@.k....9"A|....9"A|..L.....|
      +0000A0 00024280  2599F1EC 259A03F4 00000000 00000000  00000000 00000002 259ADB90 00000000  |.r1....4....................|
      +0000C0 000242A0  00000000 00000000 25993CC8 25993CB8  00000003 259938F1 00000100 24716FB0  |.........r.H.r.......r.1......?.|
⋮

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


⋮
```

*Figure 7. Example Dump Using CEE3DMP (Part 3 of 8)*

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

⋮

[5]Information for thread 24ABF69000000001

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

[6]Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    00056978  CEEOPML2      24A84558  +0000004BE  CEEOPML2      24A84558  +0000004BE              CEEOLVD            Call
    00056728                2489A658  +0000160A  EDCOWRP2      2489B31C  +00000946               CEEEV003           Call
    00056680  POSIX.CRTL.C(CELSAMP)
                            24701F00  +000000A4  thread_func   24701F00  +000000A4          45  CELSAMP   1.1.c    Call
    7F83E5F0  POSIX.CRTL.C(CELSAMP)
                            24702178  -246F57F6  main          24702178  -246F57F6               CELSAMP   1.1.c    Call

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

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

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

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

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

⋮
```

*Figure 7. Example Dump Using CEE3DMP (Part 4 of 8)*

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

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

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

   Thread Synchronization Trace Block (OTRB): 0004E000
     +000000 0004E000  00046000 00000004 000007FF 00046000  3E008000 BE008000 00008F50 0004E044  |..-...........-...........&....|
   Thread Synchronization Trace Table (OTRTBL): 00046000
     +000000 00046000  0000D4E7 40C9D540 259938BC 00000000  0001D4E7 40C14040 259938BC 00000000  |..MX IN .r........MX A  .r......|
     +000020 00046020  0002D4E7 40E64040 259938BC 00000002  0003D4E7 40E64040 259938BC 00000001  |..MX W  .r........MX W  .r......|
     +000040 00046040  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |.............................|
     +000060 00046060 - +003FFF 00049FFF             same as above
   DLL Information:
   WSA Addr  Module Addr  Thread ID         Use Count  Name
   2471CEB0  247006F0     24ABED8000000000  00000001   CELDLL
   HEAPCHK Option Control Block (HCOP): 25993028
     +000000 25993028  C8C3D6D7 00000024 00000001 00000000  00000000 259BD028 2599304C 00000000  |HCOP.....................r.<....|
     +000020 25993048  00000000 C8C3C6E3 00000200 00000000  00000000 00000000 00000000 00000000  |....HCFT........................|
   HEAPCHK Element Table (HCEL) for Heapid 259B24B4 :
   Header: 259BD028
     +000000 259BD028  C8C3C5D3 259AF028 00000000 259B24B4  000001F4 00000007 00000007 00000000  |HCEL..0............4............|
                       Address  Seg Addr Length             Address  Seg Addr Length
   Table: 259BD048
     +000000 259BD048  259BC020 259BC000 00000050 00000000  259BC070 259BC000 00000020 00000000  |...........&....................|
     +000020 259BD068  259BC090 259BC000 00000018 00000000  259BC0A8 259BC000 00000088 00000000  |...................y.......h....|
     +000040 259BD088  259BC130 259BC000 00000050 00000000  259BC180 259BC000 00000020 00000000  |..A........&......A.............|
     +000060 259BD0A8  259BF020 259BF000 000003C8 00000000  00000000 00000000 00000000 00000000  |..0...0....H....................|
   HEAPCHK Element Table (HCEL) for Heapid 259ADC14 :
   Header: 259AF028
     +000000 259AF028  C8C3C5D3 2599D028 259BD028 259ADC14  000001F4 00000001 00000001 00000000  |HCEL.r.............4............|
                       Address  Seg Addr Length             Address  Seg Addr Length
   Table: 259AF048
     +000000 259AF048  259AE020 259AE000 000001C8 00000000  00000000 00000000 00000000 00000000  |...........H....................|
   HEAPCHK Element Table (HCEL) for Heapid 00000000 :
   Header: 2599D028
     +000000 2599D028  C8C3C5D3 00000000 259AF028 00000000  000001F4 00000004 00000004 00000000  |HCEL......0........4............|
                       Address  Seg Addr Length             Address  Seg Addr Length
   Table: 2599D048
     +000000 2599D048  25995020 25995000 00000038 00000000  25995058 25995000 00000038 00000000  |.r&..r&..........r&..r&.........|
     +000020 2599D068  25995090 25995000 00000010 00000000  259950A0 25995000 00000010 00000000  |.r&..r&..........r&..r&.........|
```

*Figure 7. Example Dump Using CEE3DMP (Part 5 of 8)*

```
   Heap Storage Diagnostics
      Stg Addr  ID        Length      Entry         E Addr    E Offset    Load Mod
      25358020  25321564  000002A8    CEEV#GTS      05AB26F8  +00000000   CEEPLPKA
                                      CEEVGTST      05ABCB40  +00000072   CEEPLPKA
                                      CEEVGTHP      05ABC0C8  +000000FA   CEEPLPKA
                                      CEEOSIGG      05A72700  +00001D48   CEEPLPKA
                                      CEEV#GH       05ABA688  -FFF3E3FC   CEEPLPKA
                                      func3         25308A70  +00000078   *PATHNAM
                                      func2         25308BA0  +00000076   *PATHNAM
                                      func1         25308C78  +00000076   *PATHNAM
                                      main          25308D50  +00000076   *PATHNAM
                                      EDCZMINV      05DF29FE  -FA214608   CEEEV003
      25342038  00000000  00000120    CEEV#GH       05ABA688  +00000000   CEEPLPKA
                                      setlocale     05DABB18  +000000FC   CEEEV003
                                      tzset         05D12108  +00000594   CEEEV003
                                      _cinit        05C1F8F0  +00002C18   CEEEV003
                                      CEEZINV       05AE60B0  +00000C76   CEEPLPKA
                                      main          25308D50  -DB85A804   *PATHNAM


    Language Environment Trace Table:

      Most recent trace entry is at displacement: 002980

          Displacement                      Trace Entry in Hexadecimal                              Trace Entry in EBCDIC
          ------------ ------------------------------------------------------------------------  --------------------------------
            +000000    Time 20.32.18.430976    Date 2001.08.26    Thread ID... 24ABED8000000000
            +000010    Member ID.... 03    Flags..... 00004B    Entry Type..... 00000001
            +000018    94818995 40404040 40404040 40404040  40404040 40404040 40404040 40404040 |main                            |
            +000038    60606E4D F0F8F55D 40979989 95A3864D  5D404040 40404040 40404040 40404040 |-->(085) printf()                |
            +000058    40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040 |                                |
            +000078    40404040 40404040                                                        |                                |

            +000080    Time 20.32.18.448404    Date 2001.08.26    Thread ID... 24ABED8000000000
            +000090    Member ID.... 03    Flags..... 00004B    Entry Type..... 00000002
            +000098    4C60604D F0F8F55D 40D9F1F5 7EF0F0F0  F0F0F0F0 C540C5D9 D9D5D67E F0F0F0F0 |<--(085) R15=0000000E ERRNO=0000|
            +0000B8    F0F0F0F0 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |0000........................    |
            +0000D8    00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
            +0000F8    00000000 00000000                                                        |........                        |
:
            +002900    Time 20.32.18.718260    Date 2001.08.26    Thread ID... 24ABED8000000000
            +002910    Member ID.... 03    Flags..... 00004B    Entry Type..... 00000001
            +002918    84A49497 6D956D97 85998340 40404040  40404040 40404040 40404040 40404040 |dump_n_perc                      |
            +002938    60606E4D F0F8F55D 40979989 95A3864D  5D404040 40404040 40404040 40404040 |-->(085) printf()                |
            +002958    40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040 |                                |
            +002978    40404040 40404040                                                        |                                |

            +002980    Time 20.32.18.718278    Date 20001.08.26    Thread ID... 24ABED8000000000
            +002990    Member ID.... 03    Flags..... 00004B    Entry Type..... 00000002
            +002998    4C60604D F0F8F55D 40D9F1F5 7EF0F0F0  F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(085) R15=00000000 ERRNO=0000|
            +0029B8    F0F0F0F0 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |0000........................    |
            +0029D8    00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
            +0029F8    00000000 00000000                                                        |........                        |

[13]Enclave Storage:
    Initial (User) Heap                                    : 25995000
      +000000 25995000  C8C1D5C3 00014D48 00014D48 00000000  25995000 259950B0 00008000 00007F50 |HANC..(...(......r&..r&.......&|
      +000020 25995020  25995000 00000038 C3C4D3D3 00000000  40000000 00000000 24700F98 24703F70 |.r&.....CDLL.... ..........q....|
      +000040 25995040  25993870 00000490 00000000 00000000  00000000 00000000 25995000 00000038 |.r.......................r&.....|
      +000060 25995060  C3C4D3D3 25995028 80000000 00000000  247006F0 24700770 2471CEB0 00000150 |CDLL.r&.....0...........&|
      +000080 25995080  00000000 00000000 00000000 00000000  25995000 00000010 259ADBB8 00000000 |................r&.............|
      +0000A0 259950A0  25995000 00000010 259ADBE0 00000000  00000000 00000000 00000000 00000000 |.r&.............................|
      +0000C0 259950C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
      +0000E0 259950E0 - +007FFF 2599CFFF            same as above
```

*Figure 7. Example Dump Using CEE3DMP (Part 6 of 8)*

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

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

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

⋮

    WSA for Program Object(s)
    WSA: 2471CEB0
      +000000 2471CEB0  C3C5D3C4 D3D340E6 E2C10000 00000000  6E6E6E40 84A49497 6D956D97 8599837A  |CELDLL WSA......>>> dump_n_perc:|
      +000020 2471CED0  40D4A287 407B4089 A2406C84 15000000  E3C8D9C5 C1C44DC1 D3D35D40 C2D3D6C3  | Msg # is %d....THREAD(ALL) BLOC|
      +000040 2471CEF0  D2E240E2 E3D6D9C1 C7C50000 00000000  E2819497 93854084 A4949740 97999684  |KS STORAGE......Sample dump prod|
      +000060 2471CF10  A4838584 4082A840 83819393 89958740  C3C5C5F3 C4D4D700 6E6E6E40 84A49497  |uced by calling CEE3DMP.>>> dump|
      +000080 2471CF30  6D956D97 8599837A 40E38192 89958740  84A49497 15000000 00000000 00000000  |_n_perc: Taking dump............|
      +0000A0 2471CF50  00000000 00000000 C3C5C5F3 C4D4D740  86818993 858440A6 89A38840 94A28795  |........CEE3DMP failed with msgn|
      +0000C0 2471CF70  96406C84 15000000 6E6E6E40 84A49497  6D956D97 8599837A 40D78599 83969381  |o %d....>>> dump_n_perc: Percola|
      +0000E0 2471CF90  A3899587 40839695 8489A389 96951500  180F58F0 F01007FF 24700A70 2471CEB0  |ting condition.....00...........|
      +000100 2471CFB0  24700A80 00000000 247006F0 2471CEB0  180F58F0 F01007FF 247008A8 2471CEB0  |...........0.......00......y....|
      +000120 2471CFD0  24700A80 00000000 247006F0 2471CEB0  180F58F0 F01007FF 24700898 2471CEB0  |...........0.......00......q....|
      +000140 2471CFF0  24700A80 00000000 247006F0 2471CEB0  00000000 00000000 00000000 00000000  |...........0....................|
⋮
[14]Run-Time Options Report:
 LAST WHERE SET                OPTION
 ------------------------------------------------------------------------------
     Installation default            ABPERC(NONE)
     Installation default            ABTERMENC(ABEND)
     Installation default          NOAIXBLD
     Installation default            ALL31(ON)
     Assembler user exit             ANYHEAP(32768,16384,ANYWHERE,FREE)
     Installation default          NOAUTOTASK
     Assembler user exit             BELOWHEAP(8192,8192,FREE)
     Installation default            CBLOPTS(ON)
     Installation default            CBLPSHPOP(ON)
     Installation default            CBLQDA(OFF)
     Installation default            CHECK(ON)
     Installation default            COUNTRY(US)
     Installation default          NODEBUG
     Installation default            DEPTHCONDLMT(10)
     Installation default            ENVAR("")
         Installation default            ERRCOUNT(0)
     Installation default            ERRUNIT(6)
     Installation default            FILEHIST
     Installation default            FILETAG(NOAUTOCVT,NOAUTOTAG)
     Default setting               NOFLOW
     Assembler user exit             HEAP(49152,16384,ANYWHERE,KEEP,8192,4096)
     Installation default            HEAPCHK(OFF,1,0,0)
     Installation default            HEAPPOOLS(OFF,8,10,32,10,128,10,256,10,1024,10,2048,10)
     Installation default            INFOMSGFILTER(OFF,,,,)
     Installation default            INQPCOPN
     Installation default            INTERRUPT(OFF)
     Installation default            LIBRARY(SYSCEE)
     Installation default            LIBSTACK(4096,4096,FREE)
     Installation default            MSGFILE(SYSOUT,FBA,121,0,NOENQ)
     Installation default            MSGQ(15)
     Installation default            NATLANG(ENU)
     Ignored                       NONONIPTSTACK(See THREADSTACK)
     Installation default            OCSTATUS
     Installation default          NOPC
```

*Figure 7. Example Dump Using CEE3DMP (Part 7 of 8)*

```
     Installation default           NOPC
     Invocation command               POSIX(ON)
     Installation default             PROFILE(OFF,"")
     Installation default             PRTUNIT(6)
     Installation default             PUNUNIT(7)
     Installation default             RDRUNIT(5)
     Installation default             RECPAD(OFF)
     Invocation command               RPTOPTS(ON)
     Invocation command               RPTSTG(ON)
     Installation default           NORTEREUS
     Installation default             RTLS(OFF)
     Installation default           NOSIMVRD
     Programmer default               STACK(4096,4096,ANYWHERE,KEEP,524288,131072)
     Installation default             STORAGE(NONE,NONE,NONE,0)
     Installation default             TERMTHDACT(TRACE,,96)
     Installation default           NOTEST(ALL,"*","PROMPT","INSPPREF")
     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             VCTRSAVE(OFF)
     Installation default             VERSION()
     Installation default             XPLINK(OFF)
     Installation default             XUFLOW(AUTO)

[15]Process Control Blocks:

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

  MEML: 00014788
    +000000 00014788  00000000 00000000 247288A8 00000000  00000000 00000000 247288A8 00000000  |.........hy.............hy....|

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

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

⋮
```

*Figure 7. Example Dump Using CEE3DMP (Part 8 of 8)*

## Sections of the Language Environment dump

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

For a detailed explanation of language-specific dump output:

- For C/C++ routines, see “Finding C/C++ Information in a Language Environment Dump” on page 156.
- For COBOL routines, see “Finding COBOL information in a dump” on page 197.
- For Fortran routines, see “Finding FORTRAN information in a Language Environment dump” on page 221.
- For PL/I routines, see “Finding PL/I information in a dump” on page 241.

### [1] Page Heading

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

- CEE3DMP identifier
- Title

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

**[2] Caller Program Unit and Offset**

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

**[3] Registers on Entry to CEE3DMP**

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

- Program mask

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

  On entry to CEE3DMP, the GPRs contain:

  **GPR 0**
  : Working register

  **GPR 1**
  : Pointer to the argument list

  **GPR 2–11**
  : Working registers

  **GPR 12**
  : Address of CAA

  **GPR 13**
  : Pointer to caller’s stack frame

  **GPR 14**
  : Address of next instruction to run if the ALL31 run-time option is set to ON

  **GPR 15**
  : Entry point of CEE3DMP
- Floating point registers (FPRs) 0, 2, 4, 6
- Storage pointed to by General Purpose Registers

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

**[4] - [14] Enclave Information**

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

**[4] Enclave Identifier**

This statement names the enclave for which information in the dump is provided. If multiple enclaves exist, the dump service generates data and storage information for the most current enclave, followed by previous enclaves in a last-in-first-out (LIFO) order. For more information about dumps for multiple enclaves, see “Multiple enclave dumps” on page 75.

**[5] - [11] Thread Information**

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

**[5] Information for thread**

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

**[6] Traceback**

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

- Stack frame (DSA) address
- Program unit

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

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

  For COBOL, Fortran, PL/I, and VisualAge PL/I routines, this is the entry point name. For C/C++ routines, this is the function name. If a function name or entry point was not specified for a particular routine, then the string ’** NoName **’ will appear.
- Entry point address
- Entry point offset
- Load module
- Service level

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

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

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

### [7] Condition Information for Active Routines

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

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

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

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

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

For each active routine, this section shows:

- Routine name and stack frame address
- Arguments

  For C/C++ and Fortran, arguments are shown here rather than with the local variables. For COBOL, arguments are shown as part of local variables. PL/I arguments are not displayed in the Language Environment dump.
- Saved registers

  This lists the contents of GPRs 0–15 at the time the routine transferred control.
- Storage pointed to by the saved registers

  Treating the saved contents of each register as an address, 32 bytes before and 64 bytes after the address shown.
- Local variables

  This section displays the local variables and arguments for the routine. This section also shows the variable type. Variables are displayed only if the symbol tables are available. To generate a symbol table and display variables, use the following compile options:
  - For C, use TEST(SYM).
  - For C++, use TEST.
  - For VS COBOL II, use FDUMP.
  - For COBOL/370, use TEST(SYM).
  - For COBOL for OS/390 & VM, use TEST(SYM).
  - For Enterprise COBOL for z/OS and OS/390, use TEST(SYM)
  - For Fortran, use SDUMP.
  - For PL/I, arguments and variables are not displayed.

### [9] Control Blocks for Active Routines

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

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

**[10] Storage for Active Routines**

This displays local storage for each active routine. The storage is dumped in hexadecimal, with EBCDIC translations on the right side of the page. There can be other information, depending on the language used. For C/C++ routines, this is the stack frame storage. For COBOL programs, this is language-specific information, WORKING-STORAGE, and LOCAL-STORAGE.

**[11] Control Blocks Associated with the Thread**

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

**[12] Enclave Control Blocks**

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

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

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

**[13] Enclave Storage**

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

**[14] Run-Time Options Report**

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

**[15] Process Control Blocks**

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

# Debugging with specific sections of the Language Environment dump

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

## The tracebacks, condition information, and data values section

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

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

These sections of the dump are shown in Figure 7 on page 50.

## 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 shows the format of the upward-growing stack frame.

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

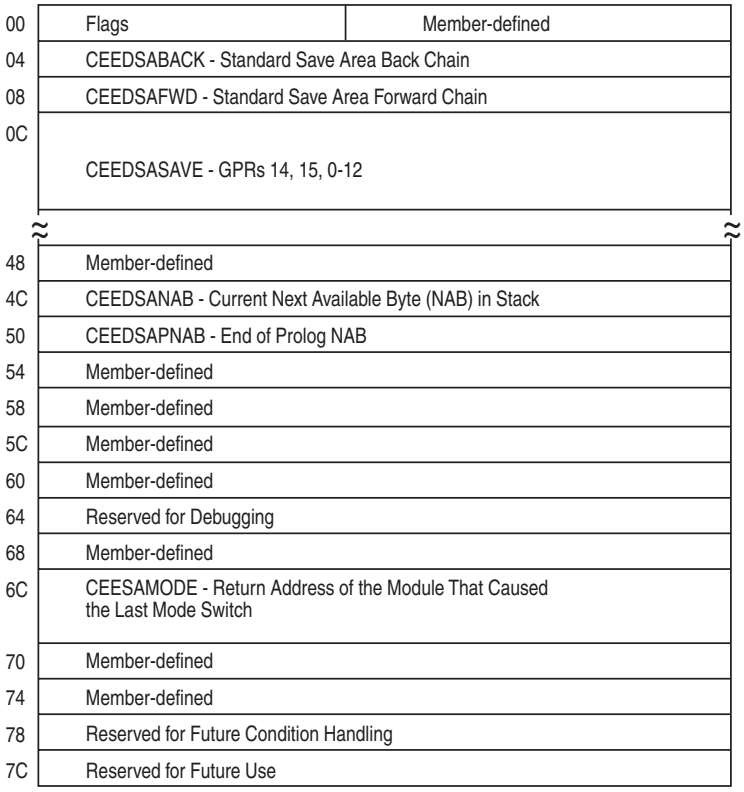



*Figure 8. Upward-Growing (Non-XPLINK) Stack Frame Format*

## The downward-growing (XPLINK) stack frame section

Figure 9 on page 64 shows the format of the downward-growing stack frame.

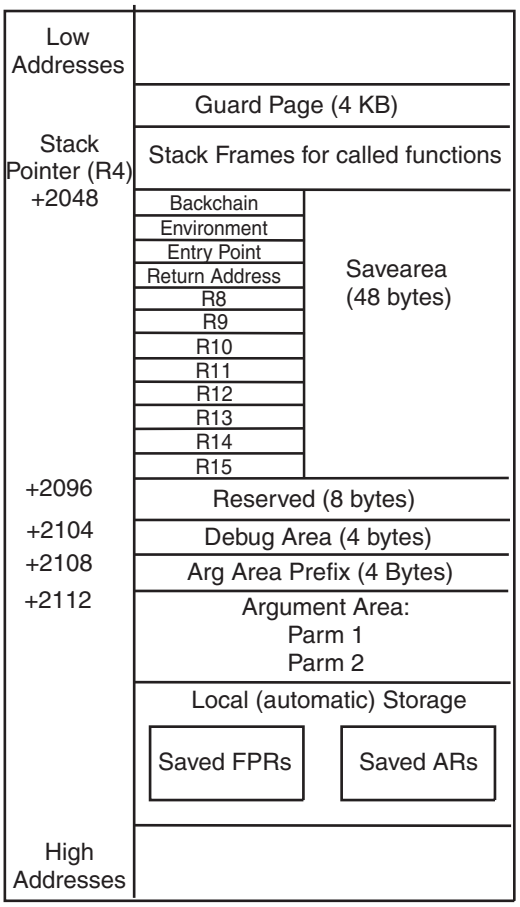



*Figure 9. Downward-Growing (XPLINK) Stack Frame Format*

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 65 shows the format of the Language Environment CAA.

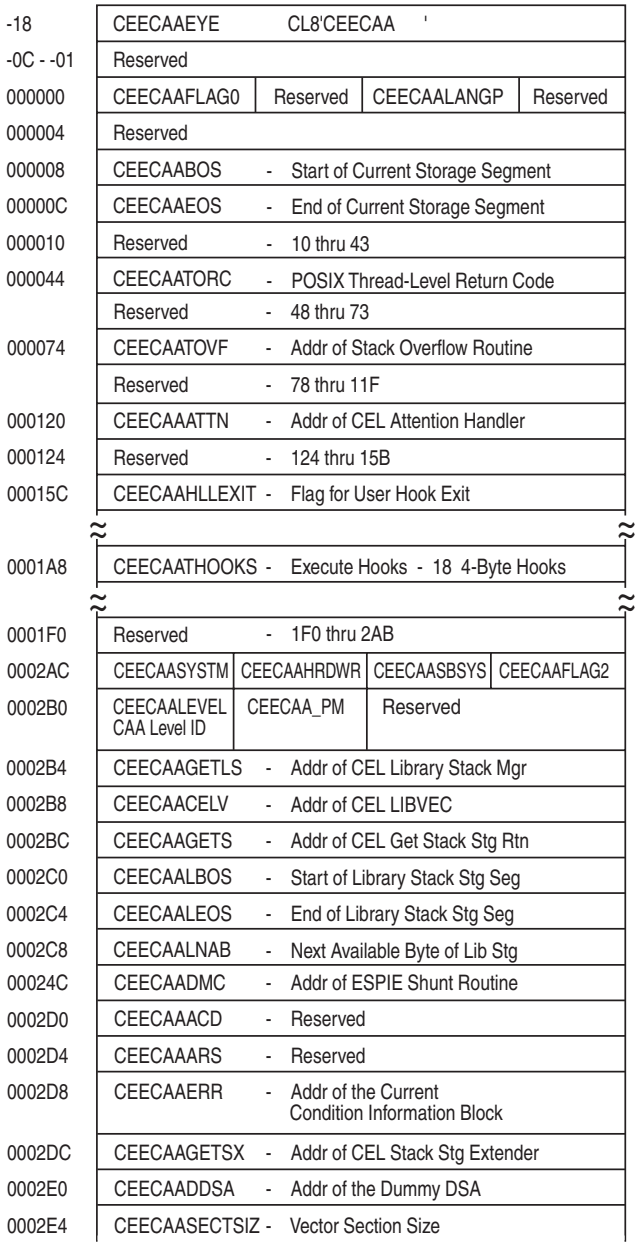



*Figure 10. Common Anchor Area (Part 1 of 2)*

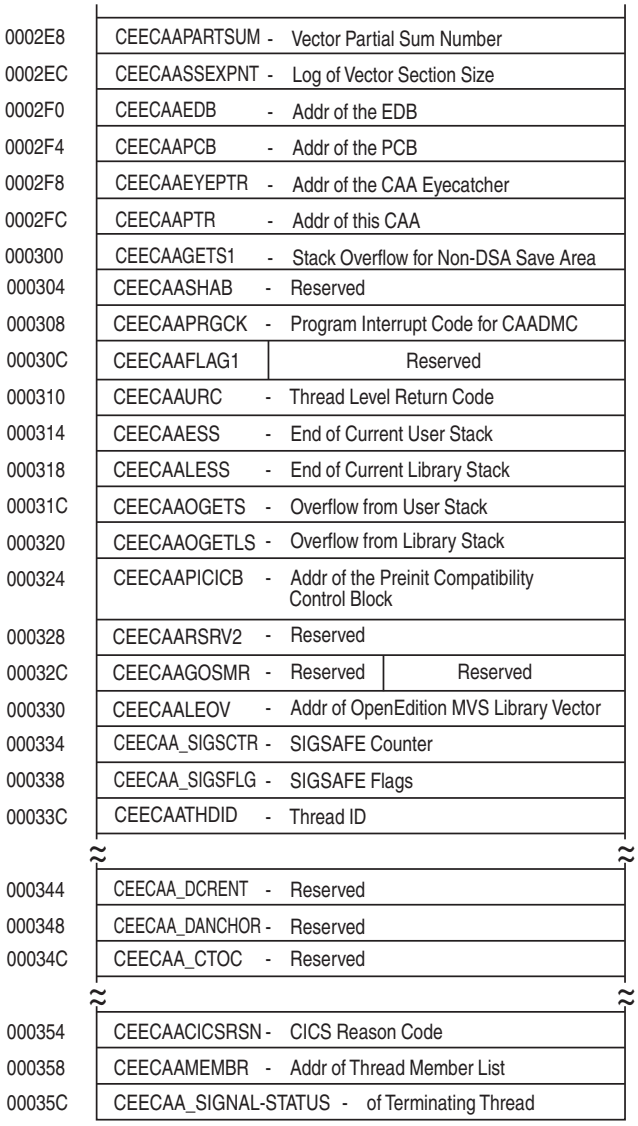



*Figure 10. Common Anchor Area (Part 2 of 2)*

Table 7 contains a list of CAA fields:

*Table 7. List of CAA Fields*

| CAA Field | Explanation |
|---|---|
| CEECAAFLAG0 | CAA flag bits. The bits are defined as follows:<br>**Bit** **Description**<br>**0–5** Reserved<br>**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.<br>**7** Reserved |

*Table 7. List of CAA Fields (continued)*

| CAA Field | Explanation |
|---|---|
| CEECAALANGP | PL/I language compatibility flags external to Language Environment. The bits are defined as follows:<br>**Bit** **Description**<br>**0–3** Reserved<br>**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.<br>**5–7** Reserved |
| CEECAABOS | Start of the current storage segment.<br>This field is initially set during thread initialization. It indicates the start of the current stack storage segment. It is altered when the current stack storage segment is changed. |
| CEECAAEOS | This field is used to determine if a stack overflow routine must be called when allocating storage from the user stack. Normally, the value of this field will represent the end of the current user stack segment. However, its value can also be zero to force the call of a stack overflow routine for every allocation of storage from the user stack. This field is used by function prologs that do not use FASTLINK linkage conventions. |
| CEECAATORC | Thread level return code. The thread level return code set by CEESRC callable service. |
| CEECAATOVF | Address of stack overflow routine. |
| CEECAAATTN | Address of the Language Environment attention handling routine. The address of the Language Environment attention handling routine supports common run-time environment's polling code convention for attention processing. |
| CEECAAHLLEXIT | Address of the Exit List Control Block set by the HLL user exit CEEBINT. |
| CEECAAHOOKS | Hook area. This is the start of 18 fullword execute hooks. Language Environment initializes each fullword to X'07000000'. The hooks can be altered to support various debugging hook mechanisms. |
| CEECAASYSTM | Underlying operating system. The value indicates the operating system supporting the active environment.<br>**Value** **Operating System**<br>**0** Undefined. This value should not appear after Language Environment is initialized.<br>**1** Unsupported<br>**3** z/OS |

*Table 7. List of CAA Fields (continued)*

| CAA Field | Explanation |
|---|---|
| CEECAAHRDWR | Underlying hardware. This value indicates the type of hardware on which the routine is running.<br>**Value** **Hardware**<br>**0** Undefined. This value should not appear after Language Environment is initialized.<br>**1** Unsupported<br>**2** System/370™, non-XA<br>**3** System/370, XA<br>**4** System/370, ESA |
| CEECAASBSYS | Underlying subsystem. This value indicates the subsystem (if any) on which the routine is running.<br>**Value** **Subsystem**<br>**0** Undefined. This value should not occur after Language Environment is initialized.<br>**1** Unsupported<br>**2** None. The routine is not running under a Language Environment-recognized subsystem.<br>**3** TSO<br>**4** IMS<br>**5** CICS |
| CEECAAFLAG2 | CAA Flag 2.<br>**Bit** **Description**<br>**0** Bimodal addressing is available.<br>**1** Vector hardware is available.<br>**2** Thread terminating.<br>**3** Initial thread<br>**4** Library trace is active. The TRACE run-time option was set.<br>**5** Reserved<br>**6** CEECAA_ENQ_Wait_Interruptible. Thread is in an enqueue wait.<br>**7** Reserved |
| CEECAALEVEL | Language Environment level identifier. This contains a unique value that identifies each release of Language Environment. This number is incremented for each new release of Language Environment. |
| CEECAA_PM | Image of current program mask. |
| CEECAAGETLS | Address of stack overflow for library routines. |
| CEECAACELV | Address of the Language Environment library vector. This field is used to locate dynamically loaded Language Environment routines. |

*Table 7. List of CAA Fields (continued)*

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

*Table 7. List of CAA Fields (continued)*

| CAA Field | Explanation |
|---|---|
| CEECAAEYEPTR | Address of the CAA eye catcher. The CAA eye catcher is CEECAA. This field can be used for validation of the CAA. |
| CEECAAPTR | Address of the CAA. This field points to the CAA itself and can be used in validation of the CAA. |
| CEECAAGETS1 | Non-DSA stack overflow. This field is the address of a stack overflow routine, which cannot guarantee that the current register 13 is pointing at a stack frame. Register 13 must point, at a minimum, to a save area. |
| CEECAASHAB | ABEND shunt routine. Its value is initially set to zero during thread initialization. If it is nonzero, this is the address of a routine used in specialized exception processing for ABENDs that are intercepted in the ESTAE exit. |
| CEECAAPRGCK | Routine interrupt code for CEECAADMC. If CEECAADMC is nonzero, and a routine interrupt occurs, this field is set to the routine interrupt code and control is passed to the address in CEECAAMDC. |
| CEECAAFLAG1 | CAA flag bits. The bits are defined as follows:<br>**Bit** **Description**<br>**0** CEECAASORT. A call to DFSORT™ is active.<br>**1–7** Reserved |
| CEECAAURC | Thread level return code. This is the common place for members to set the return codes for subroutine-to-subroutine return code processing. |
| CEECAAESS | This field is used to determine if a stack overflow routine must be called when allocating storage from the user stack. Normally, the value of this field will represent the end of the current user stack segment. However, its value can also be zero to force the call of a stack overflow routine for every allocation of storage from the user stack. This field is used by function prologs that use FASTLINK linkage conventions. |
| CEECAALESS | This field is used to determine if a stack overflow routine must be called when allocating storage from the library stack. Normally, the value of this field will represent the end of the current library stack segment. However, its value can also be zero to force the call of a stack overflow routine for every allocation of storage from the library stack. This field is used by function prologs that use FASTLINK linkage conventions. |
| CEECAAOGETS | Overflow from user stack allocations. |
| CEECAAOGETLS | Overflow from library stack allocations. |
| CEECAARSRV1 | Reserved. |
| CEECAAPICICB | 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. |

*Table 7. List of CAA Fields (continued)*

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

## The condition information block

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

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

Figure 11 on page 72 shows the condition information block.

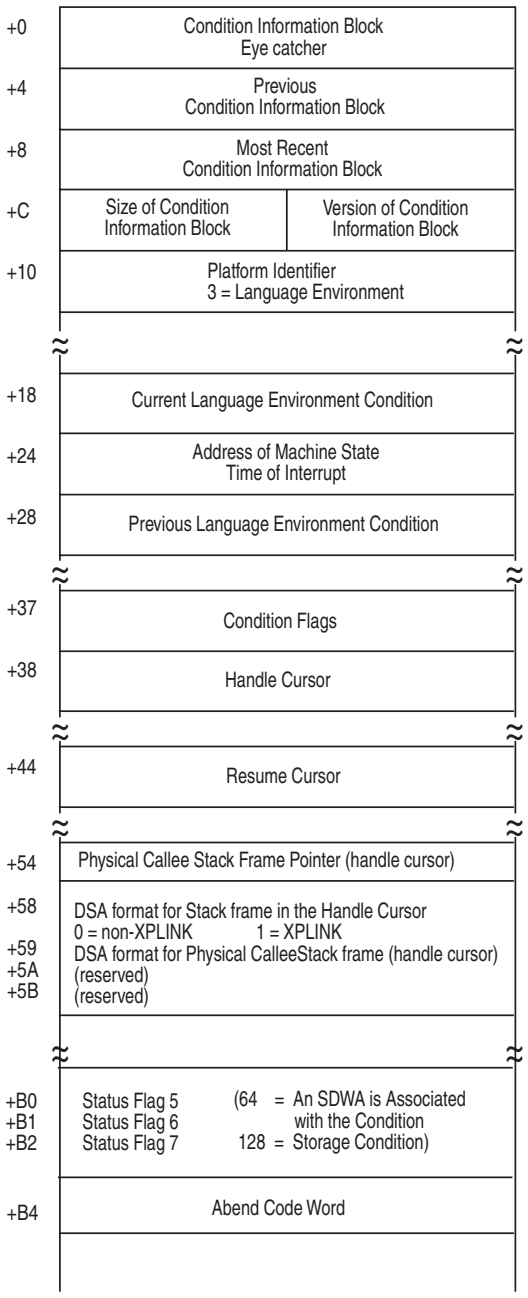



*Figure 11. Condition Information Block (Part 1 of 2)*

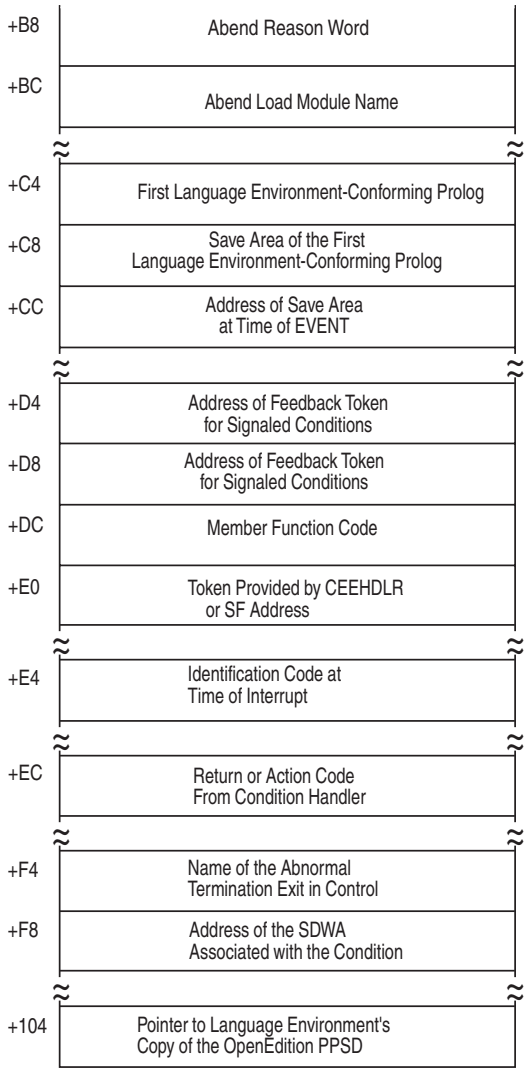



*Figure 11. Condition Information Block (Part 2 of 2)*

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 [2]
**64** Caused by a program check
**128** Caused by an abend

The flags for Status Flag 6, Language Environment actions:

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

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

The language-specific function codes for the CIB:
**X'1'** For condition procedure
**X'2'** For enablement
**X'3'** For stack frame zero conditions

## Using the machine state information block

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

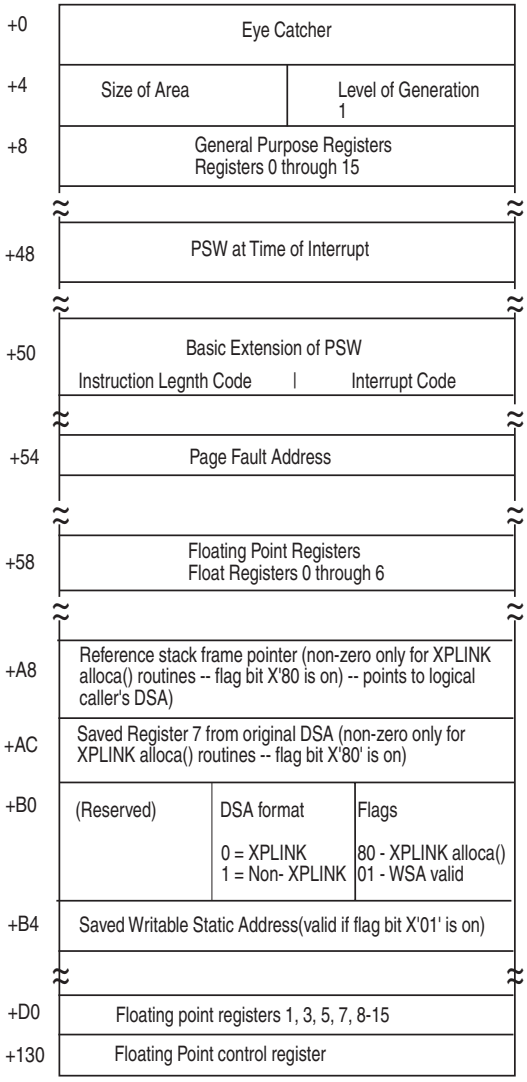



*Figure 12. Machine State Information Block*

## Multiple enclave dumps

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

Figure 13 on page 76 illustrates the information available in the Language Environment dump and the order of information for multiple enclaves.

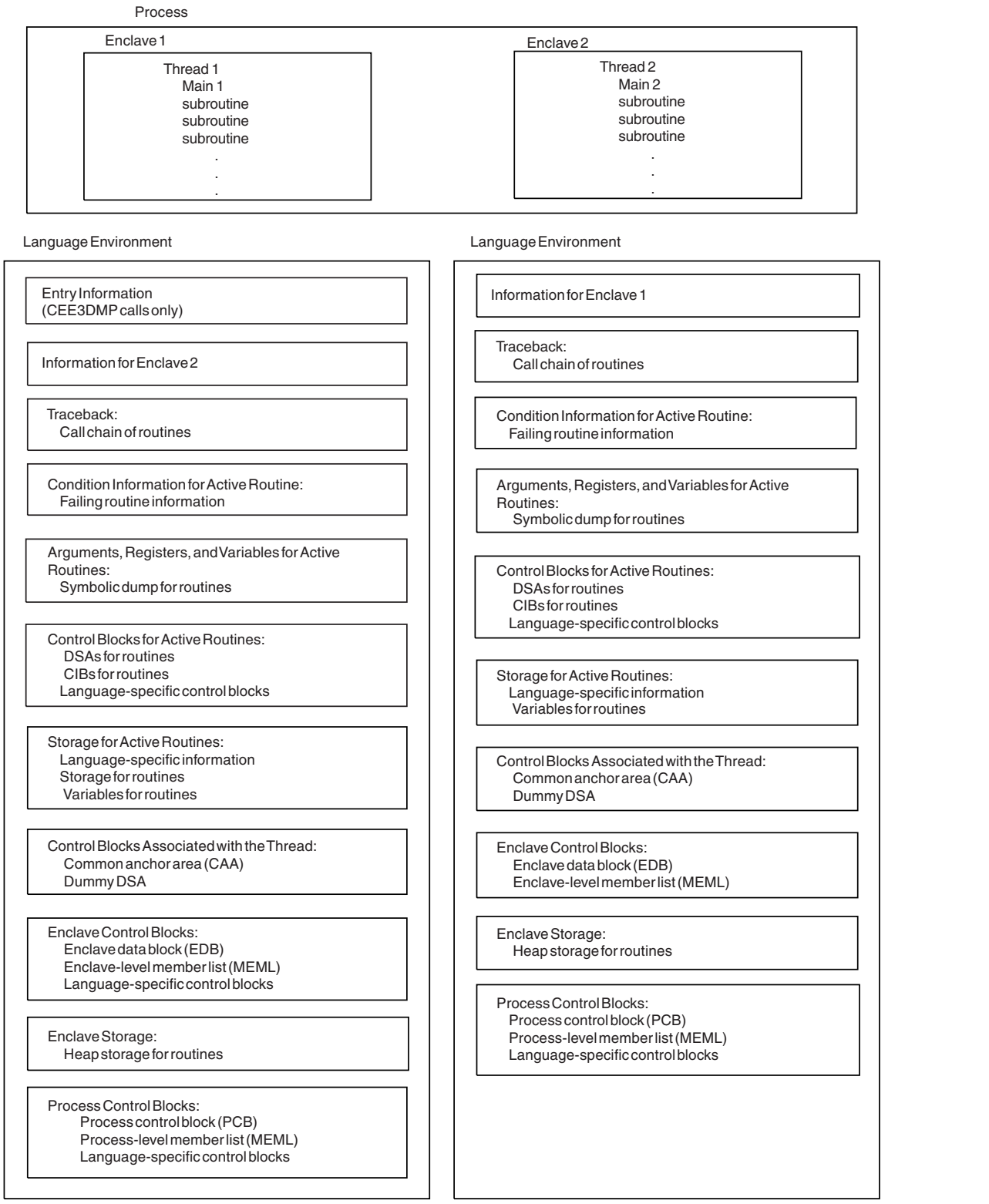



*Figure 13. Language Environment Dump of Multiple Enclaves*

## Generating a system dump

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

**TERMTHDACT(UAONLY, UATRACE, or UADUMP)**
You can use these run-time options, with TRAP(ON), to generate a system dump if an unhandled condition of severity 2 or greater occurs. For further details regarding the level of dump information produced by each of the TERMTHDACT suboptions, see "Generating a Language Environment dump with TERMTHDACT" on page 41.

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

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

ABPERC is ignored under CICS.

**Abend Codes in Initialization Assembler User Exit**
Abend codes listed in the initialization assembler user exit are passed to the operating system. The operating system can then generate a system dump.

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

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

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

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

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

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

1. Specify run-time options TERMTHDACT(UAONLY, UADUMP, UATRACE, or UAIMM), and TRAP(ON). If you specify the suboption UAIMM then you must set TRAP(ON,NOSPIE). The TERMTHDACT suboption determines the level of detail of the Language Environment formatted dump. For further details on the TERMTHDACT suboptions, see "Generating a Language Environment dump with TERMTHDACT" on page 41.
2. Include a SYSMDUMP DD card with the desired data set name and DCB information:

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

3. Rerun the program.

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

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

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

1. Specify run-time options TERMTHDACT(UAONLY, UADUMP, UATRACE, or UAIMM), ABTERM(ABEND), and TRAP(ON). If you specify the suboption UAIMM, then you must set TRAP(ON,NOSPIE). The TERMTHDACT suboption determines the level of detail of the Language Environment formatted dump. For further details on the TERMTHDACT suboptions, see “Generating a Language Environment dump with TERMTHDACT” on page 41.

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

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

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

3. Rerun the program.

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

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

**Before you begin:** Under CICS, a system dump provides the most useful information for diagnosing problems. However, if you have a Language Environment U4038 abend, CICS will not generate a system dump. In order to generate diagnostic information for a CICS run-time environment with a Language Environment U4038 abend, you must create a Language Environment U4039 abend. For instructions on how to create a Language Environment U4039 abend, see “Steps for generating a Language Environment U4039 abend” on page 79.

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 41.
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 UAIMM is processed the same as UAONLY.

## Steps for Generating a system dump in a z/OS UNIX shell

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

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

     ```
     export _BPXK_MDUMP=filename
     ```

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

     **Example:**

     ```
     export _BPXK_MDUMP=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_RUNOPT="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 41.
3. Rerun the program.

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

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

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

## Formatting and analyzing system dumps

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

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

## Preparing to use the Language Environment support for IPCS

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

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

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

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

  The Language Environment-supplied CEEIPCSP member, installed in the SYS1.PARMLIB library, contains the Language Environment-specific entries for the IPCS exit control table.
- Provide an IPCSPARM DD statement to specify the libraries containing the IPCS control tables.

  **Example:**

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

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

## Language Environment IPCS Verbexit – LEDATA

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

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

## Format

**Syntax**

```
VERBEXIT LEDATA [ 'parameter[,parameter]...']
   Report Type Parameters:
      [ SUM ]
      [ HEAP | STACK | SM ]
      [ HPT(value) ]
      [ CM ]
      [ MH ]
      [ CEEDUMP ]
      [ ALL ]
   Data Selection Parameters:
      [ DETAIL | EXCEPTION ]
   Control Block Selection Parameters:
      [ CAA(caa-address) ]
      [ DSA(dsa-address) ]
      [ TCB(tcb-address) ]
      [ ASID(address-space-id) ]
      [ NTHREADS(value) ]
```

## Parameters

### Report type parameters

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

**SUMmary**

Requests a summary of the Language Environment at the time of the dump. The following information is included:

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

**HEAP | STACK | SM**

**HEAP**

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

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

**STACK**

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

**SM**

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

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

**CM**
Requests a report on Condition Management control blocks.

**MH**
Requests a report on Message Handler control blocks.

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

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

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

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

**Note:** For the Heap and Storage Management Reports, the DETAIL parameter will provide a detailed heap segment report for each heap segment in the dump. The detailed heap segment report includes information on the free storage tree in the heap segments, and all allocated storage elements. This report will also identify problems detected in the heap management data structures. For more information about the Heap Reports, see “Understanding the HEAP LEDATA output” on page 98.

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

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

***Control block selection parameters:*** Use these parameters to select the CAA and DSA control blocks used as the starting points for formatting.

**CAA(caa-address)**
specifies the address of the CAA. If not specified, the CAA address is obtained from the TCB.

**DSA(dsa-address)**
specifies the address of the DSA. If not specified, the DSA address is assumed to be the register 13 value for the TCB.

**TCB(tcb-address)**
specifies the address of the TCB. If not specified, the TCB address of the current TCB from the CVT is used.

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

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

## Understanding the HEAPPOOLS trace output

```
HPT(5)
   ********************************************************************************
                              LANGUAGE ENVIRONMENT DATA
   ********************************************************************************

    Language Environment Product 04 V01 R05.00

[1] Heap Pool Trace Table

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

[3] Timestamp: 2003/09/03 21:01:32.828075
    Type: FREE  Cell Address: 20453850  Cpuid: 01  Tcb: 008D6A68
[4] CALL NAME                           CALL ADDRESS       CALL OFFSET
    CEEVFQT                             1FF144A0           00000000
    foo8                                1FF010B8           0000009E
    foo7                                1FF011D8           00000080
    foo6                                1FF012F8           00000080
    foo5                                1FF01418           00000080
    foo4                                1FF01538           00000080
    foo3                                1FF01658           00000080
    foo2                                1FF01778           00000080
    foo1                                1FF01898           00000080
    main                                1FF00AE8           00000000

    Timestamp: 2003/09/03 21:01:32.828074
    Type: FREE  Cell Address: 20453C58  Cpuid: 01  Tcb: 008D6A68
    CALL NAME                           CALL ADDRESS       CALL OFFSET
    CEEVFQT                             1FF144A0           00000000
    foo9                                1FF00F98           0000009E
    foo8                                1FF010B8           00000080
    foo7                                1FF011D8           00000080
    foo6                                1FF012F8           00000080
    foo5                                1FF01418           00000080
    foo4                                1FF01538           00000080
    foo3                                1FF01658           00000080
    foo2                                1FF01778           00000080
    foo1                                1FF01898           00000000

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

    Timestamp: 2003/09/03 21:01:32.819907
```

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

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

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

**[1]** HEAPPOOLS trace header information.

**[2]** 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]** The time this trace entry was taken. Note the trace entries are formatted in reverse order (most recent trace entry first).

**[4]** 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 requested. The number of entries in this traceback is limitted by the HEAPCHK run-time option

## 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 85 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 46. “Sections of the Language

Environment LEDATA verbexit formatted output" on page 95 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 95.

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

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

[2]Active Members: C/C++

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

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 1 of 11)*

```
 [4]CEEPCB:  00015560
   +000000  PCBEYE:CEEPCB     SYSTM:03    HRDWR:03    SBSYS:02    FLAG2:98
   +00000C  DBGEH:00000000    DMEMBR:00015790   ZLOD:064D3D20
   +000020  ZDEL:064CB928     ZGETST:064D1AB0   ZFREEST:064D15E0
   +00002C  LVTL:0630B978     RCB:00014918      SYSEIB:00000000
   +000038  PSL:00000000      PSA:000159D0      PSRA:064D1908
   +000044  OMVS_LEVEL:7F000000     PCB_CHAIN:00000000
   +00004C  PCB_VSSFE:00013304      PCB_PRFEH:00000000
   +000084  LPKA_LODTYP:00000003    IMS:00000000      ABENDCODE:00000000
   +000090  REASON:00000000   F3456:000080C2    MEML:00015778
   +00009C  MEMBR:00015790    PCB_EYE:00000000        PCB_BKC:00005F78
   +0000A8  PCB_FWC:00000000        PCB_R14:863002BE
   +0000B0  PCB_R15:00006EA8        PCB_R0:7D000016   PCB_R1:00005FE8
   +0000BC  PCB_R2:06300080   PCB_R3:00000000   PCB_R4:00000000
   +0000C8  PCB_R5:00000000   PCB_R6:00000000   PCB_R7:00000000
   +0000D4  PCB_R8:063040B0   PCB_R9:006E7710   PCB_R10:00000000
   +0000E0  PCB_R11:863001F2        PCB_R12:00000000        CELV24:00018038
   +0000EC  CELV31:0630ED20   SLDR:8000E108     SECTSIZ:00000000
   +0000F8  PARTSUM:00000000        SSEXPNT:00000000        BMPS:06321FC8
   +000104  BMPE:0639A708     BLEHL:0630B450    BCMXB:00014B08    BSTV:02
   +000111  PM_BYTE:00  INI_AMODE:00      FLAGS1:28   ISA:06306000
   +000118  ISA_SIZ:00018A5C        SRV_CNT:00000000
   +000120  SRV_UWORD:00000000      WORKAR:00000000   LOAD:0000E6C0
   +00012C  DELETE:0000E338   GETSTOR:00010430        FREESTOR:0000FF30
   +000138  EXCEPT:00000000   ATTN:00000000     MSGS:00000000
   +000144  ABEND:000080D8    MSGOU:0000B2B8    GLAT:0642A090
   +000150  RLAT:06452B40     ELAT:06423398     1PTQ:064634B0
   +00015C  1ENV:06462680     DBG_LODTYP:FFFFFFFF     DUMMY_STK:06306008
   +000168  DUMMY_LIB:00014000      DUMMY_CAA:0630A010
   +000170  TST_LVL:FFFFFFFF        GETCAA:00014D38   SETCAA:00014D40
   +00017C  LLTPTR:0630AD30   AUE:00000000      RC:00000000
   +000188  REASON:00000000   RC_MOD:00000000   AUE_UWORD:00000000
   +000194  FB_TOKEN:............   EOV:067C5038      PPA:068D1F44
   +0001A8  PPA_SIZ:00000A00        BELOW:00014000    BELOW_LEN:00003940
   +0001B4  PICB:06315228     UTL1:06315238     ZINA:864D26A0
   +0001C0  ZINB:000138D8     XPLINKFLAGS:00    FLAGS5:80
   +0001D4  LANGINIT:00000001 00000000 00000000 00000000 0000
   +0001E8  NUMINIT:00000001        LASTINIT:00000003
   +0001F0  LANGREUSE:00000000 00000000 00000000 00000000 0000
   +000202  REUSEMEMS:00000000 00000000 00000000 00000000 0000

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

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

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

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

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

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 2 of 11)*

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

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 3 of 11)*

```
[9]Heap Storage Control Blocks

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

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

   Enclave Level Stack Statistics

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

     SKSB:  00015FEC

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

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

   User Heap Control Blocks

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

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

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

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

   This is the last heap segment in the current heap.

   Free Storage Tree for Heap Segment 068D5000

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

   Map of Heap Segment 068D5000

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

  068D5020: Allocated storage element, length=00000110. To display: IP LIST 068D5020 LEN(X'00000110') ASID(X'001F')
  068D5028: 068D5138 068D5320 068D535D 068D539A  068D53D7 068D5414 068D5451 068D548E |...........).......P.............|

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

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 4 of 11)*

```
[9]Heap Storage Control Blocks

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

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

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

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

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

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

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

     SMCB:  000171F0
   +000000  EYE_CATCHER:SMCB  US_EYE_CATCHER:USTK     USFIRST:00022000
   +00000C  USLAST:00022000   USBOS:00022000    USEOS:00042000
   +000018  USNAB:000253C0    USINITSZ:00020000       USINCRSZ:00020000
   +000024  USANYBELOW:80000000     USKEEPFREE:00000000     USPOOL:80000002
   +000030  USPREALLOC:00000001     US_BYTES_ALLOC:00008CF8
   +000038  US_CURR_ALLOC:00003460        US_GETMAINS:00000000
   +000040  US_FREEMAINS:00000000   US_OPLINK:00      LS_THIS_IS:LSTK
   +00004C  LSFIRST:00021000        LSLAST:00021000   LSBOS:00021000
   +000058  LSEOS:00022000    LSNAB:00021018    LSINITSZ:00001000
   +000064  LSINCRSZ:00001000       LSANYBELOW:80000000
   +00006C  LSKEEPFREE:00000001     LSPOOL:80000001   LSPREALLOC:00000001
   +000078  LS_BYTES_ALLOC:00000330       LS_CURR_ALLOC:00000330
   +000080  LS_GETMAINS:00000000    LS_FREEMAINS:00000000   LS_OPLINK:00
   +00008C  RSBOS:0001F000    RSEOS:00021000    RSIZE:00002000
   +000098  RSACTIVE:00000000       SA_REG00:00025460
   +0000A0  SA_REG01:000253C0       SA_REG02:00023308
   +0000A8  SA_REG03:00000003       SA_REG04:00016058
   +0000B0  SA_REG05:00000010       SA_REG06:000230D8
   +0000B8  SA_REG07:00023C9F       SA_REG08:063CECCD
   +0000C0  SA_REG09:063CDCCE       SA_REG10:063CCCCF
   +0000C8  SA_REG11:063D8838       SA_REG12:00016AC0
   +0000D0  SA_REG13:00022CA0       SA_REG14:863D886A
   +0000D8  SA_REG15:00000000
   +0000DC  SAVEREG_XINIT:00000000 00000000 00000000 00000000
   +0000EC  CEEVGTSI:000114F8       ST_DSA_ALLOC_FLAG:00000000
   +0000F4  ST_DSA_ALLOC_VAL:00000000     ALLOCSEG:00000000
   +0000FC  BELOW16M_FLAG:00000000        LOCAL_ALLOC:FFFFFF00
   +00010C  LOCAL_GETMAINS:00000000       LOCAL_FREEMAINS:00000000
   +00015C  MOREFLAGS:00000000      DS_THIS_IS:....   DSFIRST:00017350
   +000168  DSLAST:00017350   DSBOS:00017350    DSINITSZ:00000000
   +00017C  DSINCRSZ:00000000       DSGUARDSZ:00000000
   +000184  DSKEEPFREE:00000000     DSPOOL:00000000   DSPREALLOC:00000000
   +000190  DS_BYTES_ALLOC:00000000       DS_CURR_ALLOC:00000000
   +000198  DS_GETMAINS:00000000    DS_FREEMAINS:00000000
   +0001A0  DS_FLAGS:00000000
```

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 5 of 11)*

```
DSA backchain

   DSA:  000253C0

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

Contents of DSA  at location 000253C0:

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

   DSA:  00022CA0
+000000  FLAGS:0808  MEMD:CEE1   BKC:000221F8      FWC:000253C0
+00000C  R14:863CE4C4      R15:063D8838      R0:06319A64
+000018  R1:000230B0       R2:00023308       R3:00000003
+000024  R4:00016058       R5:00000010       R6:000230D8
+000030  R7:00023C9F       R8:063CECCD       R9:063CDCCE
+00003C  R10:063CCCCF      R11:863CBCD0      R12:00016AC0
+000048  LWS:00017630      NAB:000253C0      PNAB:00000012
+000064  RENT:00022BF8     CILC:00000259     MODE:863CD462
+000078  RMR:06484AE8

Contents of DSA  at location 00022CA0:

+00000000  0808CEE1 000221F8 000253C0 863CE4C4  063D8838 06319A64 000230B0 00023308 |.......8....f.UD..h.............|
+00000020  00000003 00016058 00000010 000230D8  00023C9F 063CECCD 063CDCCE 063CCCCF |.......-........Q...............|
+00000040  863CBCD0 00016AC0 00017630 000253C0  00000012 867C5882 867C5896 86448940 |f...................f@.bf@.of.i |
+00000060  FFFFFFFF 00022BF8 00000259 863CD462  00017630 00022D80 06484AE8 00000000 |.......8....f.M.............Y....|
+00000080  863CC09A 00000000 00024218 000230B0  000159D0 063186A0 00000000 00013304 |f.....................f.........|
+000000A0  00023C9F 063CECCD 063CDCCE 063CCCCF  863CBCD0 00016AC0 06319D54 06319A64 |................f...............|
+000000C0  863CC62C 000221F8 06319A64 00023194  00000002 063186A0 00000000 00000001 |f.F....8.......m......f.........|
+000000E0  063199D0 00023C9F 063CECCD 063CDCCE  063CCCCF 863CBCD0 00016AC0 06319A64 |..r.................f...........|

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

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

Contents of DSA  at location 000221F8:

+00000000  10000000 000220E0 000222D8 863026BA  06615024 067C68F8 00022290 067C6CC0 |...........Qf..../&..@.8.....@%.|
+00000020  06301BCA 068E0414 067C6B38 000222B2  000222BC 000222C0 80000000 8669199A |.........@,.................f...|
+00000040  800082A8 00016AC0 00017630 000222D8  065D28D2 067C68F8 067C68F8 000159D0 |..by...........Q.).K.@.8.@.8....|
+00000060  00015E50 00015E68 000159D0 863022E8  8649AA90 00016AC0 00017630 000222C8 |..;&..;.....f..Yf.............H|
+00000080  00000000 00000004 04000000 F97FC14F  00000001 F97FC14F 067C6CC0 067C6C88 |............9"A]....9"A].@%..@%h|
+000000A0  00000003 067C6BA8 068ECE56 00000000  068DF1EC 068E0414 00000000 00000000 |.....@,y.........1.............|
+000000C0  00000000 00000002 068ECEE0 00000000  00000000 00000000 067C68F8 00000000 |.........................@.8....|
```

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 6 of 11)*

```
[11]Condition Management Control Blocks
   User Stack Control Blocks

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

   Library Stack Control Blocks

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

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

     CIBH:  00023818
   +000000  EYE:CIBH    BACK:063199D0     FRWD:00000000
   +000010  PTR_CIB:00000000        FLAG1:00    ERROR_LOCATION_FLAGS:00
   +000018  HDLQ:00000000     STATE:00000000    PRM_DESC:00000000
   +000024  PRM_PREFIX:00000000
   +000028  PRM_LIST:00000000 00000000 00000000 00000000
   +000038  PARM_DESC:00000000      PARM_PREFIX:00000000
   +000040  PARM_LIST:00000000 00000000 00000000 00000000   FUN:00000000
   +000054  CIB_SIZ:0000      CIB_VER:0000      FLG_5:00    FLG_6:00
   +00005A  FLG_7:00    FLG_8:00    FLG_1:00    FLG_2:00    FLG_3:00
   +00005F  FLG_4:00    ABCD:00000000     ABRC:00000000
   +000068  OLD_COND_64:00000000 00000000       OLD_MIB:00000000
   +000074  COND_64:00000000 00000000     MIB:00000000      PL:00000000
   +000084  SV2:00000000      SV1:00000000      INT:00000000
   +000090  MID:00000000      HDL_SF:00000000   HDL_EPT:00000000
   +00009C  HDL_RST:00000000        RSM_SF:00000000   RSM_POINT:00000000
   +0000A8  RSM_MACHINE:00000000    COND_DEFAULT:00000000
   +0000B4  Q_DATA_TOKEN:00000000   FDBK:00000000     ABNAME:........
            Machine State
   +000348  MCH_EYE:....
   +000350  MCH_GPR00:00000000      MCH_GPR01:00000000
   +000358  MCH_GPR02:00000000      MCH_GPR03:00000000
   +000360  MCH_GPR04:00000000      MCH_GPR05:00000000
   +000368  MCH_GPR06:00000000      MCH_GPR07:00000000
   +000370  MCH_GPR08:00000000      MCH_GPR09:00000000
   +000378  MCH_GPR10:00000000      MCH_GPR11:00000000
   +000380  MCH_GPR12:00000000      MCH_GPR13:00000000
   +000388  MCH_GPR14:00000000      MCH_GPR15:00000000
   +000390  MCH_PSW:00000000 00000000     MCH_ILC:0000      MCH_IC1:00
   +00039B  MCH_IC2:00  MCH_PFT:00000000        MCH_FLT_0:00000000 00000000
   +0003A8  MCH_FLT_2:00000000 00000000   MCH_FLT_4:00000000 00000000
   +0003B8  MCH_FLT_6:00000000 00000000   MCH_EXT:00000000
   +000418  MCH_FLT_1:00000000 00000000   MCH_FLT_3:00000000 00000000
   +000428  MCH_FLT_5:00000000 00000000   MCH_FLT_7:00000000 00000000
   +000438  MCH_FLT_8:00000000 00000000   MCH_FLT_9:00000000 00000000
   +000448  MCH_FLT_10:00000000 00000000
   +000450  MCH_FLT_11:00000000 00000000
   +000458  MCH_FLT_12:00000000 00000000
   +000460  MCH_FLT_13:00000000 00000000
   +000468  MCH_FLT_14:00000000 00000000
   +000470  MCH_FLT_15:00000000 00000000        MCH_FPC:00000000
   +00047C  MCH_APF_FLAGS:00

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

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 7 of 11)*

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

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

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

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 8 of 11)*

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

   MDST forward chain from CMXBDHEAD(1)

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

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

   MDST back chain from CMXBDHEAD(2)

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

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

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

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

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

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

[13]Information for enclave main

   [14]Information for thread 0676992000000000

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

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

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 9 of 11)*

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

     Thread Synchronization Enclave Latch Table (EPALT): 068D1544
       +000000 068D1544  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
       +000020 068D1564 - +00055F 068D1AA3             same as above
       +000560 068D1AA4  00000000 00000000 00000000 00000000  0646A478 00000000 00000000 00000000 |..................u.............|
       +000580 068D1AC4  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
       +0005A0 068D1AE4 - +00061F 068D1B63             same as above
       +000620 068D1B64  00000000 00000000 00000000 00000000  00000000 00000000 0646A478 00000000 |.........................u.....|
       +000640 068D1B84  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
       +000660 068D1BA4 - +0009FF 068D1F43             same as above
     HEAPCHK Option Control Block (HCOP): 068D3028
       +000000 068D3028  C8C3D6D7 00000024 00000001 00000000  00000000 068F3028 068D304C 00000000 |HCOP.....................<....|
       +000020 068D3048  00000000 C8C3C6E3 00000200 00000000  00000000 00000000 00000000 00000000 |....HCFT........................|
     HEAPCHK Element Table (HCEL) for Heapid 068F020C :
     Header: 068F3028
       +000000 068F3028  C8C3C5D3 068DD028 00000000 068F020C  000001F4 00000001 00000001 00000000 |HCEL.............4............|
                         Address  Seg Addr Length              Address  Seg Addr Length
     Table: 068F3048
       +000000 068F3048  068F2020 068F2000 000002A8 00000000  00000000 00000000 00000000 00000000 |...........y....................|
     HEAPCHK Element Table (HCEL) for Heapid 00000000 :
     Header: 068DD028
       +000000 068DD028  C8C3C5D3 00000000 068F3028 00000000  000001F4 00000007 00000007 00000000 |HCEL.............4............|

                         Address  Seg Addr Length              Address  Seg Addr Length
     Table: 068DD048
       +000000 068DD048  068D5020 068D5000 00000110 00000000  068D5130 068D5000 00000828 00000000 |..&...&.............&.........|
       +000020 068DD068  068D5958 068D5000 00000250 00000000  068D5BA8 068D5000 00000038 00000000 |......&....&......$y..&.........|
       +000040 068DD088  068D5BE0 068D5000 00000038 00000000  068D5C18 068D5000 00000010 00000000 |..$...&...........*...&.........|
       +000060 068DD0A8  068D5C28 068D5000 00000010 00000000  00000000 00000000 00000000 00000000 |..*...&.........................|
⋮
   [18]Language Environment Trace Table:

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

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

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

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

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

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 10 of 11)*

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

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

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

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

Exiting Language Environment Data
```

*Figure 14. Example of Formatted Output from LEDATA Verbexit (Part 11 of 11)*

## Sections of the Language Environment LEDATA verbexit formatted output

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

### [1] - [8] Summary

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

### [1] Summary Header

The summary header section contains:
- Address of Thread control block (TCB)
- Release number
- Address Space ID (ASID)

### [2] Active Members List

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

### [3] CEECAA

This section formats the contents of the Language Environment common anchor area (CAA). Refer to “The Common Anchor Area” on page 64 for a description of the fields in the CAA.

### [4] CEEPCB

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

**[5] CEERCB**

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

**[6] CEEEDB**

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

**[7] PMCB**

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

**[8] Run-Time Options**

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

**[9] Heap Storage Control Blocks**

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

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

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

This section includes a detailed heap segment report for each segment in the dump. For more information about the detailed heap segment report, see “Understanding the HEAP LEDATA output” on page 98.

**[10] Stack Storage Control Blocks**

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

This section formats:

- Storage management control block (SMCB)
- Chain of dynamic save areas (DSA)

  Refer to “The upward-growing (non-XPLINK) stack frame section” on page 62 or “The downward-growing (XPLINK) stack frame section” on page 63 for a description of the fields in the DSA.
- Chain of stack segment headers (STKH)

  An STKH immediately precedes each segment of stack storage.

**[11] Condition Management Control Blocks**

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

This section formats the chain of Condition Information Block Headers (CIBH) and Condition Information Blocks. The Machine State Information Block is contained with the CIBH starting with the field labeled MCH_EYE. Refer to “The condition information block” on page 71 for a description of fields in these control blocks.

**[12] Message Processing Control Blocks**

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

**[13] - [19] CEEDUMP Formatted Control Blocks**

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

**[13] Enclave Identifier**

This statement names the enclave for which information is provided.

**[14] Information for thread**

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

**[15] Traceback**

There will be one or more Traceback sections, depending on the setting of the NTHREADS parameter in the VERBEXIT LEDATA invocation. For a description of NTHREADS, see “Report type parameters” on page 81.

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

- Stack frame (DSA) address
- Program unit

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

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

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

  This field contains no Language Environment data.

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

**[16] Control Blocks Associated with the Thread**

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

**[17] Enclave Control Blocks**

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

**[18] Language Environment Trace Table**

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

**[19] Process Control Blocks**

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

## Understanding the HEAP LEDATA output

The Language Environment IPCS Verbexit LEDATA generates a detailed heap segment report when the HEAP option is used with the DETAIL option, or when the SM,DETAIL option is specified. The detailed heap segment report is useful when trying to pinpoint damage because it provides very specific information. The report describes the nature of the damage, and specifies where the actual damage occurred. The report can also be used to diagnose storage leaks, and to identify heap fragmentation. Figure 15 on page 99 illustrates the output produced by specifying the HEAP option. “Heap report sections of the LEDATA output” on page 102 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 15. Example Formatted Detailed Heap Segment Report from LEDATA Verbexit (Part 1 of 4)*

```
   Map of Heap Segment 24A91000

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

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

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

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

   Free Storage Tree for Heap Segment 259AC000

          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 FFFFFFFE 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.......................|

 259ADBD8: Allocated storage element, length=00000028. To display: IP LIST 259ADBD8 LEN(X'00000028') ASID(X'0021')
 259ADBE0: 00000000 259950A8 70004000 00000000  00000000 00000000 00000000 00000000 |.....r.&y.......................|

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

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

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

*Figure 15. Example Formatted Detailed Heap Segment Report from LEDATA Verbexit (Part 2 of 4)*

```
  Below Heap Control Blocks

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

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

  This is the last heap segment in the current heap.

  Free Storage Tree for Heap Segment 00044000

         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 15. Example Formatted Detailed Heap Segment Report from LEDATA Verbexit (Part 3 of 4)*

```
   This is the last heap segment in the current heap.

   Free Storage Tree for Heap Segment 259AE000

          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 15. Example Formatted Detailed Heap Segment Report from LEDATA Verbexit (Part 4 of 4)*

## Heap report sections of the LEDATA output

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

### [1]Free Storage Tree Report

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

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

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

Each node length is validated to ensure that it:

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

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

### [2]Heap Segment Map Report

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

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

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

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

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

## Diagnosing heap damage problems

Heap storage errors can occur when an application allocates a heap storage element that is too small for it to use, and therefore, accidently 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 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 16 on page 105 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 46. “C/C++-specific sections of the LEDATA output” on page 109 describes the information contained in the formatted output. Ellipses are used to summarize some sections of the dump.

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

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

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

[3]CEDB:  2471A5A4
   +000000  EYE:CEDB    SIZE:000004D0     PTR:2471A5A4      CLLST:24704B40
   +000010  CEELANG:0003      CASWITCH:0000     CLWA:2471B2DC
   +000018  CALTLWA:2471B62C        CCADDR:24702178   CFLGS:00000080
   +000028  CANCHOR:00000000        RPLLEN:00000000   ACBLEN:00000000
   +000034  LC:2471AA7C       VALID_HIGH:2483D6E0     _LOW:2483BD3C
   +000040  HEAD_FECB:00000000      ATEXIT_COUNT:00000000
   +000048  _EMPTY_COUNT:00000000   MAINPRMS:25993D08
   +000050  STDINFILE:2471A3A8      STDOUTFILE:24719FB8
   +000058  STDERRFILE:2471A1B0     CTYPE:2484029A    TZDFLT:00004650
   +000064  CINFO:2471AB8C    CMS_WRITE_DISK:4040     _DISK_SET:00000000
   +000070  MIN_FLT:00100000 00000000 00000000 00000000
   +000080  MAX_FLT:7FFFFFFF FFFFFFFF 71FFFFFF FFFFFFFF
   +000090  FLT_EPS:3C100000        DBL_EPS:34100000 00000000
   +0000A0  LDBL_EPS:26100000 00000000 18000000 00000000    FLAGS1:02000000
   +0000B4  MTF_MAINTASK_BLK:00000000     EMSG_SETTING:00   DEPTH:00000000
   +0000C0  SCREEN_WIDTH:00000000   USERID:IBMUSER.
   +0000CC  HEAP24_ANCHOR:00000000        TCIC:00000000     TKCLI:00000000
   +0000D8  ATEXIT_FUNCS01:00000000 00000000 00000000 00000000 00000000
   +0000EC  ATEXIT_FUNCS02:00000000 00000000 00000000 00000000 00000000

⋮
```

*Figure 16. Example Formatted C/C++ Output from LEDATA Verbexit (Part 1 of 5)*

```
   +000330  ATEXIT_FUNCS31:00000000 00000000 00000000 00000000 00000000
   +000344  ATEXIT_FUNCS32:00000000 00000000 00000000 00000000 00000000
   +000358  HEAD_FOREIGN_FECB:00000000    SNAP_DUMP_COUNT:00000000
   +000360  ENVIRON:00000000        GETENV_BUF:00000000
   +000368  _BUF_LEN:00000000       INSPECT_GLOBALS:00000000
   +000374  _JMP_BUFF:00025C44      _BACK_END:00000000      _FLAGS:00000000
   +000380  _TAB:00000000     INTOFFLIST:00000000     CGEN_CRENT:25993870
   +00038C  _CPRMS:000149D0   _CEDCXV:A489EB04        _CEDCOV:2489A69C
   +000398  _EPCBLIST:00000000      CAA_ADDR:00015920
   +0003A4  USERIDLENGTH:00000007   MAXUNGETCOUNT:0004
   +0003C4  IOGET_ANY:2493BFB0      _BELOW:2493B6D0   IOFREE_ANY:2493C470
   +0003D0  _BELOW:2493BC10   MTFMAINTASKBLK:00000000
   +0003E0  SIGTABLE:2471ADB4       INIT_STDIN:2471A3A8
   +0003E8  _STDOUT:24719FB8        _STDERR:2471A1B0        TABNUM:00000008
   +0003F8  FLAGS2:00000000   OPENMVS_FLAGS:00  MRPSTDR:2482D7F8
   +000408  MWPSTDR:2482DA00        MRPSTDC:2482C928
   +000410  MWPSTDC:2482CB30        OWRP1:24898BA4    OWRP3:2489EB04
   +00041C  STATIC_EDCOV:00000000   GETENV_BUF2:00000000
   +000424  _BUF2_LEN:00000000      DLCB_MUTEX:25993DA8     _CONDV:25993DAC
   +000430  EDCOV:2498B480    LCX:2471ACB4      MUTEX_ATTR:25993D48
   +000444  STOR_INIT:00003000      _INCR:00002000    DEMANGLE:00000000
   +000454  TEMPR15:00000000        TERMINATE:00000000
   +00045C  CXX_INV:00000000        D4_JOIN_MUTEX_ATTR:25993D98
   +000468  _MUTEX:25993D9C   _CONDV_ATTR:25993DA0    _CONDV:25993DA4
   +000474  DLLANCHOR:00000000      DLLLAST:00000000        MEM24P:000163C0
   +000480  RTLMUTEX_ARRAYPTR:25993D4C    MSGCATLIST:00000000
   +000488  SRCHP:00000000    ETOAP:00000000    ATOEP:00000000
   +000494  NDMGMTP:00000000        POPENP:00000000   RND48P:00000000
   +0004A0  BRK_HEAPID:00000000     _START:00000000   _CURRENT:00000000
   +0004AC  _END:00000000     RESTARTTABLE:2497BE48   SYSLOGP:00000000
   +0004BC  LOGIN_NAME:.........    PREV_UMASK_VAL:00000000

[4]CTHD:  24719964
   +000000  CTHDEYE:CTHD      SIZE:00000310     CTHDPTR:24719964
   +00000C  STORPTR:00000000        TOKPTR:24837440
   +000014  ASCTIME_RESULT:.........................
   +00002E  SNAP_DUMP_FLAG:00       GMTIME_BKDN:24719D4C
   +000034  TIMECALLED:00000000     DATECALLED:00000000
   +00003C  DTCALLED:00000000       LOC_CALLED:00000000
   +000044  DOFMTO_DISCARDS:00000000      CERRNO:00000000   AMRC:24719854
   +000050  AMRC2:2471993C    GDATE:00000000    OPTARGV:00000000
   +00005C  OPTERRV:00000001        OPTINDV:00000001
   +000064  OPTOPTV:00000000        OPTSIND:00000000        DLGHTV:00000000
   +000070  TZONEV:00000000   GTDTERRV:00000000       OPTARGP:259938A8
   +00007C  OPTERRP:259938A4        OPTINDP:259938A0
   +000084  OPTOPTP:2599389C        DLGHTP:25993890   TZONEP:25993894
   +000090  GTDTERRP:259938B0       RNDSTGP:00000000
   +000098  LOCNAME:00000000        ENCRYPTP:00000000       CRYPTP:00000000
   +0000A4  RND48P:00000000   L64AP:00000000    WCSTOKP:00000000
   +0000B0  CUSERP:00000000   GPASSP:00000000   UTMPXP:00000000
   +0000BC  NDMGMTP:00000000        RECOMP:00000000   STACKPTR:00000000
   +0000C8  STACKSIZE:00000000      STACKFLAGS:00     000000
   +0000D0  MCVTP:00000000    H_ERRNO:00000000        SD:FFFFFFFF
   +0000DC  HOSTENT_DATA_P:00000000       HOSTENT_P:00000000
   +0000E4  NETENT_DATA_P:00000000        NETENT_P:00000000
   +0000EC  PROTOENT_DATA_P:00000000      PROTOENT_P:00000000
   +0000F4  SERVENT_DATA_P:00000000       SERVENT_P:00000000
   +0000FC  NTOA_BUF:..................   __LOC1V:00000000
```

*Figure 16. Example Formatted C/C++ Output from LEDATA Verbexit (Part 2 of 5)*

```
   +000118  HERRNOP:259938AC         __LOC1P:2599388C        REXECP:00000000
   +000124  CXXEXCEPTION:00000000   TEMPDCBE:24719644
   +00012C  T_ERRNOV:00000000       T_ERRNOP:25993870
   +000148  THD_STORAGE:00000000    CONTEXT_LINK:00000000   FLAGS1:00000000
   +000154  LABEL_VAR:24719E74      ABND_CODE:00000000
   +00015C  RSN_CODE:00000000       STRFTIME_ERADTCALLED:00000000
   +000164  STRFTIME_ERADATECALLED:00000000
   +000168  STRFTIME_ERATIMECALLED:00000000
   +00016C  STRFTIME_ERAYEARCALLED:00000000     MBRLEN_STATE:0000
   +000172  MBRTOWC_STATE:0000      WCRTOMB_STATE:0000
   +000176  MBSRTOWCS_STATE:0000    WCSRTOMBS_STATE:0000    MBLEN_STATE:0000
   +00017C  MBTOWC_STATE:0000       CURR_HEAP_ID:00000000
   +000184  CURR_CAA:00000000       CURR_MOD_HANDLE:00000000
   +00018C  CURR_BMR:00000000       CU_LIST:00000000        CURR_STATUS:00
   +000198  RAND_NEXT:00000001      STRERRORBUF:247193BC
   +0001A0  TMPAREA:00000000        IOWORKAREA:2471971C
   +0001A8  TEMPDCB:00050088        TEMPJFCB:000500E8
   +0001B0  TEMPDSCB:2471967C       NAMEBUF:259A0BC8
   +0001B8  ERRNO_JR:00000000       RET_STRUCT:00000000
   +0001C0  BKDN_IS_LOCALTIME:00000000    SWPRINTF_SIZE:00008000
   +0001C8  SWPRINTF_BUF:00000000   S99P:24719624     MUTEXCTARRAY:24719EAC
   +0001D4  STRFTIME_ERANAMECALLED:00000000     FCB_MUTEX:00000000
   +000204  HSPABHWA:24719364       MUTEX_SAVE:24719EFC
   +000210  INITIAL_CPU_TIME:4D000000 00053ADF        FCB_MUTEX_OK:00000001
   +00021C  FCB_MUTEX_SAVE:00000000       ENTRY_ADDRTABLESIZE:00000000
   +000224  ADDRESS:00000000        NUMBEROFNAMES:00000000
   +00022C  NAMES1:.........................
   +000245  NAMES2:.........................
   +00025E  NAMES3:.........................
   +000277  NAMES4:.........................
   +000290  NAMES5:.........................
   +0002A9  NAMES6:.........................
   +0002C4  ENTRY_SITETABLESIZE:00000000        KIND:00
   +0002CC  NUM_ADDRS:00000000
   +0002D0  ADDRESSES:00000000 00000000 00000000 00000000 00000000 00000000
   +0002E8  NAME:00000000 00000000 00000000 00000000 00000000 00000000

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

[6]CIO:  247191AC
   +000000  EYE:CIO     SIZE:00000088     PTR:00000000      FLG1:08
   +00000D  FLG2:00     FLG3:00     FLG4:00     DUMMYF:24719234
   +000014  EDCZ24:A49BF4E0   FCBSTART:259A0408       DUMMYFCB:2471924C
   +000020  MFCBSTART:259A05F0      IOANYLIST:2599F000
   +000028  IOBELOWLIST:00050000    FCBDDLIST:24719FCC
   +000030  PERRORBUF:24719074      TMPCOUNTER:00000000
   +000038  TEMPMEM:00000000        PROMPTBUF:00000000      IO24:000502D0
   +000044  IOEXITS:00050F4C        TERMINALCHAIN:00000000
   +00004C  VANCHOR:00000000        XTI:00000000      ENOWP24:249BFFD0
   +000058  MAXNUMDESCRPS:00000000        DESCARRAY:00000000
   +000060  PROC_RES_P:00000000     TEMPFILENUM:00000000    CSS:00000000
   +00006C  DUMMY_NAME:.....     HOSTNAME_CACHE:00000000
   +000078  HOSTADDR_CACHE:00000000
```

*Figure 16. Example Formatted C/C++ Output from LEDATA Verbexit (Part 3 of 5)*

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

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

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

*Figure 16. Example Formatted C/C++ Output from LEDATA Verbexit (Part 4 of 5)*

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

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

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

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

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

     JFCB:  00050F68
   +000000  JFCBDSNM:IBMUSER.PAHBAT.JOB00018.D0000101.?
   +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       JFCFLGS1:00
.
.

Dummy FCB encountered at location 2471924C

Exiting CRTL Environment Data
```

*Figure 16. Example Formatted C/C++ Output from LEDATA Verbexit (Part 5 of 5)*

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

For the LEDATA output:

**[1] CGEN**

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

**[2] CGENE**

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

**[3] CEDB**

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

**[4] CTHD**

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

**[5] CPCB**

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

**[6] CIO**

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

**[7] File Control Blocks**

This section formats the C/C++ file control block (FCB). The FCB and its related control blocks represent the information needed by each open stream.

**Related Control Blocks**

**FSCE** The file specific category extension control block. The FSCE represents the specific type of IO being performed. The following is a list of FSCEs that may be formatted.

OSNS — OS no seek

OSFS — OS fixed text

OSVF — OS variable text

OSUT — OS undefined format text

Other FSCEs will be displayed using a generic overlay.

**OSIO** The OS IO interface control block.

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

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

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

## Understanding the COBOL-specific LEDATA output

The Language Environment IPCS Verbexit LEDATA generates formatted output of COBOL-specific control blocks from a system dump when the ALL parameter is specified and COBOL is active in the dump. Figure 17 on page 112 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 114 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:O    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 17. Example Formatted COBOL Output from LEDATA Verbexit (Part 1 of 2)*

```
[5]   TGT:  00050248
   +000048  IDENT:3TGT  LVL:05      FLAGS:40020220    RUNCOM:00049038
   +00005C  COBVEC:0001A7FC   #FCBS:00000000    WS_LEN:00000000
   +000070  SMG_WRK:00000000        CAA:00018920      LEN:00000154
   +00008C  EXT_FCBS:00000000       OUTDD:SYSOUT
   +0000AC  CALC_RSA:00000000 00000000 00000000 00000000 00000000 00000000
   +0000C4  ------->:00000000 00000000 00000000 00000000 00000000 00000000
   +0000DC  ------->:00000000       ABINF:000500A5    TESTINF:00000000
   +000100  PGMADDR:0004FF88        1STFCB:00000000   WS_ADDR:00000000
   +000118  1STEXTFCB:00000000

     CLLE:  00049440
   +000000  PGMNAME:PARM1     OPEN_NON_EXT_FILES:0000       TGT_FLAGS:00
   +00000C  LANG_LST:0004EF98       INFO_FLAGS:8891   LOAD_ADDR:8004DFE0
   +000018  TGT_ADDR:0004E258       LE_TOKEN:0BF150A0       FLAGS2:00

      TGT:  0004E258
   +000048  IDENT:3TGT  LVL:05      FLAGS:40020220    RUNCOM:00049038
   +00005C  COBVEC:0001A7FC   #FCBS:00000000    WS_LEN:00000000
   +000070  SMG_WRK:00000000        CAA:00018920      LEN:00000144
   +00008C  EXT_FCBS:00000000       OUTDD:SYSOUT
   +0000AC  CALC_RSA:00000000 00000000 00000000 00000000 00000000 00000000
   +0000C4  ------->:00000000 00000000 00000000 00000000 00000000 00000000
   +0000DC  ------->:00000000       ABINF:0004E0FD    TESTINF:00000000
   +000100  PGMADDR:0004DFE0        1STFCB:00000000   WS_ADDR:00000000
   +000118  1STEXTFCB:00000000

     CLLE:  00049370
   +000000  PGMNAME:PARM0     OPEN_NON_EXT_FILES:0000       TGT_FLAGS:00
   +00000C  LANG_LST:0004CF98       INFO_FLAGS:8891   LOAD_ADDR:8004BFF8
   +000018  TGT_ADDR:0004C260       LE_TOKEN:0BF15084       FLAGS2:00

      TGT:  0004C260
   +000048  IDENT:3TGT  LVL:05      FLAGS:40020220    RUNCOM:00049038
   +00005C  COBVEC:0001A7FC   #FCBS:00000000    WS_LEN:00000000
   +000070  SMG_WRK:00000000        CAA:00018920      LEN:00000140
   +00008C  EXT_FCBS:00000000       OUTDD:SYSOUT
   +0000AC  CALC_RSA:00000000 00000000 00000000 00000000 00000000 00000000
   +0000C4  ------->:00000000 00000000 00000000 00000000 00000000 00000000
   +0000DC  ------->:00000000       ABINF:0004C115    TESTINF:00000000
   +000100  PGMADDR:0004BFF8        1STFCB:00000000   WS_ADDR:00000000
   +000118  1STEXTFCB:00000000

     CLLE:  00049328
   +000000  PGMNAME:CALLSUBX  OPEN_NON_EXT_FILES:0000       TGT_FLAGS:00
   +00000C  LANG_LST:00000000       INFO_FLAGS:9881   LOAD_ADDR:80007DE8
   +000018  TGT_ADDR:00008220       LE_TOKEN:00000000       FLAGS2:00

      TGT:  00008220
   +000048  IDENT:3TGT  LVL:05      FLAGS:60020220    RUNCOM:00049038
   +00005C  COBVEC:0001A7FC   #FCBS:00000000    WS_LEN:0000002C
   +000070  SMG_WRK:00000000        CAA:00018920      LEN:00000150
   +00008C  EXT_FCBS:00000000       OUTDD:SYSOUT
   +0000AC  CALC_RSA:00000000 00000000 00000000 00000000 00000000 00000000
   +0000C4  ------->:00000000 00000000 00000000 00000000 00000000 00000000
   +0000DC  ------->:00000000       ABINF:00007F34    TESTINF:00000000
   +000100  PGMADDR:00007DE8        1STFCB:00000000   WS_ADDR:000083C0
   +000118  1STEXTFCB:00000000

Exiting COBOL Environment Data
```

*Figure 17. 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**

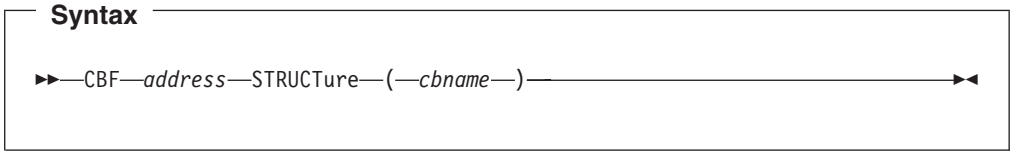
This section formats the COBOL TGT control blocks.

## 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**




*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 115. 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 18 on page 115 :

```
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        SSEXPNT:00000000        EDB:000148B0
+0002F4  PCB:00014558      EYEPTR:00015878   PTR:00015890
+000300  GETS1:00012730    SHAB:00000000     PRGCK:00000004    FLAG1:00
+000310  URC:00000000      ESS:00042F00      LESS:00022F00
+00031C  OGETS:000120F8    OGETLS:00000000   PICICB:00000000
+000328  GETSX:00000000    GOSMR:0000  LEOV:00000000
+000334  SIGSCTR:00000000        SIGSFLG:00000000
+00033C  THDID:80000000  00000000      DCRENT:00000000
+000348  DANCHOR:00000000        CTOC:00000000     RCB:00013918
+000354  CICSRSN:00000000        MEMBR:000161C8
+00035C  SIGNAL_STATUS:00000000        FOR1:00000000    FOR2:00000000
+000378  THREADHEAPID:00000000    SIGNGPTR:00015C24           SIGNG:00000001
+000398  FORDBG:00000000   AB_STATUS:00      AB_GR0:00000000
+0003A4  AB_ICD1:00000000        AB_ABCC:00000000             AB_CRC:00000000
+0003D4  SMCB:00015F70     ERRCM:06412AB0    MIB_PTR:00000000
+000434  THDSTATUS:00000000      TICB_PTR:06413840
+00047C  FWD_CHAIN:00015890      BKWD_CHAIN:00015890
```

*Figure 18. The CAA Formatted By The CBFORMAT IPCS Command*

For more information on using the IPCS CBF command refer to the ″CBFORMAT subcommand″ section in *z/OS MVS IPCS Commands*, SA22-7594.

*Table 8. Language Environment Control Blocks Which Can Be Individually Formatted*

| Control Block | Description |
|---|---|
| CEEADHP | Additional Heap Control Block |
| CEECAA | Common Anchor Area |
| CEECIB | Condition Information Block |
| CEECIBH | Condition Information Block Header |
| CEECMXB | Message Services Block |
| CEEDSA | Dynamic Storage Area |
| CEEDSATR | XPLINK Transition Area |
| CEEDSAX | Dynamic Storage Area (XPLINK style) |
| CEEEDB | Enclave Data Block |
| CEEENSM | Enclave Level Storage Management |
| CEEHANC | Heap Anchor Node |

*Table 8. Language Environment Control Blocks Which Can Be Individually Formatted (continued)*

| Control Block | Description |
|---|---|
| CEEHCOM | CEL Exception Manager Communications Area |
| CEEHPCB | Thread Level Heap Control Block |
| CEEHPSB | Heap Statistics Block |
| CEEMDST | Message Destination |
| CEEMGF | Mapping of the Message Formatter (IBM1MGF) |
| CEEPCB | Process Control Block |
| CEEPMCB | Program Management Control Block |
| CEERCB | Region Control Block |
| CEESKSB | Stack Statistics Block |
| CEESMCB | Storage Management Control Block |
| CEESTKH | Stack Header Block |
| CEESTKHX | Stack Header Block (xplink style) |
| CEESTSB | Storage Report Statistics Block |
| CEETMXB | Thread Level Messages Extension Block |

## Requesting a Language Environment trace for debugging

Language Environment provides an in-storage, wrapping trace facility that can reconstruct the events leading to the point where a dump is taken. The trace facility can record two types of events: entry and exit library calls and, if the POSIX run-time option is set to ON, user mutex and condition variable activity such as init, lock/unlock, and wait. Language Environment produces a trace table in its dump report under the following conditions:

- The CEE3DMP callable service is invoked with the BLOCKS option and the TRACE run-time option is set to ON.
- The TRACE run-time option is set to NODUMP and the TERMTHDACT run-time option is set to DUMP, UADUMP, TRACE, or UATRACE.
- The TRACE run-time option is set to DUMP (the default).

For more information about the CEE3DMP callable service, the TERMTHDACT run-time option, or the TRACE run-time option, see *z/OS Language Environment Programming Reference*.

The TRACE run-time option activates Language Environment run-time library tracing and controls the size of the trace buffer, the type of trace events to record, and it determines whether a dump containing only the trace table should be unconditionally taken when the application (enclave) terminates. The trace table contents can be written out either upon demand or at the termination of an enclave.

The contents of the Language Environment dump depend on the values set in the TERMTHDACT run-time option. Under abnormal termination, the following dump contents are generated:

- TERMTHDACT(QUIET) generates a Language Environment dump containing the trace table only
- TERMTHDACT(MSG) generates a Language Environment dump containing the trace table only

- TERMTHDACT(TRACE) generates a Language Environment dump containing the trace table and the traceback
- TERMTHDACT(DUMP) generates a Language Environment dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block)
- TERMTHDACT(UAONLY) generates a system dump of the user address space
- TERMTHDACT(UATRACE) generates a Language Environment dump that contains traceback information, and a system dump of the user address space
- TERMTHDACT(UADUMP) generates a Language Environment dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block), and a user address space dump
- TERMTHDACT(UAIMM) generates a system dump of the user address space of the original abend or program interrupt that occurred prior to the Language Environment condition manager processing the condition.

**Note:** Under CICS, UAIMM yields UAONLY behavior. Under non-CICS, TRAP(ON,NOSPIE) must be in effect. When TRAP(ON,SPIE) is in effect, UAIMM yields UAONLY behavior. For software raised conditions or signals, UAIMM behaves the same as UAONLY.

Under normal termination, the following dump contents are generated:

- Independent of the TERMTHDACT setting, Language Environment generates a dump containing the trace table only based on the TRACE run-time option

Language Environment quiesces all threads that are currently running except for the thread that issued the call to CEE3DMP. When you call CEE3DMP in a multithread environment, only the current thread is dumped. Enclave- and process-related storage could have changed from the time the dump request was issued.

## Locating the trace dump

If your application calls CEE3DMP, the Language Environment dump is written to the file specified in the FNAME parameter of CEE3DMP (the default is CEEDUMP).

If your application is running under TSO or batch, and a CEEDUMP DD is not specified, Language Environment writes the CEEDUMP to the batch log (SYSOUT=* by default). You can change the SYSOUT class by specifying a CEEDUMP DD, or by setting the environment variable, *_CEE_DMPTARG=SYSOUT(x)*, where x is the preferred SYSOUT class.

If your application is running under z/OS UNIX and is either running in an address space you issued a `fork()` to, or if it is invoked by one of the `exec` family of functions, the dump is written to the hierarchical file system (HFS). Language Environment writes the CEEDUMP to one of the following directories in the specified order:

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

The name of this file changes with each dump and uses the following format:

```
/path/Fname.Date.Time.Pid
```

**path** The path determined from the above algorithm.

**Fname**
The name specified in the FNAME parameter on the call to CEE3DMP (default is CEEDUMP).

**Date** The date the dump is taken, appearing in the format `YYYYMMDD` (such as 19980918 for September 18, 1998).

**Time** The time the dump is taken, appearing in the format `HHMMSS` (such as 175501 for 05:55:01 p.m.).

**Pid** The process ID the application is running in when the dump is taken.

## Using the Language Environment trace table format in a dump report

The Language Environment trace table is established unconditionally at enclave initialization time if the TRACE run-time option is set to ON. All threads in the enclave share the trace table; there is no thread-specific table, nor can the table be dynamically extended or enlarged.

## Understanding the trace table entry (TTE)

Each trace table entry is a fixed-length record consisting of a fixed-format portion (containing such items as the timestamp, thread ID, and member ID) and a member-specific portion. The member-specific portion has a fixed length, of which some (or all) can be unused. For information about how participating products use the trace table entry, refer to the product-specific documentation. The format of the trace table entry is as follows:

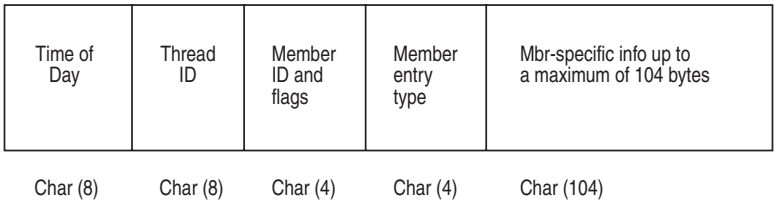



*Figure 19. Format of the Trace Table Entry*

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-rime library (RTL) function entry and exits |
| **2** | Trace all RTL mutex init/destroy and lock/unlock |
| **3** | Trace all RTL function entry and exits, and all mutex init/destroy and lock/unlock |
| **8** | Trace all RTL storage allocation/deallocation |

***When LE=1 is specified:*** The following C/C++ records may be generated.

*Table 9. LE=1 Entry Records*

| Member ID | Record Type | Description |
|---|---|---|
| 03 | 00000001 | Base C Library function Entry |
| 03 | 00000002 | Base C Library function Exit |
| 03 | 00000003 | Posix C Library function Entry |
| 03 | 00000004 | Posix C Library function Exit |
| 03 | 00000005 | XPLINK Base or Posix C Library function Entry |
| 03 | 00000006 | XPLINK Base or Posix C Library function Exit |

For a detailed description of these records, see “C/C++ Contents of the Language Environment Trace Tables” on page 167.

***When LE=2 is specified:*** The following Language Environment records may be generated.

*Table 10. LE=2 Entry Records*

| Member ID | Record Type | Class | Event | Description |
|---|---|---|---|---|
| 01 | 00000101 | LT | A | Latch Acquire |

*Table 10. LE=2 Entry Records (continued)*

| Member ID | Record Type | Class | Event | Description |
|---|---|---|---|---|
| 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 | EO | Latch already owned (not acquired) |
| 01 | 000002FD | LE | EUO | Latch unowned (not released) |
| 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 | BI | 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 | 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 | ENM | 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) |

*Table 10. LE=2 Entry Records (continued)*

| Member ID | Record Type | Class | Event | Description |
|---|---|---|---|---|
| 01 | 000004F0 | ME | EP | Mutex program check |
| 01 | 000004FA | ME | EDU | Mutex destroy failed (uninitialized) |
| 01 | 000004FB | ME | EUI | Mutex uninitialized |
| 01 | 000004FC | ME | EUO | Mutex unowned (not released) |
| 01 | 000004FD | E | EO | Mutex already owned (not acquired) |
| 01 | 000004FE | ME | EIN | Mutex initialization failed (duplicate) |
| 01 | 00000508 | CV | MR | CV release mutex |
| 01 | 00000509 | CV | MA | CV reacquire mutex |
| 01 | 0000050A | CV | MW | CV mutex wait |
| 01 | 0000050B | CV | MAW | CV reacquire mutex after wait |
| 01 | 0000050C | CV | CW | CV condition wait |
| 01 | 0000050D | CV | CTW | CV condition timeout |
| 01 | 0000050E | CV | CWP | CV wait posted |
| 01 | 0000050F | CV | CWI | CV wait interrupted |
| 01 | 00000510 | CV | CTO | CV wait timeout |
| 01 | 00000511 | CV | CSS | CV condition signal success |
| 01 | 00000512 | CV | CSM | CV condition signal miss |
| 01 | 00000513 | CV | CBS | CV condition broadcast success |
| 01 | 00000514 | CV | CBM | CV condition broadcast miss |
| 01 | 00000515 | CV | IN | CV initialize |
| 01 | 00000516 | CV | DS | CV destroy |
| 01 | 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 |

*Table 10. LE=2 Entry Records (continued)*

| Member ID | Record Type | Class | Event | Description |
|---|---|---|---|---|
| 01 | 000006FE | CE | EIN | CV 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 119.

**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 |

*Table 12. LE=8 Entry Records (continued)*

| Member ID | Record Type | Description |
|---|---|---|
| 03 | 00000001 | Storage allocation exit |

For a detailed description of these records, see “C/C++ Contents of the Language Environment Trace Tables” on page 167.

## Sample dump for the trace table entry

The following is an example of a dump of the trace table when you specify the LE=1 suboption (the library call/return trace):

```
⋮
 Enclave Control Blocks:
   EDB: 0001E920
     +000000 0001E920  C3C5C5C5 C4C24040 C5000001 00020E68  0001F040 00000000 00000000 00000000  |CEEEDB  E.........0 ............|

⋮
   MEML: 00020E68
     +000000 00020E68  00000000 00000000 0007C5B8 00000000  00000000 00000000 0007C5B8 00000000  |.........E.............E.....|

⋮
   Language Environment Trace Table:
    Most recent trace entry is at displacement: 001B80

        Displacement                      Trace Entry in Hexadecimal                                    Trace Entry in EBCDIC
        ------------ ------------------------------------------------------------------------  --------------------------------
          +000000    Time 21.41.57.595359    Date 2001.08.26    Thread ID... 8000000000000000
          +000010    Member ID.... 03    Flags..... 000000    Entry Type..... 00000001
          +000018    6D6DA289 9589A3F6 F46D6D83 82836D83  938296A2 F2F46D89 96A2A399 8581946D  |__sinit64__cbc_clbos24_iostream_|
          +000038    60606E4D F1F9F35D 406D6D87 85A38382  4D5D4040 40404040 40404040 40404040  |-->(193) __getcb()              |
          +000058    40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
          +000078    40404040 40404040                                                         |                                |

          +000080    Time 21.41.57.595367    Date 2001.08.26    Thread ID... 8000000000000000
          +000090    Member ID.... 03    Flags..... 000000    Entry Type..... 00000002
          +000098    4C60604D F1F9F35D 40D9F1F5 7EF0F0F0  F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0  |<--(193) R15=00000000 ERRNO=0000|
          +0000B8    F0F0F0F0 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |0000............................|
          +0000D8    00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
          +0000F8    00000000 00000000                                                         |........                        |

          +000100    Time 21.41.57.595374    Date 2001.08.26    Thread ID... 8000000000000000
          +000110    Member ID.... 03    Flags..... 000000    Entry Type..... 00000003
          +000118    6D6DA289 9589A3F6 F46D6D83 82836D83  938296A2 F2F46D89 96A2A399 8581946D  |__sinit64__cbc_clbos24_iostream_|
          +000138    60606E4D F1F1F35D 406D6D89 A2D796A2  89A7D695 4D5D4040 40404040 40404040  |-->(113) __isPosixOn()          |
          +000158    40404040 40404040 40404040 40404040  40404040 40404040 40000000 00000000  |                          .......|
          +000178    00000000 00000000                                                         |........                        |
          +000180    Time 21.41.57.595380    Date 2001.08.26    Thread ID... 8000000000000000
          +000190    Member ID.... 03    Flags..... 000000    Entry Type..... 00000004
          +000198    4C60604D 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 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 40404040  |                                |
          +000278    40404040 40404040                                                         |                                |

          +000280    Time 21.41.57.595690    Date 2001.08.26    Thread ID... 8000000000000000
          +000290    Member ID.... 03    Flags..... 000000    Entry Type..... 00000002
          +000298    4C60604D F1F2F45D 40D9F1F5 7EF2F4C2  F6C4F8C5 F840C5D9 D9D5D67E F0F0F0F0  |<--(124) R15=24B6D8E8 ERRNO=0000|
          +0002B8    F0F0F0F0 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |0000............................|
          +0002D8    00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
          +0002F8    00000000 00000000                                                         |........                        |

          +000300    Time 21.41.57.595743    Date 2001.08.26    Thread ID... 8000000000000000
          +000310    Member ID.... 03    Flags..... 000000    Entry Type..... 00000001
          +000318    8785A394 9684856D 86999694 6D86844D  8995A35D 40404040 40404040 40404040  |getmode_from_fd(int)            |
          +000338    60606E4D F1F9F35D 406D6D87 85A38382  4D5D4040 40404040 40404040 40404040  |-->(193) __getcb()              |
          +000358    40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
          +000378    40404040 40404040                                                         |                                |

          +000380    Time 21.41.57.595746    Date 2001.08.26    Thread ID... 8000000000000000
          +000390    Member ID.... 03    Flags..... 000000    Entry Type..... 00000002
          +000398    4C60604D F1F9F35D 40D9F1F5 7EF0F0F0  F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0  |<--(193) R15=00000000 ERRNO=0000|
          +0003B8    F0F0F0F0 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |0000............................|
          +0003D8    00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
          +0003F8    00000000 00000000                                                         |........                        |

⋮
```

*Figure 20. 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.

© Copyright IBM Corp. 1991, 2004

# Chapter 4. Debugging C/C++ Routines

This chapter provides specific information to help you debug applications that contain one or more C/C++ routines. It also provides information about debugging C/C++applications compiled with XPLINK. It includes the following topics:

- Debugging C/C++ I/O routines
- Using C/C++ compiler listings
- Generating a Language Environment dump of a C/C++ routine
- Generating a Language Environment dump of a C/C++ routine with XPLINK
- Finding C/C++ information in a Language Environment dump
- Debugging example of C/C++ routines
- Debugging example of C/C++ routines with XPLINK

There are several debugging features that are unique to C/C++ routines. Before examining the C/C++ techniques to find errors, you might want to consider the following areas of potential problems:

- If you suspect that you are using uninitialized storage, you may want to use the `STORAGE` run-time option.
- If you are using the `fetch()` function, refer to *z/OS C/C++ Programming Guide* to ensure that you are creating the fetchable module correctly.
- If you are using `DLL`s, refer to *z/OS C/C++ Programming Guide* to ensure that you are using the DLL correctly.
- For non-System Programming C routines, ensure that the entry point of the load module is CEESTART.
- You should avoid:
  - Incorrect casting
  - Referencing an array element with a subscript outside the declared bounds
  - Copying a string to a target with a shorter length than the source string
  - Declaring but not initializing a pointer variable, or using a pointer to allocated storage that has already been freed

  If a routine exception occurred and you need more information than the condition handler provided, run your routine with the following run-time options, TRAP(ON, NOSPIE) and TERMTHDACT(UAIMM). Setting these run-time options generates a system dump of the user address space of the original abend or program interrupt prior to the Language Environment condition manager processing the condition. After the system dump is taken by the operating system the Language Environment condition manager continues processing.

## Debugging C/C++ Input/Output Programs

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

## Using the __amrc and __amrc2 Structures

`__amrc`, a structure defined in **stdio.h**, can help you determine the cause of errors resulting from an I/O operation, because it contains diagnostic information (for example, the return code from a failed VSAM operation).

There are two structures:

`__amrc` (defined by type `__amrc_type`
`__amrc2` (defined by type `__amrc2_type`)

© Copyright IBM Corp. 1991, 2004

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 21 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;
    } __amrc_type;
```

*Figure 21. __amrc Structure*

Figure 22 shows the __amrc2 structure as it appears in **stdio.h**.

```
    struct {
[9]        long int   __error2;
[10]       FILE      *__fileptr;
[11]       long int   __reserved{6};
    }
```

*Figure 22. __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 13 on page 130.

**[8] __msg**
May contain the system error messages from read or write operations emitted from the DFSMS/MVS® SYNADAF macro instruction. Because the message can start with a hexadecimal address followed by a short integer, it is advisable to start printing at MSG+6 or greater so the message can be printed as a string. Because the message is not null-terminated, a maximum of 114 characters should be printed. This can be accomplished by specifying a printf format specifier as %.114s.

**[9] __error2**
A secondary error code. For example, an unsuccessful rename or remove operation places its reason code here.

**[10] __fileptr**
A pointer to the file that caused a SIGIOERR to be raised. Use an fldata() call to get the actual name of the file.

**[11] __reserved**
Reserved for future use.

## __last_op Values

The __last_op field is the most important of the __amrc fields. It defines the last I/O operation C/C++ was performing at the time of the I/O error. You should note that the structure is neither cleared nor set by non-I/O operations, so querying this field

outside of a SIGIOERR handler should only be done immediately after I/O operations. Table 13 lists __last_op values you could receive and where to look for further information.

*Table 13. __last_op Values and Diagnosis Information*

| Value | Further Information |
|---|---|
| __IO_INIT | Will never be seen by SIGIOERR exit value given at initialization. |
| __BSAM_OPEN | Sets __error with return code from OS OPEN macro. |
| __BSAM_CLOSE | Sets __error with return code from OS CLOSE macro. |
| __BSAM_READ | No return code (either __abend (errno == 92) or __msg (errno == 66) 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 __abend (errno == 92) or __msg (errno == 65) filled in). |
| __BSAM_CLOSE_T | Sets __error with return code from OS CLOSE TYPE=T. |
| __BSAM_BLDL | Sets __error with return code from OS BLDL macro. |
| __BSAM_STOW | Sets __error with return code from OS STOW macro. |
| __TGET_READ | Sets __error with return code from TSO TGET macro. |
| __TPUT_WRITE | Sets __error with return code from TSO TPUT macro. |
| __IO_DEVTYPE | Sets __error with return code from I/O DEVTYPE macro. |
| __IO_RDJFCB | Sets __error with return code from I/O RDJFCB macro. |
| __IO_TRKCALC | Sets __error with return code from I/O TRKCALC macro. |
| __IO_OBTAIN | Sets __error with return code from I/O CAMLST OBTAIN. |
| __IO_LOCATE | Sets __error with return code from I/O CAMLST LOCATE. |
| __IO_CATALOG | Sets __error with return code from I/O CAMLST CAT. The associated macro is CATALOG. |
| __IO_UNCATALOG | Sets __error with return code from I/O CAMLST UNCAT. The associated macro is CATALOG. |
| __IO_RENAME | Sets __error with return code from I/O CAMLST RENAME. |
| __SVC99_ALLOC | Sets __alloc structure with info and error codes from SVC 99 allocation. |
| __SVC99_ALLOC_NEW | Sets __alloc structure with info and error codes from SVC 99 allocation of NEW file. |
| __SVC99_UNALLOC | Sets __unalloc structure with info and error codes from SVC 99 unallocation. |
| __C_TRUNCATE | 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 fwrite() writing more data than the record can hold. Truncation is always rightmost data. There is no return code. |
| __C_FCBCHECK | Set when C or C++ FCB is corrupted. This is due to a pointer corruption somewhere. File cannot be used after this. |

*Table 13. __last_op Values and Diagnosis Information (continued)*

| Value | Further Information |
|---|---|
| __C_DBCS_TRUNCATE | 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 MB_CUR_MAX is 1. |
| __C_DBCS_SO_TRUNCATE | 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 MB_CUR_MAX is 1. |
| __C_DBCS_SI_TRUNCATE | 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 MB_CUR_MAX is 1. |
| __C_DBCS_UNEVEN | This occurs when an SI is written before the last double byte character is completed, thereby forcing C or C++ to fill in the last byte of the DBCS string with a padding byte X'FE'. Cannot happen if MB_CUR_MAX is 1. |
| __C_CANNOT_EXTEND | 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. |
| __VSAM_OPEN_FAIL | Set when a low level VSAM OPEN fails, sets __rc and __fdbk fields in the __amrc struct. |
| __VSAM_OPEN_ESDS | Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS. |
| __VSAM_OPEN_RRDS | Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS. |
| __VSAM_OPEN_KSDS | Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS. |
| __VSAM_OPEN_ESDS_PATH | Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS. |
| __VSAM_OPEN_KSDS_PATH | Does not indicate an error; set when the low level VSAM OPEN succeeds, and the file type is ESDS. |
| __VSAM_MODCB | Set when a low level VSAM MODCB macro fails, sets __rc and __fdbk fields in the __amrc struct. |
| __VSAM_TESTCB | Set when a low level VSAM TESTCB macro fails, sets __rc and __fdbk fields in the __amrc struct. |
| __VSAM_SHOWCB | Set when a low level VSAM SHOWCB macro fails, sets __rc and __fdbk fields in the __amrc struct. |
| __VSAM_GENCB | Set when a low level VSAM GENCB macro fails, sets __rc and __fdbk fields in the __amrc struct. |
| __VSAM_GET | Set when the last op was a low level VSAM GET; if the GET fails, sets __rc and __fdbk in the __amrc struct. |
| __VSAM_PUT | Set when the last op was a low level VSAM PUT; if the PUT fails, sets __rc and __fdbk in the __amrc struct. |
| __VSAM_POINT | Set when the last op was a low level VSAM POINT; if the POINT fails, sets __rc and __fdbk in the __amrc struct. |
| __VSAM_ERASE | Set when the last op was a low level VSAM ERASE; if the ERASE fails, sets __rc and __fdbk in the __amrc struct. |
| __VSAM_ENDREQ | Set when the last op was a low level VSAM ENDREQ; if the ENDREQ fails, sets __rc and __fdbk in the __amrc struct. |

*Table 13. __last_op Values and Diagnosis Information (continued)*

| Value | Further Information |
|---|---|
| __VSAM_CLOSE | Set when the last op was a low level VSAM CLOSE; if the CLOSE fails, sets __rc and __fdbk in the __amrc struct. |
| __QSAM_GET | __error is not set (if abend (errno == 92), __abend is set, otherwise if read error (errno == 66), look at __msg. |
| __QSAM_PUT | __error is not set (if abend (errno == 92), __abend is set, otherwise if write error (errno == 65), look at __msg. |
| __QSAM_TRUNC | This is an intermediate operation. You will only see this if an I/O abend occurred. |
| __QSAM_FREEPOOL | This is an intermediate operation. You will only see this if an I/O abend occurred. |
| __QSAM_CLOSE | Sets __error to result of OS CLOSE macro. |
| __QSAM_OPEN | Sets __error to result of OS OPEN macro. |
| __CMS_OPEN | Sets __error to result of FSOPEN. |
| __CMS_CLOSE | Sets __error to result of FSCLOSE. |
| __CMS_READ | Sets __error to result of FSREAD. |
| __CMS_WRITE | Sets __error to result of FSWRITE. |
| __CMS_STATE | Sets __error to result of FSSTATE. |
| __CMS_ERASE | Sets __error to result of FSERASE. |
| __CMS_RENAME | Sets __error to result of CMS RENAME command. |
| __CMS_EXTRACT | Sets __error to result of DMS EXTRACT call. |
| __CMS_LINERD | Sets __error to result of LINERD macro. |
| __CMS_LINEWRT | Sets __error to result of LINEWRT macro. |
| __CMS_QUERY | __error is not set. |
| __HSP_CREATE | Indicates last op was a DSPSERV CREATE to create a hiperspace for a hiperspace memory file. If CREATE fails, stores abend code in __amrc__code__abend__syscode, reason code in __amrc__code__abend__rc. |
| __HSP_DELETE | 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 __amrc__code__abend__syscode, reason code in __amrc__code__abend__rc. |
| __HSP_READ | Indicates last op was a HSPSERV READ from a hiperspace. If READ fails, stores abend code in __amrc__code__abend__syscode, reason code in __amrc__code__abend__rc. |
| __HSP_WRITE | Indicates last op was a HSPSERV WRITE to a hiperspace. If WRITE fails, stores abend code in __amrc__code__abend__syscode, reason code in __amrc__code__abend__rc. |
| __HSP_EXTEND | Indicates last op was a HSPSERV EXTEND during a write to a hiperspace. If EXTEND fails, stores abend code in __amrc__code__abend__syscode, reason code in __amrc__code__abend__rc. |
| __CICS_WRITEQ_TD | Sets __error with error code from EXEC CICS WRITEQ TD. |

*Table 13. __last_op Values and Diagnosis Information (continued)*

| Value | Further Information |
|---|---|
| __LFS_OPEN | Sets __error 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 *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. |
| __LFS_CLOSE | Sets __error 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 *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. |
| __LFS_READ | Sets __error 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 *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. |
| __LFS_WRITE | Sets __error 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 *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. |
| __LFS_LSEEK | Sets __error 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 *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. |
| __LFS_FSTAT | Sets __error 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 *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. |

## Displaying an Error Message with the perror() Function

To find a failing routine, check the return code of all function calls. After you have found the failing routine, use the `perror()` function after the routine to display the error message. `perror()` displays the string that you pass to it and an error message corresponding to the value of `errno`. `perror()` writes to the standard error stream (`stderr`).

If you need additional diagnostic information set the environment variable, _EDC_ADD_ERRNO2 to 1, and that will append the current errno2 value to the end of the perror() string.

Figure 23 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 23. Example of a Routine Using perror()*

## Using __errno2() to Diagnose Application Problems

Use __errno2() when diagnosing problems in a z/OS UNIX or an OpenExtensions application. This function enables C/C++ application programs to access diagnostic information returned to the C/C++ run-time library from an underlying kernel callable service. __errno2() returns the reason code of the last failing kernel callable service called by the C/C++ run-time library. The returned value is intended for diagnostic display purposes only. The function call is always successful.

**Note:** Since the __errno2() function returns the reason code of the kernel callable service that last failed, and not all function calls invoke the kernel, the value returned by __errno2() may be misleading.

Figure 24 is an example of a routine using __errno2().

```
#include <stdio.h>
#include <errno.h>
FILE *myfopen(const char *fn, const char *mode) {
   FILE *f;
   f = fopen(fn,mode);
   if (f==NULL) {
      perror("fopen() failed");
      printf("__errno2 = %08x\n", __errno2());
   }
   return(f);
}
```

*Figure 24. Example of a Routine Using __errno2()*

Figure 25 is an example of a routine using the environment variable _EDC_ADD_ERRNO2, and Figure 26 on page 135 shows the sample output from that routine.

```
#include <stdio.h>
#include <errno.h>

int main(void) {
   FILE *fp;

   /* add errno2 to perror message */
   setenv("_EDC_ADD_ERRNO2","1",1);

   fp = fopen("testfile.dat", "r");
   if (fp == NULL)
      perror("fopen error");

 }
```

*Figure 25. Example of a Routine Using _EDC_ADD_ERRNO2*

```
fopen error:  EDC5129I  No such file or directory.
(errno2=0x05620062)
```

*Figure 26. Sample Output of a Routine Using _EDC_ADD_ERRNO2*

## Using C/C++ Listings

The following sections discuss C/C++ listings generated when the executable program is created. They also explain how to use these listings to locate information, such as variable values and the timestamp, in the dump.

## Generating C/C++ Listings and Maps

The two techniques for creating an executable program are:

- When the executable program is to be stored in a PDSE or HFS, use the binder to combine the output from the C/C++ compiler.
- When the executable program is to be stored in a PDS, use Language Environment Prelinker Utility to combine the output from the C/C++ compiler and pass the prelinker output to the binder.

Note: Executable programs using XPLINK can only be created by using the binder.

The listings and maps created by the compile, prelink (optional), and link-edit steps provide many pieces of information necessary for performing problem analysis tasks. When creating an executable program without using the prelink step, the map of the Writable Static Area (WSA) is provided by the binder in the output listing in the C_WSA section.

In addition, the @STATIC is replaced by the binder with $PRIVnnnnnn and to find the source listing use the cross reference to associate $PRIVnnnnnn with the defining section name and use the section name to find the source in the module map. So, the output listing provided by the binder should be used when locating variables in executable programs created without using the prelink step.

When you are debugging, you can use various options depending upon which compiler you are using. The following section provides a overview of each listing and specifies the compiler option to use. For a detailed description of available listings, see *z/OS C/C++ User's Guide*.

*Table 14. Contents of Listing and Associated Compiler Options*

| Name | Compiler Option | Function |
|---|---|---|
| Pseudo-assembler listing | LIST | Generates a pseudo-assembler listing, which shows the source listing for the current routine. |
| Storage Offset Listing | XREF | Produces a storage offset listing, which includes in the source listing a cross reference table of names used in the routine and the line numbers on which they were declared or referenced, and a static map. |
| Structure Map | AGGREGATE (C only) | Causes a structure map to be included in the source listing. The structure map shows the layout of variables for the type `struct` or `union`. |

*Table 14. Contents of Listing and Associated Compiler Options (continued)*

| Name | Compiler Option | Function |
|---|---|---|
| Inline Report | INLRPT and INLINE(,REPORT,,) | Generates an inline report that summarizes all functions inlined and provides a detailed call structure of all the functions. |
| Prelinker Map | MAP (prelink option) | Creates the prelinker map when invoking the Prelinker. It is the default. You can use prelinker maps to determine the location of static and external C variables compiled with the RENT option and all C++ variables. |
| Link-edit Output Listing | MAP, LIST, XREF (linker option) | These options control the listing output from the link-edit process. |
| Source Listing | SOURCE | Generates the source listing, which contains the original source input statements and any compiler diagnostic messages issued for the source. |
| Cross-Reference Listing | XREF | Cross-reference table containing a list of the identifiers from the source program and the line numbers in which they appear. |
| External Symbol Cross Reference Listing | ATTR (C only) or XREF | Shows the original name and corresponding mangled name for each symbol. |
| Object File Map | IPA(MAP) LIST | Displays the names of the object files that were used as input to the IPA Link step. |
| Source File Map | IPA(MAP) LIST | Identifies the source files included in the object files. |
| Compiler Options Map | IPA(MAP) LIST | Identifies the compiler options that were specified during the IPA Compile step for each compilation unit that is encountered when the object file is processed. |
| Global Symbols Map | IPA(MAP) LIST | Shows how global symbols are mapped into members of global data structures by the global variable coalescing optimization process. |
| Inline Report for IPA Inliner | IPA(MAP) LIST | Describes the actions performed by the IPA Inliner. |

## C, C++, and C/C++ IPA Listings

The options for each listing vary depending upon which Compiler is used. The following section illustrates which options are available for each listing.

***C Compiler Listings:*** The following table specifies which listings are available for the C compiler, and which option(s) must be specified to obtain it.

*Table 15. C Compiler Listings*

| Name | Compiler Option |
|---|---|
| Source Program | SOURCE |
| Cross-Reference Listing | XREF |
| Structure and Union Maps | AGGREGATE |
| Inline Report | OPTIMIZE and INLINE(,REPORT,,) or INLRPT |
| Pseudo Assembly Listing | LIST |
| Storage Offset Listing | XREF |

***C++ Compiler Listings:*** The following table specifies which listings are available for the C++ compiler, and which option(s) must be specified to obtain it.

*Table 16. C++ Compiler Listings*

| Name | Compiler Option |
|---|---|
| Source Program | SOURCE |
| Cross-Reference Listing | ATTR and XREF |
| Inline Report | OPTIMIZE and INLINE(,REPORT,,) or INLRPT |
| Pseudo Assembly Listing | LIST |
| External Symbol Cross Reference Listing | ATTR or XREF |

***C/C++ IPA Link Step Listings:*** The following table specifies which listings are available for the C/C++ IPA Link Step, and which option(s) must be specified to obtain it.

*Table 17. C/C++ IPA Link Step Listings*

| Name | Compiler Option |
|---|---|
| Object File Map | IPA (MAP) LIST |
| Source File Map | IPA (MAP) LIST |
| Compiler Options Map | IPA (MAP) LIST |
| Global Symbols Map | IPA (MAP) LIST |
| Inline Report for IPA Inliner | IPA (MAP) LIST |
| Partition Map | IPA (MAP) LIST |

# Finding variables

You can determine the value of a variable in the routine at the point of interrupt by using the compiled code listing as a guide to its address, then finding this address in the Language Environment dump. 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

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 compile option. If you have C code compiled with the RENT option or C++ code (hereafter called RENT code) you must determine the base address of the WSA if you want to calculate the address of a static or external variable. Use the following table to determine where to find the WSA base address:

*Table 18. Finding the WSA base address*

| If you want the WSA base address for: | Locate the WSA base address in: |
|---|---|
| application code | the WSA address field in the Enclave Control Blocks section |
| a fetched module | the WSA address field of the Fetch() Information section for the fetch() function pointer for which you are interested |
| a DLL | the corresponding WSA address in the DLL Information section |

Use the WSA base address to locate the WSA in the Enclave Storage section.

## Steps for finding the static storage area

If you have C code compiled with the NORENT option (hereafter called `NORENT` code) you must determine the base address of the static storage area if you want to calculate the address of a static or external variable.

Perform the following steps to find the static storage area:

1. Name the static storage area CSECT by using the `pragma csect` directive. Once this is done, a CSECT is generated for the static storage area for each source file.
2. Determine the origin and length of the CSECT from the linker map.
3. Locate the external variables corresponding to the CSECT with the same name.
4. Determine the origin and length of the external variable CSECT from the linker map.

**Notes:**

1. Address calculation for static and external variables uses the static storage area as a base address with 1 or more offsets added to this address.
2. The storage associated with these CSECTs is not dumped when an exception occurs. It is dumped when `cdump` or CEE3DMP is called, but it is written to a separate ddname called CEESNAP. For information about `cdump`, CEE3DMP, and enabling the CEESNAP ddname, see “Generating a Language Environment Dump of a C/C++ Routine” on page 145.

## Steps for finding RENT static variables

**Before you begin:** You need to know the WSA. To find this information, see “Locating the Writable Static Area.” For this procedure’s example, the address of writable static is X'02D66E40'.

Perform the following steps to find RENT static variables:

1. Find the offset of @STATIC (associated with the file where the static variable is located) in the Writable Static Map section of the prelinker map.

   **Example:**

```
================================================================================
|                          Writable Static Map                                 |
================================================================================

  OFFSET    LENGTH  FILE ID  INPUT NAME

       0         1   00001   DFHC0011
       4         1   00001   DFHC0010
       8         2   00001   DFHDUMMY
       C         2   00001   DFHB0025
      10         2   00001   DFHB0024
      14         2   00001   DFHB0023
      18         2   00001   DFHB0022
      1C         2   00001   DFHB0021
      20         2   00001   DFHB0020
      24         2   00001   DFHEIB0
      28         4   00001   DFHEIPTR
      2C         4   00001   DFHCP011
      30         4   00001   DFHCP010
      34         4   00001   DFHBP025
      38         4   00001   DFHBP024
      3C         4   00001   DFHBP023
      40         4   00001   DFHBP022
      44         4   00001   DFHBP021
      48         4   00001   DFHBP020
      4C         4   00001   DFHEICB
      50         4   00001   DFHEID0
      54         4   00001   DFHLDVER
      58       278   00001   @STATIC
     720        30   00002   @STATIC
```

*Figure 27. 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 28 on page 140 shows the path to locate RENT C++ and C static variables by adding the address of writable static, the offset of @STATIC, and the variable offset.

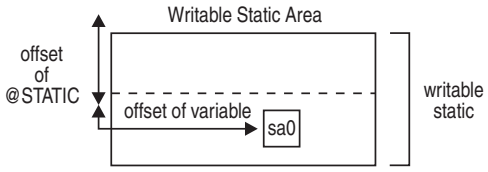



*Figure 28. Location of RENT Static Variable in Storage*

## 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" on page 138. 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 29. Writable Static Map Produced by Prelinker*

2. Add the offset of the external variable to the address of writable static.

   **Example:** X'02D66E40' + X'28' = X'2D66E68'

When you are done, you have the address of the value of the external variable in the Language Environment dump.

## Steps for finding NORENT static variables

**Before you begin:** You need to know the name and address of the static storage area. To find this information see “Steps for finding the static storage area” on page 138. For this procedure’s example, the static storage area is called **STATSTOR** and has an address of X'02D66E40'.

Perform the following steps to find external RENT variables:

1. Find the offset of the static variable in the partial storage offset compiler listing.

   **Example:**

   ```
   sa0   66-0:66   Class = static,   Location = STATSTOR +96,   Length = 4
   ```

   The offset is 96 (X'60').
2. Add the offset to the base address of static variables.

   **Example:** X'2D66E40' + X'60' = X'2D66EA0'

When you are done, you have the address of the value of the static variable in the Language Environment dump.

Figure 30 shows how to locate NORENT C static variables by adding the Static Storage Area CSECT address to the variable offset.

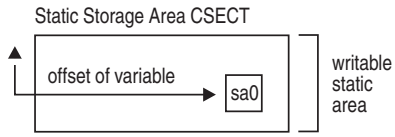



*Figure 30. Location of NORENT Static Variable in Storage*

## Steps for finding external NORENT variables

**Before you begin:** You need to find the address of the external variable CSECT. To find this information, see “Steps for finding the static storage area” on page 138. For this procedure’s example, the address of the external variable CSECT is X'02D66E40'.

The address of the external variable CSECT is the address of the value of the external variable in the Language Environment dump.

## Steps for finding the C/370 parameter list

Perform the following steps to locate a parameter in the Language Environment dump:

1. Indentify 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 31 on page 142 shows an example code for the parameter variable.

   **Example:**

```
func0() {
   :
 func1(a1,a2);
   :
}

func1(int ppx, int pp0) {
   :
}
```

*Figure 31. Example Code for Parameter Variable*

Parameters *ppx* and *pp0* correspond to copies of *a1* and *a2* in the stack frame belonging to `func0`.

2. Use the address of the start of the parameter list to find the register and offset in the partial storage offset listing.

   **Example:**

   ```
   pp0     62-0:62    Class = parameter,   Location = 4(r1),    Length = 4
   ```

   The offset is 4 (X'4') from register 1.
3. Determine the value of `GPR1` in the Saved Registers section for the function that called the function you are interested in.
4. Add this base address to the offset of the parameter.

When you are done, the contents of the variable can then be read in the DSA frame section corresponding to the function the parameter was passed from.

## Steps for finding the C++ parameter list

**Before you begin:** To locate C++ functions with `extern C` attributes, see "Steps for finding the C/370 parameter list" on page 141.

Perform the following steps to find the C++ parameter list:

1. Indentify the address of the start of the parameter list. A pointer to the parameter list is passed to the called function in register 1. This is the address of the start of the parameter list. Figure 32 shows an example code for the parameter variable.

   **Example:**

```
func0() {
   :
 func1(a1,a2);
   :
}

func1(int ppx, int pp0) {
   :
 }
```

*Figure 32. Example Code for Parameter Variable*

Parameters *ppx* and *pp0* correspond to copies of *a1* and *a2* in the stack frame belonging to `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 33. 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 34 shows an example of a static aggregate.

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

*Figure 34. Example Code for Structure Variable*

Figure 35 on page 144 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 35. 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 36 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 36. Writable Static Map Produced by Prelinker*

3. Find the offset of the static variable in the storage offset listing. The offset is 96 (X'60'). Following is an example of a partial storage offset listing.

   ```
   ss0     66-0:66    Class = static,    Location = GPR13(96),    Length = 4
   ```

   Add this offset to the result from step 2. The result is X'2D66EF8' (X'2D66E98' + X'60'). This is the address of the value of the static variable in the dump.
4. Find the offset of sz0 in the Aggregate Map, shown in Figure 35 on page 144. The offset is 60.

Add the offset from the Aggregate Map to the address of the ss0 struct. The result is X'60' (X'3C' + X'60'). This is the address of the values of sz0 in the dump.

### Finding the Timestamp

The timestamp is in the compile unit block. The address for the compile unit block is located at eight bytes past the function entry point. The compile unit block is the same for all functions in the same compilation. The fourth word of the compile unit block points to the timestamp. The timestamp is 16 bytes long and has the following format:

```
YYYYMMDDHHMMSSSS
```

## Generating a Language Environment Dump of a C/C++ Routine

You can use either the CEE3DMP callable service or the `cdump()`, `csnap()`, and `ctrace()` C/C++ functions to generate a Language Environment dump of C/C++ routines. These C/C++ functions call CEE3DMP with specific options.

## cdump()

If your routine is running under z/OS or CICS, you can generate useful diagnostic information by using the `cdump()` function. `cdump()` produces a main storage dump with the activation stack. This is equivalent to calling CEE3DMP with the option string: TRACEBACK BLOCKS VARIABLES FILES STORAGE STACKFRAME(ALL) CONDITION ENTRY.

When `cdump()` is invoked from a user routine, the C/C++ library issues an OS SNAP macro to obtain a dump of virtual storage. The first invocation of `cdump()` results in a SNAP identifier of 0. For each successive invocation, the ID is increased by one to a maximum of 256, after which the ID is reset to 0.

The output of the dump is directed to the CEESNAP data set. The DD definition for CEESNAP is as follows:

```
 //CEESNAP DD SYSOUT= *
```

If the data set is not defined, or is not usable for any reason, `cdump()` returns a failure code of 1. This occurs even if the call to CEE3DMP is successful.

If the SNAP is not successful, the CEE3DMP DUMP file displays the following message:

```
Snap was
unsuccessful
```

If the SNAP is successful, CEE3DMP displays this message:

```
 Snap was
successful; snap ID = nnn
```

Where *nnn* corresponds to the SNAP identifier described above. An unsuccessful SNAP does not result in an incrementation of the identifier.

Because `cdump()` returns a code of 0 only if the SNAP was successful or 1 if it was unsuccessful, you cannot distinguish whether a failure of `cdump()` occurred in the call to CEE3DMP or SNAP. A return code of 0 is issued only if both SNAP and CEE3DMP are successful.

Support for SNAP dumps using the `_cdump` function is provided only under z/OS and z/VM. SNAP dumps are not supported under CICS; no SNAP is produced in this environment. A successful SNAP results in a large quantity of output. A routine calling `cdump()` under CICS receives a return code of 0 if the ensuing call to CEE3DMP is successful. In addition to a SNAP dump, a Language Environment formatted dump is also taken.

## csnap()

The `csnap()` function produces a condensed storage dump. `csnap()` is equivalent to calling CEE3DMP with the option string: TRACEBACK FILES BLOCKS VARIABLES NOSTORAGE STACKFRAME(ALL) CONDITION ENTRY.

To use these functions, you must add `#include <ctest.h>` to your C/C++ code. The dump is directed to output *dumpname* , which is specified in a `//CEEDUMP DD` statement in MVS/JCL or a FILEDEF CEEDUMP command in z/VM.

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

Refer to the *z/OS 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 37 on page 147 shows a sample C routine that uses the `cdump` function to generate a dump.

Figure 42 on page 151 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 37. 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 37. Example C Routine Using cdump to Generate a Dump (Part 2 of 2)*

Figure 38 shows a fetched C module:

```
#include <ctest.h>

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

*Figure 38. Fetched module for C routine*

## Sample C++ Routine that Generates a Language Environment Dump

Figure 39 on page 149 shows a sample C++ routine that uses a protection exception to generate a dump.

```
#include <iostream.h>
#include <ctest.h>
#include "stack.h"

int main() {
  cout << "Program starting:\n";
  cerr << "Error report:\n";

  Stack<int> x;
  x.push(1);
  cout << "Top value on stack : " << x.pop() << '\n';
  cout << "Next value on stack: " << x.pop() << '\n';
  return(0);
}
```

*Figure 39. Example C++ Routine with Protection Exception Generating a Dump*

Figure 40 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 40. Template file STACK.C*

Figure 41 on page 150 shows the header file **stack.h**.

```
#ifndef __STACK_
  #define __STACK__
  template <class T> class Stack {
    public:
      Stack() {
        char* badPtr = 0; badPtr -= (0x01010101);
        head = (Node*) badPtr; /* head initialized to 0xFEFEFEFF */
      }
      T pop();
      void push(T);
    private:
      struct Node {
        T value;
        struct Node* next;
      }* head;
  };
#endif
```

*Figure 41. Header file STACK.H*

## Sample Language Environment Dump with C/C++-Specific Information

This sample dump was produced by compiling the routine in Figure 37 on page 147 with the TEST(SYM) compiler option, then running it. Notice the sequence of calls in the traceback section - EDCZMINV is the C-C++ management module that invokes `main` and @@FECBMODULE1 fetches the user-defined function `func1`, which in turn calls the library routine `__cdump`.

If source code is compiled with the `GONUMBER` or `TEST` compile option, statement numbers are shown in the traceback. If source code is compiled with the `TEST(SYM)` compile option, variables and their associated type and value are dumped out. For more information about C/C++-specific information contained in a dump, see “Finding C/C++ Information in a Language Environment Dump” on page 156.

```
CEE3DMP V1 R3.0: This is a sample dump

CEE3DMP called by program unit  (entry point __cdump) at offset +00000184.

Snap was unsuccessful

Registers on Entry to CEE3DMP:

  PM....... 0100
  GPR0..... 00000000  GPR1..... 000305D0  GPR2..... 00030564  GPR3..... 8DD5CB06
  GPR4..... 00000001  GPR5..... 00000015  GPR6..... 0DEB54D8  GPR7..... 00000001
  GPR8..... 00000000  GPR9..... 0DEB5038  GPR10.... 8DB50310  GPR11.... 8DB50310
  GPR12.... 00023890  GPR13.... 000304E0  GPR14.... 800260DE  GPR15.... 8DB6C670
  FPR0..... 4D000000  00060388            FPR2..... 00000000  00000000

  FPR4..... 00000000  00000000            FPR6..... 00000000  00000000


   .
   .
   .

Information for enclave main

  Information for thread 8000000000000000

  Registers on Entry to CEE3DMP:
    PM....... 0100
    GPR0..... 00000000  GPR1..... 000305D0  GPR2..... 00030564  GPR3..... 8DD5CB
    GPR4..... 00000001  GPR5..... 00000015  GPR6..... 0DEB54D8  GPR7..... 000000
    GPR8..... 00000000  GPR9..... 0DEB5038  GPR10.... 8DB50310  GPR11.... 8DB503
    GPR12.... 00023890  GPR13.... 000304E0  GPR14.... 800260DE  GPR15.... 8DB6C6
    FPR0..... 4D000000  00060388            FPR2..... 00000000  00000000
    FPR4..... 00000000  00000000            FPR6..... 00000000  00000000

   .
   .
   .
   Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset  Statement  Load Mod  Service  Status
    000304E0                0DD5CAB8  +00000184  __cdump       0DD5CAB8  +00000184             CEEEV003           Call
    00030440  POSIX.CRTL.C(MODULE1)
                            0DB50310  +0000006E  func1         0DB50310  +0000006E          5  MODULE1            Call
    00030350                0DEB54F8  -0DEB54F3  @@FECBMODULE1 0DEB54F8  -0DEB54F3                              Call
    00030298                0DCB4AE8  +0000001A  @@GETFN       0DCB4A40  +000000C2             CEEEV003           Call
    000301E0  POSIX.CRTL.C(CSAMPLE)
                            0DB51078  +00000392  main          0DB51078  +00000392         64  CSAMPLE            Call
    000300C8                0DC626EE  +000000B4  EDCZMINV      0DC626EE  +000000B4             CEEEV003           Call
    00030018  CEEBBEXT      0001B898  +0000013C  CEEBBEXT      0001B898  +0000013C             CEEBINIT           Call
   Parameters, Registers, and Variables for Active Routines:

   .
   .
   .
    main (DSA address 000301E0):

      Saved Registers:

        GPR0..... 0DEB5330  GPR1..... 8DC9EE8A  GPR2..... 8DC627A2  GPR3..... 8DB510C6
        GPR4..... 8001B97C  GPR5..... 0DEB5098  GPR6..... 0DEB5330  GPR7..... 0DB523DC
        GPR8..... 00000001  GPR9..... 80000000  GPR10.... 8DC626E2  GPR11.... 8001B898
        GPR12.... 00023890  GPR13.... 000301E0  GPR14.... 8DB5140C  GPR15.... 0DCB4A40

   .
   .
   .
      Local Variables:

        fetchPtr         signed int (*) (void)

                                           0xDEB5330

        fp2              struct __ffile *    0xDEBEA1C

        fp1              struct __ffile *    0xDEBD024

        rc               signed int                  0
```

*Figure 42. Example Dump from Sample C Routine (Part 1 of 6)*

```
[1] Storage for Active Routines:

  Control Blocks for Active Routines:

    :
    DSA for func1: 00030440
      +000000  FLAGS.... 1002      member... 47D0      BKC...... 00030350  FWC...... F7F20000  R14...... 8DB50380
      +000010  R15...... 0DD5CAB8  R0....... 000304E0  R1....... 000304D8  R2....... 8DC5A732  R3....... 8DB5035E
      +000024  R4....... 8001B97C  R5....... 0DEB54D8  R6....... 0DEB5330  R7....... 0DB523DC  R8....... 00000001
      +000038  R9....... 0DB6E66E  R10...... 0DB6D66F  R11...... 8DB6C670  R12...... 00023890  reserved. 000247D0
      +00004C  NAB...... 000304E0  PNAB..... 000304B0  reserved. 8DBFE208  0DB50B08  000304EC  00000000
      +000064  reserved. 000303D4  reserved. 00030350  MODE..... 00030538  reserved. 00000000  00000000
      +000078  reserved. 00000000  reserved. D3C5F140

    :
    DSA frame: 00030440
      +000000 00030440  100247D0 00030350 F7F20000 8DB50380  0DD5CAB8 000304E0 000304D8 8DC5A732  |.......&72.......N.........Q.Ex.|
      +000020 00030460  8DB5035E 8001B97C 0DEB54D8 0DEB5330  0DB523DC 00000001 0DB6E66E 0DB6D66F  |...;...@...Q.............W>..O?|
      +000040 00030480  8DB6C670 00023890 000247D0 000304E0  000304B0 8DBFE208 0DB50B08 000304EC  |..F...................S.........|
      +000060 000304A0  00000000 000303D4 00030350 00030538  00000000 00000000 00000000 D3C5F140  |.......M...&................LE1 |
      +000080 000304C0  40404040 40404040 0DB68414 40404040  40404040 40404040 0DEB54D8 40404040  |        ..d.            ...Q    |

    :
[2] Control Blocks Associated with the Thread:
    CAA: 00023890
      +000000 00023890  00000800 00000000 00030000 00050000  00000000 00000000 00000000 00000000  |................................|
      +000020 000238B0  00000000 00000000 00024A10 00000000  00000000 00000000 00000000 00000000  |................................|
      +000040 000238D0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000060 000238F0  00000000 00000000 00000000 00000000  00000000 80020760 00000000 00000000  |.......................-........|
      +000080 00023910  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|

    :
  [2A] C/370 CAA information :
    C-C++ Specific CTHD......... 0DB6740C
    C-C++ Specific CEDB......... 0DB67DE4

    C-C++ Specific Thread block: 0DB6740C
      +000000 0DB6740C  C3E3C8C4 00000308 0DB6740C 00000000  0DC3865E 00000000 00000000 00000000  |CTHD.............Cf;............|
      +000020 0DB6742C  00000000 00000000 00000000 00000000  0DB677EC 00000000 00000000 00000000  |................................|
      +000040 0DB6744C  00000000 00000000 00000000 00024640  00024728 00000000 00000000 00000001  |..............  ................|
      +000060 0DB6746C  00000001 00000000 00000000 00000000  00000000 00000000 0DEB54B8 0DEB54B4  |................................|
      +000080 0DB6748C  0DEB54B0 0DEB54AC 0DEB54A0 0DEB54A4  0DEB54C0 00000000 00000000 00000000  |..............u.................|

    :
    C-C++ Specific EDB block: 0DB67DE4
      +000000 0DB67DE4  C3C5C4C2 000004D0 0DB67DE4 0DB52B40  00030000 0DB691C4 0DB69514 0DB51078  |CEDB......'U... ......jD..n.....|
      +000020 0DB67E04  00000080 0DB50FD8 00000000 00000000  00000000 0DB682BC 0DC3DFB0 0DC3DE20  |.......Q.............b..C...C.. |
      +000040 0DB67E24  0DEB54F8 00000001 00000000 0DEB5388  0DB67CB0 0DB67A58 0DB67B84 0DC95196  |...8..........h..@...:...#d.I.o|
      +000060 0DB67E44  00004650 0DB683CC 40400000 00000000  00100000 00000000 00000000 00000000  |...&..c.  ......................|

    :
```

*Figure 42. Example Dump from Sample C Routine (Part 2 of 6)*

```
[2B] errno value................. 0
    memory file block chain..... 0DEBEBA0
    open FCB chain.............. 0DEBEA30
    GTAB table.................. 0DB6771C

   [3] signal information :
    SIGFPE   :
    function pointer... 0DB51D20    WSA address... 8DEB5038     function name... hsigfpe

    SIGTERM  :
    function pointer... 0DB51E90    WSA address... 8DEB5038     function name... hsigterm

    SIGOBJECT  :
    function pointer... 0DB67A58    WSA address... 0DB67B84     function name... (unknown)

  Enclave variables:
    *.*.C(CSAMPLE):>hsigterm
                    void ()             0xDB51E90
    *.*.C(CSAMPLE):>hsigfpe
                    void ()             0xDB51D20
    *.*.C(CSAMPLE):>xcount
                    signed int                  0
    *.*.C(CSAMPLE):>main
                    signed int (void)
                                        0xDB51078
    *.*.C(CSAMPLE):>atf1
                    void (void)         0xDB51FF8
    *.*.C(CSAMPLE):>st2
                    signed int                255
    *.*.C(CSAMPLE):>st1
                    signed int                 99
    *.*.C(MODULE1):>func1
                    signed int (void)
                                        0xDB50310

  Enclave Control Blocks:
    EDB: 000228B0
      +000000 000228B0  C3C5C5C5 C4C24040 C0000001 00023750  00022EF8 00000000 00000000 00000000  |CEEEDB  .......&...8............|
      +000020 000228D0  00022D78 00022DA8 00025038 00022558  00000000 80021808 000229D0 00008000  |.......y..&.....................|
      +000040 000228F0  00000000 00000000 0000CFB0 00000000  00000000 00000000 0DB646F0 0DEB54C8  |.........................0...H|
      +000060 00022910  8001C8D8 00000000 0DB69864 00000000  00024AE0 00000000 0DC2A620 00023890  |..HQ......q.............Bw.....|
      +000080 00022930  00000000 00000000 00000000 00000000  00000001 00000000 00008A08 008DA738  |.............................x.|
      +0000A0 00022950  00000001 00000000 00000000 00000000  00000000 00000000 00000000 00000001  |................................|
    MEML: 00023750
      +000000 00023750  00000000 00000000 0DBBB0B8 00000000  00000000 00000000 0DBBB0B8 00000000  |................................|
      +000020 00023770  00000000 00000000 0DBBB0B8 00000000  0DB67DE4 00000000 8DC2B6B8 00000000  |..................'U.....B......|
      +000040 00023790  00000000 00000000 0DBBB0B8 00000000  00000000 00000000 0DBBB0B8 00000000  |................................|
      +000060 000237B0 - +00011F 0002386F             same as above

[4] WSA address.................0DEB5038

[5] atexit information :
    function pointer... 8DB51FF8    WSA address... 0DEB5038     function name... atfl

  [6] fetch information :
    fetch pointer     : 0DEB54F8
    function pointer... 8DB50310    WSA address... 0DEB5480

  Enclave Storage:
    Initial (User) Heap                                     : 0DEB5000
      +000000 0DEB5000  C8C1D5C3 00022D48 00022D48 00000000  0DEB5000 0DEB55D0 00008000 00007A30  |HANC.............&...........:.|
      +000020 0DEB5020  0DEB5000 00000190 00000000 00000000  00000000 00000000 00000000 0DB67B98  |..&.........................#q|
      +000040 0DEB5040  0DB67A6C 0DB67CC4 00000000 00000000  00000000 00000000 00000000 00000000  |..:%..@D........................|

    .
    .
    .
    LE/370 Anywhere Heap                                    : 0DEB1000
      +000000 0DEB1000  C8C1D5C3 00022D78 00022D78 00022D78  0DEB1000 0DEB3C18 00004000 000013E8  |HANC..................... ....Y|
      +000020 0DEB1020  0DEB1000 00001008 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000040 0DEB1040  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|

    .
    .
    .
```

*Figure 42. Example Dump from Sample C Routine (Part 3 of 6)*

```
     LE/370 Below Heap                                          : 00050000
       +000000 00050000  C8C1D5C3 00022DA8 00022DA8 00022DA8  80050000 000500A8 00002000 00001F58  |HANC...y...y...y.......y........|
       +000020 00050020  00050000 00000088 C3E2E3D2 00000000  00000000 00800001 00000002 00000068  |.......hCSTK....................|
       +000040 00050040  04000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
       +000060 00050060  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
       +000080 00050080 - +001FFF 00051FFF             same as above

     Additional Heap, heapid = 0DEB3BE4                         : 0DB14000
       +000000 0DB14000  C8C1D5C3 0DEB3BE4 0DEB3BE4 0DEB3BE4  0DB14000 0DB143D0 000003E8 00000018  |HANC...U...U...U.. ........Y....|
       +000020 0DB14020  0DB14000 00000050 00000000 0DB503AC  0DB50310 0DB14098 0DEB54F8 46F11000  |.. ....&.............. q...8.1..|
       +000040 0DB14040  00000003 0DB50580 00020000 0DB14078  00000000 0DB50310 0DB50310 00000000  |.............. .................|
       +000060 0DB14060  0DB50568 0DB503C0 00000000 00000000  0DB14000 00000020 0015D7D6 E2C9E74B  |.................. .......POSIX.|
       +000080 0DB14080  C3D9E3D3 4BC34DD4 D6C4E4D3 C5F15D00  0DB14000 00000088 0DB14028 00010000  |CRTL.C(MODULE1)... ....h.. .....|

    ⋮
   File Status and Attributes:

[7] File Control Block: 0DEBEA30
       +000000 0DEBEA30  0DEBED65 00000000 000003DB 0DEBEB00  0DEBEB20 00000000 0DEBEB50 00000011  |.........................&....|
       +000020 0DEBEA50  00000014 00000000 00000000 0DEBD038  00000000 0DEBEA30 00000000 00000000  |................................|
       +000040 0DEBEA70  00000000 FFFFFFFF 00080055 0DEBEB70  0DEBEB44 0DC81810 0DC83A08 0DC83C88  |.....................H...H...H.h|
       +000060 0DEBEA90  0DC840F8 0DC71578 00000000 00000400  00000400 00000000 00000000 00000400  |.H 8.G..........................|
       +000080 0DEBEAB0  0DEBED40 0DEBED40 00000000 00000000  00000000 00000000 00000000 00000000  |... ... ........................|
       +0000A0 0DEBEAD0  00000000 00000000 00000000 00000000  0DC68140 00000000 00000000 00000000  |.................Fa ............|
       +0000C0 0DEBEAF0  43020008 40001000 00000000 0DB66DDC  58FF0008 07FF0000 0DC67BF8 00000000  |.... .........._..........F#8....|
       +0000E0 0DEBEB10  00000000 00000000 00000000 00000000  58FF0008 07FF0000 0DC80088 00000000  |.........................H.h....|
       +000100 0DEBEB30  00000000 00000000 00000000 00000000  00000000 00000000 80000020 0DEBD000  |................................|

     fldata FOR FILE:   HEALY.MEMORY.DATA
     __recfmF:1........ 1
     __recfmV:1........ 0
     __recfmU:1........ 0
     __recfmS:1........ 0
     __recfmBlk:1...... 0
     __recfmASA:1...... 0
     __recfmM:1........ 0
     __recfmPO:1....... 0
     __dsorgPDSmem:1... 0
     __dsorgPDSdir:1... 0
     __dsorgPS:1....... 0
     __dsorgConcat:1... 0
     __dsorgMem:1...... 1
     __dsorgHiper:1.... 0
     __dsorgTemp:1..... 0
     __dsorgVSAM:1..... 0
     __dsorgHFS:1...... 0
     __openmode:2...... 1
     __modeflag:4...... 2
     __dsorgPDSE:1..... 0
     __reserve2:8...... 0
     __device.......... 8
     __blksize......... 1024
     __maxreclen....... 1024
     __dsname.......... HEALY.MEMORY.DATA
     __reserve4........ 0

      FILE pointer........ 0DEBEA1C

     Buffer at current file position: 0DEBED40
       +000000 0DEBED40  A2969485 408481A3 81A29694 85409496  99854084 81A38185 A5859540 94969985  |some datasome more dataeven more|
       +000020 0DEBED60  408481A3 81000000 00000000 00000000  00000000 00000000 00000000 00000000  | data............................|
       +000040 0DEBED80  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
       +000060 0DEBEDA0 - +0003FF 0DEBF13F             same as above
     Saved Buffer........ NULL

     File Control Block: 0DEBD038
       +000000 0DEBD038  0DEBD234 00000000 00000400 0DEBD108  0DEBD128 80000000 0DEBD158 00000011  |..K...........J...J.......J.....|
       +000020 0DEBD058  00000014 00000000 00000000 0DB6701C  0DEBEA30 0DEBD038 00000000 E2E8E2F0  |.........................SYS0|
       +000040 0DEBD078  F0F2F7F2 FFFFFFFF 0000003C 0DEBD178  0DEBD14C 0DE789B8 0DE768F0 0DE81178  |0272.........J...J<.Xi..X.0.Y..|
       +000060 0DEBD098  0DE7F988 0DE85530 00000000 00000404  00001800 0DEBEA0A 00000000 00001801  |.X9h.Y.........................|
       +000080 0DEBD0B8  0DEBD208 0DEBD234 0DEBD234 00000000  00000000 00000000 00000000 00000000  |..K...K...K.....................|
       +0000A0 0DEBD0D8  0000001B 00000000 00000000 00000000  0DE01740 00000000 00000000 00000000  |.................. ..............|
       +0000C0 0DEBD0F8  43120020 28440000 00000000 0DB66DDC  58FF0008 07FF0000 0DE011F8 00000000  |.............._............8....|
       +0000E0 0DEBD118  00000000 00000000 00000000 00000000  58FF0008 07FF0000 0DE6F3E8 00000000  |.........................W3Y....|
       +000100 0DEBD138  00000000 00000000 00000000 00000000  00000000 00000000 80000020 0DEBD000  |................................|
```

*Figure 42. Example Dump from Sample C Routine (Part 4 of 6)*

```
    fldata FOR FILE:   'HEALY.MYFILE.DATA'
    __recfmF:1........ 0
    __recfmV:1........ 1
    __recfmU:1........ 0
    __recfmS:1........ 0
    __recfmBlk:1...... 1
    __recfmASA:1...... 0
    __recfmM:1........ 0
    __recfmPO:1....... 0
    __dsorgPDSmem:1... 0
    __dsorgPDSdir:1... 0
    __dsorgPS:1....... 1
    __dsorgConcat:1... 0
    __dsorgMem:1...... 0
    __dsorgHiper:1.... 0
    __dsorgTemp:1..... 0
    __dsorgVSAM:1..... 0
    __dsorgHFS:1...... 0
    __openmode:2...... 0
    __modeflag:4...... 2
    __dsorgPDSE:1..... 0
    __reserve2:8...... 0
    __device.......... 0
    __blksize......... 6144
    __maxreclen....... 1024
    __dsname.......... HEALY.MYFILE.DATA
    __reserve4........ 0

    FILE pointer........ 0DEBD024
    ddname.............. SYS00272

    Buffer at current file position: 0DEBD208
      +000000 0DEBD208  00280000 000C0000 99858396 998440F1  000C0000 99858396 998440F2 000C0000  |........record 1....record 2....|
      +000020 0DEBD228  99858396 998440F3 00040000 00000000  00000000 00000000 00000000 00000000  |record 3........................|
      +000040 0DEBD248  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000060 0DEBD268 - +0003FF 0DEBD607             same as above
    Saved Buffer........ NULL

    Write Data Control Block: 00052020
      +000000 00052020  00052E20 00000000 00000008 00EFD08C  002FE5A2 00000001 00004000 0000CE38  |..................Vs...... .....|
      +000020 00052040  8605223A 50052DAD 01582424 0089A044  12BEE1B0 00C129D8 0A0521B8 00001800  |f...&........i........A.Q........|
      +000040 00052060  30013030 0000CEA8 01D448F8 00D448F8  00000404 00D45470 47F0F026 01C3C5C5  |.......y.M.8.M.8.....M...00..CEE|
    read/update DCB..... NULL
    Write Data Control Block Extension: 00052E20
      +000000 00052E20  C4C3C2C5 00380000 00052020 00000000  C0C80000 00000000 00000000 00000000  |DCBE.............H..............|
      +000020 00052E40  00000000 00000000 00000000 00000000  00000000 00000100 800000B8 00052000  |................................|
    read/update DCBE.... NULL

    Job File Control Block: 00052E60
      +000000 00052E60  C8C5C1D3 E84BD4E8 C6C9D3C5 4BC4C1E3  C1404040 40404040 40404040 40404040  |HEALY.MYFILE.DATA               |
      +000020 00052E80  40404040 40404040 40404040 40404040  40404040 80001F1D 00000000 00000000  |                     .............|
      +000040 00052EA0  00000200 00000000 00000000 00000000  61004900 00000040 00000000 00000000  |................/...... ........|
      +000060 00052EC0  00000000 00000000 00000000 00000000  00000000 0001E2D4 E2F0F0F6 40404040  |....................SMS006    |
      +000080 00052EE0  40404040 40404040 40404040 40404040  40404040 0089DDA0 00000050 00001800  |                    .i.....&....|
      +0000A0 00052F00  00000000 00000000 00000000 20000100  80000038 00052000 00052F18 0DEBD208  |............................K.|

[8] __amrc_type structure: 00031B18
      +000000 00031B18  00000000 00000000 00000007 00000000  00000000 00000000 00000000 00000000  |................................|
      +000020 00031B38  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000040 00031B58 - +0000BF 00031BD7             same as above
      +0000C0 00031BD8  00000000 00000000 00000000 00000000  00000000 00000000 00000000 000000DC  |................................|
```

*Figure 42. Example Dump from Sample C Routine (Part 5 of 6)*

```
    amrc __code union fields
    __error..................... 0(0)
    __abend.__syscode.......... 0(0)
    __abend.__rc............... 0(0)
    __feedback.rc.............. 0(0)
    __feedback.__ftncd......... 0(0)
    __feedback.__fdbk.......... 0(0)
    __alloc.__svc99_info....... 0(0)
    __alloc.__svc99_error...... 0(0)
    __RBA...................... 0(0)
    __last_op.................. 7(7)
    __msg.__str................ NULL
    __msg.__parmr0............. 0(0)
    __msg.__parmr1............. 0(0)
    __msg.__str2............... NULL

    __amrc2_type structure: 00031A1C
      +000000 00031A1C  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
    __error2................... 0(0)
    __fileptr.................. NULL

Process Control Blocks:

  PCB: 00022558
    +000000 00022558  C3C5C5D7 C3C24040 03030398 00000000  00000000 00000000 00022788 0DC294C8  |CEEPCB  ...q..............h.BmH|
    +000020 00022578  0DC27A50 0DC2A338 0DC2A810 0DB58928  00021918 00000000 00000000 000228B0  |.B:&.Bt..By...i.................|
    +000040 00022598  0DC2A470 7C000000 00000000 00000000  00000000 00000000 00000000 00000000  |.Bu.@...........................|

  MEML: 00022788
    +000000 00022788  00000000 00000000 0DBBB0B8 00000000  00000000 00000000 0DBBB0B8 00000000  |................................|
    +000020 000227A8  00000000 00000000 0DBBB0B8 00000000  0DB66DDC 00000000 8DC2B6B8 00000000  |..................._......B......|
    +000040 000227C8  00000000 00000000 0DBBB0B8 00000000  00000000 00000000 0DBBB0B8 00000000  |................................|
    +000060 000227E8 - +00011F 000228A7             same as above

Additional Language Specific Information:
[9]   errno information :
  Thread Id .... 8000000000000000 Errno ...... 00000000 Errnojr .... 00000000
```

*Figure 42. Example Dump from Sample C Routine (Part 6 of 6)*

# Finding C/C++ Information in a Language Environment Dump

When a Language Environment traceback or dump is generated for a C/C++ routine, information is provided that is unique to C/C++ routines. C/C++-specific information includes:

- Control block information for active routines
- Condition information for active routines
- Enclave level data

Each of the unique C/C++ sections of the Language Environment dump are described.

### [1] Storage for Active Routines

The Storage for Active Routines section of the dump shows the DSAs for the active C and C++ routines. To relate a DSA frame to a particular function name, use the address associated with the frame to find the corresponding DSA. In this example, the function `func1` DSA address is X'00030440'.

### [2] Control Blocks Associated with the Active Thread

In the Control Blocks Associated with the Thread section of the dump, the following information appears:

- Fields from the CAA
- Fields specific from the CTHD and CEDB
- Signal information

### [2A] C/C++ CAA Fields

The CAA contains several fields that the C/C++ programmer can use to find information about the run-time environment. For each C/C++ program, there is a C-C++ Specific Thread area and a C-C++ Specific Enclave area.

### [2B] C-C++ Specific CAA

The C-C++ specific CAA fields that are of interest to users are described below.

**errno value**
: A variable used to display error information. Its value can be set to a positive number that corresponds to an error message. The functions `perror()` and `strerror()` print the error message that corresponds to the value of `errno`.

**Memory file control block**
: You can use the memory file control block (MFCB) to locate additional information about memory files. This control block resides at the C/C++ thread level. For more information about the MFCB, see 158.

**Open FCB chain**
: A pointer to the start of a linked list of open file control blocks (FCBs). For more information about FCBs, see 158.

### [3] Signal Information

When the POSIX(OFF) run-time option is specified, signal information is provided in the dump to aid you in debugging. For each signal that is disabled with `SIG_IGN`, an entry value of 00000001 is made in the first field of the Signal Information field for the specified signal name.

For each signal that has a handler registered, the signal name and the handler name are listed. If the handler is a fetched C function, the value `@@FECB` is entered as the function name and the address of the fetched pointer is in the first field.

If you compile a C routine as `NORENT`, the WSA address is not available (`N/A`). For more information about the `signal` function, see *z/OS 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 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 C/C++ Programming Guide*.

**[7] File Control Block Information**

This section of the dump includes the file control block (FCB) information for each C/C++ file. The FCB contains file status and attributes for files open during C/C++ active routines. You can use this information to find the data set or file name.

The FCB is a handle that points to the following file information, which is displayed when applicable, for the file:

- Access method control block (ACB) address
- Data control block (DCB) address
- Data control block extension (DCBE) address
- Job file control block (JFCB) address
- RPL address
- Current® buffer address
- Saved buffer address
- ddname

Not all FCB fields are always filled in. For example, RPLs are used only for VSAM data sets. The ddname field contains blanks if it is not used.

The save block buffer represents auxiliary buffers that are used to save the contents of the main buffers. Such saving occurs only when a reposition is performed and there is new data; for example, an incomplete text record or an incomplete fixed-block standard (FBS) block in the buffers that cannot be flushed out of the system.

Because the main buffers represent the current position in the file, while the save buffers merely indicate a save has occurred, check the save buffers only if data appears to be missing from the external device and is not found in the main buffers. Also, do not infer that the presence of save buffers means that data present there belongs at the end of the file. (The buffers remain, even when the data is eventually written.)

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

**Memory File Control Block**

This section of the dump holds the memory file control block information for each memory file the routine uses. A sample memory file control block is shown in Figure 43 on page 159.

```
Memory File Control Block: 046F5CD0
  +000000 046F5CD0  046F5D40 00000001 00000000 00000000  046F5BA8 00010000 046F5E48 00000013  |.?) .............?$y.....?;.....|
  +000020 046F5CF0  00000014 00000000 00000000 00000000  00000000 046F5BA8 00000000 00000000  |.....................?$y........|
  +000040 046F5D10  046F5CD0 00000000 00000000 00000000  00000000 00000000 00000000 046F5D40  |.?*.........................?)  |
  +000060 046F5D30  00010000 00000000 00000000 00000000  00000000 00000000 046F5E68 00000001  |.........................?;.....|
memory file name......... TSOID.MEMORY.DATA
First memory data space: 046F5E68
  +000000 046F5E68  93899585 40F19389 958540F2 93899585  40F30000 00000000 00000000 00000000  |line 1line 2line 3..............|
```

*Figure 43. Memory File Control Block*

**Memory file name**
The name assigned to this memory file.

**First memory data space**
A dump of the first 1K maximum of actual user data associated with this memory file.

#### [8] Information for __amrc

`__amrc` is a structure defined in the **stdio.h** header file to assist in determining errors resulting from I/O operations. The contents of `__amrc` can be checked for system information, such as the return code for VSAM. Certain fields of the `__amrc` structure can provide useful information about what occurred previously in your routine.

For more information about `__amrc`, refer to "Debugging C/C++ Input/Output Programs" on page 127 and to *z/OS C/C++ Programming Guide*.

#### [9] Errno Information

The Errno information shows the thread id of the thread that generated the dump and the settings of the errno and errnojr variables for that thread.

Both the errno and the errnojr variables contain the return code of the last failing z/OS UNIX system service call. These variables provide z/OS UNIX application programs access to diagnostic information returned from an underlying z/OS UNIX callable service. For more information on these return and reason codes, refer to *z/OS UNIX System Services Messages and Codes*.

## Additional Floating-Point Registers

The Language Environment dump formats Additional Floating Point (AFP™) registers and Floating Point Control (FPC) registers when the AFP suboption of the FLOAT 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 C/C++ compiler option, see *z/OS 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 44. 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 45. 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 46. Condition Information for Active Routines*

## Sample Language Environment Dump with XPLINK-Specific Information

The programs `tranmain` shown in Figure 47 on page 162 and `trandll` shown in Figure 48 on page 163 were used to produce a Language Environment dump. The dump shows XPLINK-compiled routines calling NOXPLINK-compiled routines, and NOXPLINK-compiled routines calling XPLINK-compiled routines. The program `tranmain` was compiled XPLINK and `trandll` was compiled NOXPLINK. Each was link-edited as a separate program object with the sidedeck from the other. The Language Environment dump produced by running these program is shown in Figure 49 on page 164. Explanations for some of the sections are in “Finding XPLINK Information in a Language Environment Dump” on page 166.

```
#pragma runopts(TRACE(ON,1M,NODUMP,LE=1),XPLINK(ON),TERMTHDACT(UADUMP))
#include <stdio.h>
#pragma export(tran2)

int tran1(int, int, int, long double, int);
int tran3(int, int, int, long double, int);

void main(void) {

int         parm1 = 0x11111111;
int         parm2 = 0x22222222;
int         parm3 = 0x33333333;
long double parm4 = 1234.56789;
int         parm5 = 0x55555555;
int         retval;

   printf("Main: Call Tran1\n");
   retval = tran1(parm1,parm2,parm3,parm4,parm5);
   printf("Main: Return value from Tran1 = %d\n",retval);

}
int tran2(int parm1,int parm2,int parm3,long double parm4,int parm5) {

int         retval;

   printf("Tran2: Call Tran3\n");
   retval = tran3(parm1,parm2,parm3,parm4,parm5);
   printf("Tran2: Return value from Tran3 = %d\n",retval);
   return retval;

}
```

*Figure 47. 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 48. Sample NOXPLINK-compiled Program (trandll) Which Calls an XPLINK-compiled Program*

```
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.       08/30/01 3:59:07 PM                 Page:    1

Information for enclave main

  Information for thread 8000000000000000

  [1] Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    23ED4C18  CEEHDSPR      23BA6538  +000038DA  CEEHDSPR      23BA6538  +000038DA              CEEPLPKA           Call
    23ED4998  CEEHABD       23AD34A0  +0000012A  CEEHABD       23AD34A0  +0000012A              CEEPLPKA           Exception
    23ED48E0  ./trandll.c   240848C0  +000000D6  tran3         240848C0  +000000D6          26  XNTDLL             Call
    23ED4730  CEEVRONU      23BB0470  +00000706  CEEVRONU      23BB0470  +00000706              CEEPLPKA           Call
    24077530  ./tranmain.c  23A000E8  +00000070  tran2         23A000E8  +00000070          27  XNTRAN             Call
    240775B0                23BAEE38  +000009A4  CEEVROND      23BAEE90  +0000094C              CEEPLPKA           Call
    23ED44E8  ./trandll.c   24084A40  +000000F2  tran1         24084A40  +000000F2          14  XNTDLL             Call
    23ED4338  CEEVRONU      23BB0470  +00000706  CEEVRONU      23BB0470  +00000706              CEEPLPKA           Call
    24077680  ./tranmain.c  23A00218  +0000008C  main          23A00218  +0000008C          18  XNTRAN             Call
    24077720                23BAEE38  +000009A4  CEEVROND      23BAEE90  +0000094C              CEEPLPKA           Call
    23ED40E0  EDCZHINV      23E8EC28  +0000009A  EDCZHINV      23E8EC28  +0000009A              CELHV003           Call
    23ED4018  CEEBBEXT      000082A8  +000001A6  CEEBBEXT      000082A8  +000001A6              CEEBINIT           Call

  Condition Information for Active Routines
    Condition Information for CEEHABD (DSA address 23ED4998)
      CIB Address: 23ED5438
      Current Condition:
        CEE0198S The termination of a thread was signaled due to an unhandled condition.
      Original Condition:
        CEE3250C The system or user abend U1001 R=00000000 was issued.
      Location:
        Program Unit: CEEHABD Entry: CEEHABD Statement:  Offset: +0000012A
      Machine State:
        ILC..... 0002    Interruption Code..... 000D
        PSW..... 078D1400 A3AD35CA
        GPR0..... 84000000  GPR1..... 840003E9  GPR2..... 23ED4984  GPR3..... 240848FA
        GPR4..... 23ED4980  GPR5..... 000159D0  GPR6..... 00000000  GPR7..... 00000000
        GPR8..... A3A000F2  GPR9..... 23A1BE80  GPR10.... 23ED4980  GPR11.... A3AD34A0
        GPR12.... 00016AC0  GPR13.... 23ED4998  GPR14.... A4084998  GPR15.... 00000000
      ABEND code: 000003E9 Reason code: 00000000
    Storage dump near condition, beginning at location: 23AD35BA
      +000000 23AD35BA  88100008 41000084 89000018 16100A0D  47F0B1B6 5840C2F0 9506400B 4770B1A0  |h.....di........0... B0n. .....|

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

   ⋮

    tran3 (DSA address 23ED48E0):
      UPSTACK DSA
      Parameters:
        parm5            signed int          1431655765
        parm4            long double      1.23456788999999997135303197E+03
        parm3            signed int           858993459
        parm2            signed int           572662306
        parm1            signed int           286331153
      Saved Registers:
        GPR0..... 23A1BD70  GPR1..... 23ED4978  GPR2..... 23ED4984  GPR3..... 240848FA
        GPR4..... 23ED4980  GPR5..... 23A1BE10  GPR6..... 00000000  GPR7..... 00000000
        GPR8..... A3A000F2  GPR9..... 23A1BE80  GPR10.... 240848C0  GPR11.... A3BB04FA
        GPR12.... 00016AC0  GPR13.... 23ED48E0  GPR14.... A4084998  GPR15.... A3AD34A0

   ⋮

        Local Variables:
          timing           signed long int             1
          code             signed long int          1001
```

*Figure 49. Example Dump of Calling Between XPLINK and non-XPLINK Programs (Part 1 of 3)*

```
CEEVRONU (DSA address 23ED4730):
  TRANSITION DSA
  Saved Registers:
    GPR0..... 23A1BD70  GPR1..... 24077D70  GPR2..... 23ED4818  GPR3..... 00000010
    GPR4..... 24077530  GPR5..... 23FF58D0  GPR6..... 00000000  GPR7..... 00000000
    GPR8..... A3A000F2  GPR9..... 23A1BE80  GPR10.... 240848C0  GPR11.... A3BB04FA
    GPR12.... 00016AC0  GPR13.... 23ED4730  GPR14.... A3BB0B78  GPR15.... 240848C0

⋮

tran2 (DSA address 24077530):
  DOWNSTACK DSA
  Parameters:
    parm5            signed int          1431655765
    parm4            long double      1.2345678899999999977135303197E+03
    parm3            signed int           858993459
    parm2            signed int           572662306
    parm1            signed int           286331153
  Saved Registers:
    GPR0..... 55555555  GPR1..... 11111111  GPR2..... 22222222  GPR3..... 33333333
    GPR4..... 24077530  GPR5..... 23FF58D0  GPR6..... 23BB04D8  GPR7..... A3A0015A
    GPR8..... A3A000F2  GPR9..... 23A1BE80  GPR10.... 23A000B8  GPR11.... 23BAEE38
    GPR12.... 00016AC0  GPR13.... 23ED45B0  GPR14.... 23A000B8  GPR15.... 0000000C

⋮

  Local Variables:
    retval           signed int          -455613482
CEEVROND (DSA address 240775B0):
  TRANSITION DSA
  Saved Registers:
    GPR0..... ********  GPR1..... ********  GPR2..... ********  GPR3..... ********
    GPR4..... 240775B0  GPR5..... 23A1BE80  GPR6..... 23A000E8  GPR7..... A3BAF7DE
    GPR8..... 23A000B8  GPR9..... 23BAFE37  GPR10.... ********  GPR11.... ********
    GPR12.... ********  GPR13.... ********  GPR14.... ********  GPR15.... ********

⋮

tran1 (DSA address 23ED44E8):
  UPSTACK DSA
  Saved Registers:
    GPR0..... 23FF5098  GPR1..... 23ED4580  GPR2..... 55555555  GPR3..... 24084A7A
    GPR4..... 33333333  GPR5..... 23A1BE10  GPR6..... 22222222  GPR7..... 11111111
    GPR8..... A3A00222  GPR9..... 23A00320  GPR10.... 24084A40  GPR11.... A3BB04FA
    GPR12.... 00016AC0  GPR13.... 23ED44E8  GPR14.... A4084B34  GPR15.... 23BAEE38

⋮
```

*Figure 49. Example Dump of Calling Between XPLINK and non-XPLINK Programs (Part 2 of 3)*

```
[3] Control Blocks for Active Routines:
 ⋮

DSA for tran3: 23ED48E0
  +000000  FLAGS.... 10A0      member... 1D8C      BKC...... 23ED4730  FWC...... 23ED4998  R14...... A4084998
  +000010  R15...... A3AD34A0  R0....... 23A1BD70  R1....... 23ED4978  R2....... 23ED4984  R3....... 240848FA
  +000024  R4....... 23ED4980  R5....... 23A1BE10  R6....... 00000000  R7....... 00000000  R8....... A3A000F2
  +000038  R9....... 23A1BE80  R10...... 240848C0  R11...... A3BB04FA  R12...... 00016AC0  reserved. 00017630
  +00004C  NAB...... 23ED4998  PNAB..... 00000000  reserved. 00000000  00000000  00000000  00000000
  +000064  reserved. 00000000  reserved. 23A00000  MODE..... 23A01438  reserved. 80000000  00000000
  +000078  reserved. 23A001D0  reserved. 23A00480
DSA for CEEVRONU: 23ED4730
  +000000  FLAGS.... 0000      member... 0000      BKC...... FFFFFFFF  FWC...... E3D9C1D5  R14...... A3BB0B78
  +000010  R15...... 240848C0  R0....... 23A1BD70  R1....... 24077D70  R2....... 23ED4818  R3....... 00000010
  +000024  R4....... 24077530  R5....... 23FF58D0  R6....... 00000000  R7....... 00000000  R8....... A3A000F2
  +000038  R9....... 23A1BE80  R10...... 240848C0  R11...... A3BB04FA  R12...... 00016AC0  reserved. 00017630
  +00004C  NAB...... 23ED48E0  PNAB..... 23ED48E0  reserved. 00000000  00000000  00000000  00000000
  +000064  reserved. 23ED47B0  reserved. 00000000  MODE..... 00000000  reserved. 00000000  00000000
  +000078  reserved. 00000000  reserved. 00000000
DSA for CEEVRONU: 23ED47B0
  +000000  EYE...... DOWNTOUP  TRTYPE... 00000003  BOS...... 00000000  STACKFLR. 00000000  SSTOPD... 24077680
  +000018  SSDSAU... 23ED4730  TRANEP... 23BB0470  TR_R0.... 55555555  TR_R1.... 11111111  TR_R2.... 22222222
  +00002C  TR_R3.... 33333333  TR_R4.... 24077530  TR_R5.... 23FF58D0  TR_R6.... 23BB04D8  TR_R7.... A3A0015A
  +000040  TR_R8.... A3A000F2  TR_R9.... 23A1BE80  TR_R10... 23A000B8  TR_R11... 23BAEE38  TR_R12... 00016AC0
  +000054  TR_R13... 23ED45B0  TR_R14... 23A000B8  TR_R15... 0000000C  CRENT.... 00000000  ROND_DSA. 23ED45B0
  +000068  INTF_MAP. 018F1000
DSA for tran2: 24077D30
  +000000  R4....... 240775B0  R5....... 23A1BE80  R6....... 23A000E8  R7....... A3BAF7DE  R8....... 23A000B8
  +000014  R9....... 23BAFE37  R10...... 23A000B8  R11...... 23BAEE38  R12...... 00016AC0  R13...... 23ED45B0
  +000028  R14...... 23A000B8  R15...... 0000000C  reserved. 00000A68  reserved. 23BFB78B  HPTRAN... 00000000
  +00003C  reserved. 55555555  reserved. 11111111
DSA for CEEVROND: 24077DB0
  +000000  R4....... E3D9C1D5  R5....... 00000000  R6....... 23BAEE90  R7....... 00000000  R8....... 23ED4860
  +000014  R9....... 00000000  R10...... 00000000  R11...... A3B6931C  R12...... 00000000  R13...... 00000000
  +000028  R14...... 24077E10  R15...... 00000000  reserved. 23A1BE1C  reserved. 23ED45A8  HPTRAN... 24077E10
  +00003C  reserved. FFFFFFFF  reserved. 11111111
DSA for CEEVROND: 24077E10
  +000000  EYE...... UPTODOWN  TRTYPE... 00000002  BOS...... 00000000  STACKFLR. 00000000  SSTOPD... 24077680
  +000018  SSDSAU... 23ED4338  TRANEP... 23BAEE90  TR_R0.... 00000000  TR_R1.... 00000000  TR_R2.... 00000000
  +00002C  TR_R3.... 00000000  TR_R4.... 23ED44E8  TR_R5.... 00000000  TR_R6.... 00000000  TR_R7.... A4084B34
  +000040  TR_R8.... 00000000  TR_R9.... 00000000  TR_R10... 00000000  TR_R11... 00000000  TR_R12... 00000000
  +000054  TR_R13... 00000000  TR_R14... 00000000  TR_R15... 00000000  CRENT.... 23BAEE38  ROND_DSA. 00000000
  +000068  INTF_MAP. 00000000
DSA for tran1: 23ED44E8
  +000000  FLAGS.... 1000      member... 59D0      BKC...... 23ED4338  FWC...... 23ED4860  R14...... A4084B34
  +000010  R15...... 23BAEE38  R0....... 23FF5098  R1....... 23ED4580  R2....... 55555555  R3....... 24084A7A
  +000024  R4....... 33333333  R5....... 23A1BE10  R6....... 22222222  R7....... 11111111  R8....... A3A00222
  +000038  R9....... 23A00320  R10...... 24084A40  R11...... A3BB04FA  R12...... 00016AC0  reserved. 00017630
  +00004C  NAB...... 23ED45B0  PNAB..... A3B9104C  reserved. 23ED4408  00000001  23A01CF0  0000000C
  +000064  reserved. 00000000  reserved. 23A01D38  MODE..... 00100028  reserved. 00000000  00000000
  +000078  reserved. 23A01BB0  reserved. A3B90CF8

 ⋮
```

*Figure 49. 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 ax XPLINK-compiled program that uses an 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 an upstack DSA are those saved by the DSA-owning routine on entry. Register 7 is the return address back to the caller of the DSA-owning routine. Register 6 may be the entry point of the DSA-owning routine. (This is not true when the Branch Relative and Save instruction is used to implement the call.)

## C/C++ Contents of the Language Environment Trace Tables

Language Environment provides four C/C++ trace table entry types that contain character data:

- Trace entry 1 occurs when a base C library function is called.
- Trace entry 2 occurs when a base C library function returns.
- Trace entry 3 occurs when a POSIX C library function is called.
- Trace entry 4 occurs when a POSIX C library function returns.
- Trace entry 5 occurs when an XPLINK base C or POSIX C library function is called.
- Trace entry 6 occurs when an XPLINK base C or POSIX C library function returns.

The format for trace table entry 1 is:

```
NameOfCallingFunction
—>(xxx) NameOfCalledFunction
```

or, for called functions calloc, free, malloc, and realloc:

```
NameOfCallingFunction
—>(xxx) NameOfCalledFunction<(input_parameters)>
```

In addition, when the call is due to one of these C++ operators:

```
-new,
-new[],
-delete,
-delete[]
```

then the C++ operator will appear and the format becomes:

```
NameOfCallingFunction
—>(xxx) NameOfCalledFunction<(input_parameters)>

NameOfC++Operator
```

The format for trace table entry 2 is:

```
<—(xxx) R15=value 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)>
```

Trace table entry 5 is just like trace table entry 1. The `input_parameters` and `NameOfC++Operator` only appear for the appropriate functions. The angle brackets (<>) indicate that this information does not always appear.

The format for trace table entry 6 is:

```
<--(xxxx) R1=xxxxxxxx R2=xxxxxxxx R3=xxxxxxxx ERRNO=xxxxxxxx ERRNO2=xxxxxxxx
```

In all six entry types, (xxx) and (xxxx) are numbers associated with the called library function and are used to associate a specific entry record with its corresponding return record.

For entry types 5 and 6, the number will be the same as the number of the function as seen in the C run-time library definition side-deck, SCEELIB dataset member CELHS003, on the IMPORT statement for that function.

Figure 50 on page 169 shows a non-XPLINK trace which has examples of C/C++ trace table entry types 1 thru 4.

Figure 51 on page 171 shows an XPLINK trace which has examples of the trace entries 5 and 6.

```
⋮
Language Environment Trace Table:

 Most recent trace entry is at displacement: 02D500

     Displacement                      Trace Entry in Hexadecimal                                    Trace Entry in EBCDIC
     ------------ ------------------------------------------------------------------------  --------------------------------
       +000000    Time 20.52.46.666280    Date 2001.08.26    Thread ID... 8000000000000000
       +000010    Member ID.... 03    Flags..... 000000    Entry Type..... 00000001
       +000018    94818995 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |main                            |
       +000038    60606E4D F1F3F95D 40A2A399 8397A84D  5D404040 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  |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  |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 50. 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 50. Trace Table with C/C++ Trace Table Entry Types 1 thru 4 (Part 2 of 2)*

Figure 51 on page 171 shows an XPLINK trace which has examples of the trace entries 5 and 6.

```
 ⋮
Language Environment Trace Table:

  Most recent trace entry is at displacement: 000D80

    Displacement                    Trace Entry in Hexadecimal                          Trace Entry in EBCDIC
    ------------ ------------------------------------------------------------------  --------------------------------
      +000000    Time 22.41.35.433944    Date 2001.08.30    Thread ID... 26C70D0000000000
      +000010    Member ID.... 03    Flags..... 000000    Entry Type..... 00000006
      +000018    4C60604D F0F0F5F9 5D40D9F1 7EF2F3C6  C6C3C1C2 F040D9F2 7EF2F3C3 F5F8F9C4  |<--(0059) R1=23FFCAB0 R2=23C589D|
      +000038    F040D9F3 7EF2F3C6 C6C4F0F0 F040C5D9  D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9  |0 R3=23FFD000 ERRNO=00000074 ERR|
      +000058    D5D6F27E F0F0F0F0 F0F0F0F0 00000000  00000000 00000000 00000000 00000000  |NO2=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()   |
      +0000B8    60606E4D F0F2F0F4 5D4097A3 88998581  846D94A4 A385A76D 8485A2A3 9996A84D  |-->(0204) pthread_mutex_destroy(|
      +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|
      +000138    F040D9F3 7EF0F0F0 F0F0F0F0 F040C5D9  D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9  |0 R3=00000000 ERRNO=00000074 ERR|
      +000158    D5D6F27E F0F0F0F0 F0F0F0F0 00000000  00000000 00000000 00000000 00000000  |NO2=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()   |
      +0001B8    60606E4D F0F0F5F9 5D408699 85854DF0  A7F2F4F0 F0F4C3F2 F05D4040 40404040  |-->(0059) free(0x24004C20)      |
      +0001D8    40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
      +0001F8    84859385 A3854040                                                       |delete                          |

      +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|
      +000238    F040D9F3 7EF2F3C6 C6C4F0F0 F040C5D9  D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9  |0 R3=23FFD000 ERRNO=00000074 ERR|
      +000258    D5D6F27E F0F0F0F0 F0F0F0F0 00000000  00000000 00000000 00000000 00000000  |NO2=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()   |
      +0002B8    60606E4D F0F2F0F4 5D4097A3 88998581  846D94A4 A385A76D 8485A2A3 9996A84D  |-->(0204) pthread_mutex_destroy(|
      +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|
      +000338    F040D9F3 7EF0F0F0 F0F0F0F0 F040C5D9  D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9  |0 R3=00000000 ERRNO=00000074 ERR|
      +000358    D5D6F27E F0F0F0F0 F0F0F0F0 00000000  00000000 00000000 00000000 00000000  |NO2=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()   |
      +0003B8    60606E4D F0F0F5F9 5D408699 85854DF0  A7F2F4F0 F0F4C3F3 F85D4040 40404040  |-->(0059) free(0x24004C38)      |
      +0003D8    40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
      +0003F8    84859385 A3854040                                                       |delete                          |

      +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|
      +000438    F040D9F3 7EF2F3C6 C6C4F0F0 F040C5D9  D9D5D67E F0F0F0F0 F0F0F7F4 40C5D9D9  |0 R3=23FFD000 ERRNO=00000074 ERR|
      +000458    D5D6F27E F0F0F0F0 F0F0F0F0 00000000  00000000 00000000 00000000 00000000  |NO2=00000000....................|
      +000478    00000000 00000000                                                       |........                        |
 ⋮
```

*Figure 51. Trace Table with XPLINK Trace Table Entries 5 and 6.*

For more information about the Language Environment trace table format, see “Understanding the trace table entry (TTE)” on page 118.

# 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 52 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 52. C Routine with a Divide-by-Zero Error*

To debug this routine, use the following steps:

1. Locate the Current Condition message in the Condition Information for Active Routines section of the dump. In this example, the message is CEE3209S. The system detected a fixed—point divide exception. This message indicates the error was caused by an attempt to divide by zero. For additional information about CEE3209S, see *z/OS Language Environment Run-Time Messages* .

   The traceback section of the dump indicates that the exception occurred at offset X'7E' within function funcb. This information is used along with the compiler-generated Pseudo Assembly Listing to determine where the problem occurred.

   If the TEST compiler option is specified, variable information is in the dump. If the GONUMBER compiler option is specified, statement number information is in the dump. Figure 53 on page 173 shows the generated traceback from the dump.

```
Information for enclave main

  Information for thread 0B672E6800000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    00015018  CEEHDSP       088AFB00  +000025D2  CEEHDSP       088AFB00  +000025D2              CEEPLPKA  UQ13157  Call
    00017288                0B309C18  +0000007E  funcb         0B309C18  +0000007E              *PATHNAM           Exception
    000171E0                0B309B28  +0000006E  main          0B309B28  +0000006E              *PATHNAM           Call
    000170C8                0876ED36  -08765998  EDCZMINV      0876ED36  -08765998              CEEEV003           Call
    00017018  CEEBBEXT      00CA0508  +0000013C  CEEBBEXT      00CA0508  +0000013C              CEEBINIT  UQ09246  Call

  Condition Information for Active Routines
    Condition Information for  (DSA address 00017288)
      CIB Address: 00015498
      Current Condition:
        CEE0198S The termination of a thread was signaled due to an unhandled condition.
      Original Condition:
        CEE3209S The system detected a fixed-point divide exception.
      Location:
        Program Unit:  Entry: funcb Statement:  Offset: +0000007E
      Machine State:
        ILC..... 0002    Interruption Code..... 0009
        PSW..... 078D2400 8B309C98
        GPR0..... 00017330  GPR1..... 00017280  GPR2..... 8876EDEA  GPR3..... 8B309C62
        GPR4..... 80CA05EC  GPR5..... 0B3076C8  GPR6..... 00000000  GPR7..... 00000001
        GPR8..... 00000000  GPR9..... 80000000  GPR10.... 8876ED2A  GPR11.... 80CA0508
        GPR12.... 00008910  GPR13.... 00017288  GPR14.... 00017288  GPR15.... 0B309C18
    Storage dump near condition, beginning at location: 0B309C86
      +000000 0B309C86  4B803052 5860304A 58656000 8E600020  1D685070 D0A058D0 D00458E0 D00C9838  |.....-....-..-....&...........q.|
    ⋮
```

*Figure 53. 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 54 on page 174.

   The offset (within `funcb`) of the exception from the traceback (X'7E') reveals the divide instruction: `DR  r6,r8` at that location. Instructions X'66' through X'80' refer to the `result = fa/(statint-73);` line of the C/C++ routine.

```
OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                                        *
 000000                    00015 |        funcb    DS    0D
 000000  47F0  F026        00015 |                  B     38(,r15)
 000004  01C3  C5C5                                       CEE eyecatcher
 000008  0000  00A8                                       DSA size
 00000C  ****  ****                                       =A(PPA1-funcb)
 000010  47F0  F001        00015 |                  B     1(,r15)
 000014  183F              00015 |                  LR    r3,r15
 000016  58F0  C31C        00015 |                  L     r15,796(,r12)
 00001A  184E              00015 |                  LR    r4,r14
 00001C  05EF              00015 |                  BALR  r14,r15
 00001E  0000  0000                                       =F'0'
 000022  47F0  303A        00015 |                  B     58(,r3)
 000026  90E8  D00C        00015 |                  STM   r14,r8,12(r13)
 00002A  58E0  D04C        00015 |                  L     r14,76(,r13)
 00002E  4100  E0A8        00015 |                  LA    r0,168(,r14)
 000032  5500  C314        00015 |                  CL    r0,788(,r12)
 000036  4720  F014        00015 |                  BH    20(,r15)
 00003A  5000  E04C        00015 |                  ST    r0,76(,r14)
 00003E  9210  E000        00015 |                  MVI   0(r14),16
 000042  50D0  E004        00015 |                  ST    r13,4(,r14)
 000046  18DE              00015 |                  LR    r13,r14
 000048  0530              00015 |                  BALR  r3,r0
 00004A                    End of Prolog
                           00015 |      *  void funcb(int *pp) {
 00004A  5010  D098        00015 |                  ST    r1,152(,r13)
 00004E  5850  C1F4        00015 |                  L     r5,500(,r12)
                                        *    int result;
                           00017 |      *    fa = *pp;
 000052  5860  D098        00017 |                  L     r6,152(,r13)
 000056  5860  6000        00017 |                  L     r6,0(,r6)
 00005A  5870  ****        00017 |                  L     r7,=Q(fa)
 00005E  5860  6000        00017 |                  L     r6,0(,r6)
 000062  5065  7000        00017 |                  ST    r6,0(r5,r7)
                           00018 |      *    result = fa/(statint-73);
 000066  5880  ****        00018 |                  L     r8,=Q(statint)
 00006A  5885  8000        00018 |                  L     r8,0(r5,r8)
 00006E  4B80  ****        00018 |                  SH    r8,=H'73'
 000072  5860  ****        00018 |                  L     r6,=Q(fa)
 000076  5865  6000        00018 |                  L     r6,0(r5,r6)
 00007A  8E60  0020        00018 |                  SRDA  r6,32
 00007E  1D68              00018 |                  DR    r6,r8
 000080  5070  D0A0        00018 |                  ST    r7,160(,r13)
                           00019 |      *    return;
                           00020 |      *  }
 000084                    Start of Epilog
 000084  58D0  D004        00020 |                  L     r13,4(,r13)
 000088  58E0  D00C        00020 |                  L     r14,12(,r13)
 00008C  9838  D020        00020 |                  LM    r3,r8,32(r13)
 000090  051E              00020 |                  BALR  r1,r14
 000092  0707              00020 |                  NOPR  r7
 000094                    Start of Literals
 000094  0000  0000                                       =Q(fa)
 000098  0000  0000                                       =Q(statint)
 00009C  0049                                             =H'73'
 00009E                    End of Literals

                           PPA1: Entry Point Constants
 00009E  10CE  A106                                       =F'281977094'      Flags
 0000A2  FFFF  FF9C                                       =A(PPA2-funcb)
 0000A6  0000  0000                                       =F'0'              No PPA3
 0000AA  0000  0000                                       =F'0'              No EPD
 0000AE  FFE0  0000                                       =F'-2097152'       Register save mask
 0000B2  0000  0000                                       =F'0'              Member flags
 0000B6  90                                               AL1(144)           Flags
 0000B7  0000  00                                         AL3(0)             Callee's DSA use/8
 0000BA  0240                                             =H'576'            Flags
 0000BC  0014                                             =H'20'             Offset/2 to CDL
 0000BE  0005  ****                                       AL2(5),C'funcb'
 0000C6  5000  0049                                       =F'1342177353'     CDL function length/2
 0000CA  FFFF  FF62                                       =F'-158'           CDL function EP offset
 0000CE  3825  0000                                       =F'941948928'      CDL prolog
 0000D2  4007  0042                                       =F'1074200642'     CDL epilog
 0000D6  0000                                             =H'0'              CDL end
                           PPA1 End
            .
            .
            .
```

*Figure 54. 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 137.

   Because this routine was compiled with the RENT option, find the WSA address in the Enclave Control Blocks section of the dump. In this example, this address is X'0B3076C8'. Figure 55 shows the WSA address.

```
Enclave Control Blocks:

   ⋮
  WSA address..................... 0B3076C8
   ⋮
```

*Figure 55. C/C++ CAA Information in Dump*

4. 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 56. In this example, the offset is X'4'.

```
   ⋮
=======================================================================
|                    *** MODULE MAP ***                               |
=======================================================================


   ⋮
 ---------------
 CLASS  C_WSA            LENGTH =       60  ATTRIBUTES = MRG, DEFER , RMODE=ANY ALIGN = DBLWORD
 ---------------

            CLASS
           OFFSET  NAME                TYPE    LENGTH   SECTION

                0  fa                PART             4  fa
                4  statint           PART             4  statint
   ⋮
```

*Figure 56. Writable Static Map*

5. Add the WSA address of X'0B3076C8' to the offset of `statint`. The result is X'0B3076CC'. This is the address of the variable `statint`, which is in the writable static area. The writable static area is storage allocated by the C/C++ run-time for the C/C++ user, so it is in the user heap. The heap content is displayed in the Enclave Storage section of the dump, shown in Figure 57 on page 176.
6. To find the variable `statint` in the heap, locate the closest address listed that is before the address of `statint`. In this case, that address is X'0B3076CB'. Count across X'04' to location X'0B3076CC'. The value at that location is X'49' (that is, `statint` is 73), and hence the fixed point divide exception.

```
  ⋮
Enclave Storage:
  Initial (User) Heap                                    : 0B325000
  ⋮

  WSA for Program Object(s)
  WSA: 0B3076C8
    +000000 0B3076C8  00000001 00000049 0B31B6F0 00000000  00000000 00000000 00000000 00000000  |...........0.....................|
    +000020 0B3076E8  00000001 00000001 00000000 00000000  00004650 00000001 00000000 0B31EEEC  |..................&............|
    +000040 0B307708  00000000 00000000 0B31F018 0B31EDC0  00000000 00000000 0B31F85C 0B31F866  |..........0..............8*..8.|

  ⋮
```

*Figure 57. Enclave Storage Section of Dump*

## Calling a Nonexistent Non-XPLINK Function

Figure 58 demonstrates the error of calling a nonexistent function. This routine was compiled with the compiler options LIST and RENT and was run with the option TERMTHDACT(DUMP). The code was processed by the binder with MAP to generate a binder map, which is used to calculate the addresses of static and external variables.

This routine was not compiled with the TEST(ALL) compile option. As a result, arguments and variables do not appear in the dump.

The only prelinker option used was MAP.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
void funca(int* aa);
int (*func_ptr)(void)=0;
int main(void) {
  int aa;
  funca(&aa);
  printf("result of funca = %d\n",aa);
  return;
}
void funca(int* aa) {
  *aa = func_ptr();
  return;
}
```

*Figure 58. C/C++ Example of Calling a Nonexistent Subroutine*

To debug this routine, use the following steps:

1. Locate the Current Condition message in the Condition Information for Active Routines section of the dump, shown in Figure 59 on page 177. In this example, the message is `CEE3201S The system detected an operation exception (System Completion Code=0C1).` This message suggests that the error was caused by an attempt to branch to an unknown address. For additional information about CEE3201S, see *z/OS Language Environment Run-Time Messages*.

   The traceback section of the dump indicates that the exception occurred at offset X'-04500616' within function `funca`. The negative offset indicates that the

offset cannot be used to locate the instruction that caused the error. Another indication of bad data is the value of X'80000004' in the instruction address of the PSW. This address indicates that an instruction in the routine branched outside the bounds of the routine.

```
Information for enclave main

  Information for thread 8000000000000000

  Registers on Entry to CEE3DMP:

    PM....... 0100
    GPR0..... 00000000  GPR1..... 00077448  GPR2..... 053AD9AF  GPR3..... 853AD514
    GPR4..... 00000001  GPR5..... 053AD314  GPR6..... 80077454  GPR7..... 00000000
    GPR8..... 00000001  GPR9..... 80000000  GPR10.... 00077470  GPR11.... 000F7490
    GPR12.... 0006A520  GPR13.... 000773C8  GPR14.... 80060712  GPR15.... 853F7918
    FPR0..... 4D000000  00043C31            FPR2..... 00000000  00000000
    FPR4..... 00000000  00000000            FPR6..... 00000000  00000000

   ⋮
  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry    E Addr    E  Offset  Statement  Load Mod  Service  Status
    0006B018  CEEHDSP       0001A5B8  +00001A18  CEEHDSP  0001A5B8  +00001A18             CEEPLPKA           Call
    00075278                04500618  -04500616  funca    04500618  -04500616                                **Exception**
    000751D8                04500558  +00000068  main     04500558  +00000068                                Call
    000750C8                046ABBAE  +000000B0  @@MNINV  046ABBAE  +000000B0             CEEEV003           Call
    00075018  CEEBBEXT      00007768  +00000138  CEEBBEXT 00007768  +00000138             CEEBINIT           Call

  Condition Information for Active Routines
    Condition Information for  (DSA address 00075278)
      CIB Address: 0006B3C8
      Current Condition:
        CEE0198S The termination of a thread was signalled.
      Original Condition:
        CEE3201S The system detected an operation exception (System Completion Code=0C1).
      Location:
        Program Unit:  Entry: funca Statement:  Offset: -04500616
      Machine State:
        ILC..... 0002    Interruption Code..... 0001
        PSW..... 078D0000 80000004
        GPR0..... 00075308  GPR1..... 00075270  GPR2..... 00075278  GPR3..... 84500666
        GPR4..... 046E5618  GPR5..... 046E5620  GPR6..... 00075268  GPR7..... 00000000
        GPR8..... 04500698  GPR9..... 80000000  GPR10.... 846ABBA2  GPR11.... 80007768
        GPR12.... 00068520  GPR13.... 00075278  GPR14.... 84500680  GPR15.... 00000000

   ⋮
  Parameters, Registers, and Variables for Active Routines:
    CEEHDSP (DSA address 0006B018):
      Saved Registers:
        GPR0..... 0001BE62  GPR1..... 0006B32C  GPR2..... 00000003  GPR3..... 0006BEC8
        GPR4..... 00000000  GPR5..... 0005D8C0  GPR6..... 0001C8BB  GPR7..... 00000003
        GPR8..... 00000001  GPR9..... 0001C5B6  GPR10.... 0001B5B7  GPR11.... 0001A5B8
        GPR12.... 00068520  GPR13.... 0006B018  GPR14.... 8005E712  GPR15.... 846FC918

   ⋮
funca (DSA address 00075278):
      Saved Registers:
        GPR0..... 00075308  GPR1..... 00075270  GPR2..... 00075278  GPR3..... 84500666
        GPR4..... 046E5618  GPR5..... 046E5620  GPR6..... 00075268  GPR7..... 00000000
        GPR8..... 04500698  GPR9..... 80000000  GPR10.... 846ABBA2  GPR11.... 80007768
        GPR12.... 00068520  GPR13.... 00075278  GPR14.... 84500680  GPR15.... 00000000
   ⋮
```

*Figure 59. Sections of the Dump from Example C Routine*

2. Find the branch instructions for `funca` in the listing in Figure 60 on page 178. Notice the `BALR r14,r15` instruction at offset X'126'. This branch is part of the instruction.

```
0000C0                        92  funca    DS    0F
  ⋮
00010C  0530                 107           BALR  r3,r0
00010E                   End of Prolog
00010E  5840  C1F4           109           L     r4,500(,r12)
000112  5010  D088           110           ST    r1,136(,r13)
                                  *    return;
                                  *  }
                                  *
                     00016 |      *  void funca(int *aa) {
                     00017 |      *    *aa = func_ptr();
000116  5870  D088           116           L     r7,136(,r13)
00011A  5860  7000           117           L     r6,0(,r7)
00011E  5870  ****           118           L     r7,=Q(func_ptr)
000122  58F4  7000           119           L     r15,0(r4,r7)
000126  05EF                 120           BALR  r14,r15
000128  50F0  6000           125           ST    r15,0(,r6)
  ⋮
```

*Figure 60. Pseudo Assembly Listing*

3. Find the offset of FUNC@PTR in the Writable Static Map, shown in Figure 61, as produced by the prelinker.

```
  ⋮
================================================================================
|                          Writable Static Map                                 |
================================================================================

  OFFSET    LENGTH  FILE ID  INPUT NAME

      0         4   00001    FUNC@PTR
      8        18   00001    @STATIC
  ⋮
```

*Figure 61. Writable Static Map*

4. Add the offset of FUNC@PTR (X'0') to the address of WSA (X'46E5618'). The result ( X'46E5618') is the address of the function pointer func_ptr in the writable static storage area within the heap. This value is 0, indicating the variable is uninitialized.

   Figure 62 on page 179 shows the sections of the dump.

```
Enclave Control Blocks:
   ⋮
    WSA address..................  046E5618
   ⋮
   Enclave Storage:
     Initial (User) Heap: 046E4000
   ⋮
      +001600 046E5600  C05FA15A 7B4F5B7C 79000000 00000000  046E4000 00000028 00000000 00000000  |.^.!#|$@.........> .............|
      +001620 046E5620  9985A2A4 93A34096 864086A4 95838140  7E406C84 15000000 00000000 00000000  |result of funca = %d............|
      +001640 046E5640  00000000 00000000 046E4000 00000010  00000000 80054152 046E4000 00000018  |.........> .............> .....|
      +001660 046E5660  00000000 00000000 00000000 00000000  046E5638 00000000 00000010 00000000  |.................>.............|
      +001680 046E5680  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
   ⋮
```

*Figure 62. Enclave Control Blocks and Storage sections in Dump*

## Calling a Nonexistent XPLINK Function

Figure 63 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 63. C/C++ Example of Calling a Nonexistent Subroutine*

To debug this routine, use the following steps:

1. Locate the Current Condition message in the Condition Information for Active Routines section of the dump, shown in Figure 64 on page 180. In this example, the message is `CEE3201S The system detected an operation exception (System Completion Code=0C1).` This message suggests that the error was caused by an attempt to branch to an unknown address. For additional information about CEE3201S, see *z/OS Language Environment Run-Time Messages*.

   The traceback section of the dump indicates that the exception occurred at offset X'-23B553DE7' within function `funca`. The negative offset indicates that the offset cannot be used to locate the instruction that caused the error. Another indication of bad data is the value of X'80000004' in the instruction address of the PSW. This address indicates that an instruction in the routine branched outside the bounds of the routine.

```
Information for enclave main

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    241694E8  CEEHDSPR      23D7D3F8  +000038DA  CEEHDSPR      23D7D3F8  +000038DA              CEEPLPKA           Call
    24169338  CEEHRNUH      23E78028  +00000082  CEEHRNUH      23E78028  +00000082              CEEPLPKA           Call
    24209620                23B553E0  -23B553DE  funca         23B553E0  -23B553DE              XEXIST             Exception
    242096A0                23B55358  +00000012  main          23B55358  +00000012              XEXIST             Call
    24209720                23E7AD10  +000009A4  CEEVROND      23E7AD68  +0000094C              CEEPLPKA           Call
    241690E0  EDCZHINV      2413BFC0  +0000009A  EDCZHINV      2413BFC0  +0000009A              CELHV003           Call
    24169018  CEEBBEXT      00073380  +000001A6  CEEBBEXT      00073380  +000001A6              CEEBINIT           Call

  Condition Information for Active Routines
    Condition Information for  (DSA address 24209620)
      CIB Address: 24169D08
      Current Condition:
        CEE0198S The termination of a thread was signaled due to an unhandled condition.
      Original Condition:
        CEE3201S The system detected an operation exception (System Completion Code=0C1).

      Location:
        Program Unit:  Entry: funca Statement:  Offset: -23B553DE
      Machine State:
        ILC..... 0002    Interruption Code..... 0001
        PSW..... 078D2300 80000004
        GPR0..... 23C7BFC0  GPR1..... 24209F00  GPR2..... 24169128  GPR3..... 2416912C
        GPR4..... 24209620  GPR5..... 00FCD178  GPR6..... 00000000  GPR7..... A3B553FE
        GPR8..... A3B55362  GPR9..... 23E7BD0F  GPR10.... 00000000  GPR11.... A3E7AD10
        GPR12.... 0007FAC0  GPR13.... 241691B8  GPR14.... 23E7AD68  GPR15.... 00000000
    Storage dump near condition, beginning at location: 00000000
      +000000 00000000  Inaccessible storage.

Information for enclave main

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    241694E8  CEEHDSPR      23D7D3F8  +000038DA  CEEHDSPR      23D7D3F8  +000038DA              CEEPLPKA           Call
    24169338  CEEHRNUH      23E78028  +00000082  CEEHRNUH      23E78028  +00000082              CEEPLPKA           Call
    24209620                23B553E0  -23B553DE  funca         23B553E0  -23B553DE              XEXIST             Exception
    242096A0                23B55358  +00000012  main          23B55358  +00000012              XEXIST             Call
    24209720                23E7AD10  +000009A4  CEEVROND      23E7AD68  +0000094C              CEEPLPKA           Call
    241690E0  EDCZHINV      2413BFC0  +0000009A  EDCZHINV      2413BFC0  +0000009A              CELHV003           Call
    24169018  CEEBBEXT      00073380  +000001A6  CEEBBEXT      00073380  +000001A6              CEEBINIT           Call

  Condition Information for Active Routines
    Condition Information for  (DSA address 24209620)
      CIB Address: 24169D08
      Current Condition:
        CEE0198S The termination of a thread was signaled due to an unhandled condition.
      Original Condition:
        CEE3201S The system detected an operation exception (System Completion Code=0C1).

      Location:
        Program Unit:  Entry: funca Statement:  Offset: -23B553DE
      Machine State:
        ILC..... 0002    Interruption Code..... 0001
        PSW..... 078D2300 80000004
        GPR0..... 23C7BFC0  GPR1..... 24209F00  GPR2..... 24169128  GPR3..... 2416912C
        GPR4..... 24209620  GPR5..... 00FCD178  GPR6..... 00000000  GPR7..... A3B553FE
        GPR8..... A3B55362  GPR9..... 23E7BD0F  GPR10.... 00000000  GPR11.... A3E7AD10
        GPR12.... 0007FAC0  GPR13.... 241691B8  GPR14.... 23E7AD68  GPR15.... 00000000
    Storage dump near condition, beginning at location: 00000000
      +000000 00000000  Inaccessible storage.
```

*Figure 64. Sections of the Dump from Example C Routine (Part 1 of 2)*

```
Parameters, Registers, and Variables for Active Routines:
  CEEHDSPR (DSA address 241694E8):
    UPSTACK DSA
    Saved Registers:
      GPR0..... 23C7BFC0  GPR1..... 24169904  GPR2..... 2416AF40  GPR3..... 00000003
      GPR4..... 23CA7D20  GPR5..... 00000002  GPR6..... 23CB29D0  GPR7..... 00000000
      GPR8..... A3D80A78  GPR9..... 2416A4E7  GPR10.... 23D817FC  GPR11.... 23D7D3F8
      GPR12.... 0007FAC0  GPR13.... 241694E8  GPR14.... A3D80CD4  GPR15.... A3D615D8
      FPR8..... ********  ********            FPR9..... ********  ********
      FPR10.... ********  ********            FPR11.... ********  ********
      FPR12.... ********  ********            FPR13.... ********  ********
      FPR14.... ********  ********            FPR15.... ********  ********

.
.
.

  CEEHRNUH (DSA address 24169338):
    TRANSITION DSA
    Saved Registers:
      GPR0..... 23C7BFC0  GPR1..... 00000000  GPR2..... 23CB1CA0  GPR3..... 23CB1D38
      GPR4..... 24209620  GPR5..... 24169338  GPR6..... 23CB29D0  GPR7..... 24209620
      GPR8..... 940C1000  GPR9..... 00000000  GPR10.... 23CB1D20  GPR11.... 23E78028
      GPR12.... 0007FAC0  GPR13.... 24169338  GPR14.... A3E780AC  GPR15.... 23D7D3F8
      FPR8..... ********  ********            FPR9..... ********  ********
      FPR10.... ********  ********            FPR11.... ********  ********
      FPR12.... ********  ********            FPR13.... ********  ********
      FPR14.... ********  ********            FPR15.... ********  ********

.
.
.

  funca (DSA address 24209620):
    DOWNSTACK DSA
    Saved Registers:
      GPR0..... 23C7BFC0  GPR1..... 24209F00  GPR2..... 24169128  GPR3..... 2416912C
      GPR4..... 24209620  GPR5..... 00FCD178  GPR6..... 00000000  GPR7..... A3B553FE
      GPR8..... A3B55362  GPR9..... 23E7BD0F  GPR10.... 00000000  GPR11.... A3E7AD10
      GPR12.... 0007FAC0  GPR13.... 241691B8  GPR14.... 23E7AD68  GPR15.... 00000000
      FPR8..... ********  ********            FPR9..... ********  ********
      FPR10.... ********  ********            FPR11.... ********  ********
      FPR12.... ********  ********            FPR13.... ********  ********
      FPR14.... ********  ********            FPR15.... ********  ********
```

*Figure 64. Sections of the Dump from Example C Routine (Part 2 of 2)*

2. Find the branch instructions for `funca` in the listing in Figure 65 on page 182. Notice the `BASR r7,r6` instruction at offset X'001C'. This branch is part of the instruction.

```
                      00015 |       *  void funca(int* aa) {
  ⋮
000020                                 @2L0     DS    0D
000020  00C300C5                                =F'12779717'       XPLink entrypoint marker
000024  00C500F1                                =F'12910833'
000028  FFFFFFE0                                =F'-32'
00002C  00000080                                =F'128'
000000                      00015 |        funca    DS    0D
000000  9057  4784          00015 |                STM   r5,r7,1924(r4)
000004  A74A  FF80          00015 |                AHI   r4,H'-128'
000008                      End of Prolog

000008  5010  48C0          00015 |                ST    r1,aa(,r4,2240)
                            00016 |       *    *aa = func_ptr();
00000C  5860  4804          00016 |                L     r6,#Save_ADA_Ptr_2(,r4,2052)
000010  5860  6018          00016 |                L     r6,=A(func_ptr)(,r6,24)
000014  5860  6000          00016 |                L     r6,func_ptr(,r6,0)
000018  9856  6010          00016 |                LM    r5,r6,&ADA_&EPA(r6,16)
00001C  0D76                00016 |                BASR  r7,r6
00001E  4700  0004          00016 |                NOP   4
000022  1803                00016 |                LR    r0,r3
000024  5860  48C0          00016 |                L     r6,aa(,r4,2240)
000028  5000  6000          00016 |                ST    r0,(*)int(,r6,0)
                            00017 |       *    return;
                            00018 |       *  }
00002C                      00018 |        @2L3     DS    0H

00002C                      Start of Epilog
00002C  5870  480C          00018 |                L     r7,2060(,r4)
000030  4140  4080          00018 |                LA    r4,128(,r4)
000034  07F7                00018 |                BR    r7
  ⋮
```

*Figure 65. Pseudo Assembly Listing*

3. Find the offset of `func_ptr` in the Writable Static Map, shown in Figure 66.

```
  ⋮
---------------
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 66. Writable Static Map*

4. Add the offset of `func_ptr` (X'38') to the address of WSA (X'23C7BFC0'). The result ( X'23C7BFF8') is the address of the function pointer `func_ptr` in the writable static storage area within the heap. This value is 0, indicating the variable is uninitialized.

   Figure 67 on page 183 shows the sections of the dump.

```
Enclave Control Blocks:
   :
    DLL Information:
    WSA Addr  Module Addr  Thread ID         Use Count  Name
    23C7BFC0                                 00000001   main

    WSA address.................23C7BFC0

   :

  Enclave Storage:

   :

    WSA for Program Object(s)
    WSA: 23C7BFC0
      +000000 23C7BFC0  C36DE6E2 C1404040 40404040 40404040  9985A2A4 93A34096 864086A4 95838140  |C_WSA           result of funca |
      +000020 23C7BFE0  7E406C84 15000000 23C7BFF8 00000000  00000060 23EA78A0 00000000 00000000  |=%d.....G.8.......-......>......|

   :
```

*Figure 67. 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 SYSMDUMP DD allocation information is not inherited. Even though the SYSMDUMP allocation is not inherited, a SYSMDUMP allocation must exist in the parent in order to obtain a HFS core dump. If the program terminates abnormally while running in this new address space, the kernel causes an unformatted core 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 79.

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 68. 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 core 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 core dump file to the data set, you can use IPCS to analyze the dump. Refer to "Formatting and analyzing system dumps" on page 80 for information about formatting Language Environment control blocks.

## Multithreading Consideration

Certain control blocks are locked while a dump is in progress. For example, a `csnap()` of the file control block would prevent another thread from using or dumping the same information. An attempt to do so causes the second thread to wait until the first one completes before it can continue.

## Understanding C/C++ Heap Information in Storage Reports

Storage reports that contain specific C/C++ heap information can be generated in two ways:

- By setting the Language Environment RPTSTG(ON) run-time option for Language Environment created heaps
- By issuing a stand-alone call to the C function, __uheapreport() for user–created heaps.

Details on how to request and interpret the reports are provided in the following sections.

## Language Environment Storage Report with HeapPools Statistics

To request a Language Environment storage report set RPTSTG(ON). If the C/C++ application specified the HEAPPOOLS(ON) run-time option, then the storage report displays HeapPools statistics. For a sample storage report showing HeapPools statistics for a multithreaded C/C++ application, see Figure 3 on page 15.

The following describes the C/C++ specific heap pool information.

### HeapPools Storage Statistics

The HEAPPOOLS run-time option controls usage of the heap pools storage algorithm at the enclave level. The heap pools algorithm allows for the definition of one to twelve heap pools, each consisting of a number of storage cells of a specified length.

Usage Note: The use of an alternative Vendor Heap Manager (VHM) overrides the use of the HEAPPOOLS run-time option.

***HeapPools Statistics:***

- Pool *p* size: *ssss*
  - *p* — the number of the pool
  - *ssss* — the cell size specified for the pool.
- Successful Get Heap requests: *xxxx-yyyy n*
  - *xxxx* — the low side of the 8 byte range
  - *yyyy* — the high side of the 8 byte range
  - *n* — the number of requests in the 8 byte range.
- Requests greater than the largest cell size — the number of storage requests that are not satisfied by heap pools.

**Note:** Values displayed in the HeapPools Statistics report are not serialized when collected, therefore the values are not necessarily exact.

***HeapPools Summary:*** The HeapPools Summary displays a report of the HeapPool Statistics and provides suggested percentages for current cell sizes as well as suggested cell sizes.

- Cell Size — the size of the cell specified in the HEAPPOOLS run-time option
- Extent Percent — the cell pool percent specified by the HEAPPOOLS run-time option
- Cells Per Extent — the number of cells per extent. This number is calculated using the following formula:

  ```
  Initial Heap Size * (Extent Percent/100))/(8 + Cell Size)
  ```

  with a minimum of four cells.

  **Note:** Having a small number of cells per extent is not recommended since the pool could allocate many extents, which would cause the HeapPool algorithm to perform inefficiently.
- Extents Allocated — the number of times that each pool allocated an extent.

  In order to optimize storage usage, the extents allocated should be either one or two. If the number of extents allocated is too high, then increase the percentage for the pool.

- Maximum Cells Used — the maximum number of cells used for each pool.
- Cells In Use — the number of cells that were never freed.

  **Note:** A large number in this field could indicate a storage leak.
- Suggested Percentages for current Cell Sizes — percentages calculated to find the optimal size of the cell pool extent. The calculation is based on the following formula:

  ```
  (Maximum Cells Used * (Cell Size + 8) * 100) / Initial Heap Size
  With a minimum of 1% and a maximum of 90%
  ```

  Make sure that your cell pool extents are neither too large nor too small. If your percentages are too large then additional, unreferenced virtual storage will be allocated, thereby causing the program to exhaust the region size. If the percentages are too small then the HeapPools algorithm will run inefficiently.
- Suggested Cell Sizes — sizes that are calculated to optimally use storage (assuming that the application will malloc/free with the same frequency).

  **Note:** The suggested cell sizes are given with no percentages because the usage of each new cell pool size is not known. If there are less than 12 cell sizes calculated and the last calculated cell size is smaller than the largest cell size currently in effect, the largest cell size currently in effect will be used for the last suggested cell size.

For more information about stack and heap storage, see *z/OS Language Environment Programming Guide*.

## C Function, __uheapreport, Storage Report

To generate a user-created heap storage report use the C function, *__uheapreport()*. Use the information in the report to assist with tuning your application's use of the user-created heap. For a description of the information contained in the report, see "HeapPools Storage Statistics" on page 185.

For more information on the *__uheapreport()* function, see *z/OS 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 69 on page 187.

```
Storage Report for Enclave  08/30/01 11:42:23 AM
Language Environment V01 R03.00

    HeapPools Statistics:
      Pool 1 size:    32
        Successful Get Heap requests:     1-   32          11250
      Pool 2 size:   128
        Successful Get Heap requests:    97-  128           3306
      Pool 3 size:   512
        Successful Get Heap requests:   481-  512            864
      Pool 4 size:  2048
        Successful Get Heap requests:  2017- 2048            216
      Pool 5 size:  8192
        Successful Get Heap requests:  8161- 8192             54
      Pool 6 size: 16384
        Successful Get Heap requests: 16353-16384             27
      Requests greater than the largest cell size:             0
    HeapPools Summary:
      Cell  Extent   Cells Per  Extents    Maximum     Cells In
      Size  Percent  Extent     Allocated  Cells Used  Use
      ------------------------------------------------------------
         32   15        3750           1        3750           0
        128   15        1102           1        1102           0
        512   15         288           1         288           0
       2048   15          72           1          72           0
       8192   15          18           1          18           0
      16384   15           9           1           9           0
      ------------------------------------------------------------
      Suggested Percentages for current Cell Sizes:
        32,15,128,15,512,15,2048,15,8192,15,16384,15
      Suggested Cell Sizes:
        32,,128,,512,,2048,,8192,,16384,
End of Storage Report
```

*Figure 69. Storage Report Generated by __uheapreport()*

# 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 and OS/390 Language Reference*.

Scope terminators can also help you trace the logic of your program because they clearly indicate the end of a statement. For a detailed description of scope terminators, see *Enterprise COBOL for z/OS and OS/390 Programming Guide* or *COBOL for OS/390 & VM Programming Guide*.

## Finding input/output errors

VSAM file status keys can help you determine whether routine errors are due to the logic of your routine or are I/O errors occurring on the storage media.

© Copyright IBM Corp. 1991, 2004

To use file status keys as a debugging aid, include a test after each I/O statement to check for a value other than 0 in the file status key. If the value is other than 0, you can expect to receive an error message. You can use a nonzero value to indicate how the I/O procedures in the routine were coded. You can also include procedures to correct the error based on the file status key value.

The file status key values and their associated meanings are described in *Enterprise COBOL for z/OS and OS/390 Language Reference* and in *COBOL for OS/390 & VM Language Reference*.

## Handling input/output errors

If you have determined that the problem lies in one of the I/O procedures in your program, you can include the USE EXCEPTION/ERROR declarative to help debug the problem. If the file does not open, the appropriate USE EXCEPTION/ERROR declarative is activated. You can specify the appropriate declarative for the file or for the different open attributes—INPUT, OUTPUT, I/O, or EXTEND.

Code each USE AFTER STANDARD ERROR statement in a separate section immediately after the Declarative Section keyword of the Procedure Division. See the rules for coding such usage statements in *Enterprise COBOL for z/OS and OS/390 Language Reference* or in *COBOL for OS/390 & VM Language Reference*.

## Validating data (class test)

If you suspect that your program is trying to perform arithmetic on nonnumeric data or is somehow receiving the wrong type of data on an input record, you can use the class test to validate the type of data. For a detailed discussion of how to use the class test to check for incompatible data, see *Enterprise COBOL for z/OS and OS/390 Programming Guide* or *COBOL for OS/390 & VM Programming Guide* .

## Assessing switch problems

Using INITIALIZE or SET statements to initialize a table or data item is useful when you suspect that a problem is caused by residual data left in those fields. If your problem occurs intermittently and not always with the same data, the problem could be that a switch is not initialized, but is generally set to the right value (0 or 1). By including a SET statement to ensure that the switch is initialized, you can determine whether or not the uninitialized switch is the cause of the problem. For a detailed discussion of how to use the INITIALIZE and SET statements, see *Enterprise COBOL for z/OS and OS/390 Programming Guide* or *COBOL for OS/390 & VM Programming Guide*.

## Generating information about procedures

You can use the USE FOR DEBUGGING declarative to include COBOL statements in a COBOL program and specify when they should run. Use these statements to generate information about your program and how it is running.

For example, to check how many times a procedure is run, include a special procedure for debugging (in the USE FOR DEBUGGING declarative) that adds 1 to a counter each time control passes to that procedure. The adding-to-a-counter technique can be used as a check for:

- How many times a PERFORM ran. This shows you whether the control flow you are using is correct.
- How many times a loop routine actually runs. This tells you whether the loop is running and whether the number you have used for the loop is accurate.

Code each USE FOR DEBUGGING declarative in a separate section in the DECLARATIVES SECTION of the PROCEDURE DIVISION. See the rules for coding them in *Enterprise COBOL for z/OS and OS/390 Language Reference* or in *COBOL for OS/390 & VM Language Reference*.

You can use debugging lines, debugging statements, or both in your program. Debugging lines are placed in your program, and are identified by a `D` in position 7. Debugging statements are coded in the DECLARATIVES SECTION of the PROCEDURE DIVISION.

- The USE FOR DEBUGGING declaratives must:
  - Be only in the DECLARATIVES SECTION
  - Follow a DECLARATIVES header USE FOR DEBUGGING

  With USE FOR DEBUGGING, the TEST compiler option must have the NONE hook-location suboption specified or the NOTEST compiler option must be specified. The TEST compiler option and the DEBUG run-time option are mutually exclusive, with DEBUG taking precedence.
- Debugging lines must have a `D` in position 7 to identify them.

To use debugging lines and statements in your program, you must include both:

- WITH DEBUGGING MODE in the SOURCE-COMPUTER paragraph in the ENVIRONMENT DIVISION
- The DEBUG run-time option

Figure 70 on page 192 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 70. Example of Using the WITH DEBUGGING MODE Clause*

In the example in Figure 70, portions of a program are shown to illustrate the kind of statements needed to use the USE FOR DEBUGGING declarative. The DISPLAY statement specified in the DECLARATIVES SECTION issues the:

```
Trace For Procedure-Name : 501-Some-Routine nn
```

message every time the PERFORM 501-SOME-ROUTINE runs. The total shown, *nn*, is the value accumulated in the data item named TOTAL.

Another use for the DISPLAY statement technique shown above is to show the flow through your program. You do this by changing the USE FOR DEBUGGING declarative in the DECLARATIVES SECTION to:

```
USE FOR DEBUGGING ON ALL PROCEDURES.
```

and dropping the word TOTAL from the DISPLAY statement.

## Using COBOL listings

When you are debugging, you can use one or more of the following listings:
- Sorted Cross-Reference listing
- Data Map listing
- Verb Cross-Reference listing
- Procedure Division Listings

This section gives an overview of each of these listings and specifies the compiler option you use to obtain each listing. For a detailed description of available listings, sample listings, and a complete description of COBOL compiler options, see *Enterprise COBOL for z/OS and OS/390 Programming Guide* or *COBOL for OS/390 & VM Programming Guide*.

*Table 19. Compiler-generated COBOL Listings and Their Contents*

| Name | Contents | Compiler Option |
|---|---|---|
| Sorted Cross-Reference Listings | Provides sorted cross-reference listings of DATA DIVISION, PROCEDURE DIVISION, and program names. The listings provide the location of all references to this information. | XREF |
| Data Map listing | Provides information about the locations of all DATA DIVISION items and all implicitly declared variables. This option also supplies a nested program map, which indicates where the programs are defined and provides program attribute information. | MAP |
| Verb Cross-Reference listing | Produces an alphabetic listing of all the verbs in your program and indicates where each is referenced. | VBREF |
| Procedure Division listings | Tells the COBOL compiler to generate a listing of the PROCEDURE DIVISION along with the assembler coding produced by the compiler. The list output includes the assembler source code, a map of the task global table (TGT), information about the location and size of WORKING-STORAGE and control blocks, and information about the location of literals and code for dynamic storage usage. | LIST |
| Procedure Division listings | Instead of the full PROCEDURE DIVISION listing with assembler expansion information, you can use the OFFSET compiler option to get a condensed listing that provides information about the program verb usage, global tables, WORKING-STORAGE, and literals. The OFFSET option takes precedence over the LIST option. That is, OFFSET and LIST are mutually exclusive; if you specify both, only OFFSET takes effect. | OFFSET |

## Generating a Language Environment dump of a COBOL program

The two sample programs shown in Figure 71 on page 194 and Figure 72 on page 195 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 71. COBOL Program COBDUMP1 Calling COBDUMP2*

## COBOL program that calls the Language Environment CEE3DMP callable service

In the example in Figure 72 on page 195, 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 72. 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 73 on page 196. 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 197.

```
⋮
CEE3DMP V1 R3.0: COBOL DUMP                                                    08/30/01 11:49:15 AM                Page:    1

CEE3DMP called by program unit COBDUMP2 at statement 40 (offset +00000430).

Registers on Entry to CEE3DMP:

  PM....... 0000
  GPR0..... 0D41F838  GPR1..... 00027158  GPR2..... 0D4232C8  GPR3..... 0D302302
  GPR4..... 0D301FA0  GPR5..... 00047038  GPR6..... 00000000  GPR7..... 00FCAB00
  GPR8..... 0D423160  GPR9..... 0D41F700  GPR10.... 0D302070  GPR11.... 0D302234
  GPR12.... 00016A48  GPR13.... 000270C0  GPR14.... 8001E0E2  GPR15.... 8D353858
  FPR0..... 00000000  00000000            FPR2..... 00000000  00000000
  FPR4..... 00000000  00000000            FPR6..... 00000000  00000000
    GPREG STORAGE:
      Storage around GPR0 (0D41F838)
        -0020 0D41F818  00000000 00000000 00000000 0D423110  0D423160 00000000 0D4230D8 0004C038  |...................-.......Q...|
        +0000 0D41F838  00000000 00000000 0D302450 07FE07FE  00000000 00000000 00001FFF 07FE0000  |...........&....................|
        +0020 0D41F858  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|


⋮
Information for enclave COBDUMP1

  Information for thread 8000000000000000

  Registers on Entry to CEE3DMP:
    PM....... 0000
    GPR0..... 0D41F838  GPR1..... 00027158  GPR2..... 0D4232C8  GPR3..... 0D302302
    GPR4..... 0D301FA0  GPR5..... 00047038  GPR6..... 00000000  GPR7..... 00FCAB00
    GPR8..... 0D423160  GPR9..... 0D41F700  GPR10.... 0D302070  GPR11.... 0D302234
    GPR12.... 00016A48  GPR13.... 000270C0  GPR14.... 8001E0E2  GPR15.... 8D353858
    FPR0..... 00000000  00000000            FPR2..... 00000000  00000000
    FPR4..... 00000000  00000000            FPR6..... 00000000  00000000
    GPREG STORAGE:
      Storage around GPR0 (0D41F838)
        -0020 0D41F818  00000000 00000000 00000000 0D423110  0D423160 00000000 0D4230D8 0004C038  |...................-.......Q...|
        +0000 0D41F838  00000000 00000000 0D302450 07FE07FE  00000000 00000000 00001FFF 07FE0000  |...........&....................|
        +0020 0D41F858  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|

⋮
   Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    000270C0  COBDUMP2      0D301F68  +00000430  COBDUMP2      0D301F68  +00000430          40  GO                Call
    00027018  COBDUMP1      0D300100  +0000033E  COBDUMP1      0D300100  +0000033E          23  GO                Call

   Parameters, Registers, and Variables for Active Routines:    COBDUMP2 (DSA address 000270C0):
      Saved Registers:
        GPR0..... 0D41F838  GPR1..... 00027158  GPR2..... 0D4232C8  GPR3..... 0D302302
        GPR4..... 0D301FA0  GPR5..... 00047038  GPR6..... 00000000  GPR7..... 00FCAB00
        GPR8..... 0D423160  GPR9..... 0D41F700  GPR10.... 0D302070  GPR11.... 0D302234
        GPR12.... 00016A48  GPR13.... 000270C0  GPR14.... 8001E0E2  GPR15.... 8D353858
    GPREG STORAGE:
      Storage around GPR0 (0D41F838)
        -0020 0D41F818  00000000 00000000 00000000 0D423110  0D423160 00000000 0D4230D8 0004C038  |...................-.......Q...|
        +0000 0D41F838  00000000 00000000 0D302450 07FE07FE  00000000 00000000 00001FFF 07FE0000  |...........&....................|
        +0020 0D41F858  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|

⋮
```

*Figure 73. Sections of the Language Environment Dump Called from COBDUMP2 (Part 1 of 2)*

```
Local Variables:
          13 IOFSS1                           FILE SPECIFIED AS: OPTIONAL, ORGANIZATION=VSAM SEQUENTIAL,
                                             ACCESS MODE=SEQUENTIAL, RECFM=FIXED.  CURRENT STATUS OF
                                             FILE IS: NOT OPEN, VSAM 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
          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)                          'YYY'
                   SUB(2,1)                          'ZZZ'
                   SUB(2,2)                          '   '
          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 00027018):
   Saved Registers:
     GPR0..... 0D41F1A8  GPR1..... 000270B0  GPR2..... 0D4230D8  GPR3..... 0D3003EA
     GPR4..... 0D300138  GPR5..... 00015AE8  GPR6..... 00000000  GPR7..... 00000000
     GPR8..... 0D423088  GPR9..... 0D41F078  GPR10.... 0D300208  GPR11.... 0D300328
     GPR12.... 00016A48  GPR13.... 00027018  GPR14.... 8D300440  GPR15.... 0D301F68
 GPREG STORAGE:
   Storage around GPR0 (0D41F1A8)
     -0020 0D41F188  00000000 0D423088 00000000 00000000  00000000 0D423038 0D423088 00000000  |.......h...................h...|
     +0000 0D41F1A8  00047370 00000000 0D300528 07FE07FE  00000000 00000000 00001FFF 07FE0000  |............................|
     +0020 0D41F1C8  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |............................|

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 73. 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 74 on page 198, displays the following information for each active COBOL program:

- DSA
- Program name and date/time of compile
- COBOL compiler Version, Release, Modification, and User Level
- COBOL control blocks TGT and CLLE. The layout of the TGT can be found by looking at the compiler listing of the COBOL program. The CLLE is a COBOL

control block that is allocated by the COBOL runtime for each program. The CLLE is dumped for IBM service personnel use.

```
⋮
Control Blocks for Active Routines:
    DSA for COBDUMP2: 000270C0
      +000000  FLAGS.... 0010      member... 4001      BKC...... 00027018  FWC...... 00027168  R14...... 8001E0E2
      +000010  R15...... 8D353858  R0....... 0D41F838  R1....... 00027158  R2....... 0D4232C8  R3....... 0D302302
      +000024  R4....... 0D301FA0  R5....... 00047038  R6....... 00000000  R7....... 00FCAB00  R8....... 0D423160
      +000038  R9....... 0D41F700  R10...... 0D302070  R11...... 0D302234  R12...... 00016A48  reserved. 00000000
      +00004C  NAB...... 00027168  PNAB..... 00000000  reserved. 00000000  000270C0  0D41F700  00101001
      +000064  reserved. 00027018  reserved. 000270E4  MODE..... 8D30239A  reserved. 000270EC  000270F4
      +000078  reserved. 000270F0  reserved. 000270F8

⋮
Program COBDUMP2 was compiled 08/04/01 11:49:09 AM
    COBOL Version = 02 Release = 01 Modification = 01     User Level = '    '
    TGT for COBDUMP2: 0D41F700
      +000000 0D41F700  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000020 0D41F720 - +00003F 0D41F73F             same as above
      +000040 0D41F740  00000000 00000000 F3E3C7E3 00000000  05000000 42030220 00047038 000187FC  |........3TGT.................g.|
      +000060 0D41F760  0D41F920 00000001 00000174 00000000  00000000 0D423100 00000000 00000000  |..9.............................|
      +000080 0D41F780  00016A48 0000021C 00000000 00000000  00000000 00000001 E2E8E2D6 E4E34040  |.......................SYSOUT   |
      +0000A0 0D41F7A0  C9C7E9E2 D9E3C3C4 00000000 00000000  00000000 00000000 00000000 00000000  |IGZSRTCD........................|
      +0000C0 0D41F7C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +0000E0 0D41F7E0  00000000 00000000 0D302064 00000001  0D41F908 00047370 0D3020F3 0D41F83C  |..................9........3..8.|
      +000100 0D41F800  0D301F68 0D302078 0D41F904 0D30206C  0D41F904 0D423160 00000000 00000000  |..........9....%..9....-........|
      +000120 0D41F820  00000000 0D423110 0D423160 00000000  0D4230D8 0004C038 00000000 00000000  |............-.......Q............|
      +000140 0D41F840  0D302450 07FE07FE 00000000 00000000  00001FFF 07FE0000 00000000 00000000  |...&............................|
      +000160 0D41F860  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000180 0D41F880 - +0001FF 0D41F8FF             same as above
      +000200 0D41F900  00000000 0D41F950 40000000 00000000  00000000 0D41FA50 00000001 00000000  |......9& ..............&........|

⋮
     CLLE for COBDUMP2: 00047370
      +000000 00047370  C3D6C2C4 E4D4D7F2 00000100 00000000  84810000 0D301F68 0D41F700 00000000  |COBDUMP2........da........7.....|
      +000020 00047390  00000000 0D41F5AC 0D41F6B8 00047328  00047000 000000C8 000000C0 00000000  |......5...6............H........|

⋮
```

*Figure 74. Control Block Information for Active COBOL Routines*

## Storage for each active routine

The Storage for Active Routines section of the dump, shown in Figure 75 on page 199, displays the following information for each COBOL program:

- Program name
- Contents of the base locators for files, WORKING-STORAGE, LINKAGE SECTION, LOCAL-STORAGE SECTION, variably-located areas, and EXTERNAL data.
- File record contents.
- WORKING-STORAGE, including the base locator for WORKING-STORAGE (BLW) and program class storage.

```
⋮
  Storage for Active Routines:
    COBDUMP2:
      Contents of base locators for files are:
           0-0004C038

      Contents of base locators for working storage are:
           0-0D423160

      Contents of base locators for the linkage section are:
           0-00000000        1-0D4230D8

      No variably located areas were used in this program.

      No EXTERNAL data was used in this program.

      No object instance data were used in this program.

      No local storage was used in this program.

      No DSA indexes were used in this program.

      No indexes were used in this program.
    File record contents for COBDUMP2
    ESDS1DD  (BLF-0): 0004C038
      +000000 0004C038  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000020 0004C058 - +00003F 0004C077             same as above

    Working storage for COBDUMP2
    BLW-0: 0D423160
      +000000 0D423160  000000E7 E7E70000 00E8E8E8 000000E9  E9E90000 00000000 C3D6C2D6 D340C4E4  |...XXX...YYY...ZZZ......COBOL DU|
      +000020 0D423180  D4D74040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |MP                              |
      +000040 0D4231A0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
      +000060 0D4231C0  40404040 40404040 40C2D3D6 C3D2E240  E2E3D6D9 C1C7C540 D7C1C7C5 4DF5F55D  |         BLOCKS STORAGE PAGE(55)|
      +000080 0D4231E0  40C6C9D3 C5E24040 40404040 40404040  40404040 40404040 40404040 40404040  | FILES                          |
      +0000A0 0D423200  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
      +0000C0 0D423220 - +00015F 0D4232BF             same as above
      +000160 0D4232C0  40404040 40404000 00000000 00000000  00000000 00000000 00000000 00000000  |       ........................|

    Program class storage: 0D423100
      +000000 0D423100  000001DB 00000000 00000000 0D41F940  00000000 00000000 00000000 00000000  |.............9 ................|
      +000020 0D423120  C9C7E9E2 D9E3C3C4 00000000 00000000  00000000 00000000 00000000 00000000  |IGZSRTCD........................|
      +000040 0D423140  E2E8E2D6 E4E34040 00000000 00000000  0E000000 00000000 0F000000 00000000  |SYSOUT  ........................|
      +000060 0D423160  000000E7 E7E70000 00E8E8E8 000000E9  E9E90000 00000000 C3D6C2D6 D340C4E4  |...XXX...YYY...ZZZ......COBOL DU|
      +000080 0D423180  D4D74040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |MP                              |
      +0000A0 0D4231A0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
      +0000C0 0D4231C0  40404040 40404040 40C2D3D6 C3D2E240  E2E3D6D9 C1C7C540 D7C1C7C5 4DF5F55D  |         BLOCKS STORAGE PAGE(55)|
      +0000E0 0D4231E0  40C6C9D3 C5E24040 40404040 40404040  40404040 40404040 40404040 40404040  | FILES                          |
      +000100 0D423200  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
      +000120 0D423220 - +0001BF 0D4232BF             same as above
      +0001C0 0D4232C0  40404040 40404000 00000000 00000000  00000000 00000000 00000000 00000000  |       ........................|
    Program class storage: 0D41F940
      +000000 0D41F940  000001BE 00000000 0D423100 0004C028  C6C3C200 01020000 FFFFFFFF FFFFFFFF  |................FCB.............|
      +000020 0D41F960  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  FFFFFFFF 00000000 00000000 00000000  |................................|
      +000040 0D41F980  00000000 00000000 00000000 8001B9E0  8001B9E0 8001B9E0 8D414938 8001B9E0  |................................|
      +000060 0D41F9A0  8001B9E0 8001B9E0 8D414938 00000000  00000000 00000000 00000000 00000000  |................................|
      +000080 0D41F9C0  00000000 00000000 00000000 00000000  00000000 00000000 C3D6C2C4 E4D4D7F2  |........................COBDUMP2|
      +0000A0 0D41F9E0  C5E2C4E2 F1C4C440 00000000 00000000  00000000 0D302194 00000000 00000000  |ESDS1DD ...............m........|
      +0000C0 0D41FA00  00000000 00000000 00000000 00000000  00000000 00000000 00008800 00000000  |..........................h.....|
      +0000E0 0D41FA20  00000000 00000000 00000028 00000000  00000000 00000000 00000000 00000000  |................................|
      +000100 0D41FA40  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000120 0D41FA60 - +0001BF 0D41FAFF             same as above
    Program class storage: 0004C028
      +000000 0004C028  0000003F 00000000 0D41F940 00000000  00000000 00000000 00000000 00000000  |.........9 .....................|
      +000020 0004C048  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
⋮
```

*Figure 75. Storage for Active COBOL Programs*

## Enclave-level data

The Enclave Control Blocks section of the dump, shown in Figure 76 on page 200, displays the following information:

- RUNCOM control block. The RUNCOM is a control block that is allocated by the COBOL runtime to anchor enclave level resources. The RUNCOM is dumped for IBM service personnel use.
- Storage for all run units
- COBOL control blocks FCB, FIB, and GMAREA. The FCB, FIB, and GMAREA are control blocks used for COBOL file processing. These control blocks are dumped for IBM service personnel use.

```
Enclave Control Blocks:

⋮
 RUNCOM: 00047038
  +000000 00047038  C3F3D9E4 D5C3D6D4 000002D8 04860000  000159C8 00000001 00005F78 00000000  |C3RUNCOM...Q.f.....H......^.....|
  +000020 00047058  00000000 0D300100 00047328 00000000  00015AE8 00000000 00000000 00000000  |..................!Y............|
  +000040 00047078  00018A80 000181BC 00000000 000187FC  00000000 00000000 00016A48 00000000  |......a.......g.............|
  +000060 00047098  00000000 000473A8 00000000 00000000  00000000 F0F0F0F0 F0F0F0F0 0D41F6B8  |.......y............00000000..6.|

⋮

Enclave Storage:

⋮
 Rununit class storage: 000473A8
  +000000 000473A8  000000C0 00000000 00000000 0D41F6F0  00000000 00000000 00000000 00000000  |..............60................|
  +000020 000473C8  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000040 000473E8 - +0000BF 00047467             same as above
 Rununit class storage: 0D41F6F0
  +000000 0D41F6F0  00000248 00000000 000473A8 00047360  00000000 00000000 00000000 00000000  |...........y...-................|
  +000020 0D41F710  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000040 0D41F730  00000000 00000000 00000000 00000000  00000000 00000000 F3E3C7E3 00000000  |........................3TGT....|
  +000060 0D41F750  05000000 42030220 00047038 000187FC  0D41F920 00000001 00000174 00000000  |..............g...9.............|

⋮
 Rununit class storage: 00047360
  +000000 00047360  00000040 00000000 0D41F6F0 0D41F2A0  C3D6C2C4 E4D4D7F2 00000100 00000000  |... ......60..2.COBDUMP2........|
  +000020 00047380  84810000 0D301F68 0D41F700 00000000  00000000 0D41F5AC 0D41F6B8 00047328  |da........7...........5...6.....|
 Rununit class storage: 0D41F2A0
  +000000 0D41F2A0  00000448 00000000 00047360 0D41F068  C8C1E340 00000100 00000000 00000000  |...........-..0.HAT ............|
  +000020 0D41F2C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000040 0D41F2E0 - +00031F 0D41F5BF             same as above

⋮
 Rununit class storage: 0D41F068
  +000000 0D41F068  0000022C 00000000 0D41F2A0 00047318  00000000 00000000 00000000 00000000  |.........2......................|
  +000020 0D41F088  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000040 0D41F0A8  00000000 00000000 00000000 00000000  00000000 00000000 F3E3C7E3 00000000  |........................3TGT....|
  +000060 0D41F0C8  05000000 60030220 00047038 000187FC  0D41F288 00000000 00000064 00000000  |....-.........g...2h............|

⋮
 Rununit class storage: 00047318
  +000000 00047318  00000040 00000000 0D41F068 00000000  C3D6C2C4 E4D4D7F1 00000100 00000000  |... ......0.....COBDUMP1........|
  +000020 00047338  94810000 8D300100 0D41F078 00000000  00000000 0D41F5A8 0D41F6B8 00000000  |ma........0...........5y..6.....|

File Control Blocks:
 FCB for file ESDS1DD  in program COBDUMP2: 0D41F950
  +000000 0D41F950  C6C3C200 01020000 FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  |FCB.............................|
  +000020 0D41F970  FFFFFFFF 00000000 00000000 00000000  00000000 00000000 00000000 8001B9E0  |................................|
  +000040 0D41F990  8001B9E0 8001B9E0 8D414938 8001B9E0  8001B9E0 8001B9E0 8D414938 00000000  |.............I.....I............|
  +000060 0D41F9B0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000080 0D41F9D0  00000000 00000000 C3D6C2C4 E4D4D7F2  C5E2C4E2 F1C4C440 00000000 00000000  |........COBDUMP2ESDS1DD ........|
  +0000A0 0D41F9F0  00000000 0D302194 00000000 00000000  00000000 00000000 00000000 00000000  |.......m........................|
  +0000C0 0D41FA10  00000000 00000000 00008800 00000000  00000000 00000000 00000028 00000000  |..........h.....................|
  +0000E0 0D41FA30  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
 FIB for file ESDS1DD  in program COBDUMP2: 0D302194
  +000000 0D302194  C6C9C200 0103C5E2 C4E2F1C4 C4400088  8080A000 00008000 00000000 00000028  |FIB...ESDS1DD .h................|
  +000020 0D3021B4  00010000 00000000 00000000 00000000  00000000 00000000 0D30218D 00000000  |.............................0..|
  +000040 0D3021D4  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000060 0D3021F4 - +00007F 0D302213             same as above
  +000080 0D302214  0000C9D6 C6E2E2F1 40404040 40404040  40404040 40404040 40404040 40404040  |..IOFSS1                        |
 GMAREA for file ESDS1DD  in program COBDUMP2: 00000000
  +000000 00000000  Inaccessible storage.
```

*Figure 76. Enclave-Level Data for COBOL Programs*

### Process-level data

The Process Control Block section of the dump, shown in Figure 77, displays COBOL process-level control blocks THDCOM, COBCOM, COBVEC, and ITBLK.

In a non-CICS environment, the ITBLK control block only appears when a VS COBOL II program is active. In a CICS environment, the ITBLK control block always appears.

COBOL control blocks THDCOM, COBCOM, COBVEC and ITBLK are dumped for IBM service personnel use.

```
 Process Control Blocks:

⋮
   THDCOM: 00018A80
     +000000 00018A80  C3F3E3C8 C4C3D6D4 000001E8 81000000  00000100 00000000 00018108 000181BC  |C3THDCOM...Ya.............a...a.|
     +000020 00018AA0  00047038 00000000 C3D6C2C4 E4D4D7F1  00000000 00000000 00000000 00000000  |........COBDUMP1.................|
     +000040 00018AC0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
     +000060 00018AE0 - +00007F 00018AFF             same as above

⋮
   COBCOM: 00018108
     +000000 00018108  C3F3C3D6 C2C3D6D4 00000978 F0F2F0F9  F0F00000 00000000 00000000 00000000  |C3COBCOM....020900..............|
     +000020 00018128  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
     +000040 00018148  00000000 00000000 00000000 00000000  00000000 00000000 906000D6 000181BC  |........................-.O..a.|
     +000060 00018168  000187FC 00000100 00820000 00000000  00008000 08000000 00018A80 00000000  |..g.....b.......................|

⋮
   COBVEC: 000181BC
     +000000 000181BC  0001843C 00018442 00018448 0001844E  00018454 0001845A 00018460 00018466  |..d...d...d...d+..d...d!..d-..d.|
     +000020 000181DC  0001846C 00018472 00018478 0001847E  00018484 0001848A 00018490 00018496  |..d%..d...d...d=..dd..d...d...do|
     +000040 000181FC  0001849C 000184A2 000184A8 000184AE  000184B4 000184BA 000184C0 000184C6  |..d...ds..dy..d...d...d...d...dF|
     +000060 0001821C  000184CC 000184D2 000184D8 000184DE  000184E4 000184EA 000184F0 000184F6  |..d...dK..dQ..d...dU..d...d0..d6|

⋮
```

*Figure 77. 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 78 on page 202 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 79 on page 202.

```
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 78. 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 79. The message is `IGZ0006S The reference to table SLOT by verb number 01 on line 000011 addressed an area outside the region of the table.` The message indicates that line 11 was the current COBOL statement when the error occurred. For more information about this message, see *z/OS Language Environment Run-Time Messages*.
2. Statement 11 in the traceback section of the dump occurred in program COBOLX.

```
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.        08/30/01 11:48:58 AM                Page:    1

Information for enclave COBOLX

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    0002A768  CEEHDSP       0D3386F0  +00003032  CEEHDSP       0D3386F0  +00003032              CEEPLPKA           Call
    0002A5D0  CEEHSGLT      0D344250  +0000005C  CEEHSGLT      0D344250  +0000005C              CEEPLPKA           Exception
    0002A0B8  IGZCMSG       0D400BF8  +0000038C  IGZCMSG       0D400BF8  +0000038C              IGZCPAC            Call
    0002A018  COBOLX        00007808  +00000286  COBOLX        00007808  +00000286          11  GO                 Call

    Condition Information for Active Routines
    Condition Information for CEEHSGLT (DSA address 0002A5D0)
      CIB Address: 0002ADE0
      Current Condition:
        IGZ0006S The reference to table SLOT by verb number 01 on line 000011 addressed an area outside the region of the table.

      Location:
        Program Unit: CEEHSGLT Entry: CEEHSGLT Statement:  Offset: +0000005C
    Storage dump near condition, beginning at location: 0D34429C
      +000000 0D34429C  F010D20B D0801000 58A0C2B8 58F0A01C  05EFD20B D098B108 41A0D098 50A0D08C  |0.K.......B..0....K..q.....q&...|
```

*Figure 79. Sections of Language Environment Dump for COBOLX (Part 1 of 2)*

```
    Parameters, Registers, and Variables for Active Routines:
    CEEHDSP (DSA address 0002A768):
      Saved Registers:
        GPR0..... 0D33BB6C  GPR1..... 0002AB80  GPR2..... 00000001  GPR3..... 00000003
        GPR4..... 00000008  GPR5..... 0002ADE0  GPR6..... 00020038  GPR7..... 0002B767
        GPR8..... 0D33B6ED  GPR9..... 0D33A6EE  GPR10.... 0D3396EF  GPR11.... 8D3386F0
        GPR12.... 00019A48  GPR13.... 0002A768  GPR14.... 800210E2  GPR15.... 8D34E858
    GPREG STORAGE:
      Storage around GPR0 (0D33BB6C)
        -0020 0D33BB4C  0D33BB78 0D33BBBC 0D33BB80 0D33BBC0  0D33BBB0 00000000 00000001 00000002  |................................|
        +0000 0D33BB6C  00000003 00000004 00000006 00000007  00000008 00000009 0000000A 0000000B  |................................|
        +0020 0D33BB8C  0000000D 0000000E 00000010 00000015  00000017 00000064 00000069 00000080  |................................|
⋮
    CEEHSGLT (DSA address 0002A5D0):
      Saved Registers:
        GPR0..... 000079B8  GPR1..... 0002A290  GPR2..... 0002A290  GPR3..... 00000000
        GPR4..... 00020038  GPR5..... 00020038  GPR6..... 0002A3D4  GPR7..... 00000005
        GPR8..... 0001BA80  GPR9..... 00009140  GPR10.... 00020038  GPR11.... 8D344250
        GPR12.... 00019A48  GPR13.... 0002A5D0  GPR14.... 800210CE  GPR15.... 8D343AA8
    GPREG STORAGE:
      Storage around GPR0 (000079B8)
        -0020 00007998  08000004 00224000 00000006 C0000140  00040800 00040028 02400002 08000004  |...... ........ ........ ......|
        +0000 000079B8  001F03C0 00060800 0004001C 40000000  0040C000 01400006 08000004 002202C0  |............ .... ... ..........|
        +0020 000079D8  00060800 00040022 183F4100 10A05500  C00C05F0 47D0F00C 58F0C300 05EF181F  |...................0..0..0C.....|
⋮
    IGZCMSG  (DSA address 0002A0B8):
      Saved Registers:
        GPR0..... 000079B8  GPR1..... 0002A290  GPR2..... 0002A290  GPR3..... 00000000
        GPR4..... 00020038  GPR5..... 00020038  GPR6..... 0002A3D4  GPR7..... 00000005
        GPR8..... 0001BA80  GPR9..... 00009140  GPR10.... 0004A038  GPR11.... 8D400BF8
        GPR12.... 00019A48  GPR13.... 0002A0B8  GPR14.... 8002463E  GPR15.... 8D344250
    GPREG STORAGE:
      Storage around GPR0 (000079B8)
        -0020 00007998  08000004 00224000 00000006 C0000140  00040800 00040028 02400002 08000004  |...... ........ ........ ......|
        +0000 000079B8  001F03C0 00060800 0004001C 40000000  0040C000 01400006 08000004 002202C0  |............ .... ... ..........|
        +0020 000079D8  00060800 00040022 183F4100 10A05500  C00C05F0 47D0F00C 58F0C300 05EF181F  |...................0..0..0C.....|
⋮
    COBOLX (DSA address 0002A018):
      Saved Registers:
        GPR0..... 0002A0B8  GPR1..... 0000799E  GPR2..... 00000010  GPR3..... 0001B7FC
        GPR4..... 00007840  GPR5..... 00018AE8  GPR6..... 00000000  GPR7..... 00000000
        GPR8..... 000093A0  GPR9..... 00009140  GPR10.... 00007910  GPR11.... 00007A3E
        GPR12.... 00007904  GPR13.... 0002A018  GPR14.... 80007A90  GPR15.... 8D400BF8
    GPREG STORAGE:
      Storage around GPR0 (0002A0B8)
        -0020 0002A098  0D30021C 0D300218 0D300220 000179A4  00000000 00000000 00000000 00000000  |...............u................|
        +0000 0002A0B8  00101001 0002A018 0002A5D0 8002463E  8D344250 000079B8 0002A290 0002A290  |..........v........&......s...s.|
        +0020 0002A0D8  00000000 00020038 00020038 0002A3D4  00000005 0001BA80 00009140 0004A038  |..............tM..........j ....|
⋮
      Local Variables:
              6 77 J                9999 COMP        00009
              7 01 TABLE-X          AN-GR
              8  02 SLOT             9999  OCCURS 8
                    SUB(1)           COMP             00000
```

*Figure 79. Sections of Language Environment Dump for COBOLX (Part 2 of 2)*

3. Find the statement on line 11 in the listing for program COBOLX, shown in Figure 80 on page 204. This statement moves the 1 value to the array SLOT (J).

```
  PP 5648-A25 IBM COBOL for OS/390 & VM  2.1.1                  COBOLX    Date 11/04/1999  Time 11:48:54   Page     3
  LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  Map and Cross Reference
/* COBOLX
  000001               ID DIVISION.
  000002               PROGRAM-ID. COBOLX.
  000003               ENVIRONMENT DIVISION.
  000004               DATA DIVISION.
  000005               WORKING-STORAGE SECTION.
  000006               77  J     PIC 9(4) USAGE COMP.
  000007               01  TABLE-X.
  000008                02  SLOT PIC 9(4) USAGE COMP OCCURS 8 TIMES.
  000009               PROCEDURE DIVISION.
  000010                   MOVE 9 TO J.
  000011                   MOVE 1 TO SLOT (J).
  000012                   GOBACK.
*/ COBOLX
⋮
```

*Figure 80. COBOL Listing for COBOLX*

4. Find the values of the local variables in the Parameters, Registers, and Variables for Active Routines section of the traceback, shown in Figure 79 on page 202. 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 81 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 81 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 81. 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 82 on page 205. 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 +00000338.

```
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.        08/30/01 5:17:40 PM                Page:    1

Information for enclave COBOLY

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    0002A5A8  CEEHDSP       0D3386F0  +000028DE  CEEHDSP       0D3386F0  +000028DE              CEEPLPKA           Call
    0002A410  CEEHSGLT      0D344250  +0000005C  CEEHSGLT      0D344250  +0000005C              CEEPLPKA           Exception
    0002A2A8  IGZCLDL       0D3FF098  +0000011A  IGZCLDL       0D3FF098  +0000011A              IGZCPAC            Call
    0002A0C0  IGZCFCC       0001E128  +00000398  IGZCFCC       0001E128  +00000398              IGZCFCC            Call
    0002A018  COBOLY        000077E0  +00000338  COBOLY        000077E0  +00000338           8  GO                 Call

  Condition Information for Active Routines
    Condition Information for CEEHSGLT (DSA address 0002A410)
      CIB Address: 0002AC20
      Current Condition:
        CEE0198S The termination of a thread was signaled due to an unhandled condition.
      Original Condition:
        CEE3501S The module UNKNOWN  was not found.
      Location:
        Program Unit: CEEHSGLT Entry: CEEHSGLT Statement:  Offset: +0000005C
    Storage dump near condition, beginning at location: 0D34429C
      +000000 0D34429C  F010D20B D0801000 58A0C2B8 58F0A01C  05EFD20B D098B108 41A0D098 50A0D08C  |0.K.......B..0....K..q.....q&...|
⋮
```

*Figure 82. Sections of Language Environment Dump for COBOLY*

3. Use the offset of X'338' from the COBOL listing, shown in Figure 83 on page 206, to locate the statement that caused the exception in the COBOLY program. At offset X'338' is an instruction for statement 8. Statement 8 is a call with the identifier SUBNAME specified.

```
⋮
PP 5648-A25 IBM COBOL for OS/390 & VM  2.1.1                 COBOLY    Date 11/08/1999  Time 17:17:35   Page    11
    0002A8                START    EQU  *                     COBOLY
    0002A8  183F                   LR   3,15
    0002AA  4100 10A8              LA   0,168(0,1)
    0002AE  5500 C00C              CL   0,12(0,12)
    0002B2  05F0                   BALR 15,0
    0002B4  47D0 F00C              BC   13,12(0,15)
    0002B8  58F0 C300              L    15,768(0,12)
    0002BC  05EF                   BALR 14,15
    0002BE  181F                   LR   1,15
    0002C0  50D0 1004              ST   13,4(0,1)
    0002C4  5000 104C              ST   0,76(0,1)
    0002C8  D203 1000 3058         MVC  0(4,1),88(3)
    0002CE  18D1                   LR   13,1
    0002D0  58C0 90E8              L    12,232(0,9)           TGTFIXD+232
    0002D4  1812                   LR   1,2
    0002D6  50D0 D058              ST   13,88(0,13)
    0002DA  5090 D05C              ST   9,92(0,13)
    0002DE  58A0 C004              L    10,4(0,12)            CBL=1
    0002E2  5880 9128              L    8,296(0,9)            BLW=0
    0002E6  D203 90EC A010         MVC  236(4,9),16(10)       TGTFIXD+236                    PGMLIT AT +8
    0002EC  BF2F 9208              ICM  2,15,520(9)           IPCB=1+16
    0002F0  58B0 C008              L    11,8(0,12)            PBL=1
    0002F4  4780 B000              BC   8,0(0,11)             GN=7(000306)
    0002F8  5830 905C              L    3,92(0,9)             TGTFIXD+92
    0002FC  58F0 30F4              L    15,244(0,3)           V(IGZCMSG )
    000300  4110 A180              LA   1,384(0,10)           PGMLIT AT +376
    000304  05EF                   BALR 14,15
    000306                GN=7     EQU  *
    000306  5A20 C000              A    2,0(0,12)             SYSLIT AT +0
    00030A  5020 9208              ST   2,520(0,9)            IPCB=1+16
    00030E  9640 91F8              OI   504(9),X'40'          IPCB=1
000008  *
000008  CALL
    000312  D207 D098 8000         MVC  152(8,13),0(8)        TS2=0                          SUBNAME
    000318  DC07 D098 A01A         TR   152(8,13),26(10)      TS2=0                          PGMLIT AT +18
    00031E  D203 D0A0 A15A         MVC  160(4,13),346(10)     TS2=8                          PGMLIT AT +338
    000324  4120 D098              LA   2,152(0,13)           TS2=0
    000328  5020 D0A4              ST   2,164(0,13)           TS2=12
    00032C  4110 D0A0              LA   1,160(0,13)           TS2=8
    000330  5820 905C              L    2,92(0,9)             TGTFIXD+92
    000334  58F0 2100              L    15,256(0,2)           V(IGZCFCC )
    000338  05EF                   BALR 14,15
    00033A  5830 9124              L    3,292(0,9)            BL=1
    00033E  40F0 3000              STH  15,0(0,3)             RETURN-CODE
000009  GOBACK
    000342  47F0 B052              BC   15,82(0,11)           GN=2(000358)
    000346  9120 9054              TM   84(9),X'20'           TGTFIXD+84
    00034A  47E0 B052              BC   14,82(0,11)           GN=2(000358)
    00034E  58F0 20F4              L    15,244(0,2)           V(IGZCMSG )
    000352  4110 A16E              LA   1,366(0,10)           PGMLIT AT +358
    000356  05EF                   BALR 14,15
    000358                GN=2     EQU  *
    000358  5840 9208              L    4,520(0,9)            IPCB=1+16
    00035C  5B40 C000              S    4,0(0,12)             SYSLIT AT +0
    000360  5040 9208              ST   4,520(0,9)            IPCB=1+16
    000364  9140 9055              TM   85(9),X'40'           TGTFIXD+85
    000368  47E0 B070              BC   14,112(0,11)          GN=8(000376)
    00036C  4110 0008              LA   1,8(0,0)
    000370  58F0 2020              L    15,32(0,2)            V(IGZCCTL )
    000374  05EF                   BALR 14,15
    000376                GN=8     EQU  *
    000376  9128 9054              TM   84(9),X'28'           TGTFIXD+84
    00037A  4770 B08A              BC   7,138(0,11)           GN=9(000390)
    00037E  48F0 3000              LH   15,0(0,3)             RETURN-CODE
    000382  58D0 D004              L    13,4(0,13)
    000386  58E0 D00C              L    14,12(0,13)
    00038A  980C D014              LM   0,12,20(13)
    00038E  07FE                   BCR  15,14
```

*Figure 83. COBOL Listing for COBOLY (Part 1 of 2)*

```
    000390                GN=9     EQU  *
    000390  D20B D098 A14E         MVC  152(12,13),334(10)      TS2=0                          PGMLIT AT +326
    000396  4840 3000             LH   4,0(0,3)                RETURN-CODE
    00039A  5040 D0A4             ST   4,164(0,13)             TS2=12
    00039E  4110 D098             LA   1,152(0,13)             TS2=0
    0003A2  58F0 2224             L    15,548(0,2)             V(IGZETRM )
    0003A6  05EF                  BALR 14,15
⋮
```

*Figure 83. COBOL Listing for COBOLY (Part 2 of 2)*

4. Find the value of the local variables in the Parameters, Registers, and Variables for Active Routines section of the dump, shown in Figure 84. Notice that the value of SUBNAME with usage DISP, has a value of 'UNKNOWN'.

   Correct the problem by either changing the subroutine name to one that is defined, or by ensuring that the subroutine is available at compile time.

```
⋮
Parameters, Registers, and Variables for Active Routines:
⋮
COBOLY (DSA address 0002A018):
      Saved Registers:
        GPR0..... 0002A0C0  GPR1..... 0002A0B8  GPR2..... 0001B7FC  GPR3..... 000077E0
        GPR4..... 00007818  GPR5..... 00018AE8  GPR6..... 00000000  GPR7..... 00000000
        GPR8..... 000093B0  GPR9..... 00009150  GPR10.... 000078E8  GPR11.... 00007AE6
        GPR12.... 000078DC  GPR13.... 0002A018  GPR14.... 80007B1A  GPR15.... 8001E128
    GPREG STORAGE:
      Storage around GPR0 (0002A0C0)
        -0020 0002A0A0  0D300220 000179A4 00000000 00000000  E4D5D2D5 D6E6D540 A2080000 0002A0B0  |.......u........UNKNOWN s.......|
        +0000 0002A0C0  00102001 0002A018 00000000 8001E4C2  8D3FF098 0002A2A8 0002A260 0001B7FC  |..............UB..0q..sy..s-....|
        +0020 0002A0E0  000077E0 0002A0B8 0001B7FC 00000000  0002A250 0001BA80 00009150 0004A038  |..................s&......j&....|
⋮
        Local Variables:
              6 77 SUBNAME          X(8) DISP        'UNKNOWN '
```

*Figure 84. 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 85 on page 208 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 quitient
         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 85. 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 86 on page 209. 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.

```
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.        08/30/01 12:13:36 PM                Page:    1

Information for enclave COBOLZ1

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    0002A5E0  CEEHDSP       0D3386F0  +00003032  CEEHDSP       0D3386F0  +00003032              CEEPLPKA           Call
    0002A560  ASSEMZ3       0D315048  +00000064  ASSEMZ3       0D315048  +00000064              ASSEMZ3            Exception
    0002A378  IGZCFCC       0001E128  +00000270  IGZCFCC       0001E128  +00000270              IGZCFCC            Call
    0002A2C0  COBOLZ2       0004C758  +0000026E  COBOLZ2       0004C758  +0000026E          11  COBOLZ2            Call
    0002A0D8  IGZCFCC       0001E128  +00000270  IGZCFCC       0001E128  +00000270              IGZCFCC            Call
    0002A018  COBOLZ1       000077C0  +00000258  COBOLZ1       000077C0  +00000258           8  GO                 Call

  Condition Information for Active Routines
    Condition Information for ASSEMZ3 (DSA address 0002A560)
      CIB Address: 0002AC58
      Current Condition:
        CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

      Location:
        Program Unit: ASSEMZ3 Entry: ASSEMZ3 Statement:  Offset: +00000064
      Machine State:
        ILC..... 0004    Interruption Code..... 0009
        PSW..... 078D0000 8D3150B0
        GPR0..... 0002A5E0  GPR1..... 0002A360  GPR2..... 0D41AA84  GPR3..... 0004E27C
        GPR4..... 00000000  GPR5..... 0000092C  GPR6..... 0004E3B0  GPR7..... 0002A360
        GPR8..... 00000000  GPR9..... 0004E148  GPR10.... 0004A038  GPR11.... 8D315048
        GPR12.... 00019A48  GPR13.... 0002A560  GPR14.... 8001E39A  GPR15.... 8D315048
    Storage dump near condition, beginning at location: 0D31509C
      +000000 0D31509C  10184150 092C1B44 58610000 41660000  5D460000 58F0B0F0 5800B0F0 58DD0004  |...&...../......)....0.0...0....|

  Parameters, Registers, and Variables for Active Routines:
:

    COBOLZ2 (DSA address 0002A2C0):
      Saved Registers:
        GPR0..... 0002A378  GPR1..... 0002A368  GPR2..... 0001B7FC  GPR3..... 0004E27C
        GPR4..... 0002A360  GPR5..... 0001B1BC  GPR6..... 0004A370  GPR7..... 00FCAB00
        GPR8..... 0004E3B0  GPR9..... 0004E148  GPR10.... 0004C860  GPR11.... 0004C96E
        GPR12.... 0004C854  GPR13.... 0002A2C0  GPR14.... 8004C9C8  GPR15.... 8001E128
    GPREG STORAGE:
      Storage around GPR0 (0002A378)
        -0020 0002A358  0002A378 0D41ABA8 8004E3B0 00000000  96080000 0004C870 0004E27C 0002A360  |..t....y..T.....o.....H...S@..t-|
        +0000 0002A378  00102401 0002A2C0 0002A560 8001E39A  8D315048 0002A560 0002A360 0D41AA84  |......s...v-..T...&...v-..t-...d|
        +0020 0002A398  0004E27C 0002A368 0004E27C 0004A3B8  0002A360 00000000 0004E148 0004A038  |..S@..t...S@..t...t-.....|

:
Local Variables:
              6 77 DV-VAL           9999 COMP        00000
              8 77 D-VAL            9999 COMP        00000
:
    COBOLZ1 (DSA address 0002A018):
      Saved Registers:
        GPR0..... 0002A0D8  GPR1..... 0002A0C8  GPR2..... 0001B7FC  GPR3..... 00009280
        GPR4..... 0002A0C0  GPR5..... 00018AE8  GPR6..... 00000000  GPR7..... 00000000
        GPR8..... 000093B0  GPR9..... 00009150  GPR10.... 000078C8  GPR11.... 000079CE
        GPR12.... 000078BC  GPR13.... 0002A018  GPR14.... 80007A1A  GPR15.... 8001E128
    GPREG STORAGE:
      Storage around GPR0 (0002A0D8)
        -0020 0002A0B8  00000000 00000000 800093B0 00000000  96080000 000078DC 00009280 0002A0C0  |.........l.....o.........k.....|
        +0000 0002A0D8  00102401 0002A018 0002A2C0 8001E39A  0004C758 0002A2C0 0002A0C0 0001B7FC  |.........s...T...G...s.........|
        +0020 0002A0F8  00009280 0002A0C8 00009280 0004A370  0002A0C0 00000000 00009150 0004A038  |..k....H..k...t..........j&....|
:
           Local Variables:
              6 77 D-VAL            9999 COMP        00000

:
```

*Figure 86. Sections of Language Environment Dump for Program COBOLZ1*

3. Locate statement 11 in the COBOL listing for the COBOLZ2 program, shown in Figure 87. This is a call to the assembler routine ASSEMZ3.

```
PP 5648-A25 IBM COBOL for OS/390 & VM  2.1.1                  COBOLZ2   Date 11/04/1999  Time 12:13:28   Page     3
  LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  Map and Cross Reference
/* COBOLZ2
  000001               ID DIVISION.
  000002               PROGRAM-ID. COBOLZ2.
  000003               ENVIRONMENT DIVISION.
  000004               DATA DIVISION.
  000005               WORKING-STORAGE SECTION.
  000006               77  DV-VAL PIC 9(4) USAGE COMP.                                    BLW=0000+000         2C
  000007               LINKAGE SECTION.
  000008               77  D-VAL PIC 9(4) USAGE COMP.                                     BLL=0001+000         2C
  000009               PROCEDURE DIVISION USING D-VAL.                                  8
  000010                   MOVE D-VAL TO DV-VAL.                                        8 6
  000011                   CALL "ASSEMZ3" USING DV-VAL.                                 EXT 6
  000012                   GOBACK.
*/ COBOLZ2
⋮
```

*Figure 87. COBOL Listing for COBOLZ2*

4. Check offset +64 in the listing for the assembler routine ASSEMZ3, shown in Figure 88.

   This shows an instruction to divide the contents of register 4 by the variable pointed to by register 6. You can see the two instructions preceding the divide instruction load register 6 from the first word pointed to by register 1 and prepare register 6 for the divide. Because of linkage conventions, you can infer that register 1 contains a pointer to a parameter list that passed to ASSEMZ3. Register 6 points to a 0 value because that was the value passed to ASSEMZ3 when it was called by a higher level routine.

   **Note:** To translate assembler instructions, see *z/Architecture™ Principles of Operation, SA22-7832.*

```
                                    IBMC    HLASM Option Summary   IBMC                 (PTF R3PLUS9)   Page    1
                                                                                     HLASM R3.0  1999/11/04 12.13
                                        IBMC External Symbols IBMC                                      Page    2
Symbol   Type   Id     Address  Length     LD ID  Flags Alias-of                    HLASM R3.0  1999/11/04 12.13
ASSEMZ3   SD 00000001 00000000 000000F4            07
CEESTART  ER 00000002
CEEBETBL  ER 00000003

                                                                                                        Page    3
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Stmt IBMC                          HLASM R3.0  1999/11/04 12.13
                                      1          PRINT NOGEN
000000 47F0 F014            00014     2 ASSEMZ3  CEEENTRY MAIN=NO,PPA=MAINPPA
000056 4150 092C            0092C    37          LA    5,2348             Low order part of quotient
00005A 1B44                          38          SR    4,4                Hi order part of quitient
00005C 5861 0000            00000    39          L     6,0(1)             Get pointer to divisor
000060 4166 0000            00000    40          LA    6,0(6)             Clear hi bit
000064 5D46 0000            00000    41          D     4,0(6)             Do division
000068 58F0 B0F0            000F0    42          CEETERM  RC=0            Terminate with return code zero
                                     49 *
000080 10                            50 MAINPPA  CEEPPA                   Constants describing the code block
                                    116+*,Time Stamp = 1999/11/04 12:13:00                                01-CEEPP
                                    117+*,Version 1 Release 1 Modification 0                              01-CEEPP
                                    128          CEEDSA                  Mapping of the Dynamic Save Area
                                    173          CEECAA                  Mapping of the Common Anchor Area
000000                              376          END   ASSEMZ3
0000F0 00000000                     377                =A(0)
```

*Figure 88. Listing for ASSEMZ3*

5. Check local variables for COBOLZ2 in the Local Variables section of the dump shown in Figure 89 on page 211. 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 89. 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 90, indicating that the value did pass from COBOLZ1 to COBOLZ2.

```
PP 5648-A25 IBM COBOL for OS/390 & VM  2.1.1                 COBOLZ2   Date 11/04/1999  Time 12:13:28   Page     3
  LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  Map and Cross Reference
/* COBOLZ2
  000001               ID DIVISION.
  000002               PROGRAM-ID. COBOLZ2.
  000003               ENVIRONMENT DIVISION.
  000004               DATA DIVISION.
  000005               WORKING-STORAGE SECTION.
  000006               77  DV-VAL PIC 9(4) USAGE COMP.                                    BLW=0000+000         2C
  000007               LINKAGE SECTION.
  000008               77  D-VAL PIC 9(4) USAGE COMP.                                     BLL=0001+000         2C
  000009               PROCEDURE DIVISION USING D-VAL.                                 8
  000010                   MOVE D-VAL TO DV-VAL.                                       8 6
  000011                   CALL "ASSEMZ3" USING DV-VAL.                                EXT 6
  000012                   GOBACK.
*/ COBOLZ2
```

*Figure 90. Listing for COBOLZ2*

7. In the Local Variables section of the dump for program COBOLZ1, shown in Figure 91, 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 91. Variables Section of Language Environment Dump for COBOLZ1*

# Chapter 6. Debugging FORTRAN routines

This chapter provides information to help you debug applications that contain one or more FORTRAN routines. It includes the following topics:

- Determining the source of errors in FORTRAN routines
- Using FORTRAN compiler listings
- Generating a Language Environment dump of a FORTRAN routine
- Finding FORTRAN information in a dump
- Examples of debugging FORTRAN routines

## Determining the source of errors in FORTRAN routines

Most errors in FORTRAN routines can be identified by the information provided in FORTRAN run-time messages, which begin with the prefix FOR.

The FORTRAN compiler cannot identify all possible errors. The following list identifies several errors not detected by the compiler that could potentially result in problems:

- Failing to assign values to variables and arrays before using them in your program.
- Specifying subscript values that are not within the bounds of an array. If you assign data outside the array bounds, you can inadvertently destroy data and instructions.
- Moving data into an item that is too small for it, resulting in truncation.
- Making invalid data references to EQUIVALENCE items of differing types (for example, integer or real).
- Transferring control into the range of a DO loop from outside the range of the loop. The compiler issues a warning message for all such branches if you specify OPT(2), OPT(3), or VECTOR.
- Using arithmetic variables and constants that are too small to give the precision you need in the result. For example, to obtain more than 6 decimal digits in floating-point results, you must use double precision.
- Concatenating character strings in such a way that overlap can occur.
- Trying to access services that are not available in the operating system or hardware.
- Failing to resolve name conflicts between FORTRAN and C library routines using the procedures described in *z/OS Language Environment Programming Guide*.

## Identifying run-time errors

FORTRAN has several features that help you find run-time errors. FORTRAN run-time messages are discussed in *z/OS Language Environment Run-Time Messages*. Other debugging aids include the optional traceback map, program interruption messages, abnormal termination dumps, and operator messages.

- The optional traceback map helps you identify where errors occurred while running your application. The 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.

© Copyright IBM Corp. 1991, 2004

- 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 215.
- 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:

  | | |
  |---|---|
  | *n* | String of 1–5 decimal digits you specified in the PAUSE or STOP statement. For the STOP statement, this number is placed in R15. |
  | *'message'* | Character constant you specified in the PAUSE or STOP statement. |
  | **0** | Printed when a PAUSE statement containing no characters is executed (not printed for a STOP statement). |

  A PAUSE message causes the program to stop running pending an operator response. The format of the operator's response to the message depends on the system being used.
- Under Language Environment, error messages produced by Language Environment and FORTRAN are written to a common message file. Its ddname is specified in the MSGFILE run-time option. The default ddname is SYSOUT.

  FORTRAN information directed to the message file includes:
  - Error messages resulting from unhandled conditions
  - Printed output from any of the dump services (SDUMP, DUMP/PDUMP, CDUMP/CPDUMP)
  - Output produced by a WRITE statement with a unit identifier having the same value as the FORTRAN error message unit
  - Output produced by a WRITE statement with * given as the unit identifier (assuming the FORTRAN error message unit and standard print unit are the same unit)
  - Output produced by the PRINT statement (assuming the FORTRAN error message unit and the standard print unit are the same unit)

  For more information about handling message output using the Language Environment MSGFILE run-time option, see *z/OS Language Environment Programming Guide*.

## Using FORTRAN compiler listings

FORTRAN listings provide you with:
- The date of compilation including information about the compiler
- A listing of your source program
- Diagnostic messages telling you of errors in the source program
- Informative messages telling you the status of the compilation

The following table contains a list of the contents of the various compiler-generated listings that you might find helpful when you use information in dumps to debug FORTRAN programs.

*Table 20. Compiler-generated FORTRAN Listings and Their Contents*

| Name | Contents | Compiler Option |
|---|---|---|
| Diagnostic message listing | Error messages detected during compilation. | FLAG |
| Source program | Source program statements. | SOURCE |
| Source program | Source program statements and error messages. | SRCFLG |
| Storage map and cross reference | Variable use, statement function, subprogram, or intrinsic function within a program. | MAP and XREF |
| Cross reference | Cross reference of names with attributes. | XREF |
| Source program map | Offsets of automatic and static internal variables (from their defining base). | MAP |
| Object code | Contents of the program control section in hexadecimal notation and translated into a pseudo-assembler format. To limit the size of the object code listing, specify the statement or range of statements to be listed; for example, LIST(20) or LIST(10,30). | LIST |
| Variable map, object code, static storage | Same as MAP and LIST options above, plus contents of static internal and static external control sections in hexadecimal notation with comments. | MAP and LIST |
| Symbolic dump | Internal statement numbers, sequence numbers, and symbol (variable) information. | SDUMP |

## Generating a Language Environment dump of a FORTRAN routine

To generate a dump containing FORTRAN information, call either DUMP/PDUMP, CDUMP/CPDUMP, or SDUMP.

DUMP/PDUMP and CDUMP/CPDUMP produce output that is unchanged from the output generated under FORTRAN. Under Language Environment, however, the output is directed to the message file.

When SDUMP is invoked, the output is also directed to the Language Environment message file. The dump format differs from other FORTRAN dumps, however, reflecting a common format shared by the various HLLs under Language Environment.

You cannot make a direct call to CEE3DMP from a FORTRAN program. It is possible to call CEE3DMP through an assembler routine called by your FORTRAN program. FORTRAN programs are currently restricted from directly invoking Language Environment callable services.

**DUMP/PDUMP**
Provides a dump of a specified area of storage.

**CDUMP/CPDUMP**
Provides a dump of a specified area of storage in character format.

**SDUMP**
Provides a dump of all variables in a program unit.

## DUMP/PDUMP subroutines

The DUMP/PDUMP subroutine dynamically dumps a specified area of storage to the system output data set. When you use DUMP, the processing stops after the dump; when you use PDUMP, the processing continues after the dump.

**Syntax**

**CALL {DUMP | PDUMP} (**$a_1$, $b_1$,$k_1$, $a_2$,$b_2$, $k_2$,...**)**

*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} (**$a_1$, $b_1$, $a_2$, $b_2$,...**)**

*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 [(***rtn*$_1$**,***rtn*$_2$**,...)]**

***rtn*$_1$**,***rtn*$_2$**,...**

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 92 on page 220 shows a sample program calling SDUMP and Figure 93 on page 221 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 NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                   NOREORDER NOPC
                   OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
 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 NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                   NOREORDER NOPC
                   OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
 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 NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                   NOREORDER NOPC
                   OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
 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 NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                   NOREORDER NOPC
                   OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
 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 92. Example Program That Calls SDUMP*

Figure 93 on page 221 shows the resulting output generated by the example in Figure 92.

```
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 93. 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.

```
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.         08/30/01 10:32:56 AM                Page:    1

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..0....J...N.L.p....0..K.|

  Parameters, Registers, and Variables for Active Routines:
    CEEHDSP (DSA address 0002D018):
      Saved Registers:
        GPR0..... 000003E7  GPR1..... 0002D3B4  GPR2..... 0002DFD7  GPR3..... 0002E027
        GPR4..... 0002DF94  GPR5..... 00000000  GPR6..... 00000004  GPR7..... 00000000
        GPR8..... 0002E017  GPR9..... 0593875E  GPR10.... 0593775F  GPR11.... 85936760
        GPR12.... 00014770  GPR13.... 0002D018  GPR14.... 800250DE  GPR15.... 85949C70

  GPREG STORAGE:
    Storage around GPR0 (000003E7)
      -000020 000003C7  Inaccessible storage.
      +000000 000003E7  Inaccessible storage.
      +000020 00000407  Inaccessible storage.
    Storage around GPR1 (0002D3B4)
      -000020 0002D394  00000006 00000000 0002E017 0593875E  0593775F 85936760 00014770 00000000  |.............lg;.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 94. Sections of the Language Environment Dump*

# Understanding the Language Environment traceback table

Examine the traceback section of the dump, labeled with **[1]** in Figure 94, 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 94 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 94 on page 222, 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 94 on page 222 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 95 illustrates an error caused by calling a nonexistent routine. The options in effect at compile time appear at the top of the listing.

```
OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
                   NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                   NOREORDER NOPC
                   OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
            1        PROGRAM CALLNON
            2        INTEGER*4 ARRAY_END
                C
            3        CALL SUBNAM
            4        STOP
            5        END
```

*Figure 95. Example of Calling a Nonexistent Routine*

Figure 96 on page 224 shows sections of the dump generated by a call to SDUMP.

```
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.        08/30/01 10:33:01 AM                Page:    1

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..|

⋮
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.        08/30/01 10:33:01 AM                Page:    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 96. 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 95 on page 223. 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 97 demonstrates a divide-by-zero error. In this example, the main FORTRAN program passed 0 to subroutine DIVZEROSUB, and the error occurred when DIVZEROSUB attempted to use this data as a divisor.

```
OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
                   NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                   NOREORDER NOPC
                   OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
           1         PROGRAM DIVZERO
           2         INTEGER*4 ANY_NUMBER
           3         INTEGER*4 ANY_ARRAY(3)
           4         PRINT *,'EXAMPLE STARTING'
           5         ANY_NUMBER = 0
           6         DO I = 1,3
     1     7         ANY_ARRAY(I) = I
     1     8         END DO
           9         CALL DIVZEROSUB(ANY_NUMBER, ANY_ARRAY)
          10         PRINT *,'EXAMPLE ENDING'
          11         STOP
          12         END
OPTIONS IN EFFECT: LIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED TRMFLG SRCFLG NODDIM NORENT SDUMP(ISN)
                   NOSXM NOVECTOR IL(DIM) NOTEST SC(*) NODC NOEC NOEMODE NOICA NODIRECTIVE NODBCS NOSAA NOPARALLEL NODYNAMIC NOSYM
                   NOREORDER NOPC
                   OPT(0) LANGLVL(77) NOFIPS FLAG(I) HALT(S) AUTODBL(NONE) PTRSIZE(8) LINECOUNT(60) CHARLEN(500) NAME(MAIN#)
           1         SUBROUTINE DIVZEROSUB(DIVISOR, DIVIDEND)
           2         INTEGER*4 DIVISOR
           3         INTEGER*4 DIVIDEND(3)
           4         PRINT *,'IN SUBROUTINE DIVZEROSUB'
           5         DIVIDEND(1) = DIVIDEND(3) / DIVISOR
           6         PRINT *,'END OF SUBROUTINE DIVZEROSUB'
           7         RETURN
           8         END
```

*Figure 97. FORTRAN Routine with a Divide-by-Zero Error*

Figure 98 on page 226 shows the Language Environment dump for routine DIVZERO.

```
CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.         08/30/01 10:33:01 AM                Page:    1

Information for enclave DIVZERO

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    0002D018  CEEHDSP       05936760  +0000277C  CEEHDSP       05936760  +0000277C              CEEPLPKA           Call
    05900640  DIVZSUB       05900558  +00000258  DIVZSUB       05900558  +00000258       5_ISN  GO                 Exception
    0002F018  AFHLCLNR      0001B150  +00000000  AFHLCLNR      0001B150  +00000000              AFHPRNBG           Call
    059002E8  DIVZERO       05900200  +00000298  DIVZERO       05900200  +00000298       9_ISN  GO                 Call

  Condition Information for Active Routines
    Condition Information for DIVZSUB   (DSA address 05900640)
      CIB Address: 0002D468
      Current Condition:
        CEE3209S The system detected a fixed-point divide exception.
      Location:
        Program Unit: DIVZSUB
        Entry:        DIVZSUB
        Statement:    5_ISN Offset: +00000258
      Machine State:
        ILC..... 0004    Interruption Code..... 0009
        PSW..... 078D2A00 859007B4
        GPR0..... 00000000  GPR1..... 00000003  GPR2..... 059003FC  GPR3..... 05900400
        GPR4..... 007F6930  GPR5..... 05900468  GPR6..... 0000000C  GPR7..... 059003E0
        GPR8..... 85900400  GPR9..... 807FD4F8  GPR10.... 00000000  GPR11.... 007FD238
        GPR12.... 00E21ED2  GPR13.... 05900640  GPR14.... 8590079C  GPR15.... 05900BA0
    Storage dump near condition, beginning at location: 059007A0
      +000000 059007A0  5870D118 5800700C 5870D120 8E000020  5D007000 5870D118 50107004 58F0D128  |..J.......J.....).....J.&....0J.|

  Parameters, Registers, and Variables for Active Routines:
    CEEHDSP (DSA address 0002D018):
      Saved Registers:
        GPR0..... 00000000  GPR1..... 0002D3B4  GPR2..... 0002DFD7  GPR3..... 0002E027
        GPR4..... 0002DF94  GPR5..... 00000000  GPR6..... 00000004  GPR7..... 00000000
        GPR8..... 0002E017  GPR9..... 0593875E  GPR10.... 0593775F  GPR11.... 05936760
        GPR12.... 00014770  GPR13.... 0002D018  GPR14.... 800250DE  GPR15.... 85949C70

  GPREG STORAGE:
    Storage around GPR0 (00000000)
      +000000 00000000  Inaccessible storage.
      +000020 00000020  Inaccessible storage.
      +000040 00000040  Inaccessible storage.
    Storage around GPR1 (0002D3B4)
      -000020 0002D394  00000006 00000000 0002E017 0593875E  0593775F 05936760 00014770 00000000  |............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..|

⋮
      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 98. 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 98. 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 98 on page 226. 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 98 on page 226. The parameter DIVISOR shows a value of 0.
5. Since DIVISOR contains the value passed to DIVZEROSUB, check its value. ANY_NUMBER is the actual argument passed to DIVZEROSUB, and the dump and listing of DIVZERO indicate that ANY_NUMBER had value 0 when passed to DIVZEROSUB, leading to the divide-by-zero exception.

# Chapter 7. Debugging PL/I routines

This chapter contains information that can help you debug applications that contain one or more PL/I routines. Following a discussion about potential errors in PL/I routines, the first part of this chapter discusses how to use compiler-generated listings to obtain information about PL/I routines, and how to use PLIDUMP to generate a Language Environment dump of a PL/I routine. The last part of the chapter provides examples of PL/I routines and explains how to debug them using information contained in the traceback information provided in the dump. The topics covered are listed below.

- Determining the source of errors in PL/I routines
- Using PL/I compiler listings
- Generating a Language Environment dump of a PL/I routine
- Finding PL/I information in a dump
- Debugging example of PL/I routines

## Determining the source of errors in PL/I routines

Most errors in PL/I routines can be identified by the information provided in PL/I run-time messages, which begin with the prefix IBM. For a list of these messages, see *z/OS Language Environment Run-Time Messages*.

A malfunction in running a PL/I routine can be caused by:

- Logic errors in the source routine
- Invalid use of PL/I
- Unforeseen errors
- Invalid input data
- Compiler or run-time routine malfunction
- System malfunction
- Unidentified routine malfunction
- Overlaid storage

## Logic errors in the source routine

Errors of this type are often difficult to detect because they often appear as compiler or library malfunctions.

Some common errors in source routines are:

- Incorrect conversion from arithmetic data
- Incorrect arithmetic and string manipulation operations
- Unmatched data lists and format lists

## Invalid use of PL/I

A misunderstanding of the language or a failure to provide the correct environment for using PL/I can result in an apparent malfunction of a PL/I routine.

Any of the following, for example, might cause a malfunction:

- Using uninitialized variables
- Using controlled variables that have not been allocated
- Reading records into incorrect structures
- Misusing array subscripts
- Misusing pointer variables
- Incorrect conversion
- Incorrect arithmetic operations
- Incorrect string manipulation operations

© Copyright IBM Corp. 1991, 2004

## Unforeseen errors

If an error is detected during run time and no ON-unit is provided in the routine to terminate the run or attempt recovery, the job terminates abnormally. However, the status of a routine at the point where the error occurred can be recorded by using an ERROR ON-unit that contains the statements:

```
ON ERROR
 BEGIN;
  ON ERROR SYSTEM;
  CALL PLIDUMP;         /*generates a dump*/
  PUT DATA;             /*displays variables*/
 END;
```

The statement ON ERROR SYSTEM ensures that further errors do not result in a permanent loop.

## Invalid input data

A routine should contain checks to ensure that any incorrect input data is detected before it can cause the routine to malfunction.

Use the COPY option of the GET statement to check values obtained by stream-oriented input. The values are listed on the file named in the COPY option. If no file name is given, SYSPRINT is assumed.

## Compiler or run-time routine malfunction

If you are certain that the malfunction is caused by a compiler or run-time routine error, you can either open a PMR or submit an APAR for the error. See either *PL/I for MVS & VM Diagnosis Guide* or *VisualAge PL/I for OS/390 Diagnosis Guide* for more information about handling compiler and run-time routine malfunctions. Meanwhile, you can try an alternative way to perform the operation that is causing the trouble. A bypass is often feasible, since the PL/I language frequently provides an alternative method of performing operations.

## System malfunction

System malfunctions include machine malfunctions and operating system errors. System messages identify these malfunctions and errors to the operator.

## Unidentified routine malfunction

In most circumstances, an unidentified routine malfunction does not occur when using the compiler. If your routine terminates abnormally without an accompanying Language Environment run-time diagnostic message, the error causing the termination might also be inhibiting the production of a message. Check for the following:

- Your job control statements might be in error, particularly in defining data sets.
- Your routine might overwrite main storage areas containing executable instructions. This can happen if you have accidentally:
  - Assigned a value to a nonexistent array element. For example:

    ```
    DCL ARRAY(10);

    ⋮
    DO I = 1 TO 100;
    ARRAY(I) = VALUE;
    ```

    To detect this type of error in a compiled module, set the SUBSCRIPTRANGE condition so that each attempt to access an element outside the declared

range of subscript values raises the SUBSCRIPTRANGE condition. If there is no ON-unit for this condition, a diagnostic message is printed and the ERROR condition is raised. This facility, though expensive in run time and storage space, is a valuable routine-testing aid.

– Used an incorrect locator value for a locator (pointer or offset) variable. This type of error can occur if a locator value is obtained by means of record-oriented transmission. Ensure that locator values created in one routine, transmitted to a data set, and subsequently retrieved for use in another routine, are valid for use in the second routine.

– Attempted to free a nonbased variable. This can happen when you free a based variable after its qualifying pointer value has been changed. For example:

```
DCL A STATIC,B BASED (P);
ALLOCATE B;
P = ADDR(A);
FREE B;
```

– Used the SUBSTR pseudovariable to assign a string to a location beyond the end of the target string. For example:

```
DCL X CHAR(3);
I=3
SUBSTR(X,2,I) = 'ABC';
```

To detect this type of error, enable the STRINGRANGE condition during compilation.

## Storage overlay problems

If you suspect an error in your PL/I application is a storage overlay problem, check for the following:

- The use of a subscript outside the declared bounds (check the SUBSCRIPTRANGE condition)
- An attempt to assign a string to a target with an insufficient maximum length (check the STRINGSIZE condition)
- The failure of the arguments to a SUBSTR reference to comply with the rules described for the SUBSTR built-in function (check the STRINGRANGE condition)
- The loss of significant last high-order (left-most) binary or decimal digits during assignment to an intermediate result or variable or during an input/output operation (check the SIZE condition)
- The reading of a variable-length file into a variable
- The misuse of a pointer variable
- The invocation of a Language Environment callable service with fewer arguments than are required

The first four situations are associated with the listed PL/I conditions, all of which are disabled by default. If you suspect one of these problems exists in your routine, use the appropriate condition prefix on the suspected statement or on the BEGIN or PROCEDURE statement of the containing block.

The fifth situation occurs when you read a data record into a variable that is too small. This type of problem only happens with variable-length files. You can often isolate the problem by examining the data in the file information and buffer.

The sixth situation occurs when you misuse a pointer variable. This type of storage overlay is particularly difficult to isolate. There are a number of ways pointer variables can be misused:

- When a READ statement runs with the SET option, a value is placed in a pointer. If you then run a WRITE statement or another READ SET option with another pointer, you overlay your storage if you try to use the original pointer.
- When you try to use a pointer to allocate storage that has already been freed, you can also cause a storage overlay.
- When you attempt to use a pointer set with the ADDR built-in function as a base for data with different attributes, you can cause a storage overlay.

The seventh situation occurs when a Language Environment callable service is passed fewer arguments than its interface requires. The following example might cause a storage overlay because Language Environment assumes that the fourth item in the argument list is the address of a feedback code, when in reality it could be residue data pointing anywhere in storage.

Invalid calls:

```
DCL CEEDATE ENTRY OPTIONS(ASM);
CALL CEEDATE(x,y,z);      /* invalid */
```

Valid calls:

```
DCL CEEDATE ENTRY(*,*,*,* OPTIONAL) OPTIONS(ASM);
CALL CEEDATE(x,y,z,*);    /* valid   */
CALL CEEDATE(x,y,z,fc);   /* valid   */
```

## Using PL/I compiler listings

The following sections explain how to generate listings that contain information about your routine. PL/I listings show machine instructions, constants, and external or internal addresses that the linkage editor resolves. This information can help you find other information, such as variable values, in a dump of a PL/I routine.

**Note:** VisualAge PL/I shares a common compiler back-end with C/C++. The VisualAge PL/I assembler listing will, consequently, have a similar form to those from the C/C++ compiler.

The PL/I compiler listings included below are from the PL/I for MVS & VM product.

## Generating PL/I listings and maps

The following table shows compiler-generated listings that you might find helpful when you use information in dumps to debug PL/I routines. For more information about supported compiler options that generate listings, reference either the *PL/I for MVS & VM Programming Guide* or the *VisualAge PL/I for OS/390 Programming Guide*.

*Table 21. Compiler-generated PL/I Listings and Their Contents*

| Name | Contents | Compiler Option |
|---|---|---|
| Source program | Source program statements | SOURCE |
| Cross reference | Cross reference of names with attributes | XREF and ATTRIBUTES |
| Aggregate table | Names and layouts of structures and arrays | AGGREGATE |
| Variable map | Offsets of automatic and static internal variables (from their defining base) | MAP |

*Table 21. Compiler-generated PL/I Listings and Their Contents (continued)*

| Name | Contents | Compiler Option |
|---|---|---|
| Object code | Contents of the program control section in hexadecimal notation and translated into a pseudo-assembler format. To limit the size of the object code listing, specify a certain statement or range of statements to be listed; for example, LIST(20) or LIST(10,30). | LIST |
| Variable map, object code, static storage | Same as MAP and LIST options above, plus contents of static internal and static external control sections in hexadecimal notation with comments | MAP and LIST |

## Finding information in PL/I listings

Figure 99 shows an example PL/I routine that was compiled with LIST and MAP.

```
*PROCESS SOURCE, LIST, MAP;

                  SOURCE LISTING

   STMT

     1   |EXAMPLE: PROC OPTIONS(MAIN);
     2   |           DCL EXTR ENTRY EXTERNAL;
     3   |           DCL A FIXED BIN(31);
     4   |           DCL B(2,2)  FIXED BIN(31) STATIC EXTERNAL INIT((4)0);
     5   |           DCL C CHAR(20) STATIC INIT('SAMPLE CONSTANT');
     6   |           DCL D FIXED BIN(31) STATIC;
     7   |           DCL E FIXED BIN(31);
     8   |           FETCH EXTR;
     9   |           CALL EXTR(A,B,C,D,E);
    10   |           DISPLAY(C);
    11   |         END;
```

*Figure 99. PL/I Routine Compiled with LIST and MAP*

Figure 100 on page 234 shows the output generated from this routine, including the static storage map, variable storage map, and the object code listing. The sections following this example describe the contents of each type of listing.

```
                        STATIC INTERNAL STORAGE MAP

000000  E00000E8             PROGRAM ADCON
000004  00000008             PROGRAM ADCON
000008  00000096             PROGRAM ADCON
00000C  00000096             PROGRAM ADCON
000010  00000096             PROGRAM ADCON
000014  00000000             A..IBMSJDSA
000018  00000000             A..IBMSPFRA
00001C  00000000             A..STATIC
000020  0000000000000044     LOCATOR..B
000028  0000008800140000     LOCATOR..C
000030  91E091E0             CONSTANT
000034  0A000000C5E7E3D9     FECB..EXTR
        40404040
000040  80000034             A..FECB..EXTR
000044  0000000C00000008     DESCRIPTOR
        0000000200000001
        0000000400000002
        00000001
000060  80000034             A..FECB..EXTR
000064  00000000             A..B
000068  00000000             A..A
00006C  00000020             A..LOCATOR
000070  00000028             A..LOCATOR
000074  000000A0             A..D
000078  80000000             A..E
00007C  00000000             A..ENTRY EXTR
000080  80000028             A..LOCATOR
000084
000088  E2C1D4D7D3C540C3     INITIAL VALUE..C
        D6D5E2E3C1D5E340
        40404040

                        STATIC EXTERNAL CSECTS

000000  0000000000000000     CSECT FOR EXTERNAL VARIABLE
        0000000000000000

⋮
                                VARIABLE STORAGE MAP

IDENTIFIER                      LEVEL         OFFSET       (HEX)   CLASS     BLOCK

E                                   1            184          B8   AUTO      EXAMPLE
D                                   1            160          A0   STATIC    EXAMPLE
C                                   1            136          88   STATIC    EXAMPLE
A                                   1            188          BC   AUTO      EXAMPLE

⋮
  OBJECT LISTING
                                                      000096  58 B0 C 004              L     11,4(0,12)
                                                      00009A  58 FB 0 000              L     15,PR..EXTR
* STATEMENT NUMBER  1                                 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(COMPILATION 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 100. Compiler-Generated Listings from Example PL/I Routine (Part 1 of 2)*

```
000038  00000000                DC    X'00000000'         0000DA  41 10 3 060              LA    1,96(0,3)
00003C  01008000                DC    X'01008000'         0000DE  58 F0 3 018              L     15,A..IBMSPFRA
000040  00000000                DC    A(REGION TABLE)     0000E2  05 EF                    BALR  14,15
000044  00000002                DC    X'00000002'         0000E4  58 FB 0 000              L     15,PR..EXTR
000048  00000000                DC    A(PRIMARY ENTRY)    0000E8                    CL.6    EQU   *
00004C  00000000                DC    X'00000000'         0000E8  1B 55                    SR    5,5
000050  00000000                DC    X'00000000'         0000EA  41 10 D 0C0              LA    1,192(0,13)
000054  58 30 F 010             L     3,16(0,15)          0000EE  05 EF                    BALR  14,15
000058  58 10 D 04C             L     1,76(0,13)
00005C  58 00 F 00C             L     0,12(0,15)
000060  1E 01                   ALR   0,1                 * 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 100. Compiler-Generated Listings from Example PL/I Routine (Part 2 of 2)*

## Static internal storage map

To get a complete variable storage map and static storage map, but not a complete LIST, specify a single statement for LIST to minimize the size of the listing; for example, LIST(1).

Each line of the static storage map contains the following information:

1. Six-digit hexadecimal offset.
2. Hexadecimal text, in 8-byte sections where possible.
3. Comment, indicating the type of item to which the text refers. The comment appears on the first line of the text for an item.

Some typical comments you might find in a static storage listing:

*Table 22. Typical Comments in a PL/I Static Storage Listing*

| Comment | Explanation |
|---|---|
| A..*xxx* | Address constant for *xxx* |
| COMPILER LABEL CL.*n* | Compiler-generated label *n* |
| CONDITION CSECT | Control section for programmer-named condition |
| CONSTANT | Constant |
| CSECT FOR EXTERNAL VARIABLE | Control section for external variable |
| D..*xxx* | Descriptor for *xxx* |
| DED..*xxx* | Data element descriptor for *xxx* |
| DESCRIPTOR | Data descriptor |
| ENVB | Environment control block |
| FECB..*xxx* | Fetch control block for *xxx* |
| DCLCB | Declare control block |

*Table 22. Typical Comments in a PL/I Static Storage Listing (continued)*

| Comment | Explanation |
|---|---|
| FED..*xxx* | Format element descriptor for *xxx* |
| KD..*xxx* | Key descriptor for *xxx* |
| LOCATOR..*xxx* | Locator for *xxx* |
| ONCB | ON statement control block |
| PICTURED DED..*xxx* | Pictured data element descriptor for *xxx* |
| PROGRAM ADCON | Program address constant |
| RD..*xxx* | Record descriptor for *xxx* |
| SYMBOL TABLE ELEMENT | Symbol table address |
| SYMBOL TABLE..*xxx* | Symbol table for *xxx* |
| SYMTAB DED..*xxx* | Symbol table DED for *xxx* |
| USER LABEL..*xxx* | Source program label for *xxx* |
| *xxx* | Variable with name *xxx*. If the variable is not initialized, no text appears against the comment. There is also no static offset if the variable is an array (the static offset can be calculated from the array descriptor, if required). |

## Variable storage map

For automatic and static internal variables, the variable storage map contains the following information:

- PL/I identifier name
- Level
- Storage class
- Name of the PL/I block in which it is declared
- Offset from the start of the storage area, in both decimal and hexadecimal form

If the LIST option is also specified, a map of the static internal and external control sections, called the static storage map, is also produced.

## Object code listing

The object code listing consists of the machine instructions and a translation of these instructions into a form that resembles assembler and includes comments, such as source program statement numbers.

The machine instructions are formatted into blocks of code, headed by the statement or line number in the PL/I source program listing. Generally, only executable statements appear in the listing. DECLARE statements are not normally included. The names of PL/I variables, rather than the addresses that appear in the machine code, are listed. Special mnemonics are used to refer to some items, including test hooks, descriptors, and address constants.

Statements in the object code listing are ordered by block, as they are sequentially encountered in the source program. Statements in the external procedure are given first, followed by the statements in each inner block. As a result, the order of statements frequently differs from that of the source program.

Every object code listing begins with the name of the external procedure. The actual entry point of the external procedure immediately follows the heading comment `REAL ENTRY`. The subsequent machine code is the prolog for the block,

which performs block activation. The comment PROCEDURE BASE marks the end of the prolog. Following this is a translation of the first executable statement in the PL/I source program.

Following are the comments used in the listing:

*Table 23. Comments in a PL/I Object Code Listing*

| Comment | Function |
|---|---|
| BEGIN BLOCK *xxx* | Indicates the start of the begin block with label *xxx* |
| BEGIN BLOCK NUMBER *n* | Indicates the start of the begin block with number *n* |
| 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 *n* | 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 *xxx* | Indicates the start of compiler-generated subroutine *xxx* |
| 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 *xxx* | Identifies the end of an ILC procedure *xxx* |
| 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 *xxx* | Indicates the start of initialization code for variable *xxx* |
| 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 *xxx* | Identifies the start of an implicitly generated ILC procedure *xxx* |
| 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 *n* | Indicates the start of an ON-unit block with number *n* |
| ON-UNIT BLOCK END | Indicates the end of the ON-unit block |
| PROCEDURE *xxx* | Identifies the start of the procedure labeled *xxx* |
| PROCEDURE BASE | Identifies the address loaded into the base register for the procedure |
| PROGRAM ADDRESSABILITY REGION BASE | Identifies the address where the routine base is updated if the routine size exceeds 4096 bytes and consequently cannot be addressed from one base |
| PROLOGUE BASE | Identifies the start of the prolog code common to all entry points into that procedure |
| REAL ENTRY | Precedes the actual executable entry point for a procedure |

*Table 23. Comments in a PL/I Object Code Listing (continued)*

| Comment | Function |
|---|---|
| STATEMENT LABEL *xxx* | Identifies the position of source program statement label *xxx* |
| STATEMENT NUMBER *n* | Identifies the start of code generated for statement number *n* in the source listing |

In certain cases the compiler uses mnemonics to identify the type of operand in an instruction and, where applicable, follows the mnemonic by the name of a PL/I variable.

*Table 24. PL/I Mnemonics*

| Mnemonic | Explanation |
|---|---|
| A..*xxx* | Address constant for *xxx* |
| ADD..*xxx* | Aggregate descriptor for *xxx* |
| BASE..*xxx* | Base address of variable *xxx* |
| BLOCK.*n* | Identifier created for an otherwise unlabeled block |
| CL.*n* | Compiler-generated label number *n* |
| D..*xxx* | Descriptor for *xxx* |
| DED..*xxx* | Data element descriptor for *xxx* |
| 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.*n* | Workspace, followed by identifying number *n* |
| L..*xxx* | Length of variable *xxx* |
| PR..*xxx* | Pseudoregister vector slot for *xxx* |
| LOCATOR..*xxx* | Locator for *xxx* |
| RKD..*xxx* | Record or key descriptor for *xxx* |
| VO..*xxx* | Virtual origin for *xxx* (the address where element 0 is held for a one-dimensional array, element 0,0 for a two-dimensional array, and so on) |

## Generating a Language Environment dump of a PL/I routine

To generate a dump of a PL/I routine, you can call either the Language Environment callable service CEE3DMP or PLIDUMP. For information about calling CEE3DMP, see “Generating a Language Environment dump with CEE3DMP” on page 37.

## PLIDUMP syntax and options

PLIDUMP calls intermediate PL/I library routines, which convert most PLIDUMP options to CEE3DMP options. The following list contains PLIDUMP options and the corresponding CEE3DMP option, if applicable.

Some PLIDUMP options do not have corresponding CEE3DMP options, but continue to function as PL/I default options. The list following the syntax diagram provides a description of those options.

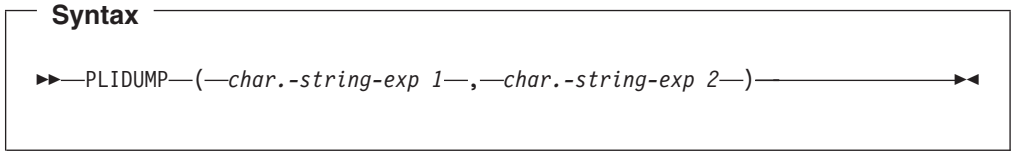
**Note:** VisualAge PL/I does not support multitasking, therefore, the PLIDUMP options that refer to multitasking do not apply to VisualAge PL/I.

PLIDUMP now conforms to National Language Support standards.

PLIDUMP can supply information across multiple Language Environment enclaves. If an application running in one enclave fetches a main procedure (an action that creates another enclave), PLIDUMP contains information about both procedures.

The syntax and options for PLIDUMP are shown below.

**Syntax**




**char.-string-exp 1**

A dump options character string consisting of one or more of the following:

**A** All. Results in a dump of all tasks including the ones in the WAIT state.

**B** BLOCKS (PL/I hexadecimal dump). Dumps the control blocks used in Language Environment and member language libraries. For PL/I, this includes the DSA for every routine on the call chain and PL/I ″global″ control blocks, such as Tasking Implementation Appendage (TIA), Task Communication Area (TCA), and the PL/I Tasking Control Block (PTCB). PL/I file control blocks and file buffers are also dumped if the F option is specified.

**C** Continue. The routine continues after the dump.

**E** Exit. The enclave terminates after the dump. In a multitasking environment, if PLIDUMP is called from the main task, the enclave terminates after the dump. If PLIDUMP is called from a subtask, the subtask and any subsequent tasks created from the subtask terminate after the dump. In a multithreaded environment, if PLIDUMP is called from the Initial Process Thread (IPT), the enclave terminates after the dump. If PLIDUMP is called from a non-IPT, only the non-IPT terminates after the dump.

**F** FILE INFORMATION. A set of attributes for all open files is given. The contents of the file buffers are displayed if the B option is specified.

**H** STORAGE in hexadecimal. A SNAP dump of the region is produced. A ddname of CEESNAP must be provided to direct the CEESNAP dump report.

**K** BLOCKS (when running under CICS). The Transaction Work Area is included.

**Note:** This option is not supported under VisualAge PL/I.

**NB** NOBLOCKS.

**NF** NOFILES.

**NH** NOSTORAGE.

**NK** NOBLOCKS (when running under CICS).

**NT** NOTRACEBACK.

**O** THREAD(CURRENT). Results in a dump of only the current task or current thread (the invoker of PLIDUMP).

**S** Stop. The enclave terminates after the dump. In a multitasking environment, regardless of whether PLIDUMP is called from the main task or a subtask, the enclave terminates after the dump. In a multithreaded environment, regardless of whether PLIDUMP is called from the IPT or a non-IPT, the enclave terminates after the dump (in which case there is no fixed order as to which thread terminates first).

**T** TRACEBACK. Includes a traceback of all routines on the call chain. The traceback shows transfers of control from either calls or exceptions. BEGIN blocks and ON-units are also control transfers and are included in the trace. The traceback extends backwards to the main program of the current thread.

T, F, C, and A are the default options.

**char.-string-exp 2**
A user-identified character string up to 80 characters long that is printed as the dump header.

## PLIDUMP usage notes

If you use PLIDUMP, the following considerations apply:

- If a routine calls PLIDUMP a number of times, use a unique user-identifier for each PLIDUMP invocation. This simplifies identifying the beginning of each dump.
- In MVS or TSO, you can use ddnames of CEEDUMP, PLIDUMP, or PL1DUMP to direct dump output. If no ddname is specified, CEEDUMP is used.
- The data set defined by the PLIDUMP, PL1DUMP, or CEEDUMP DD statement should specify a logical record length (LRECL) of at least 131 to prevent dump records from wrapping.
- When you specify the H option in a call to PLIDUMP, the PL/I library issues an OS SNAP macro to obtain a dump of virtual storage. The first invocation of PLIDUMP results in a SNAP identifier of 0. For each successive invocation, the ID is increased by one to a maximum of 256, after which the ID is reset to 0.
- Support for SNAP dumps using PLIDUMP is provided only under MVS. SNAP dumps are not produced in a CICS environment.

– If the SNAP does not succeed, the CEE3DMP DUMP file displays the message:

  ```
  Snap was unsuccessful
  ```

  Failure to define a CEESNAP data set is the most likely cause of an unsuccessful CEESNAP.

– If the SNAP is successful, CEE3DMP displays the message:

  ```
  Snap was successful; snap ID = nnn
  ```

  where *nnn* corresponds to the SNAP identifier described above. An unsuccessful SNAP does not result in an incrementation of the identifier.

- To ensure portability across system platforms, use PLIDUMP to generate a dump of your PL/I routine.

## Finding PL/I information in a dump

The following sections discuss PL/I-specific information located in the following sections of a Language Environment dump:

- Traceback
- Control Blocks for Active Routines
- Control Block Associated with the Thread
- File Status and Attributes

## Traceback

Examine the traceback section of the dump, shown in Figure 101 on page 242, for condition information about your routine and information about the statement number and address where the exception occurred.

```
CEE3DMP V1 r5.0: PLIDUMP called from error ON-unit.                              08/05/95 4:04:12 PM         Page:   1

PLIDUMP was called from statement number 6 at offset +000000D6 from ERROR  ON-UNIT with entry address 00020168

Information for enclave EXAMPLE

  Information for thread 8000000000000000

  Registers on Entry to CEE3DMP:

    PM....... 0100
    GPR0..... 00000000  GPR1..... 00077448  GPR2..... 053AD9AF  GPR3..... 853AD514
    GPR4..... 00000001  GPR5..... 053AD314  GPR6..... 80077454  GPR7..... 00000000
    GPR8..... 00000001  GPR9..... 80000000  GPR10.... 00077470  GPR11.... 000F7490
    GPR12.... 0006A520  GPR13.... 000773C8  GPR14.... 80060712  GPR15.... 853F7918
    FPR0..... 4D000000  00043C31            FPR2..... 00000000  00000000
    FPR4..... 00000000  00000000            FPR6..... 00000000  00000000

  Traceback:
     DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset  Load     Service    Statement  Status
     005359A0  CEEKKMRA      00654438  +00000748  CEEKKMRA      00654438  +00000748  CELE38                         Call
     006D9810  LIBRARY(PLI)  005CBE98  +000000B2  LIBRARY(PLI)  005CBE98  +000000B2  CEEPLPKA                       Call
     005358A0  EXAMPLE       00020080  +000001BE  ERR  ON-UNIT  00020168  +000000D6  CELE38                       6  Call
     00535698  IBMRERPL      007CB410  +00000528  IBMRERPL      007CB410  +00000528                                 Call
     005355B0  CEEEV010      005B5000  +000000E8  CEEEV010      005B5000  +000000E8                                 Call
     006C3018  CEEHDSP       005F6D00  +00000970  CEEHDSP       005F6D00  +00000970                                 Call
     00535428  IBMRERRI      007CB040  +00000254  IBMRERRI      007CB040  +00000254                               Exception
     00535358  EXAMPLE       00020080  +00000296  LABL1: BEGIN  00020258  +000000BE                              11  Call
     00535258  EXAMPLE       00020080  +000000D0  EXAMPLE       00020088  +000000C8                               8  Call
     005351B0  IBMRPMIA      007CABD0  +000002FA  IBMRPMIA      007CABD0  +000002FA                                 Call
     005350C8  CEEEV010      005B5000  +000001FE  CEEEV010      005B5000  +000001FE                                 Call
     00535018  CEEBBEXT      005E55F0  +0000012E  CEEBBEXT      005E55F0  +0000012E                                 Call

    Condition Information for Active Routines
     Condition Information for IBMRERRI (DSA address 00535428)
       CIB Address: 006C33C8
       Current Condition:
         IBM0930S
       Original Condition:
         IBM0421S ONCODE=520  The SUBSCRIPTRANGE condition was raised.
       Location:
         Program Unit: IBMRERRI Entry: IBMRERRI Statement:  Offset: +00000254

⋮
```

*Figure 101. Traceback Section of Dump*

## PL/I task traceback

A task traceback table is produced for multitasking programs showing the task invocation sequence (trace). For each task, the CAA address, task variable address, event variable address, thread ID, and absolute priority appear in the traceback table. An example is shown in Figure 102 on page 243.

```
CEE3DMP V1 R5.0: called from SUBTSK2                                    08/18/95 2:36:49 PM
Page:    1

PLIDUMP was called from statement number 23 at offset +000000D2 from BEGIN BLOCK6 within task SUBTSK2

PL/I Task Traceback:
  Task     Attached by  Thread ID         TCA Addr   EV Addr    TV Addr   Absolute Priority
  SUBTSK2  SUBTSK1      03B2CB7800000003  00070708   000684F8   000684E8  000
  SUBTSK1  SUBTASK      03B2C2D000000002  00066708   00034498   00034488  000
  SUBTASK  TASKING      03B2BA2800000001  0005D708   00034468   00034458  056
  TASKING               03B2B18000000000  00016658   000545D4   0005423C  254

Information for enclave TASKING

  Information for thread 03B2CB7800000003

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset  Load     Service   Statement  Status
    000726A8  CEEKKMRA      0394AAC0  +00000860  CEEKKMRA      0394AAC0  +00000860  CELE38                         Call
    0006E2E8  IBMRKDM       039D2450  +000000BA  IBMRKDM       039D2450  +000000BA  CEEPLPKA                       Call
    000725B8  SUBTSK1       00007760  +0000052A  BEGIN BLOCK6  00007BB8  +000000D2  CELE38                    23   Call
    000724F8  SUBTSK1       00007760  +0000043E  PROCA          00007B1C  +00000082                         21   Call
    00072430  SUBTSK1       00007760  +0000036E  SUBTSK2       00007A24  +000000AA                         19   Call
    000715D8  IBMUPTMM      00023920  +000000F6  IBMUPTMM      00023920  +000000F6                              Call
    7F6653A0                00006E38  +00000000                00006E38  +00000000                              Call
```

*Figure 102. Task Traceback Section*

### Condition information

If the dump was called from an ON-unit, the type of ON-unit is identified in the traceback as part of the entry information. For ON-units, the values of any relevant condition built-in functions (for example, ONCHAR and ONSOURCE for conversion errors) appear. In cases where the cause of entry into the ON-unit is not stated, usually when the ERROR ON-unit is called, the cause of entry appears in the condition information.

### Statement number and address where error occurred

This information, which is the point at which the condition that caused entry to the ON-unit occurred, can be found in the traceback section of the dump.

If the condition occurs in compiled code, and you compiled your routine with either GOSTMT or GONUMBER, the statement numbers appear in the dump. To identify the assembler instruction that caused the error, use the traceback information in the dump to find the program unit (PU) offset of the statement number in which the error occurred. Then find that offset and the corresponding instruction in the object code listing.

## Control blocks for active routines

This section shows the stack frames for all active routines, and the static storage. Use this section of the dump to identify variable values, determine the contents of parameter lists, and locate the timestamp.

Figure 103 on page 244 shows this section of the dump.

```
Control Blocks for Active Routines:

⋮
     DSA for PLIDMPB: 003CA438
       +000000  FLAGS.... 8025      member... 0000      BKC...... 003CA348  FWC...... 00000000  R14...... 4E020360
       +000010  R15...... 805611C0  R0....... 003CA588  R1....... 000204AC  R2....... 5E0202BE  R3....... 000203B8
       +000024  R4....... 003CA57C  R5....... 00000000  R6....... 00000009  R7....... 00000001  R8....... 003CA554
       +000038  R9....... 003CA4F8  R10...... 00000004  R11...... 00000008  R12...... 003CA4EC  reserved. 00542280
       +00004C  NAB...... 003CA588  PNAB..... 003CA588  reserved. 91E091E0  003CA348  007C8090  00627188
       +000064  reserved. 003CA4E8  reserved. 005C8E10  MODE..... 0058C848  reserved. 003CA608  00620258
       +000078  reserved. 003CA4E8  reserved. 003CA4EC
     DYNAMIC SAVE AREA (PLIDMPB): 003CA438
       +000000 003CA438  80250000 003CA348 00000000 4E020360  805611C0 003CA588 000204AC 5E0202BE  |.....t.....+..-......vh....;...|
       +000020 003CA458  000203B8 003CA57C 00000000 00000009  00000001 003CA554 003CA4F8 00000004  |......v@.............v...u8....|
       +000040 003CA478  00000008 003CA4EC 00542280 003CA588  003CA588 91E091E0 003CA348 007C8090  |......u.......vh..vhj.j...t..@..|
       +000060 003CA498  00627188 003CA4E8 005C8E10 0058C848  003CA608 00620258 003CA4E8 003CA4EC  |...h..uY.*....H...w.......uY..u.|
       +000080 003CA4B8  00627250 003CA608 00627258 000204D7  00100000 00627250 003CA4E8 00627250  |...&.w........P.......&.uY...&..|
       +0000A0 003CA4D8  003CA4EC 003CA530 003CA608 00000000  00000248 00000003 003CA4F8 00020474  |..u...v...w.............u8....|
       +0000C0 003CA4F8  0004E385 9989B870 00000000 00000008  C7899393 89A297A8 8E572778 40404040  |..Teri..........Gillispy....    |
       +0000E0 003CA518  003CA520 40404040 00020438 00020418  40404040 8002046C 00404000 007C8004  |..v.    ........    ...%.  ..@..|
       +000100 003CA538  40404040 40404040 00014040 00542460  E3C2C6C3 003CA548 00040000 D7D3C9C4  |        ..  ...-TBFC..v.....PLID|
       +000120 003CA558  E4D4D740 83819393 85844086 99969440  97999683 8584A499 8540D7D3 C9C4D4D7  |UMP called from procedure PLIDMP|
       +000140 003CA578  C2404040 003CA554 00250000 40404040  88004040 00542280 003CAA58 6E579B82  |B   ..v.....    h.  ........>..b|
     STATIC FOR PROCEDURE PLIDMP    TIMESTAMP: 2 DEC 92   11:26:26
     STARTING FROM: 000203B8
       +000000 000203B8  E0000300 00020088 00020116 00020188  000201E2 00020254 000202AE 000202BE  |.......h.......h...S............|
       +000020 000203D8  000202BE 000202BE 000202BE 80020A38  80020A50 80020A68 80020A80 80020A98  |...................&..........q|
       +000040 000203F8  80020AB0 80021340 80021148 80020AC8  800213B8 800213D0 80020AE0 80020AF8  |....... .......H..............8|
       +000060 00020418  20000002 1F802800 00040008 00000000  000204C8 000F0000 000204D7 00100000  |..................H.......P....|
       +000080 00020438  000204F3 00100000 00000000 00040000  00000000 00250000 00020608 00110000  |...3...........................|
       +0000A0 00020458  00000000 00020474 91E091E0 00000005  00000009 00000001 00000003 00000000  |........j.j.....................|
       +0000C0 00020478  000C8000 0000000E 000C8000 000206D0  000206D0 003CA320 8002046C 000206D0  |.....................t....%....|
       +0000E0 00020498  003CA410 8002046C 000206D0 003CA520  8002046C 003CA54C 803CA57C 80020A08  |..u....%......v....%..v<..v@....|

⋮
```

*Figure 103. Control Blocks for Active Routines Section of the Dump*

## Automatic variables

To find automatic variables, use an offset from the stack frame of the block in which they are declared. This information appears in the variable storage map generated when the MAP compiler option is in effect. If you have not used the MAP option, you can determine the offset by studying the listing of compiled code instructions.

## Static Variables

If your routine is compiled with the MAP option, you can find static variables by using an offset in the variable storage map. If the MAP option is not in effect, you can determine the offset by studying the listing of compiled code.

## Based variables

To locate based variables, use the value of the defining pointer. Find this value by using one of the methods described above to find static and automatic variables. If the pointer is itself based, you must find its defining pointer and follow the chain until you find the correct value.

The following is an example of typical code for X BASED (P), with P AUTOMATIC:

```
58 60 D 0C8        L 6,P

58 E0 6 000        L 14,X
```

P is held at offset X'C8' from register 13. This address points to X.

Take care when examining a based variable to ensure that the pointers are still valid.

### Area variables

Area variables are located using one of the methods described above, according to their storage class.

The following is an example of typical code: for an area variable A declared AUTOMATIC:

```
41 60 D 0F8        LA 6,A
```

The area starts at offset X'F8' from register 13.

### Variables in areas

To find variables in areas, locate the area and use the offset to find the variable.

### Contents of parameter lists

To find the contents of a passed parameter list, first find the register 1 value in the save area of the calling routine's stack frame. Use this value to locate the parameter list in the dump. If R1=0, no parameters passed. For additional information about parameter lists, see either *PL/I for MVS & VM Programming Guide* or *VisualAge PL/I for OS/390 Programming Guide*.

### Timestamp

If the TSTAMP compiler installation option is in effect, the date and time of compilation appear within the last 32 bytes of the static internal control section. The last three bytes of the first *word* give the offset to this information. The offset indicates the end of the timestamp. Register 3 addresses the static internal control section. If the BLOCK option is in effect, the timestamp appears in the static storage section of the dump.

## Control blocks associated with the thread

This section of the dump, shown in Figure 104 on page 246, includes information about PL/I fields of the CAA and other control block information.

```
Control Blocks Associated with the Thread:
  CAA: 0058C848
    +000000 0058C848  00000800 00542648 003CA000 0044A000  00000000 0058C858 00000000 005421A0  |.....................H.........|
    +000020 0058C868  00000000 00000000 00542030 00000000  00542230 007C8004 005421F8 00000000  |....................@.....8....|
    +000040 0058C888  005421C0 00000000 006CB660 00000000  006CBBB8 006C66E0 00000000 00000000  |.........%.-.....%...%..........|
    +000060 0058C8A8  00000000 006C6660 00000000 006CB200  006CB4A0 006CB620 006C7028 04001010  |.....%.-.....%...%...%...%......|

⋮
  DUMMY DSA: 0058E040
    +000000  FLAGS.... 0000      member... 0000      BKC...... 000095E8  FWC...... 003CA018  R14...... 40020810
    +000010  R15...... 805905F0  R0....... 00000E08  R1....... 0058AE14  R2....... 000206F0  R3....... 00000002
    +000024  R4....... 00000000  R5....... 00000000  R6....... 00000000  R7....... 005801E0  R8....... 000206C0
    +000038  R9....... 00D44A30  R10...... 00000000  R11...... 4002073A  R12...... 0058C848  reserved. 00542280
    +00004C  NAB...... 003CA018  PNAB..... 003CA018  reserved. 00000000  00000000  00000000  00000000
    +000064  reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 00000000  00000000
    +000078  reserved. 00000000  reserved. 00000000

CEE3DMP V1 R5.0: PLIDUMP called from procedure PLIDMPB.                         08/05/95 11:29:13 AM         Page:  5

     PL/I TCA APPENDAGE: 00542030
       +000000 00542030  00000000 00000000 00000000 00000030  00000000 00000000 00000000 00000000  |................................|
       +000020 00542050  005421D8 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |...Q............................|
       +000040 00542070  00000000 00000000 00000000 00000000  0058C848 00000000 00000000 00000000  |.................H..............|
       +000060 00542090  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|

   Enclave Control Blocks:
     EDB: 0058AD68
       +000000 0058AD68  C3C5C5C5 C4C24040 80400001 0058C6B8  0058B3A8 00000000 00000000 00000000  |CEEEDB  . ....F....y............|
       +000020 0058AD88  0058B0D8 0058B108 005801E0 0057F198  00000000 8057F088 0058AE14 00008000  |...Q..........1q......0h........|
     MEML: 0058C6B8
       +000000 0058C6B8  00000000 00000000 00592030 00000000  00000000 00000000 00592030 00000000  |................................|
       +000020 0058C6D8 - +00009F 0058C757             same as above
       +0000A0 0058C758  00000000 00000000 0054A000 00000000  00000000 00000000 00592030 00000000  |................................|

   File Status and Attributes:
     ATTRIBUTES OF FILE: SYSPRINT
      STREAM OUTPUT PRINT ENVIRONMENT( F BLKSIZE(80) RECSIZE(80) BUFFERS(2) )
     CONTENTS OF BUFFERS
     BUFFER: 007CDF60
       +000000 007CDF60  40D7D3C9 C4D4D7C2 40E2A381 99A38995  87404040 40404040 40404040 40404040  | PLIDMPB Starting               |
       +000020 007CDF80  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
       +000040 007CDFA0  40404040 40404040 40404040 40404040  40D7D3C9 C4D4D7C1 40E2A381 99A38995  |                PLIDMPA Starting|
     BUFFER: 007CDFB0
       +000000 007CDFB0  40D7D3C9 C4D4D7C1 40E2A381 99A38995  87404040 40404040 40404040 40404040  | PLIDMPA Starting               |
       +000020 007CDFD0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                |
       +000040 007CDFF0  40404040 40404040 40404040 40404040  007CE031 7A958983 924BE3C5 D9C94040  |                                |

   File Control Blocks:
     FILE CONTROL BLOCK (FCB): 007C8004
       +000000 007C8004  00000000 00000000 0056B3CA 005CB5EC  000206D0 007C8090 00000000 00000000  |.............*.......@..........|
       +000020 007C8024  00000000 41211100 82000000 00000104  00500000 00000050 007CDF60 E3C600F4  |........b........&.....&@.-TF.4 |
       +000040 007C8044  00000000 00000000 00000000 007CDF72  003E0001 003C004F 00030000 00000000  |.............@.........|........|
       +000060 007C8064  00000000 006CBBB8 00000000 003CA520  00000000 00000000 00000000 00000000  |.....%........v.................|
     DATA CONTROL BLOCK (DCB): 007C8090
       +000000 007C8090  00000000 00000000 00000000 00000000  00280000 027CDF58 00504000 007D94B0  |.....................@...& ..'m.|
       +000020 007C80B0  40000001 84000000 00000048 007D94A0  92D5927E 00000001 0C5CB7FA 00090050  | ...d........'m.kNk=.....*.....&|
       +000040 007C80D0  00000000 007D94B0 007CDFB0 007CDFB0  00000050 80000001 00000000 00000001  |.....'m..@...@.....&............|
     DECLARE CONTROL BLOCK (DCLCB): 000206D0
       +000000 000206D0  FFFFFFFC 41201000 02D70F00 00000000  00000014 0008E2E8 E2D7D9C9 D5E30000  |.........P.............SYSPRINT..|

Process Control Blocks:

   PCB: 0057F198
     +000000 0057F198  C3C5C5D7 C3C24040 02020220 00000000  00000000 00000000 0057FB30 005C8E10  |CEEPCB  .....................*..|
     +000020 0057F1B8  005C75B8 005C2288 005C1C80 00000000  00000000 00000000 0057FB18 0057FB30  |.*...*.h.*.....................|

   MEML: 0057FB30
     +000000 0057FB30  00000000 00000000 00592030 00000000  00000000 00000000 00592030 00000000  |................................|
     +000020 0057FB50 - +00009F 0057FBCF             same as above
```

*Figure 104. Control Blocks Associated with the Thread Section of the Dump*

### The CAA

The address of the CAA control block appears in this section of the dump. If the BLOCK option is in effect, the complete CAA (including the PL/I implementation appendage) appears separately from the body of the dump. Register 12 addresses the CAA.

### File status and attribute information

This part of the dump includes the following information:

- The default and declared attributes of all open files
- Buffer contents of all file buffers
- The contents of FCBs, DCBs, DCLCBs, IOCBs, and control blocks for the process or enclave

## PL/I contents of the Language Environment trace table

Language Environment provides three PL/I trace table entry types that contain character data:

- Trace entry 100 occurs when a task is created.
- Trace entry 101 occurs when a task that contains the tasking CALL statements is terminated.
- Trace entry 102 occurs when a task that does not contain a tasking CALL statement is terminated.

The format for trace table entries 100, 101, and 102 is:

```
—>(100) NameOfCallingTask NameOfCalledTask OffsetOfCallStmt
            UserAgrPtr CalledTaskPtr TaskVarPtr EventVarPtr
            PriorityPtr CallingR2-R5 CallingR12-R14

—>(101) NameOfReturnTask ReturnerR2-R5 ReturnerR12-R14

—>(102) NameOfReturnTask
```

For more information about the Language Environment trace table format, see “Understanding the trace table entry (TTE)” on page 118.

## Debugging example of PL/I routines

This section contains examples of PL/I routines and instructions for using information in the Language Environment dump to debug them. Important areas in the source code and in the dump for each routine are highlighted.

## Subscript range error

Figure 105 on page 248 illustrates an error caused by an array subscript value outside the declared range. In this example, the declared array value is 10.

This routine was compiled with the options LIST, TEST, GOSTMT, and MAP. It was run with the TERMTHDACT(TRACE) option to generate a traceback for the condition.

```
15688-235 IBM SAA AD/Cycle PL/I          Ver 1 Rel 2 Mod 0                          18 NOV 92   15:57:56    PAGE   1
-OPTIONS SPECIFIED
0*PROCESS  GOSTMT LIST S STG LC(100) TEST MAP;                          00001000
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE:  PROC  OPTIONS(MAIN);                                      PAGE   2
-                   SOURCE LISTING
-    STMT
0
        1  |EXAMPLE:  PROC  OPTIONS(MAIN);                                    |00002000
           |                                                                  |00003000
        2  | DCL Array(10) Fixed bin(31);                                     |00004000
        3  | DCL (I,Array_End)   Fixed bin(31);                               |00005000
           |                                                                  |00006000
        4  | On error                                                         |00007000
           |   Begin;                                                         |00008000
        5  |     On error system;                                             |00009000
        6  |     Call plidump('tbnfs','Plidump called from error ON-unit');   |00010000
        7  |   End;                                                           |00011000
           |                                                                  |00012000
        8  | (subrg):             /* Enable subscriptrange condition*/        |00013000
           |   Labl1: Begin;                                                  |00014000
        9  |     Array_End = 20;                                              |00015000
       10  |     Do I = 1 to Array_End;   /* Loop to initialize array      */ |00016000
       11  |       Array(I) = 2;    /* Set array elements to 2       */        |00017000
       12  |     End;                                                         |00018000
       13  |   End Labl1;                                                     |00019000
           |                                                                  |00020000
       14  |End Example;                                                      |00021000
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE:  PROC  OPTIONS(MAIN);                                      PAGE   3
-                           VARIABLE STORAGE MAP
-IDENTIFIER                 LEVEL         OFFSET       (HEX)   CLASS     BLOCK

 I                             1            200          C8   AUTO      EXAMPLE
 ARRAY_END                     1            204          CC   AUTO      EXAMPLE
 ARRAY                         1            208          D0   AUTO      EXAMPLE
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE:  PROC  OPTIONS(MAIN);                                      PAGE   6
-  OBJECT LISTING
```

*Figure 105. Example of Moving a Value Outside an Array Range*

Figure 106 on page 249 shows sections of the dump generated by a call to PLIDUMP.

```
CEE3DMP V1 R5.0: PLIDUMP called from error ON-unit.                              02/05/95 4:04:12 PM         Page:   1

 PLIDUMP was called from statement number 6 at offset +000000D6 from ERROR ON-unit with entry address 00020168

 Information for enclave EXAMPLE

  Information for thread 8000000000000000

  Registers on Entry to CEE3DMP:

    PM....... 0100
    GPR0..... 00000000  GPR1..... 00077448  GPR2..... 053AD9AF  GPR3..... 853AD514
    GPR4..... 00000001  GPR5..... 053AD314  GPR6..... 80077454  GPR7..... 00000000
    GPR8..... 00000001  GPR9..... 80000000  GPR10.... 00077470  GPR11.... 000F7490
    GPR12.... 0006A520  GPR13.... 000773C8  GPR14.... 80060712  GPR15.... 853F7918
    FPR0..... 4D000000  00043C31            FPR2..... 00000000  00000000
    FPR4..... 00000000  00000000            FPR6..... 00000000  00000000

   Traceback:
     DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset  Load     Service  Statement  Status
     005359A0  CEEKKMRA      00654438  +00000748  CEEKKMRA      00654438  +00000748  CELE38                       Call
     006D9810  LIBRARY(PLI)  005CBE98  +000000B2  LIBRARY(PLI)  005CBE98  +000000B2  CEEPLPKA                     Call
                                                                                     CELE38
     005358A0  EXAMPLE       00020080  +000001BE  ERR  ON-UNIT  00020168  +000000D6                          6  Call
     00535698  IBMRERPL      007CB410  +00000528  IBMRERPL      007CB410  +00000528                             Call
     005355B0  CEEEV010      005B5000  +000000E8  CEEEV010      005B5000  +000000E8                             Call
     006C3018  CEEHDSP       005F6D00  +00000970  CEEHDSP       005F6D00  +00000970                             Call
     00535428  IBMRERRI      007CB040  +00000254  IBMRERRI      007CB040  +00000254                       Exception
     00535358  EXAMPLE       00020080  +00000296  LABL1: BEGIN  00020258  +000000BE                         11  Call
     00535258  EXAMPLE       00020080  +000000D0  EXAMPLE       00020088  +000000C8                          8  Call
     005351B0  IBMRPMIA      007CABD0  +000002FA  IBMRPMIA      007CABD0  +000002FA                             Call
     005350C8  CEEEV010      005B5000  +000001FE  CEEEV010      005B5000  +000001FE                             Call
     00535018  CEEBBEXT      005E55F0  +0000012E  CEEBBEXT      005E55F0  +0000012E                             Call

    Condition Information for Active Routines
     Condition Information for IBMRERRI (DSA address 00535428)
       CIB Address: 006C33C8
       Current Condition:
         IBM0930S
       Original Condition:
         IBM0421S ONCODE=520  The SUBSCRIPTRANGE condition was raised.
       Location:
         Program Unit: IBMRERRI Entry: IBMRERRI Statement:  Offset: +00000254

   Control Blocks for Active Routines:
     DYNAMIC SAVE AREA (EXAMPLE): 00535258
       +000000 00535258  C0250000 005351B0 00000000 4E020152  00020258 00535358 00535258 4E020140  |......é.....+.............+..|
       +000020 00535278  00020350 0002053C 00535258 00535328  0002053C 00020408 00000001 00020088  |...&.....................h|
       +000040 00535298  00000005 006F3848 006D9410 00535358  00535358 91E091A0 00000000 00020408  |.....?..._m.........j.j.....|
       +000060 005352B8  00000000 00000000 00000000 00000001  00535310 00000200 00000001 00000000  |............................|
       +000080 005352D8  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |............................|
       +0000A0 005352F8  00000000 00000000 00000000 00000000  00000000 00000000 0C010000 00000000  |............................|
       +0000C0 00535318  00535328 000203A4 0000000B 00000014  00000002 00000002 00000002 00000002  |............................|
       +0000E0 00535338  00000002 00000002 00000002 00000002  00000002 00000002 00000000 00000000  |............................|

⋮
```

*Figure 106. Sections of the Language Environment Dump*

To debug this routine, use the following steps:

1. In the dump, PLIDUMP was called by the ERROR ON-unit in statement 6. The traceback information in the dump shows that the exception occurred following statement 11.
2. Locate the Original Condition message in the Condition Information for Active Routines section of the dump. The message is `IBM0421S ONCODE=520 The SUBSCRIPTRANGE condition was raised.` This message indicates that the exception occurred when an array element value exceeded the subscript range value (in this case, 10). For more information about this message, see *z/OS Language Environment Run-Time Messages*.
3. Locate statement 9 in the routine in Figure 105 on page 248. 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 105 on page 248. Use this offset to find the I value at the time of the dump. In this example, the offset is X'C8'.
2. Now find offset X'C8' from the start of the stack frame in Figure 106 on page 249.

   The block located at this offset contains the value that exceeded the array range, X'B' or 11.

## Calling a nonexistent subroutine

Figure 107 demonstrates the error of calling a nonexistent subroutine. This routine was compiled with the LIST, MAP, and GOSTMT compiler options. It was run with the TERMTHDACT(DUMP) run-time option to generate a traceback.

```
15688-235 IBM SAA AD/Cycle PL/I          Ver 1 Rel 2 Mod 0                          25 NOV 92   13:47:13    PAGE   1
-OPTIONS SPECIFIED
0*PROCESS  GOSTMT LIST S STG LC(100) TEST MAP;                          00001000
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE1:  PROC  OPTIONS(MAIN);                                    PAGE   2
-                  SOURCE LISTING
-    STMT
0
        1   |EXAMPLE1:  PROC  OPTIONS(MAIN);                                        |00002000
            |                                                                       |00003000
        2   | DCL Prog01      entry external;                                       |00004000
            |                                                                       |00005000
        3   | On error                                                              |00006000
            |   Begin;                                                              |00007000
        4   |     On error system;                                                  |00008000
        5   |     Call plidump('tbnfs','Plidump called from error ON-unit');        |00009000
        6   |   End;                                                                |00010000
            |                                                                       |00011000
        7   | Call Prog01;              /* Call external program PROG01   */        |00012000
            |                                                                       |00013000
        8   |End Example1;                                                          |00014000
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE1:  PROC  OPTIONS(MAIN);                                    PAGE   3
-                  STORAGE REQUIREMENTS
-BLOCK, SECTION OR STATEMENT     TYPE                LENGTH   (HEX)    DSA SIZE   (HEX)
-EXAMLE11                       PROGRAM CSECT          444     1BC
 EXAMLE12                       STATIC CSECT           292     124
 EXAMPLE1                       PROCEDURE BLOCK        210      D2         192      C0
 BLOCK 2      STMT 3            ON UNIT                232      E8         256     100
15688-235 IBM SAA AD/Cycle PL/I          EXAMPLE1:  PROC  OPTIONS(MAIN);                                    PAGE   4
-                       STATIC INTERNAL STORAGE MAP
```

*Figure 107. Example of Calling a Nonexistent Subroutine*

Figure 108 on page 251 shows the traceback and condition information from the dump.

```
CEE3DMP V1 R5.0: PLIDUMP called from error ON-unit.                    12/05/95 1:57:32 PM        Page:   1

 PLIDUMP was called from statement number 5 at offset +000000D6 from ERROR ON-unit with entry address 00020154

 Information for enclave EXAMPLE1

  Information for thread 8000000000000000

  Registers on Entry to CEE3DMP:

    PM....... 0100
    GPR0..... 00000000  GPR1..... 00077448  GPR2..... 053AD9AF  GPR3..... 853AD514
    GPR4..... 00000001  GPR5..... 053AD314  GPR6..... 80077454  GPR7..... 00000000
    GPR8..... 00000001  GPR9..... 80000000  GPR10.... 00077470  GPR11.... 000F7490
    GPR12.... 0006A520  GPR13.... 000773C8  GPR14.... 80060712  GPR15.... 853F7918
    FPR0..... 4D000000  00043C31            FPR2..... 00000000  00000000
    FPR4..... 00000000  00000000            FPR6..... 00000000  00000000
```

*Figure 108. Sections of the Language Environment Dump (Part 1 of 2)*

```
   Traceback:
     DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Status
     003FC708  CEEKKMRA      004AB438  +00000748  CEEKKMRA      004AB438  +00000748              Call
     006D4680  LIBRARY(PLI)  004931C0  +000000B2  LIBRARY(PLI)  004931C0  +000000B2              Call
     003FC608  EXAMPLE1      00020080  +000001AA  ERR  ON-UNIT  00020154  +000000D6           5  Call
     003FC400  IBMRERPL      006E33D8  +00000528  IBMRERPL      006E33D8  +00000528              Call
     003FC318  CEEEV010      0047C000  +0000011C  CEEEV010      0047C000  +0000011C              Call
     00623018  CEEHDSP       005A1D00  +000009B8  CEEHDSP       005A1D00  +000009B8              Call
     003FC258  EXAMPLE1      00020080  +000000C0  EXAMPLE1      0002008C  +000000B4           7  Exception
     003FC1B0  IBMRPMIA      006E2BA8  +00000316  IBMRPMIA      006E2BA8  +00000316              Call
     003FC0C8  CEEEV010      0047C000  +000003DE  CEEEV010      0047C000  +000003DE              Call
     003FC018  CEEBBEXT      005905F0  +0000012E  CEEBBEXT      005905F0  +0000012E              Call

   Condition Information for Active Routines
     Condition Information for EXAMPLE1 (DSA address 003FC258)
       CIB Address: 006233C8
       Current Condition:
         CEE3201S The system detected an Operations exception.
       Location:
         Program Unit: EXAMPLE1 Entry: EXAMPLE1 Statement: 7 Offset: +000000C0
       Machine State:
         ILC..... 0003    Interruption Code..... 0001
         PSW..... FFE40001 CE000006
         GPR0..... 003FC318  GPR1..... 00000000  GPR2..... 4E020134  GPR3..... 00020240
         GPR4..... 0002036C  GPR5..... 00000000  GPR6..... 003FC310  GPR7..... 0002036C
         GPR8..... 000202B0  GPR9..... 0002008C  GPR10.... 00567198  GPR11.... 00000003
         GPR12.... 00574848  GPR13.... 003FC258  GPR14.... 4E020142  GPR15.... 00000000
```

*Figure 108. Sections of the Language Environment Dump (Part 2 of 2)*

To understand the traceback and debug this example routine, use the following steps:

1. Find the Current Condition message in the Condition Information for Active Routines section of the dump. The message is `CEE3201S. The system detected an Operation exception.` For more information about this message, see *z/OS Language Environment Run-Time Messages*.

   This section of the dump also provides such information as the name of the active routine and the current statement number at the time of the dump.

2. Locate statement 7 in the routine (Figure 107 on page 250). 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 109 demonstrates a divide-by-zero error. In this example, the main PL/I routine passed bad data to a PL/I subroutine. The bad data in this example is 0, and the error occurred when the subroutine SUB1 attempted to use this data as a divisor.

```
-                  SOURCE LISTING
-   STMT
0
       1  |SAMPLE: PROC  OPTIONS(MAIN) ;                                          |00002000
          |                                                                       |00003000
       2  | On error                                                              |00004000
          | begin;                                                                |00005000
       3  |   On error system;        /* prevent nested error conditions */      |00006000
       4  |   Call PLIDUMP('TBC','PLIDUMP called from error ON-unit');            |00007000
      5   |   Put Data;               /* Display variables              */      |00008000
       6  | End;                                                                  |00009000
          |                                                                       |00010000
       7  |  DECLARE                                                              |00011000
          |    A_number    Fixed Bin(31),                                         |00012000
          |    My_Name     Char(13),                                              |00013000
          |    An_Array(3) Fixed Bin(31) init(1,3,5);                             |00014000
          |                                                                       |00015000
       8  |  Put skip list('Sample Starting');                                    |00016000
       9  |  A_number = 0;                                                        |00017000
      10  |  My_Name = 'Tery Gillaspy';                                           |00018000
          |                                                                       |00019000
      11  |  Call Sub1(a_number, my_name, an_array);                              |00020000
          |                                                                       |00021000
      12  |    SUB1:  PROC(divisor, name1, Array1);                               |00022000
      13  |      Declare                                                          |00023000
          |        Divisor   Fixed Bin(31),                                       |00024000
          |        Name1     Char(13),                                            |00025000
          |        Array1(3) Fixed Bin(31);                                       |00026000
          |                                                                       |00027000
      14  |      Put skip list('Sub1 Starting');                                  |00028000
      15  |      Array1(1) = Array1(2) / Divisor;                                 |00029000
      16  |      Put skip list('Sub1 Ending');                                    |00030000
      17  |    End SUB1;                                                          |00031000
          |                                                                       |00032000
      18  |  Put skip list('Sample Ending');                                      |00033000
      19  |End;                                                                   |00034000
```

*Figure 109. 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 110 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 110. Variables from Routine SAMPLE*

The routine in Figure 109 on page 252 was compiled with the LIST compiler option, which generated the object code listing shown in Figure 111.

```
-  OBJECT LISTING

 * STATEMENT NUMBER  15
 000372  58 B0 D 0C8              L     11,200(0,13)
 000376  58 40 B 004              L     4,4(0,11)
 00037A  58 90 3 0AC              L     9,172(0,3)
 00037E  5C 80 4 004              M     8,4(0,4)
 000382  58 70 3 0CC              L     7,204(0,3)
 000386  5C 60 4 004              M     6,4(0,4)
 00038A  58 80 D 0C0              L     8,192(0,13)
 00038E  58 60 B 000              L     6,0(0,11)
 000392  5F 60 4 000              SL    6,0(0,4)
 000396  58 E7 6 000              L     14,VO..ARRAY1(7)
 00039A  8E E0 0 020              SRDA  14,32
 00039E  5D E0 8 000              D     14,DIVISOR
 0003A2  50 F9 6 000              ST    15,VO..ARRAY1(9)
```

*Figure 111. Object Code Listing from Example PL/I Routine*

Figure 112 on page 254 shows the Language Environment dump for routine SAMPLE.

```
CEE3DMP V1 R5.0: PLIDUMP called from error ON-unit.                          02/05/95 3:17:13 PM         Page:   1

 PLIDUMP was called from statement number 4 at offset +000000BE from ERROR ON-unit with entry address 0002022C

Information for enclave SAMPLE

  Information for thread 8000000000000000

  Registers on Entry to CEE3DMP:
     PM....... 0100
    GPR0..... 00000000  GPR1..... 00077448  GPR2..... 053AD9AF  GPR3..... 853AD514
    GPR4..... 00000001  GPR5..... 053AD314  GPR6..... 80077454  GPR7..... 00000000
    GPR8..... 00000001  GPR9..... 80000000  GPR10.... 00077470  GPR11.... 000F7490
    GPR12.... 0006A520  GPR13.... 000773C8  GPR14.... 80060712  GPR15.... 853F7918
    FPR0..... 4D000000  00043C31            FPR2..... 00000000  00000000
    FPR4..... 00000000  00000000            FPR6..... 00000000  00000000

   Traceback:
     DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset  Load     Service Statement  Status
     004C58A0  CEEKKMRA      00572438  +00000748  CEEKKMRA      00572438  +00000748  CELE38                      Call
     004BD680  LIBRARY(PLI)  00558498  +000000B2  LIBRARY(PLI)  00558498  +000000B2  CEEPLPKA                    Call
CELE38
     004C5778  SAMPLE        00020080  +0000026A  ERR  ON-UNIT  0002022C  +000000BE                        4  Call
     004C5570  IBMRERPL      006D7470  +00000528  IBMRERPL      006D7470  +00000528                        Call
     004C5488  CEEEV010      00545000  +000000E8  CEEEV010      00545000  +000000E8                        Call
     00675018  CEEHDSP       0059AD58  +000009DC  CEEHDSP       0059AD58  +000009DC                        Call
     004C5388  SAMPLE        00020080  +0000039E  SUB1          00020348  +000000D6                   15  Exception
     004C5258  SAMPLE        00020080  +0000015C  SAMPLE        00020088  +00000154                       11  Call
     004C51B0  IBMRPMIA      006D6BB8  +000003A2  IBMRPMIA      006D6BB8  +000003A2                        Call
     004C50C8  CEEEV010      00545000  +000001FE  CEEEV010      00545000  +000001FE                        Call
     004C5018  CEEBBEXT      005895F0  +0000012E  CEEBBEXT      005895F0  +0000012E                        Call

  Condition Information for Active Routines
     Condition Information for SAMPLE (DSA address 004C5388)
       CIB Address: 006753C8
       Current Condition:
         IBM0281S A prior condition was promoted to the ERROR condition.
       Original Condition:
         CEE3209S The system detected a Fixed Point divide exception.
       Location:
         Program Unit: SAMPLE Entry: SUB1 Statement: 15 Offset: +0000039E
       Machine State:
         ILC..... 0002    Interruption Code..... 0009
         PSW..... FFE40009 AE020422
         GPR0..... 004C5488  GPR1..... 004C5460  GPR2..... 4E0203B4  GPR3..... 00020478
         GPR4..... 00020534  GPR5..... 004C5258  GPR6..... 004C532C  GPR7..... 00000008
         GPR8..... 004C5328  GPR9..... 00000004  GPR10.... 00000003  GPR11.... 004C5320
         GPR12.... 00585848  GPR13.... 004C5388  GPR14.... 00000000  GPR15.... 00000003
      DSA for SUB1: 004C5388
       +000000  FLAGS.... 8025      member... 0000      BKC...... 004C5258  FWC...... 00000000  R14...... 00000000
       +000010  R15...... 00000003  R0....... 004C5488  R1....... 004C5460  R2....... 4E0203B4  R3....... 00020478
       +000024  R4....... 00020534  R5....... 004C5258  R6....... 004C532C  R7....... 00000008  R8....... 004C5328
       +000038  R9....... 00000004  R10...... 00000003  R11...... 004C5320  R12...... 00585848  reserved. 004BD280
       +00004C  NAB...... 004C5488  PNAB..... 004C5488  reserved. 91E091E0  004C5258  00000000  00000000
       +000064  reserved. 00000000  reserved. 00000000  MODE..... 00000000  reserved. 00000000  00000200
       +000078  reserved. 00000000  reserved. 00000000
```

*Figure 112. Language Environment Dump from Example PL/I Routine (Part 1 of 2)*

```
CEE3DMP V1 R5.0: PLIDUMP called from error ON-unit.                              02/05/95 3:17:13 PM         Page:   5
      CIB for SUB1: 006753C8
       +000000 006753C8  C3C9C240 00000000 00000000 010C0002  00000000 00000000 00030119 59C9C2D4  |CIB .........................IBM|
       +000020 006753E8  00000000 00675D08 00030C89 59C3C5C5  00000000 00000004 004C5258 00545000  |......)....i.CEE.........<...&  |
       +000040 00675408  00000000 004C5388 00020422 00586028  0000000A 00000000 00000000 00000000  |.....<.h......-.................|
       +000060 00675428  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
       +000080 00675448 - +00009F 00675467             same as above
       +0000A0 00675468  00000000 00000000 00000000 00000000  44230000 00000000 00000000 00000000  |................................|
       +0000C0 00675488  00000000 00000000 004C5388 004C5388  0002041E 00000000 00000000 00000001  |.........<.h.<.h................|
       +0000E0 006754A8  004C5258 0000000A 00000064 00000000  FFFFFFFC 00000000 00000000 00000000  |.<..............................|
       +000100 006754C8  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
<     DYNAMIC SAVE AREA (SUB1): 004C5388
       +000000 004C5388  80250000 004C5258 00000000 00000000  00000003 004C5488 004C5460 4E0203B4  |.....<..............<.h.<.-+...|
       +000020 004C53A8  00020478 00020534 004C5258 004C532C  00000008 004C5328 00000004 00000003  |.........<...<.......<..........|
       +000040 004C53C8  004C5320 00585848 004BD280 004C5488  004C5488 91E091E0 004C5258 00000000  |.<........K..<.h.<.hj.j..<......|
       +000060 004C53E8  00000000 00000000 00000000 00000000  00000000 00000200 00000000 00000000  |................................|
       +000080 004C5408  00000000 C9C2D4D9 D6D7C1C1 000205A4  000F0000 00675188 004C56A0 5E5C20CA  |....IBMROPAA...u......éh.<..;*..|
       +0000A0 004C5428  005B88C8 00000000 004C54FC 00578198  00675250 00675258 006DB968 006DAF44  |.$hH.....<....aq...&;....._..._..|
       +0000C0 004C5448  004C5328 004C5318 004C5320 00000001  004C5460 00000003 00020510 000204D8  |.<...<...<.......<.-...........Q|
       +0000E0 004C5468  00000000 80020524 00400000 007D1004  00000000 00000000 00010000 004BD460  |......... ...'..............M-|
     DSA for SAMPLE: 004C5258
       +000000  FLAGS.... C025      member... 0000      BKC...... 004C51B0  FWC...... 00000000  R14...... 5E0201DE
       +000010  R15...... 00020348  R0....... 004C5388  R1....... 00020558  R2....... 4E020166  R3.....8
       +000024  R4....... 00000005  R5....... 004C5258  R6....... 004C5330  R7....... 004C5320  R8....... 00000001
       +000038  R9....... 00020088  R10...... 00000003  R11...... 00578198  R12...... 00585848  reserved. 004BD280
       +00004C  NAB...... 004C5388  PNAB..... 004C5388  reserved. 91E091E0  00000000  00020610  00000000
       +000064  reserved. 00000000  reserved. 00000000  MODE..... 00000001  reserved. 004C5310  00000200
       +000078  reserved. 00000001  reserved. 00000000
     DYNAMIC SAVE AREA (SAMPLE): 004C5258
       +000000 004C5258  C0250000 004C51B0 00000000 5E0201DE  00020348 004C5388 00020558 4E020166  |.....<é.....;........<.h....+...|
       +000020 004C5278  00020478 00000005 004C5258 004C5330  004C5320 00000001 00020088 00000003  |.........<...<...<.........h....|
       +000040 004C5298  00578198 00585848 004BD280 004C5388  004C5388 91E091E0 00000000 00020610  |..aq......K..<.h.<.hj.j.........|
       +000060 004C52B8  00000000 00000000 00000000 00000001  004C5310 00000200 00000001 00000000  |.................<..............|
       +000080 004C52D8  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
       +0000A0 004C52F8  00000000 00000000 00000000 00000000  00000000 00000000 0C010000 00000000  |................................|
       +0000C0 004C5318  004C533C 000D0000 004C5330 00020534  00000000 00000000 00000001 00000003  |.<.............................|
       +0000E0 004C5338  00000005 E38599A8 40C78993 9381A297  A8000000 00000000 00000000 00000000  |....Tery Gillaspy...............|
       +000100 004C5358  004C5360 00000000 000204E0 000204D8  00000000 80020524 00400000 007D1004  |.<.-...........Q......... ...'..|
       +000120 004C5378  00000000 00000000 00010000 004BD460  80250000 004C5258 00000000 00000000  |..............M-.....<..........|

     STATIC FOR PROCEDURE SAMPLE    TIMESTAMP: 15 JAN 93  15:10:05
     STARTING FROM: 00020478
       +000000 00020478  E00002A4 00020088 00020130 00020166  0002022C 000202A0 00020348 000203B4  |...u...h........................|
       +000020 00020498  000203B4 000203B4 000203B4 80020AA0  80020AB8 80020AD0 80020AE8 80020B00  |.........................Y....|
       +000040 000204B8  80020B18 800213A8 800211B0 80020B30  80021420 80021438 80020B48 80020B60  |.......y.......................-|
       +000060 000204D8  20000002 1F800000 000205A4 000F0000  00000000 000D0000 00000000 00020534  |...........u....................|
       +000080 000204F8  000205C0 000D0000 00000000 00030000  00000000 00210000 000205F1 000D0000  |.........................1....|
       +0000A0 00020518  000205FE 000B0000 91E091E0 00000001  00000003 00000005 00000000 00000004  |........j.j.....................|
       +0000C0 00020538  00000004 00000003 00000001 00000002  00020738 00020738 004C5360 80020524  |.........................<.-....|
       +0000E0 00020558  004C5328 004C5318 804C5320 00020738  00205240 004C586C 804C5898 00000000  |.<...<...<.............<.%.<.q   |
       +000100 00020578  80020A70 00020738 80000000 00000000  80020620 00020738 004C5460 80020524  |.........................<.-....|
       +000120 00020598  00020738 00000000 80020524 E2819497  938540E2 A38199A3 899587E3 8599A840  |...........Sample StartingTery  |
       +000140 000205B8  C7899393 81A297A8 E2819497 938540C5  95848995 87E3C2C3 D7D3C9C4 E4D4D740  |GillaspySample EndingTBCPLIDUMP |
       +000160 000205D8  83819393 85844086 99969440 85999996  9940D6D5 60A49589 A3E2A482 F140E2A3  |called from error ON-unitSub1 St|
       +000180 000205F8  8199A389 9587E2A4 82F140C5 95848995  87000000 00000000 0C160000 0002022C  |artingSub1 Ending...............|
       +0001A0 00020618  0C960000 00000000 00020634 00020650  0002066C 00000000 00000000 85000001  |.o............&..%.......e.....|
       +0001C0 00020638  000204DA 000000D0 00000000 0008C16D  D5E4D4C2 C5D90000 81000001 000204D8  |..............A_NUMBER..a......Q|
       +0001E0 00020658  000000C0 00000000 0007D4E8 6DD5C1D4  C5000000 81000101 000204DA 000000C8  |..........MY_NAME...a..........H|
       +000200 00020678  00000000 0008C1D5 6DC1D9D9 C1E80000  00000001 00020088 000001A4 000206BC  |......AN_ARRAY.........h...u....|
       +000220 00020698  00000001 00DE0002 00E20008 01200009  0128000A 012E000B 01560012 01940013  |.........S...................m..|
CEE3DMP V1 R5.0: PLIDUMP called from error ON-unit.                              02/05/95 3:17:13 PM         Page:   4

       +000240 000206B8  01A40002 0002022C 00000112 000206E0  00000002 00740003 00780004 00C00005  |.u.............................|
       +000260 000206D8  01020006 0114000C 00020348 0000012C  00020704 0000000C 006C000E 00AA000F  |.......................%......|
       +000280 000206F8  00DE0010 011C0011 011C0011 0E0E0E0E  F1F540D1 C1D540F9 F34040F1 F57AF1F0  |................15 JAN 93  15:10|
       +0002A0 00020718  7AF0F540 80000010 00020000 00000000  01000001 00020088 00020758 00000000  |:05 ...................h........|
```

*Figure 112. Language Environment Dump from Example PL/I Routine (Part 2 of 2)*

To understand the dump information and debug this routine, use the following steps:

1. Notice the title of the dump:`PLIDUMP called from error ON-unit`. This was the title specified when PLIDUMP was invoked, and it indicates that the ERROR condition was raised and PLIDUMP was called from within the ERROR ON-unit.
2. Locate the messages in the Condition Information section of the dump.

   There are two messages. The current condition message indicates that a prior condition was promoted to the ERROR condition. The promotion of a condition

occurs when the original condition is left unhandled (no PL/I ON-units are assigned to gain control). The original condition message is `CEE3209S. The system detected a Fixed Point divide exception`. The original condition usually indicates the actual problem. For more information about this message, see *z/OS Language Environment Run-Time Messages*.

3. In the traceback section, note the sequence of calls in the call chain. SAMPLE called SUB1 at statement 11, and SUB1 raised an exception at statement 15, PU offset X'39E'.
4. Find the statement in the listing for SUB1 that raised the ZERODIVIDE condition. If SUB1 was compiled with GOSTMT and SOURCE, find statement 15 in the source listing.

   Since the object listing was generated in this example, you can also locate the actual assembler instruction causing the exception at offset X'39E' in the object listing for this routine, shown in Figure 111 on page 253. Either method shows that *divisor* was used as the divisor in a divide operation.
5. You can see from the declaration of SUB1 that *divisor* is a parameter passed from SAMPLE. Because of linkage conventions, you can infer that register 1 in the SAMPLE save area points to a parameter list that was passed to SUB1. *divisor* is the first parameter in the list.
6. In the SAMPLE DSA, the R1 value is X'20558'. This is the address of the parameter list, which is located in static storage.
7. Find the parameter list in the stack frame; the value of the first parameter is X'00000000'. Thus, the exception occurred when SAMPLE passed a 0 value used as a divisor in subroutine SUB1.

# Chapter 8. Debugging under CICS

This chapter provides information for debugging under the Customer Information Control System (CICS). The following sections explain how to access debugging information under CICS, and describe features unique to debugging under CICS.

Use the following list as a quick reference for debugging information:
- Language Environment run-time messages (CESE Transient Data Queue)
- Language Environment traceback (CESE Transient Data Queue)
- Language Environment dump output (CESE Transient Data Queue)
- CICS Transaction Dump (CICS DFHDMPA or DFHDMPB data set)
- Language Environment abend and reason codes (system console)
- Language Environment return codes to CICS (system console)

If the EXEC CICS HANDLE ABEND command is active and the application, or CICS, initiates an abend or application interrupt, then Language Environment does not produce any run-time messages, tracebacks, or dumps.

If EXEC CICS ABEND NODUMP is issued, then no Language Environment dumps or CICS transaction dumps are produced.

## Accessing debugging information

The following sections list the debugging information available to CICS users, and describe where you can find this information.

Under CICS, the Language Environment run-time messages, Language Environment traceback, and Language Environment dump output are written to the CESE transient data queue. The transaction identifier, terminal identifier, date, and time precede the data in the queue. For detailed information about the format of records written to the transient data queue, see *z/OS Language Environment Programming Guide*.

The CESE transient data queue is defined in the CICS destination control table (DCT). The CICS macro DFHDCT is used to define entries in the DCT. See *CICS Resource Definition Guide* for a detailed explanation of how to define a transient data queue in the DCT. If you are not sure how to define the CESE transient data queue, see your system programmer.

## Locating Language Environment run-time messages

Under CICS, Language Environment run-time messages are written to the CESE transient data queue. A sample Language Environment message that appears when an application abends due to an unhandled condition from an EXEC CICS command is:

```
P039UTV9 19910916145313 CEE3250C The System or User ABEND AEI0 was issued.
P039UTV9 19910916145313          From program unit UT9CVERI at entry point UT9CVERIT
                                 +0000011E at P039UTV9 19910916145313
                                 at offset address 0006051E.
```

## Locating the Language Environment traceback

Under CICS, the Language Environment traceback is written to the CESE transient data queue. Because Language Environment invokes your application routine, the Language Environment routines that invoked your routine appear in the traceback. Figure 113 on page 258 shows an example Language Environment traceback

© Copyright IBM Corp. 1991, 2004

written to the CESE transient data queue. Data unnecessary for this example has been replaced by ellipses.

```
 P023T2AB 19911011084413 CEE3211S The system detected a Decimal-divide exception.
 P023T2AB 19911011084413          From program unit T2AB at entry point T2AB at offset +000014A8 at address 0A9444E0.
1P023T2AB 19911011084413 CEE3DMP V1 R3.0: Condition processing resulted in the unhandled condition.          08/30/01 09:23:10
 P023T2AB 19911011084413
 P023T2AB 19911011084413 Information for enclave T2AB
 P023T2AB 19911011084413
 P023T2AB 19911011084413   Information for thread 8000000000000000
 P023T2AB 19911011084413
 P023T2AB 19911011084413   Registers on Entry to CEE3DMP:
 P023T2AB 19911011084413
 P023T2AB 19911011084413     PM....... 0100
 P023T2AB 19911011084413     GPR0..... 00000000  GPR1..... 00077448  GPR2..... 053AD9AF  GPR3..... 853AD514
 P023T2AB 19911011084413     GPR4..... 00000001  GPR5..... 053AD314  GPR6..... 80077454  GPR7..... 00000000
 P023T2AB 19911011084413     GPR8..... 00000001  GPR9..... 80000000  GPR10.... 00077470  GPR11.... 000F7490
 P023T2AB 19911011084413     GPR12.... 0006A520  GPR13.... 000773C8  GPR14.... 80060712  GPR15.... 853F7918
 P023T2AB 19911011084413     FPR0..... 4D000000  00043C31            FPR2..... 00000000  00000000
 P023T2AB 19911011084413     FPR4..... 00000000  00000000            FPR6..... 00000000  00000000
 P023T2AB 19911011084413
 P023T2AB 19911011084413   Traceback:
 P023T2AB 19911011084413     DSA Addr  Program Unit  PU Addr   Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
 P023T2AB 19911011084413     0A9A5630  CEEHDSP       001DB878  c9101         05500028  +00000152                                Call
 P023T2AB 19911011084413     0A9AD1D0  CEECGEX       001D6BF0  CB2C9101      055001E0  +000003D0                                Call
 P023T2AB 19911011084413     0A9ABD88  T2AB          0A943038  IGZCEV5       04CF9000  +00000836                                Call
 P023T2AB 19911011084413     0A9AD098  CEECRINV      001D9F90  CEECRINV      0522A708  +0000036E                                Call
 P023T2AB 19911011084413     0A9AD010  CEECCICS      001C6370  CEECCICS      0001FF28  +00000456              CEECCICS  UQ00568  Call
⋮
```

*Figure 113. 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 45.

## 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 insideLanguage 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 happpened 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 bee 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        Runounit_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        Runounit_End_Invocation NAMETEST                                         =000386=
00036 1 AP 1949 APLI  EVENT RETURN-FROM-LE/370    Runounit_End_Invocation OK NAMETEST                                      =000387=
00036 1 AP 1948 APLI  EVENT CALL-TO-LE/370        Runounit_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    Runounit_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 114. CICS trace output in the ABBREV format.*

The Domain Name field is replaced with a ″Feature″ short name (for example, Lang.Env.) and module name (for example, CEE.....) which are coded into the ″Feature Trace″ initialization (short name) and header formatting call (module name). See the following macro example.

The FULL version includes the heading from the ABBREV version and then dumps each captured block in Hex and Character formats. For example:

```
AP 1948 APLI EVENT CALL-TO-LE/370 - Rununit_Initialization    Program_name(NAMETEST)

              TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-868218FE TIME-05:58:55.2643333923 INTERVAL-00.0000020781    =000343=
                1-0000  0000001E                                                        *....                           *
                2-0000  06878DE0 00140148 0005848C 0014014C  00045A4C 00140130 0014001C 067F3CE8  *.g.\......d....<..!<.........".Y*
                  0020  0678FAD8 06878F37 867F3DD0                                     *...Q.g..f".}                   *
                3-0000  D5C1D4C5 E3C5E2E3                                             *NAMETEST                       *
                4-0000  00000030 20000000 07500000 00001B00  87500020 00000000 06C03800 00000000  *.........&......g&.......{......*
                  0020  00140050 00000000 00000000 0678FABC                           *...&............               *

FT Lang.Env.  1014 CEEZCREN EVENT - CEEEVNT-ID(PRCINIT)   R13(06C00E10), PARMS(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 - CEEEVNT-ID(OPTP)      R13(06C00E10), PARMS(06C049B0, 07500F28, 06C0403C, 06C010B4)

              TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F0A23A TIME-05:58:55.2644148454 INTERVAL-00.0000178125    =000345=
                  0000  0000C9C2 D4404040 40404040 40404040  40404040 40404040 40404040 40404040  *..IBM                          *
                  0020  D3819587 A4818785 40C595A5 89999695  948595A3 40404040 40404040 4040F0F0  *Language Environment          00*
                  0040  F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3  D3819587 4BC595A5 4B000000          *00000001CEECTFMTLang.Env....   *
                1-0000  06C00E10 00000004 06C049B0 07500F28  06C0403C 06C010B4                 *.{.......{...&...{ ..{..       *

FT Lang.Env.  1101 CEECRINI EVENT - SET_ANCHOR R13(06C009B8), PARMS(06C06180, 00000002)

              TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F02F90 TIME-05:58:55.2644493767 INTERVAL-00.0000345312    =000346=
                  0000  0000C9C2 D4404040 40404040 40404040  40404040 40404040 40404040 40404040  *..IBM                          *
                  0020  D3819587 A4818785 40C595A5 89999695  948595A3 40404040 40404040 4040F0F0  *Language Environment          00*
                  0040  F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3  D3819587 4BC595A5 4B0092B4          *00000001CEECTFMTLang.Env..k.   *
                1-0000  06C009B8 06C06180 00000002                                     *.{...{/.....                   *

FT Lang.Env.  1018 CEEZINV  EVENT - CEEEVNT-ID(ENCINIT)   R13(06C06D80), PARMS(00000000, 06C0403C, 00000000, 06C041B4, 00000000,
                         01000000, 00000000, 00000000)

              TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F0C9B4 TIME-05:58:55.2644710798 INTERVAL-00.0000217031    =000347=
                  0000  000AC9C2 D4404040 40404040 40404040  40404040 40404040 40404040 40404040  *..IBM                          *
                  0020  D3819587 A4818785 40C595A5 89999695  948595A3 40404040 40404040 4040F0F0  *Language Environment          00*
                  0040  F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3  D3819587 4BC595A5 4B0072B4          *00000001CEECTFMTLang.Env....   *
                1-0000  06C06D80 00000012 00000000 06C0403C  00000000 06C041B4 00000000 01000000  *.{_...........{ ......{..........*
                  0020  00000000 00000000                                              *........                       *
                2-0000  D8C3C5E2 C9000000 00000000 00000000  D8C3C5E2 D6000000 00000000 00000000  *QCESI...........QCESO...........*
                  0020  D8C3C5E2 C5000000 00000000 00000000                            *QCESE...........               *

FT Lang.Env.  1008 CEECRINV EVENT - CEEEVNT-ID(MAININV)   R13(06C06D80), PARMS(87500020, 00000001, 00000000, 00140050, 87500020)

              TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-06F038D2 TIME-05:58:55.2645123298 INTERVAL-00.0000412500    =000348=
                  0000  0000C9C2 D4404040 40404040 40404040  40404040 40404040 40404040 40404040  *..IBM                          *
                  0020  D3819587 A4818785 40C595A5 89999695  948595A3 40404040 40404040 4040F0F0  *Language Environment          00*
                  0040  F0F0F0F0 F0F0F0F1 C3C5C5C3 E3C6D4E3  D3819587 4BC595A5 4B000000          *00000001CEECTFMTLang.Env....   *
                1-0000  06C06D80 0000000E 87500020 00000001  00000000 00140050 87500020          *.{_.....g&.............&g&..    *

AP 1948 APLI EVENT CALL-TO-LE/370 - Rununit_End_Invocation    Program_name(NAMETEST)

              TASK-00036 KE_NUM-0026 TCB-006FA1D0 RET-868218FE TIME-05:58:55.2670554079 INTERVAL-00.0000107187    =000386=
                1-0000  00000021                                                        *....                           *
                2-0000  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 00000000  *................................*
```

*Figure 115. 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 (CEECXITA), 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 24 and *z/OS Language Environment Programming Guide*.

## Finding data when Language Environment returns a nonzero return code

Language Environment does not write any messages to the CESE transient data queue. Following is the output generated when Language Environment returns a nonzero reason code to CICS and the location where the output appears:

*Table 25. Finding Data When Language Environment Returns a Nonzero Return Code*

| Output Message | Location | Issued By |
|---|---|---|
| DFHAC2206 14:43:54 LE03CC01 Transaction UTV2 has failed with abend AEC7. Resource backout was successful. | User's terminal | CICS |
| DFHAP1200I LE03CC01 A CICS request to the Language Environment has failed. Reason code '0012030'. | System console | CICS |
| DFHAC2236 06/05/91 14:43:48 LE03CC01 Transaction UTV2 abend AEC7 in routine UT2CVERI term P021 backout successful. | Transient data queue CSMT | CICS |

## Finding data when Language Environment abends internally

Language Environment does not write any messages to the CESE transient data queue. Following is the output generated when Language Environment abends internally and the location where the output appears:

*Table 26. Finding Data When Language Environment Abends Internally*

| Output Message | Location | Issued By |
|---|---|---|
| DFHAC2206 14:35:24 LE03CC01 Transaction UTV8 has failed with abend 4095. Resource backout was successful. | User's terminal | CICS |
| CEE1000S LE INTERNAL abend. ABCODE = 00000FFF REASON = 00001234 | System console | Language Environment |

*Table 26. Finding Data When Language Environment Abends Internally (continued)*

| Output Message | Location | Issued By |
|---|---|---|
| DFHAC2236 06/05/91 14:35:24 LE03CC01 Transaction UTV8 abend 4095 in routine UT8CVERI term P021 backout successful. | Transient data queue CSMT | CICS |

## Finding data when Language Environment Abends from an EXEC CICS command

This section shows the output generated when an application abends from an EXEC CICS command and the location where the output appears.

This error assumes the use of Language Environment run-time option TERMTHDACT(MSG).

*Table 27. Finding Data When Language Environment Abends from an EXEC CICS Command*

| Output Message | Location | Issued By |
|---|---|---|
| DFHAC2206 14:35:34 LE03CC01 Transaction UTV8 has failed with abend AEI. Resource backout was successful. | User's terminal | CICS |
| No message. | System console | CICS |
| DFHAC2236 06/05/91 14:35:17 LE03CC01 Transaction UTV9 abend AEI0 in routine UT9CVERI term P021 backout successful. | Transient data queue CSMT | CICS |
| P021UTV9 091156 143516 CEE3250C The System or User Abend AEI0 was issued. | Transient data queue CESE | Language Environment |

## Displaying and modifying run-time options with the CLER transaction

The CLER transaction can be used to:

- Display the current run-time options in effect for the region.
- Modify the following subset of the region run-time options:
  - TRAP(ON|OFF)
  - TERMTHDACT(QUIET|MSG|TRACE|DUMP|UAONLY|UATRACE|UADUMP|UAIMM)
  - RPTOPTS(ON|OFF)
  - RPTSTG(ON|OFF)
  - ALL31(ON|OFF)
  - CBLPSHPOP(ON|OFF)
- 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(CEL4RTO) GROUP(CEE) LANGUAGE(ASSEMBLER) EXECKEY(CICS)
DEFINE MAPSET(CELCLEM) GROUP(CEE)
DEFINE MAPSET(CELCLRH) GROUP(CEE)
DEFINE TRANS(CLER) PROG(CEL4RTO) GROUP(CEE)
```

Use the CEECCSD job to activate these definitions, or you must define them dynamically with the CICS CEDA transaction.

**Note:** If the run-time option ALL31 is modified to OFF, the stack is forced to BELOW. A warning message, asking if want to continue, is presented on the panel. Once the stack is modified to BELOW, it will remain below for the duration of the region, even if you set ALL31 back to ON.

To send the run-time option report to the CESE queue for output display or printing, press `PF10` on the panel which displays the run-time option report.

For detailed information on the use of CLER, select `PF1` from the main menu that is displayed when the CLER transaction is invoked.

# Part 3. Appendixes

© Copyright IBM Corp. 1991, 2004

# Appendix A. Diagnosing Problems with Language Environment

This appendix provides information for diagnosing problems in the Language Environment product. It helps you determine if a correction for a product failure similar to yours has been previously documented. If the problem has not been previously reported, it tells you how to open a Problem Management Record (PMR) to report the problem to IBM, and if the problem is with an IBM product, what documentation you need for an Authorized Program Analysis Report (APAR).

## Diagnosis Checklist

Step through each of the items in the diagnosis checklist below to see if they apply to your problem. The checklist is designed to either solve your problem or help you gather the diagnostic information required for determining the source of the error. It can also help you confirm that the suspected failure is not a user error; that is, it was not caused by incorrect usage of the Language Environment product or by an error in the logic of the routine.

1. If your failing application contains programs that were changed since they last ran successfully, review the output of the compile or assembly (listings) for any unresolved errors.
2. If there have not been any changes in your applications, check the output (job or console logs, CICS transient (CESE) queues) for any messages from the failing run.
3. Check the message prefix to identify the system or subsystem that issued the message. This can help you determine the cause of the problem. Following are some of the prefixes and their respective origins.

   **EDC** The prefix for C/C++ messages. The following series of messages are from the C/C++ run-time component of Language Environment: 5000 (except for 5500, which are from the DSECT utility), 6000, and 7000.

   **IGZ** The prefix for messages from the COBOL run-time component of Language Environment.

   **FOR** The prefix for messages from the Fortran run-time component of Language Environment.

   **IBM** The prefix for messages from the PL/I run-time component of Language Environment.

   **CEE** The prefix for messages from the common run-time component of Language Environment.
4. For any messages received, check for recommendations in the “Programmer Response” sections of the messages in this manual.
5. Verify that abends are caused by product failures and not by program errors. See the appropriate chapters in this manual for a list of Language Environment-related abend codes.
6. Your installation may have received an IBM Program Temporary Fix (PTF) for the problem. Verify that you have received all issued PTFs and have installed them, so that your installation is at the most current maintenance level.
7. The preventive service planning (PSP) bucket, an online database available to IBM customers through IBM service channels, gives information about product installation problems and other problems. Check to see whether it contains information related to your problem.

© Copyright IBM Corp. 1991, 2004

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 XPLINK and non-XPLINK applications, IBM recommends that you use the IPCS verbexit LEDATA 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, seeChapter 3, “Using Language Environment debugging facilities,” on page 37.
   - Alternatively, in a z/VM or a non-XPLINK environment, you can follow the save area chain to find out the name of the failing module and whether IBM owns it. For information on finding the routine name, see “Locating the Name of the Failing Routine for a Non-XPLINK Application.”
9. After you identify the failure, consider writing a small test case that re-creates the problem. The test case could help you determine whether the error is in a user routine or in the Language Environment product. Do not make the test case larger than 75 lines of code. The test case is not required, but it could expedite the process of finding the problem.

   If the error is not a Language Environment failure, refer to the diagnosis procedures for the product that failed.
10. Record the conditions and options in effect at the time the problem occurred. Compile your program with the appropriate options to obtain an assembler listing and data map. If possible, obtain the binder or linkage editor output listing if running on MVS or the LOAD/GENMOD map if running on z/VM. Note any changes from the previous successful compilation or run. For an explanation of compiler options, refer to the compiler-specific programming guide.
11. If you are experiencing a no-response problem, try to force a dump. under z/VM in the CP mode, enter the DUMP command. Under other systems, 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 on page 272.

If the entry point type is nonconforming, go to step 6 on page 273.

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.

   Figure 116 on page 270 illustrates the non-XPLINK Language Environment-conforming PPA1 and PPA2.

**PPA1: Entry Point Block**

| | | | | |
|---|---|---|---|---|
| X'00' | Offset to the length of name | X'CE' (Lang Env Signature) | Lan Env flags | Member flags |
| X'04' | Address of PPA2 | | | |
| X'08' | Signed offset to BDI from the entry point of zero | | | |
| X'0C' | Reserved | | | |
| X'10' | Reserved | | | |
| X'14' | Reserved | | | |
| X'18' | Reserved | | | |
| X'1C' | Language Environment flags (16 bits) | | | |
| | ⋮ | | | |
| | Length of name | Untruncated entry/label name | | |

**PPA2: Compile Unit Block**

| | | | | |
|---|---|---|---|---|
| X'00' | Member identifier | Member Subid | Member Defined | Control Level (= 1) |
| X'04' | V(CEESTART) for load module | | | |
| X'08' | Offset from PPA2 to CDI, zero if no compile unit debug info | | | |
| X'0C' | Offset from PPA2 to timestamp/version information, or zero | | | |
| X'10' | A(PEP) - address of the compilation unit's Primary Entry Point | | | |
| | ⋮ | | | |

*Figure 116. Language Environment Non-XPLINK PPA1 and PPA2*

Figure 117 on page 271 illustrates the XPLINK Language Environment-conforming PPA1, followed by the XPLINK PPA1 optional area fields.

**PPA1: XPLINK Entry Point Block Fixed Area (Version 3)**

| | | | | | |
|---|---|---|---|---|---|
| +00 | Version | LE Signature X'CE' (Lan Env Signature) | Saved GPR Mask | | |
| +04 | Signed Offset to PPA2 from start of PPA1 | | | | |
| +08 | PPA1 Flags 1 | PPA1 Flags 2 | PPA1 Flags 3 | PPA1 Flags 4 | |
| +0C | Length/4 of Parms | | Length/2 of Prolog | Alloca Reg | Offs/2 R4 Chg |
| +10 | Length of Code | | | | |

*Figure 117. Language Environment PPA1 for XPLINK*

Figure 118 illustrates the non-XPLINK Language Environment PPA2.

PPA2: Compile Unit Block

| | | | | |
|---|---|---|---|---|
| 00<br>04 | Member Identifier | Member Subid | Member Defined | Control Level (= 1) |
| 08 | V(CEESTART) for load module | | | |
| 0C | Offset from PPA2 to Compile Debugging Information (CDI) or zero | | | |
| 10 | Offset from PPA2 to timestamp/version information or zero | | | |
| | A(PEP) - address to the compilation unit's Primary Entry Point | | | |
| | :<br>:<br>: | | | |

*Figure 118. PPA2: Compile Unit Block (Non-XPLINK)*

Figure 119 on page 272 illustrates the Language Environment PPA2: Compile Unit Block for XPLINK, and Figure 120 on page 272 illustrates the PPA2 timestamp and version information.

**PPA2: Compile Unit Block**

| | | | | |
|---|---|---|---|---|
| +00 | Member Identifier | Member Subid | Member Defined | Control Level (= 4) |
| +04 | Signed offset from PPA2 to CEESTART for load module | | | |
| +08 | Signed offset from PPA2 to CDI or zero if no compile unit debug info | | | |
| +0C | Signed offset from PPA2 to timestamp/version information or zero | | | |
| +10 | Signed offset from PPA2 to the compilation unit's Primary Entry Point | | | |
| +14 | Compilation flags | Reserved (must be zero) | | |
| | | | | |

*Figure 119. PPA2 Compile Unit Block for XPLINK*

| | | |
|---|---|---|
| 00 | CL4'yyyy' Year of compilation | |
| 04 | CL4'mmdd' Date of compilation | |
| 08 | CL4'hhmm' Time of compilation | |
| 0C | CL2'ss' Time of compilation | CL2'vv' Version |
| 10 | CL4'rrmm' Release/Modification | |
| 14 | Service level string length | Untruncated service level string |

*Figure 120. PPA2 Timestamp and Version Information for XPLINK*

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 121 on page 273 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 121. 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 122 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 122. Nonconforming Entry Point Type with Sample Dump*

## Searching the IBM Software Support Database

Failures in the Language Environment product can be described through the use of keywords. A keyword is a descriptive word or abbreviation assigned to describe one aspect of a product failure. A set of keywords, called a keyword string, describes the failure in detail. You can use a keyword or keyword string as a search argument against an IBM software support database, such as the Service Information Search

(SIS). The database contains keyword and text information describing all current problems reported through APARs and associated PTFs. IBM Support Center personnel have access to the software support database and are responsible for storing and retrieving the information. Using keywords or a keyword string, they will search the database to retrieve records that describe similar known problems.

If you have IBMLink™ or some other connection to the IBM databases, you can do your own search for previously recorded product failures before calling the IBM Support Center.

If your keyword or keyword string matches an entry in the software support database, the search may yield a more complete description of the problem and possibly identify a correction or circumvention. Such a search may yield several matches to previously reported problems. Review each error description carefully to determine if the problem description in the database matches the failure.

If a match is not found, go to “Preparing Documentation for an Authorized Program Analysis Report (APAR).”

## Preparing Documentation for an Authorized Program Analysis Report (APAR)

Prepare documentation for an APAR only after you have done the following:

- Eliminated user errors as a possible cause of the problem.
- Followed the diagnostic procedures.
- You or your local IBM Support Center has been unsuccessful with the keyword search.

Having met these criteria, follow the instructions below.

1. Report the problem to IBM.

   If you have not already done so, report the problem to IBM by opening a Program 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 28 describes how to produce documentation required for submission with the APAR.

*Table 28. 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:<br>Source listing<br>Object listing<br>Storage map<br>Traceback<br>Cross-reference listing<br>JCL listing and linkage editor listing<br>Assembler-language expansion | Use appropriate compiler options |
| 3 | Dumps<br>Language Environment dump<br>System dump | See instructions in Chapter 3, "Using Language Environment debugging facilities," on page 37 (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 it 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 Volume 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/servers/eserver/zseries/zos/bkserv/`

One exception is command syntax that is published in railroad track format; screen-readable copies of z/OS books with that syntax information are separately available in HTML zipfile form upon request to mhvrcfs@us.ibm.com.

© Copyright IBM Corp. 1991, 2004

# 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

© Copyright IBM Corp. 1991, 2004

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Programming Interface Information

This publication documents information NOT intended to be used as a Programming Interface of Language Environment in z/OS.

## Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

| | | |
|---|---|---|
| AD/Cycle | IBMLink | SAA |
| AFP | IMS | SXM |
| C/370 | IMS/ESA | System/370 |
| CICS | Language Environment | TCS |
| COBOL/370 | MVS | VisualAge |
| DB2 | MVS/ESA | z/OS |
| DFSMS/MVS | Open Class | z/OS.e |
| DFSORT | OS/390 | zSeries |
| IBM | Resource Link | |

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.

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 Run-Time Application Migration Guide*, GA22-7565
- *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 C/C++***
- *z/OS C/C++ Language Reference*, SC09-4815
- *z/OS C/C++ Compiler and Run-Time Migration Guide*, GC09-4913
- *z/OS C/C++ Programming Guide*, SC09-4765
- *z/OS C/C++ User's Guide*, SC09-4767
- *z/OS C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS C/C++ Messages*, GC09-4819
- *Standard C++ Library Reference*, SC09-4949
- *C/C++ Legacy Class Libraries Reference*, SC09-7652
- *IBM Open Class Library Transition Guide*, SC09-4948

***Enterprise COBOL for z/OS and OS/390***
- *Enterprise COBOL for z/OS and OS/390 Licensed Program Specifications*, GC27-1411
- *Enterprise COBOL for z/OS and OS/390 Customization*, GC27-1410
- *Enterprise COBOL for z/OS and OS/390 Language Reference*, SC27-1408
- *Enterprise COBOL for z/OS and OS/390 Programming Guide*, SC27-1412
- *Enterprise COBOL for z/OS and OS/390 Migration Guide*, GC27-1409

***COBOL for OS/390 & VM***
- *COBOL for OS/390 & VM Licensed Program Specifications*, GC26-9044
- *COBOL for OS/390 & VM Customization under OS/390*, GC26-9045
- *COBOL for OS/390 & VM Language Reference*, SC26-9046
- *COBOL for OS/390 & VM Programming Guide*, SC26-9049
- *COBOL for OS/390 & VM Compiler and Run-Time Migration Guide*, GC26-4764

***COBOL for MVS & VM*** (Release 2)
- *Licensed Program Specifications*, GC26-4761
- *Programming Guide*, SC26-4767
- *Language Reference*, SC26-4769
- *Compiler and Run-Time Migration Guide*, GC26-4764
- *Installation and Customization under MVS*, SC26-4766
- *Reference Summary*, SX26-3788
- *Diagnosis Guide*, SC26-3138

***VS COBOL II***

© Copyright IBM Corp. 1991, 2004

*VS COBOL II Application Programming Guide for MVS and CMS*, SC26-4045

***Debug Tool***

- Debug Tool documentation is available at:
  `http://www.ibm.com/software/ad/debugtool/library/`

***VS FORTRAN Version 2***

- *Language Environment Fortran Run-Time Migration Guide*, SC26-8499
- *Language and Library Reference*, SC26-4221
- *Programming Guide for CMS and MVS*, SC26-4222

***Enterprise PL/I for z/OS and OS/390, V3R1***

- *Enterprise PL/I for z/OS and OS/390 Licensed Program Specifications*, GC27-1456
- *Enterprise PL/I for z/OS and OS/390 Programming Guide*, SC27-1457
- *Enterprise PL/I for z/OS and OS/390 Language Reference*, SC27-1460
- *Enterprise PL/I for z/OS and OS/390 Migration Guide*, GC27-1458
- *Enterprise PL/I for z/OS and OS/390 Messages and Codes*, SC27-1461
- *Enterprise PL/I for z/OS and OS/390 Diagnosis Guide*, SC27-1459

***VisualAge PL/I***

- *VisualAge PL/I for OS/390 Licensed Program Specifications*, GC26-9471
- *VisualAge PL/I for OS/390 Programming Guide*, SC26-9473
- *VisualAge PL/I Language Reference*, SC26-9476
- *VisualAge PL/I for OS/390 Compiler and Run-Time Migration Guide*, SC26-9474
- *VisualAge PL/I Messages and Codes*, SC26-9478
- *VisualAge PL/I for OS/390 Diagnosis Guide*, SC26-9475

***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*, GC34-5985
- *CICS Operations and Utilities Guide*, SC34-5991
- *CICS Problem Determination Guide*, GC34-6002
- *CICS Resource Definition Guide*, SC34-5990
- *CICS Data Areas*, LY33-6100
- *CICS Application Programming Guide*, SC34-5993
- *CICS Application Programming Reference*, SC34-5994
- *CICS System Definition Guide*, SC34-5988

***DB2***

*Database 2 Application Programming and SQL Guide*, SC26-4377

***DFSMS/MVS***

*z/OS DFSMS Program Management*, SC27-1130
*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/ESA Application Programming: Design Guide*, SC26-8728
*IMS/ESA Application Programming: Database Manager*, SC26-8727
*IMS/ESA Application Programming: Transaction Manager*, SC26-8729
*IMS/ESA Application Programming: EXEC DLI Commands for CICS and IMS*, SC26-8726

***msys for Setup***

- *z/OS Managed System Infrastructure for Setup User's Guide*, SC33-7985

***z/OS***

- *z/OS Introduction and Release Guide*, GA22-7502
- *z/OS Program Directory*, GI10-0670
- *z/OS and z/OS.e Planning for Installation*, GA22-7504
- *z/OS Information Roadmap*, SA22-7500
- *z/OS Hot Topics Newsletter*, GA22-7501
- *z/OS Licensed Program Specifications*, GA22-7503
-
- *z/OS ISPF Dialog Tag Language Guide and Reference*, SC34-4824
- *z/OS ISPF Planning and Customizing*, GC34-4814
- *z/OS ISPF Dialog Developer's Guide and Reference*, SC34-4821
-
- *z/OS UNIX System Services User's Guide*, SA22-7801
- *z/OS UNIX System Services Command Reference*, SA22-7802
- *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803
- *z/OS UNIX System Services Planning*, GA22-7800
-
- *z/OS TSO/E Customization*, SA22-7783
- *z/OS TSO/E Programming Services*, SA22-7789
- *z/OS TSO/E System Programming Command Reference*, SA22-7793

***z/OS.e***

- *z/OS.e Overview*, GA22-7869
- *z/OS.e Licensed Program Specifications*, GA22-7868

## Softcopy Publications

*z/OS Collection*, SK3T-4269

# Index

## Special characters


__abend 128
__alloc 128
__amrc 128
__code 128
__error 128
__feedback 128
__last_op 128
__msg 129
_BPXK_MDUMP 79


## A


abend codes
  >= 4000 32
  < 4000 32
  passing to operating system 24
  system, example of 34
  user-specified, example of 34
  user, example of 34
  using 34
abends
  internal, table of output 262, 263
  Language Environment 34, 259
  requested by assembler user exit 24
  system 34
  under CICS 259
  user 34
ABPERC run-time option
  function 9
  generating a system dump and 77
  modifying condition handling behavior and 22
ABTERMENC run-time option 9, 25
  using 25
accessibility 277
AGGREGATE compiler option 4, 8
anywhere heap
  statistics 20
APAR (Authorized Program Analysis Report) 274
  documentation 275
application programs
  debugging
    handling a core dump written to a BFS file 183
    handling a core dump written to an HFS file 183
argument
  in dump 62
arguments, registers, and variables for active routines 60
assembler language
  user exit 24, 25
    for CICS 262
    generating a system dump with 77
    modifying condition handling behavior and 25
    using 24, 25
atexit
  information in dump 157
Authorized Program Analysis Report (APAR) 274
automatic variables
  locating in dump 244


## B


base locator
  for working storage 198
  in dump 198
below heap
  statistics 20
BLOCKS option of CEE3DMP callable service 39


## C


C return codes to CICS 259
C-CAA (C-specific common anchor area)
  *See* C/C++, C-specific common anchor area (C-CAA)
C/C++
  __amrc
    example of structure 128
    information in dump 159
  __msg 129
  atexit
    information in dump 157
  C-specific common anchor area (C-CAA) 157
  cdump() function 145
  compiler listings 135
    IPA link step listing 137
  compiler options 3
  debugging examples 172, 179
  dump
    fetch information in 157
    parameter in 141
    signal information in 156
    structure variables, locating in 143
    system, structures in 143
  file
    control block information 158
    status and attributes in dump 158
  functions
    calling dump, example 145
    cdump() 45, 145
    csnap() 45, 145, 146
    ctrace() 45, 145, 146
    fetch() 127
    fopen() 129
    perror() 127
    printf() 128
    to produce dump output 45
  memory file control block 158
  perror() function 133
  stdio.h 128
  timestamp 145
CAA (common anchor area) 64
call chain 62
CALL statement
  CDUMP/CPDUMP 217


© Copyright IBM Corp. 1991, 2004

CALL statement *(continued)*
DUMP/PDUMP 216
SDUMP 218
callable services 21
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 27
case 2 condition token 27
cdump() 145
CEE prefix 31, 33
CEE3ABD—terminate enclave with an abend 21, 34, 77
generating a dump and 77
handling user abends and 21, 34
modifying condition handling behavior and 21
CEE3DMP—generate dump 37, 62
*See also* Language Environment dump
generating a Language Environment dump with 37
options 38
relationship to PLIDUMP 239
syntax 38
CEE3GRO—returns location offset 22
CEE3SRP—set resume point 22
CEEBXITA assembler user exit 24
CEECXITA assembler user exit 262
CEEDCOD—decompose a condition token 27
CEEDUMP — Language Environment Dump Service
*See also* Language Environment dump
control blocks 97
locating 117
CEEHDLR—register user condition handler 23
CEEMGET—get a message 27
CEEMOUT—dispatch a message 25
CEEMRCE—move resume cursor to designated label 21
CEEMRCR—move resume cursor relative to handle cursor 21
CEEMSG—get, format, and dispatch a message 27
CEESGL—signal a condition 27
CEESTART 127
character
data dump 217
CHECK run-time option
function 9
modifying condition handling behavior and 22
CHECKOUT compiler option 4
CICS
abends 259
application, from an EXEC CICS command 263
debugging for 257
debugging information, table of locations 257
destination control table (DCT) 257
example traceback in CESE transient data queue 257
examples of output 257
nonzero reason code returned, table of output 262
reason codes 259
register and program status word contents 259
return codes
Language Environment 259
run-time messages 257
transaction
dump 258
rollback 262
class test 190
classifying errors table 31
CLLE (COBOL load list entry) 198
COBCOM control block 201
COBOL
base locator for working storage 198
compiler options 6
debugging examples 201, 213
dump
external data in 198
file information in 198
linkage section in 198
local variables in 195
routine information in 195
run unit storage in 199
stack frames for active routines in 195
working storage in 198
errors 189
listings 193
memory file control block 157
program class storage 198
return codes to CICS 259
routine
calling Language Environment dump service 193
COBVEC control block 201
command
syntax diagrams xvii
COMMAREA (Communication Area) 258
compiler options
C 3

compiler options *(continued)*
COBOL 6
Fortran 7
PL/I 7
Compiler options map 136
condition
information
for active routines 60
in dump 60
POSIX 27
unhandled 28
condition handling
behavior, modifying 21
user-written condition handler 21, 23
condition information block 62, 71
condition manager 28
CONDITION option of CEE3DMP callable service 40, 62
condition token 27
case 1 27
case 2 27
example of 28
conditions, nested 28
control block
for active routines 60
core dump
written to an HFS file 183
Cross-Reference listing 136
csnap() 146
ctrace() 146

# D

data
map listing 193
values 62
DCB (data control block) 198
DCT (destination control table) 257
DEBUG run-time option 9
debugging
C, examples 172, 179
COBOL, examples 201, 213
for CICS 257
Fortran, examples 223, 229
PL/I, examples 247, 252
tool 37
DEPTHCONDLMT run-time option
function 9
modifying condition handling behavior and 22
wait/loop error and 32
diagnosis checklist 267
disability 277
DISPLAY statement 26, 189
documents, licensed xix
DSA (dynamic save area) 62
*See* stack, frame
dummy DSA 62
dump
an area of storage 217
core
written to an HFS file 183
dump *(continued)*
date in 58
dynamically allocated storage in 61
symbolic 218
DUMP suboption of TERMTHDACT run-time option 41
DUMP/PDUMP routine 216
format specifications 216
output 216
usage considerations 216

# E

ECB (enclave control block) 61
EDB (enclave data block) 61
EDC prefix 31, 33
EIB (exec interface block) 258
enclave
identifier in dump 58
member list 61, 62
storage 61
termination
behavior, establishing 25
entry information 58
ENTRY option of CEE3DMP callable service 40
entry point
name of active routines in dump 59
ERRCOUNT run-time option
function 9
modifying condition handling behavior and 22
wait/loop error and 32
errno 157
error
determining source of 267
message while Language Environment was handling another error 28
unanticipated 31
ESD compiler option 8
examples
application abends from 263
C routines 172, 179
calling a nonexistent subroutine 176, 204, 250
COBOL routines 201, 213
divide-by-zero error 172, 207, 252
output under CICS 257
PL/I routines 247, 252
SUBSCRIPTANGE error 201, 247
EXEC CICS DUMP statements 258
external data
for COBOL programs in dump 198
External symbol cross reference listing 136

# F

fetch
fetch information in dump 157
fetch() 127
fetchable module 127
file
for COBOL, in dump 198
status key 189
file control block (FCB) 158

FILES option of CEE3DMP callable service 39
FLAG compiler option 4
floating point registers
in dump 58
fopen() 129
FOR prefix 31, 33
Fortran
compiler options 7
debugging examples 223, 229
dump services 215
errors, determining the source of 213
listings 215

## G

general purpose registers 58
Global symbols map 136
GMAREA 198
GONUMBER compiler option 4

## H

HANDLE ABEND EXEC CICS command 257
header files, C
ctest.h 145
errno.h 172
stdio.h 128
stdlib.h 172
heap storage
created by CEECRHP callable service 21
in LEDATA Output 98
reports 102
storage in dump 61
user 18
HEAPCHK run-time option
function 9, 103
HEAPPOOL Storage
statistics 185
user-created, _uheapreport 186

## I

I/O
conventions 127
IBM prefix 31, 33
IGZ prefix 31, 33
in dump 58
INFOMSGFILTER run-time option
function 9
INITIALIZE statement 190
inline
report 136
Inline report for IPA 136
instruction length counter in dump 60
interactive problem control system (IPCS)
analyzing a core dump 183
INTERRUPT compiler option
function 8
INTERRUPT run-time option 9
interruption code in dump 60
ITBLK in dump 201

## K

keyboard 277

## L

language constructs 189
Language Environment (Language Environment/370)
return codes to CICS 259
symbolic feedback code 27
Language Environment dump
C information in 156
CEEDUMP 37
COBOL information in 197
default options 40
example traceback in 62
Fortran information in 221
multiple enclaves and 76
options
BLOCKS 39
CONDITION 40, 62
ENCLAVE 38
ENTRY 40
FILES 39
FNAME 40
NOBLOCKS 39
NOCONDITION 40
NOENTRY 40
NOFILES 39
NOSTORAGE 39
NOTRACEBACK 39
NOVARIABLES 39
PAGESIZE(n) 40
STACKFRAME 39
STORAGE 39
THREAD 39
TRACEBACK 39, 62
VARIABLES 39, 62
output
for C routines 150
for COBOL program 193
for Fortran routines 221
for PL/I routines 239
information for multiple enclaves 45
PL/I information in 241
section descriptions 57
TERMTHDACT suboptions 42
title 58
traceback with condition information
C routine 150
COBOL program 195
Language Environment routine 57
PL/I routine 239
using C functions 45
using CDUMP/CPDUMP subroutine 215
using CEE3DMP callable service 37, 57
using DUMP/PDUMP subroutine 215
using PLIDUMP subroutine 45, 239
using SDUMP subroutine 215
using TERMTHDACT run-time option 41

Language Environment/370 (Language Environment)
*See* Language Environment (Language Environment/370)
LEDATA
IPCS Verbexit 80
C/C++ Output 104
COBOL Output 110
Format 81
Parameters 81
Understanding Output 84
licensed documents xix
linkage editor
module map 136
linkage section
for COBOL programs in dump 198
LIST compiler option 4, 6, 8
listings generated by compiler
C 135
COBOL 193
Fortran 215
PL/I 232
LMESSAGE compiler option 8
local
variables 60
LookAt message retrieval tool xx

# M

machine state information
in dump 60
MAP compiler option 6, 8
memory file control block (MFCB) 157, 158
message
classifying errors and 32
run-time, CICS 257
user-created 25
using in your routine 25
message retrieval tool, LookAt xx
module
fetchable 127
module name prefixes, Language Environment 31
MSG suboption
of TERMTHDACT 41
MSGFILE run-time option
function 9
run-time messages and 32
MSGQ run-time option 10

# N

nested condition 28
no response (wait/loop) 32
NOBLOCKS option of CEE3DMP callable service 39
NOCONDITION option of CEE3DMP callable service 40
NOENTRY option of CEE3DMP callable service 40
NOFILES option of CEE3DMP callable service 39
NOSTORAGE option of CEE3DMP callable service 39
Notices 279
NOTRACEBACK option of CEE3DMP callable service 39
NOVARIABLES option of CEE3DMP callable service 39

# O

Object file map 136
OFFSET compiler option 4, 6, 8
optimizing
C 3, 62
COBOL 6
PL/I 8
options
C compiler 3
COBOL compiler 6
defaults for dump 42
determining run-time in effect 10, 12
Fortran compiler 7
Language Environment run-time 9
PL/I compiler 8
OUTDD compiler option 6
output
incorrect 32
missing 32

# P

page
number in dump 58
PAGESIZE(n) option of CEE3DMP callable service 40
parameter
checking value of 24
perror() function 133
PL/I
address of interrupt, finding in dump 243
CAA address, finding in dump 247
common anchor area (CAA) 247
compiler listings
object code listing 236
static storage map 235
variable storage map 236
compiler options
generating listings with 232
list of 7
CSECT 235
debugging examples 247, 252
dump
error type, finding in 243
parameter list, finding contents in 245
PL/I information, finding in 241, 245
PLIDUMP subroutine and 239
statement number, finding in 243
timestamp, finding in 245
variables, finding in 244
ERROR ON-unit 230, 243
errors 229, 232
floating-point register 230
object code listing 236
ON statement control block 236
static storage listing 235
SUBSCRIPTRANGE condition 230, 231, 249
PLIDUMP subroutine 239

PMR (Problem Management Record) 274
pointer
  variable 127
PPA 269
Prelinker map 136
preventive service planning (PSP) bucket 267
printf() function 26, 128
Problem Management Record (PMR) 274
procedure division listings 193
process
  control block 62
  member list 62
process control block (PCB) 62
PROFILE run-time option
  function 10
program
  class storage 198
program prolog area 269
program status word (PSW) 60
pseudo-assembler listing 135
PSP (preventive service planning) bucket 267

# Q

QUIET suboption of TERMTHDACT run-time option 41

# R

reason code
  nonzero returned to CICS 262
  under CICS 259
registers 0–15
  in dump 58
release number
  in dump 58
return code
  bad or nonzero 32
RPTOPTS run-time option 10
RPTSTG run-time option 12
run unit
  COBOL 199
  level control block 199
  storage in dump 61, 199
run-time
  messages
    under CICS 257
run-time options 22
  determining those in effect 10, 12
  sample options report 10
  specifying 24

# S

scope
  terminator 189
SDUMP routine
  description 218
  format specifications 218
  output 218
  usage considerations 218
service routines
  CDUMP/CPDUMP 217
  DUMP/PDUMP 216
  SDUMP 218
SET statement 190
shortcut keys 277
signal information in dump 156
sorted cross-reference listing 193
SOURCE compiler option 4, 6, 8
Source file map 136
source listing 136
stack
  frame 62
STACKFRAME option of CEE3DMP callable service 39
statement
  numbers
    in dump 58
static
  variables in dump 244
  writable map 139, 140, 144
status
  of routines in dump 59
stderr 26
stdio.h 128
stdout 26
storage
  evaluating use of 12
  for active routines 61
  leak detecting 61, 103
  offset listing 135
  report 12
  statistics 18, 20
STORAGE compiler option 8
STORAGE option of CEE3DMP callable service 39
STORAGE run-time option 10
structure
  map 135, 143
  variable example code 143
symbolic
  feedback code 27
symbolic dumps 218
  how to call under FORTRAN 218
syntax diagrams
  how to read xvii
system
  abend
    with TRAP(OFF) 32
    with TRAP(ON) 32
system dump
  generating 77
    in z/OS UNIX shell 79

# T

task global table (TGT) 198
TERMINAL compiler option 4, 8
TERMTHDACT run-time option
  function 9, 41, 250
  generating a dump and 23
  modifying condition handling behavior and 10

TERMTHDACT run-time option *(continued)*
suboptions 41
TEST compiler option 4, 6, 7
TEST run-time option 10
text file name prefixes, Language Environment 31
THDCOM in dump 201
THREAD option of CEE3DMP callable service 39
time
in dump 58
TRACE run-time option
function 10
trace table 118
TRACE suboption of TERMTHDACT run-time option 41
TRACEBACK option of CEE3DMP callable service 39, 62
transaction
dump 258
rollback 262
rollback effects of assembler user exit on 262
work area 258
TRAP run-time option
function 10
Language Environment condition handling and 23, 77
user abends and 34

# U

UADUMP suboption of TERMTHDACT run-time option 23, 42
UAIMM suboption of TERMTHDACT run-time option 23, 42
UAONLY suboption of TERMTHDACT run-time option 23, 41
UATRACE suboption of TERMTHDACT run-time option 23, 41
unhandled conditions 25, 28
establishing enclave termination behavior for 25
USE EXCEPTION/ERROR declaratives 190
USE FOR DEBUGGING declarative 190, 191
user
abend 34
code 32
exit 24, 25
heap
statistics 20
stack
statistics 18
user-specified abends 34
USRHDLR run-time option 9, 23
utility and service subroutines
CDUMP/CPDUMP 217
DUMP/PDUMP 216
SDUMP 218

# V

variables
in Language Environment dump 62
structure example code 143
VARIABLES option of CEE3DMP callable service 39, 62
VBREF compiler option 6
verb cross-reference 193
verbexit
LEDATA 80
version number
in dump 58

# W

working storage
in dump 61, 198

# X

XPLINK
downward-growing stack 63
finding XPLINK information in a dump 166
storage stastics 19
trace table entries for 167
XREF compiler option 6, 8
XUFLOW run-time option
function 10
modifying condition handling behavior and 23

# Z

z/OS UNIX System Services
C application program and 183
generating a system dump 79

# Readers' Comments — We'd Like to Hear from You

**z/OS**
**Language Environment**
**Debugging Guide**

**Publication No. GA22-7560-04**

**Overall, how satisfied are you with the information in this book?**

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Overall satisfaction | ☐ | ☐ | ☐ | ☐ | ☐ |

**How satisfied are you that the information in this book is:**

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Accurate | ☐ | ☐ | ☐ | ☐ | ☐ |
| Complete | ☐ | ☐ | ☐ | ☐ | ☐ |
| Easy to find | ☐ | ☐ | ☐ | ☐ | ☐ |
| Easy to understand | ☐ | ☐ | ☐ | ☐ | ☐ |
| Well organized | ☐ | ☐ | ☐ | ☐ | ☐ |
| Applicable to your tasks | ☐ | ☐ | ☐ | ☐ | ☐ |

**Please tell us how we can improve this book:**

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

**Readers' Comments — We'd Like to Hear from You**
GA22-7560-04

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape



Cut or Fold
Along Line

Program Number: 5694-A01, 5655-G52

Printed in USA

GA22-7560-04