Contents

Figures ........................................ ix

Tables .......................................... xiii

Abstract for MVS Diagnosis: Tools and Service Aids ............... xv

Who must use this information .................................... xv
Where to find more information .................................... xv

How to send your comments to IBM ................................. xvii
If you have a technical problem ................................... xvii

Summary of changes ........................................... xix

Summary of changes for z/OS MVS Diagnosis:
Tools and Service Aids for Version 2 Release 2
(V2R2) as updated September 2016 ........................ xix
General content changes for z/OS MVS
Diagnosis: Tools and Service Aids .................. xix
Summary of changes for z/OS MVS Diagnosis:
Tools and Service Aids for Version 2 Release 2
(V2R2) as updated December 2015 ........... xx
General content changes for z/OS MVS
Diagnosis: Tools and Service Aids ........... xx
Summary of changes for z/OS MVS Diagnosis:
Tools and Service Aids for Version 2 Release 1
(V2R1) as updated February 2015 ............ xx
Summary of changes for z/OS Version 2 Release 1
(V2R1) as updated March 2014 ................ xxi
z/OS Version 2 Release 1 summary of changes .... xxi

Chapter 1. Selecting tools and service aids ........................... 1
How do I know which tool or service aid to select? ........ 1
What tools and service aids are available? ................ 2

Chapter 2. SVC dump ........................................... 7
Planning data set management for SVC dumps .......... 8
Using automatically allocated dump data sets ........ 8
Using pre-allocated dump data sets ................ 13
Choosing SVC dump data sets ........................ 15
Finding automatically allocated dump data sets .... 16
Communication from the system ...................... 16
Specifying SYS1.DUMPxx data sets ............... 16
Controlling SYS1.DUMPxx data sets ............. 17
Obtaining an SVC dump .................................. 17
Issuing a macro for SVC dump ...................... 19
Operator activities .................................. 19
Making a dump data set available .................. 21
Determining current SVC dump options and status .... 22
Finding SVC dumps .................................. 22
Printing, viewing, copying, and clearing a pre-allocated or SYS1.DUMPxx data set .... 24
Contents of SVC dumps .................................. 25
Customizing SVC dump contents ................... 26
Tailoring SVC dumps ................................ 35
Analyzing summary SVC dumps .................... 35
SUMDUMP output for SVC-Entry SDUMPX .... 37
SUMDUMP output for branch-entry SDUMPX .... 37
Analyzing disabled summary dumps .............. 38
Analyzing suspend summary dumps .............. 38
Analyzing an SVC dump ............................. 39
Specifying the source of the dump .................. 39
Formatting the SVC dump header .................. 40
Looking at the dump title ......................... 41
Displaying the incident token, time and type of dump .... 42
Locating error information ......................... 43
Analyze TCB structure ................................ 46
Examining the LOGREC buffer .................. 48
Examining the system trace .................. 50
Looking at the registers ......................... 50
Other useful reports for SVC dump analysis .... 52
Reading the SDUMPX 4K SQA buffer ........ 53

Chapter 3. Transaction dump ..................................... 55
Planning data sets for transaction dumps ........ 55
Planning data set management for transaction dumps .......... 55
Using preallocated dump data sets .............. 56
Setting up allocation authority .................. 56
Choices for IEATDUMP Data Sets .............. 56
Obtaining transaction dumps .................... 58
Printing, viewing, copying, and clearing a dump data set .... 59
Contents of transaction dumps ............... 59
Customizing transaction dump contents .......... 59

Chapter 4. Stand-alone dump .................................... 65
Planning for stand-alone dump .................. 66
Should I take a stand-alone dump to DASD or to tape? ........ 66
Can I use my current version of the stand-alone dump program to dump a new version of z/OS? .... 70
Creating the stand-alone dump program .......... 70
MNOTES from the AMDSADMP macro .......... 70
Coding the AMDSADMP macro .................. 75
Using the AMDSADDD utility ................. 88
Generating the stand-alone dump program ........ 96
One-stage generation ......................... 96
Two-stage generation ......................... 103
Running the stand-alone dump program ........ 107
Procedure A: Initialize and run stand-alone dump .......... 108
Procedure B: Restart stand-alone dump .......... 112
Procedure C: ReIPL stand-alone dump .......... 113
Procedure D: Dump the stand-alone dump program .......... 113
| Starting GTF to trace VTAM remote network activity | 244 |
| Stopping GTF | 245 |
| GTF trace options | 247 |
| Combining GTF options | 252 |
| Examples of sample prompting sequences | 258 |
| Receiving GTF trace | 260 |
| Combining, extracting, and merging GTF trace output | 261 |
| Merging trace output | 262 |
| Reading GTF output | 263 |
| Formatted GTF trace output | 264 |
| Trace record identifiers | 265 |
| Example of formatted GTF trace output | 267 |
| Formatted trace records for events | 269 |
| Time stamp records | 269 |
| Source index records | 269 |
| Lost event records | 270 |
| ADINT trace records | 270 |
| CCW trace records | 271 |
| CSCH and HSCH trace records | 273 |
| DSP and SDSP trace records | 274 |
| EOS, INTG, IO, IOCS, and PCI trace records | 276 |
| EXT trace records | 280 |
| FRR trace records | 282 |
| HEXFORMAT, SUBSYS, and SYSTEM trace records | 283 |
| IOX trace records | 284 |
| LSR trace records | 287 |
| MSCH trace records | 288 |
| PCIDMX trace records | 289 |
| PCILG trace records | 290 |
| PCISTG trace records | 291 |
| PGM and PI trace records | 292 |
| RNIO trace records | 293 |
| RSCH trace records | 294 |
| SLIP trace records | 295 |
| SLIP standard trace record | 296 |
| SLIP standard/user trace record | 299 |
| SLIP user trace record | 299 |
| SLIP debug trace record | 300 |
| SRB trace records | 301 |
| SRM trace records | 302 |
| SSCH trace records | 303 |
| STAE trace records | 304 |
| SVC and SVCR trace records | 306 |
| TCW trace records | 308 |
| USR trace records | 308 |
| Unformatted USR trace record | 312 |
| Formatted USR trace record | 313 |
| USRF9 trace record for VSAM | 313 |
| USRFD trace record for VTAM | 314 |
| USRFE trace record for BSAM, QSAM, BPAM, and BDAM | 314 |
| USRFF trace record for open/close/EOV abnormal end | 315 |
| USRFF trace record for user requested work area | 316 |
| XSCH trace record | 317 |
| Event Identifiers (EIDs) for USR trace records | 317 |
| Format Identifiers (FIDs) for USR trace records | 319 |
| Unformatted GTF trace output | 320 |
| Control records | 321 |
| Unformatted lost event records | 322 |
| User data records | 323 |
| System data records | 324 |
| Unformatted trace records for events | 325 |

### Chapter 11. The generic tracker facility

- GTZLQRY REXX callable function | 346 |
- GTZLQRY "STATUS" interface | 346 |
- GTZLQRY "TRACKDATA" interface | 351 |
- GTZLQRY "EXCLUDE" interface | 360 |
- GTZLQRY "DEBUG" interface | 364 |
- Data persistence and simplified generic tracking services | 369 |
- DFSMS tracking | 371 |
- MVS Allocation tracking | 372 |
- JES3 control statement tracking | 372 |

### Chapter 12. Component trace

Planning for component tracing | 377 |
- Create CTnccxxx parmlib members for some components | 377 |
- Select the trace options for the component trace | 383 |
- Decide where to collect the trace records | 384 |
- Obtaining a component trace | 385 |
- Request component tracing to address space or data space trace buffers | 385 |
- Request writing component trace data to trace data sets | 388 |
- Create a parmlib member | 391 |
- Request component tracing for systems in a sysplex | 393 |
- Verifying component tracing | 395 |
- Verify that the writer is active | 397 |
- Viewing the component trace data | 398 |
- SYSPC component trace | 399 |
- Requesting a SYSPC trace | 400 |
- Formatting a SYSPC trace | 403 |
- Output from a SYSPC Trace | 407 |
- FMH-5 trace data | 409 |
- SYXAR component trace | 413 |
- Requesting a SYXAR trace | 413 |
- Formatting a SYXAR trace | 415 |
- Output from a SYXAR Variables Trace | 415 |
- SYSBCPII component trace | 417 |
- Requesting a SYSBCPII trace | 417 |
- Formatting a SYSBCPII trace | 419 |
- Output from a SYSBCPII Trace | 419 |
- SYSHI component trace | 420 |
- Requesting a SYSHI trace | 421 |
- Formatting a SYSHI trace | 423 |
- Output from a SYSHI Trace | 423 |
- SYSCA component trace | 425 |
- Requesting a SYSCA trace | 425 |
- Formatting a SYSCA trace | 427 |
- Output from a SYSCA Trace | 427 |
- SYSDLF component trace | 427 |
- Requesting a SYSDLF trace | 428 |
## Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default name pattern for automatically allocated dump data set</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Example: verifying dump status</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Example of dump status</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Example: Output from DISPLAY,DUMP ERRDATA command</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>Example: JCL to print, copy, and clear an SVC dump data set</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>Example: Format of IPCS VERBx SUMDUMP command</td>
<td>36</td>
</tr>
<tr>
<td>7</td>
<td>Example: IPCS VERBx SUMDUMP command</td>
<td>36</td>
</tr>
<tr>
<td>8</td>
<td>Example: Examining storage</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>IPCS Default Values menu</td>
<td>40</td>
</tr>
<tr>
<td>10</td>
<td>IPCS primary option menu</td>
<td>41</td>
</tr>
<tr>
<td>11</td>
<td>IPCS MVS analysis of dump contents menu</td>
<td>41</td>
</tr>
<tr>
<td>12</td>
<td>STATUS WORKSHEET subcommand sample output — dump title</td>
<td>42</td>
</tr>
<tr>
<td>13</td>
<td>Sample output from the STATUS SYSTEM subcommand</td>
<td>43</td>
</tr>
<tr>
<td>14</td>
<td>IPCS Subcommand Entry menu</td>
<td>43</td>
</tr>
<tr>
<td>15</td>
<td>Search argument abstract in the STATUS FAILDATA report</td>
<td>44</td>
</tr>
<tr>
<td>16</td>
<td>System mode information in the STATUS FAILDATA report</td>
<td>44</td>
</tr>
<tr>
<td>17</td>
<td>Time of error information in the STATUS FAILDATA report</td>
<td>45</td>
</tr>
<tr>
<td>18</td>
<td>Example: the SUMMARY TCBERROR report</td>
<td>47</td>
</tr>
<tr>
<td>19</td>
<td>Example: output from the IPCS subcommand SYSTRACE</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>Sample of the STATUS REGISTERS report</td>
<td>51</td>
</tr>
<tr>
<td>21</td>
<td>Sample of the STATUS REGISTERS report run in z/Architecture mode</td>
<td>52</td>
</tr>
<tr>
<td>22</td>
<td>SPFUSER name pattern for automatically allocated dump data set</td>
<td>58</td>
</tr>
<tr>
<td>23</td>
<td>Example: JCL to Print, Copy, and Clear the Dump Data Set</td>
<td>59</td>
</tr>
<tr>
<td>24</td>
<td>Format of AMDSADMP Macro Instruction</td>
<td>76</td>
</tr>
<tr>
<td>25</td>
<td>Example: Accepting All Defaults</td>
<td>81</td>
</tr>
<tr>
<td>26</td>
<td>Example: Generating an unformatted, tape resident dump program</td>
<td>82</td>
</tr>
<tr>
<td>27</td>
<td>Example: Generating a dump program with output to DASD</td>
<td>82</td>
</tr>
<tr>
<td>28</td>
<td>Example: Generating a dump program with output to DASD</td>
<td>82</td>
</tr>
<tr>
<td>29</td>
<td>Sample Console output from the stand-alone Dump Program</td>
<td>84</td>
</tr>
<tr>
<td>30</td>
<td>Example of valid specifications</td>
<td>87</td>
</tr>
<tr>
<td>31</td>
<td>Using AMDSADDD to Allocate and Initialize a Dump Data Set</td>
<td>93</td>
</tr>
<tr>
<td>32</td>
<td>Using AMDSADDD to Allocate and Initialize an Extended Dump Data Set</td>
<td>94</td>
</tr>
<tr>
<td>33</td>
<td>Using AMDSADDD to Clear an Existing Dump Data Set</td>
<td>94</td>
</tr>
<tr>
<td>34</td>
<td>Using AMDSADDD to Reallocate the Dump Data Set</td>
<td>95</td>
</tr>
<tr>
<td>35</td>
<td>Example: Using JCL to allocate and initialize a dump data set</td>
<td>95</td>
</tr>
<tr>
<td>36</td>
<td>Example: Using JCL to allocate and initialize an extended format dump data set</td>
<td>96</td>
</tr>
<tr>
<td>37</td>
<td>Example: One-stage generation</td>
<td>96</td>
</tr>
<tr>
<td>38</td>
<td>Example: One-stage generation of stand-alone dump to a DASD</td>
<td>98</td>
</tr>
<tr>
<td>39</td>
<td>Example: One-stage generation of stand-alone dump to tape</td>
<td>99</td>
</tr>
<tr>
<td>40</td>
<td>One-stage generation JCL for a DASD (beginning with z/OS V1R12)</td>
<td>101</td>
</tr>
<tr>
<td>41</td>
<td>Example: One-stage generation JCL (any release) for a DASD (any release)</td>
<td>102</td>
</tr>
<tr>
<td>42</td>
<td>Example: One-stage JCL (beginning with z/OS V1R12) for tape</td>
<td>103</td>
</tr>
<tr>
<td>43</td>
<td>Example: One-stage JCL (any release) for tape</td>
<td>103</td>
</tr>
<tr>
<td>44</td>
<td>Example: Stage-two JCL to assemble the AMDSADMP macro</td>
<td>104</td>
</tr>
<tr>
<td>45</td>
<td>Example: Assembling multiple versions of AMDSADMP Macro</td>
<td>104</td>
</tr>
<tr>
<td>46</td>
<td>Example: Stage-two JCL to assemble the AMDSADMP macro</td>
<td>106</td>
</tr>
<tr>
<td>47</td>
<td>Example: Stage-two JCL to assemble the AMDSADMP macro</td>
<td>106</td>
</tr>
<tr>
<td>48</td>
<td>Example: Stage-Two JCL to assemble the AMDSADMP macro with overrides</td>
<td>107</td>
</tr>
<tr>
<td>49</td>
<td>Example: Using a load parm to perform a stand-alone dump</td>
<td>111</td>
</tr>
<tr>
<td>50</td>
<td>Example: Terminating a stand-alone dump</td>
<td>116</td>
</tr>
<tr>
<td>51</td>
<td>Example: Copying stand-alone dump output from tape to DASD</td>
<td>117</td>
</tr>
<tr>
<td>52</td>
<td>Example: Copying stand-alone dump output from DASD to DASD</td>
<td>117</td>
</tr>
<tr>
<td>53</td>
<td>Example: Copying a stand-alone dump from multiple DASD data sets</td>
<td>118</td>
</tr>
<tr>
<td>54</td>
<td>Example: Copying stand-alone dump output from DASD and tape</td>
<td>118</td>
</tr>
<tr>
<td>55</td>
<td>Example: Printing an unformatted stand-alone dump</td>
<td>119</td>
</tr>
<tr>
<td>56</td>
<td>Example: VERBEXIT SYMPTOMS output</td>
<td>122</td>
</tr>
<tr>
<td>57</td>
<td>Example: STATUS WORKSHEET output</td>
<td>123</td>
</tr>
<tr>
<td>58</td>
<td>Example: STATUS CPU REGISTERS output</td>
<td>124</td>
</tr>
<tr>
<td>59</td>
<td>Example: WHERE subcommand output</td>
<td>124</td>
</tr>
<tr>
<td>60</td>
<td>Example: IOSCHECK ACTVUCBS Subcommand output</td>
<td>125</td>
</tr>
<tr>
<td>61</td>
<td>Example: ANALYZE subcommand output</td>
<td>125</td>
</tr>
<tr>
<td>62</td>
<td>Example: RSMDATA output</td>
<td>125</td>
</tr>
<tr>
<td>63</td>
<td>Example: ASMCHECK output</td>
<td>126</td>
</tr>
<tr>
<td>64</td>
<td>Example: SUMMARY FORMAT output (determining ready work)</td>
<td>127</td>
</tr>
<tr>
<td>65</td>
<td>Example: SUMMARY FORMAT output (determining TCB in normal wait)</td>
<td>128</td>
</tr>
<tr>
<td>66</td>
<td>Example: SYSTRACE output (recognizing an enabled loop)</td>
<td>129</td>
</tr>
<tr>
<td>67</td>
<td>Example: Using IEBPTPC to print a dump</td>
<td>141</td>
</tr>
<tr>
<td>68</td>
<td>Example: Recognizing a pattern</td>
<td>164</td>
</tr>
<tr>
<td>Example: Entering SPZAP control statements through console.</td>
<td>657</td>
<td></td>
</tr>
<tr>
<td>Example: Using the DUMP control statement with a class name</td>
<td>658</td>
<td></td>
</tr>
<tr>
<td>Sample formatted hexadecimal dump</td>
<td>663</td>
<td></td>
</tr>
<tr>
<td>Sample translated dump</td>
<td>664</td>
<td></td>
</tr>
<tr>
<td>Sample formatted hexadecimal dump for PDSE program object module</td>
<td>665</td>
<td></td>
</tr>
<tr>
<td>Sample translated dump for PDSE data library</td>
<td>666</td>
<td></td>
</tr>
<tr>
<td>Example: AMATERSE JCL</td>
<td>668</td>
<td></td>
</tr>
<tr>
<td>Example: AMATERSE JCL from a problem program</td>
<td>670</td>
<td></td>
</tr>
<tr>
<td>Simple FTP connection</td>
<td>679</td>
<td></td>
</tr>
<tr>
<td>FTP connection using a proxy server</td>
<td>679</td>
<td></td>
</tr>
<tr>
<td>FTP with a proxy user ID</td>
<td>680</td>
<td></td>
</tr>
<tr>
<td>FTP using the FTPCMDS DD statement</td>
<td>680</td>
<td></td>
</tr>
<tr>
<td>FTPCMDS data set example</td>
<td>680</td>
<td></td>
</tr>
<tr>
<td>FTP specifying port 2121 on TARGET_SYS</td>
<td>681</td>
<td></td>
</tr>
<tr>
<td>FTP forcing PASSIVE mode</td>
<td>681</td>
<td></td>
</tr>
<tr>
<td>FTP using a userid.NETRC data set</td>
<td>682</td>
<td></td>
</tr>
<tr>
<td>FTP connection with the DEBUG DD statement</td>
<td>682</td>
<td></td>
</tr>
<tr>
<td>Using SYSUT2 statement for allocating an unload data set</td>
<td>683</td>
<td></td>
</tr>
<tr>
<td>Using a multiple record control statement</td>
<td>683</td>
<td></td>
</tr>
<tr>
<td>Using the NO_FTP option</td>
<td>684</td>
<td></td>
</tr>
<tr>
<td>FTP specifying multiple statements in an inline FTPCMDS DD</td>
<td>684</td>
<td></td>
</tr>
<tr>
<td>Example: An ADYSETxx Member for a System in a Sysplex</td>
<td>692</td>
<td></td>
</tr>
<tr>
<td>Example: An ADYSETxx Member with GLOBAL</td>
<td>692</td>
<td></td>
</tr>
<tr>
<td>Example: DAE Data Set for Single System</td>
<td>693</td>
<td></td>
</tr>
<tr>
<td>Example: DAE Data Set Shared by Sysplex Systems</td>
<td>693</td>
<td></td>
</tr>
<tr>
<td>Example: Symptom Dump Output</td>
<td>703</td>
<td></td>
</tr>
<tr>
<td>Change LPAR Security panel for active LPAR</td>
<td>707</td>
<td></td>
</tr>
<tr>
<td>Activation profile security panel for new LPAR</td>
<td>708</td>
<td></td>
</tr>
</tbody>
</table>
Tables

1. Selecting a dump ........... 1
2. Selecting a trace ........... 1
3. Selecting a service aid ......... 2
4. Description of dumps ........... 3
5. Description of traces .......... 4
6. Description of service aids ........ 5
7. Sample operator DUMP command members in SYS1.SAMPLIB ........... 2
8. Customizing SVC dump contents through the SDATA parameter .......... 2
9. Affects on the CSA storage captured in an SVC dump ........... 2
10. Customizing SVC dump contents through summary dumps. ........... 3
11. Customizing SVC dump contents through operator commands ........... 3
12. Fields in SQA bugger ........... 3
13. Customizing transaction dump contents through the SDATA Parameter ........ 4
14. Customizing transaction dump contents through operator commands ......... 4
15. DDNAMES and defaults used by AMDSAOSG 97
16. AMDSAOSG return codes. ........... 99
17. Directing the output of assembly field ........... 104
18. Work area pointed to by the PSAWTCOD field ........... 129
19. Problem data saved by the SVC FLIH for task and SRB code ........... 130
20. Problem data saved for a program check for task and SRB code ........... 131
21. Problem data saved by the I/O FLIH for task and SRB code ........... 132
22. Problem data saved by the external FLIH for task and SRB code ........... 132
23. Types of ABEND dumps ........... 136
24. Summary: DD statements to specify for specific ABEND dumps ........... 137
25. Summary: dump contents by parameter ........... 143
26. Default contents of summary dumps in ABEND dumps ........... 146
27. Customizing SYSABEND dump contents ........... 148
28. Customizing SYSDUMP dump contents ........... 150
29. Customizing SYSDUMP dump contents ........... 152
30. Customizing dumps using through the SNAP or SNAPX macro ........... 160
31. Dumps that have TRT in their default options ........... 171
32. References for system trace entry format description ........... 173
33. RCVY trace events that require reentry ........... 194
34. Summary of dumps that contain master trace data ........... 223
35. How to locate master trace table from CVT ........... 225
36. Combining GTF options ........... 252
37. GTF trace options and corresponding prompting keywords ........... 252
38. CCW defaults for selected TRACE options ........... 254
39. Event identifiers and the types of events they represent ........... 258
40. Summary of trace record identifiers ........... 265
41. Event identifiers for USR trace records ........... 318
42. Format identifiers for USR trace records ........... 320
43. DSP trace record offset, size, and description ........... 326
44. Values for DSP minimal trace record ........... 326
45. EXT comprehensive trace record offset, size, and description ........... 327
46. CCW error codes ........... 331
47. Basic SVC comprehensive trace record ........... 339
48. REQUEST="STATUS" output variable names and descriptions ........... 347
49. REQUEST="STATUS" ABEND reason codes and descriptions ........... 350
50. REQUEST="TRACKDATA" output variable names and descriptions ........... 353
51. REQUEST="TRACKDATA" ABEND reason codes and descriptions ........... 359
52. REQUEST="EXCLUDE" output variable names and descriptions ........... 360
53. REQUEST="EXCLUDE" ABEND reason codes and descriptions ........... 364
54. REQUEST="DEBUG" output variable names and descriptions ........... 364
55. REQUEST="DEBUG" ABEND reason codes and descriptions ........... 369
56. Summary of BCP component traces that use the component trace service ........... 376
57. Determining if a component has a parmlib member ........... 377
58. Component trace options ........... 379
59. Location of trace buffers for components ........... 384
60. How to request SVC dumps for component traces ........... 386
61. Subcommands that format component trace records ........... 398
62. Requesting SYSAPPC component trace for APPC/MVS ........... 399
63. CTnAPPxx parameters ........... 400
64. Parameters allowed on TRACE CT ........... 400
65. Parameters allowed on REPLY ........... 401
66. Summary of the title prefixes and APPC/MVS subcomponents ........... 407
67. FMH-5 trace entries in the SYSAPPC component trace ........... 410
68. Summary of SYSLOGR component trace request ........... 470
69. Example: Using EREP parameters ........... 566
70. Program object and load module attributes ........... 592
71. PMR number variables for PDUU ........... 677
72. Return codes for z/OS Problem Documentation Upload Utility ........... 684
73. Summary of required symptoms ........... 688
74. Summary of optional symptoms ........... 688

© Copyright IBM Corp. 1988, 2016
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>75. VRADAE and VRANODAE keys on dump suppression when SUPPRESS and SUPPRESSALL keywords are specified in ADYSETxx</td>
<td>690</td>
</tr>
<tr>
<td>76. Examples of when DAE parameters may change</td>
<td>695</td>
</tr>
<tr>
<td>77. Summary of installation exit routines for dump suppression</td>
<td>698</td>
</tr>
<tr>
<td>78. Suppressing symptom dumps</td>
<td>705</td>
</tr>
</tbody>
</table>
Abstract for MVS Diagnosis: Tools and Service Aids

This information covers the tools and service aids that IBM® provides for use in diagnosing MVS™ problems.

The first chapter, Chapter 1, “Selecting tools and service aids,” on page 1, contains a guide on how to select the appropriate tool or service aid for your purposes. It also provides an overview of all the tools and service aids available.

Each subsequent chapter covers one of the tools or service aids. While topics vary, the following topics are typically covered for each tool or service aid:

- Customizing and planning information
- Starting and stopping the tool or service aid
- Receiving, formatting, and reading the output from the tool or service aid.

At the beginning of each chapter, there is a short editorial style comment that is intended to characterize the tool or service aid that is covered in the chapter.

Who must use this information

This information is for anyone who diagnoses software problems that occur on the operating system. This person is usually a system programmer for the installation. This information is also for application programmers who are testing their programs.

This information assumes that the reader:

- Understands basic system concepts and the use of system services
- Codes in Assembler language, and reads Assembler and linkage editor output
- Codes JCL statements for batch jobs and cataloged procedures
- Understands the commonly used diagnostic tasks and aids, such as message logs, dumps, and the interactive problem control system (IPCS)
- Understands how to search the problem reporting databases
- Understands the techniques for reporting problems to IBM

Where to find more information

Where necessary, this information references information in other documents, using shortened versions of the information title. For complete titles and order numbers of the documents for all products that are part of z/OS, see z/OS V2R2 Information Roadmap.
How to send your comments to IBM

We appreciate your input on this documentation. Please provide us with any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

Use one of the following methods to send your comments:

**Important:** If your comment regards a technical problem, see instead “If you have a technical problem.”

- Send an email to mhvrdfs@us.ibm.com
- Send an email from the “Contact us” web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html)

Include the following information:
- Your name and address
- Your email address
- Your phone or fax number
- The publication title and order number:
  - z/OS V2R2 MVS Diagnosis: Tools and Service Aids
  - GA32-0905-05
- The topic and page number or URL of the specific information to which your comment relates
- The text of your comment.

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

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one or more of the following actions:
- Visit the IBM Support Portal (support.ibm.com)
- Contact your IBM service representative.
- Call IBM technical support.
Summary of changes

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

Summary of changes for z/OS MVS Diagnosis: Tools and Service Aids for Version 2 Release 2 (V2R2) as updated September 2016

The following changes are made for z/OS® V2R2 as updated September 2016.

General content changes for z/OS MVS Diagnosis: Tools and Service Aids

The following content is new, changed, or no longer included in V2R2 as updated September 2016.

New

The following content is new.

- MGMTCLAS was added to the JCL statements for the Problem Documentation Upload Utility (PDUU). For more information, see "JCL statements for PDUU" on page 675.
- A new PDUU example was added. For more information, see "Example 12: Using a proxy server with multiple FTPCMDs DD statements" on page 684.
- The REAL parameter was added to the AMDSADMP macro. For more information, see "Syntax of the AMDSADMP macro" on page 75.

Changed

The following content is changed.

- All of the system trace entries in Chapter 8, “System trace,” on page 169 were updated with the latest format.
- The event identifiers for USR trace records were updated. For more information, see "Event Identifiers (EIDs) for USR trace records" on page 317.
- The information about naming a logrec log stream was updated. For more information, see "Defining a logrec log stream" on page 554.
- The information about the available DASD space needed for SADMP was updated. For more information, see "If I do dump to DASD, how much space do I need?" on page 66.

Summary of changes for z/OS MVS Diagnosis: Tools and Service Aids for Version 2 Release 2 (V2R2) as updated December 2015

The following changes are made for z/OS V2R2 as updated December 2015.

© Copyright IBM Corp. 1988, 2016
General content changes for z/OS MVS Diagnosis: Tools and Service Aids

The following content is new, changed, or no longer included in V2R2 as updated December 2015.

New

The following content is new.

- SYSDUMP component trace was added to the SDUMP component. For more information, see “SYSDUMP component trace” on page 432.
- CTIDMPxx parmlib member was added. For more information, see “Planning for component tracing” on page 377.

Summary of changes for z/OS MVS Diagnosis: Tools and Service Aids for Version 2 Release 2

The following changes are made for z/OS Version 2 Release 2 (V2R2).

New

- New section about JES3 control statement tracking in Chapter 11, “The generic tracker facility,” on page 343. For more information, see “JES3 control statement tracking” on page 372.
- New topic about changing the logrec recording medium. For more information, see “Changing the logrec recording medium” on page 556.
- New subsections for the GTZLQRY REXX callable function, data persistence, and simplified generic tracking services added to the generic tracker facility section. For more information, see Chapter 11, “The generic tracker facility,” on page 343.

Changed

- Information about recording logrec error records was updated for new functionality. For more information, see Chapter 15, “Recording logrec error records,” on page 551.
- An update to the RMODE field was added to the AMBLIST output for LISTLOAD OUTPUT=XREF. For more information, see “LISTLOAD OUTPUT=XREF output” on page 617.
- The SYSXES component trace information has been updated about the SYSXES trace buffer size. For more information, see “Prepare for specific component traces on systems in a sysplex” on page 393 and “CTnXSxx parmlib member” on page 533, and “Requesting a SYSXES trace” on page 533.

Summary of changes for z/OS Version 2 Release 1 (V2R1) as updated February 2015

The following changes are made for z/OS Version 2 Release 1 (V2R1) as updated February 2015. In this revision, all technical changes for z/OS V2R1 are indicated by a vertical line to the left of the change.

New

- The HCAS (HCSAByASID), HCNO (HCSANoOwner), and HCSY (HCSASysOwner) options were added to the SDATA parameter. For details, see “Customized contents using the SDATA parameter” on page 26.
Summary of changes for z/OS Version 2 Release 1 (V2R1) as updated March 2014

The following changes are made for z/OS Version 2 Release 1 (V2R1) as updated March 2014. In this revision, all technical changes for z/OS V2R1 are indicated by a vertical line to the left of the change.

New

• Various new and changed topics document new support for internal flash memory exploitation. The coupling facility can now migrate objects out to storage-class memory (SCM) when the number of objects exceeds a calculated threshold, then it can fetch the objects back into main CF storage when requested.

• Various new and changed topics document support for the new XCF Note Pad Service function. The XCF Note Pad Service is a new application programming interface that allows programs to manipulate notes in an XCF note pad. A note pad is an abstraction layered on top of the existing coupling facility list structure interfaces. You can use the new IXCNOTE macro to manipulate data in a coupling facility list structure, provided the note pad abstraction meets the needs of the application.

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

• z/OS Migration
• z/OS Planning for Installation
• z/OS Summary of Message and Interface Changes
• z/OS Introduction and Release Guide
Chapter 1. Selecting tools and service aids

This topic introduces the tools and service aids that MVS provides for diagnosis. For the purposes of this document, tools includes dumps and traces, while service aids include the other facilities provided for diagnosis. For example:

- SVC dump and system trace are tools
- Logrec data set and AMBLIST are service aids.

There are major two topics:

1. "How do I know which tool or service aid to select?" - This topic lists problem types and matches them with the appropriate service aid or the appropriate tool. Use this topic to select the tool or service aid you need for a particular problem.
2. "What tools and service aids are available?" on page 2 - This topic describes each tools and service aids, including when to use it for diagnosis. Use this topic when you need an overview of tools and service aids available or to find the appropriate time to use a particular tool or service aid.

How do I know which tool or service aid to select?

This topic contains tables that provide criterion for selecting a tool or service aid, depending on the problem or need. The tables show the problem or need, the corresponding tool or service aid, and the topic or document that covers it in complete detail. (Most of the detailed information on tools and service aids is in this document.) Use these tables to quickly find a tool or service aid.

Table 1 provides guidance on how to select the type of dump to use for a specific problem.

<table>
<thead>
<tr>
<th>What is the problem or need?</th>
<th>Type of dump to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing of an authorized program or a problem program while it is running, especially for 64-bit applications</td>
<td>Transaction dump (see Chapter 3, “Transaction dump,” on page 55)</td>
</tr>
<tr>
<td>Testing of a problem program while it is running</td>
<td>SNAP dump (see Chapter 6, “SNAP dump,” on page 157)</td>
</tr>
<tr>
<td>Abnormal end of an authorized program or a problem program</td>
<td>ABEND dump (see Chapter 5, “ABEND dump,” on page 135)</td>
</tr>
<tr>
<td>System problem when the system continues processing</td>
<td>SVC dump (see Chapter 2, “SVC dump,” on page 7)</td>
</tr>
<tr>
<td>System problem when the system stops processing or is stopped by the operator because of slowdown or looping</td>
<td>Stand-alone dump (see Chapter 4, “Stand-alone dump,” on page 65)</td>
</tr>
</tbody>
</table>

Table 2 provides guidance on how to select the type of trace to use for a specific problem.

<table>
<thead>
<tr>
<th>What is the problem or need?</th>
<th>Type of trace to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>System problem: diagnosis requires checking of component events</td>
<td>Component trace (see Chapter 12, “Component trace,” on page 375)</td>
</tr>
</tbody>
</table>
Selecting tools and service aids

Table 2. Selecting a trace (continued)

<table>
<thead>
<tr>
<th>What is the problem or need?</th>
<th>Type of trace to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>System problem: diagnosis requires detailed checking of one or two system events</td>
<td>Generalized trace facility (GTF) trace (see Chapter 10, “The Generalized Trace Facility (GTF),” on page 229)</td>
</tr>
<tr>
<td>System or authorized program problem: diagnosis requires the messages related to a dump</td>
<td>Master trace (see Chapter 9, “Master trace,” on page 221)</td>
</tr>
<tr>
<td>System problem: diagnosis requires checking many system events</td>
<td>System trace (see Chapter 8, “System trace,” on page 169)</td>
</tr>
<tr>
<td>System or problem program: diagnosis requires information about allocation of virtual storage.</td>
<td>GETMAIN, FREEMAIN, STORAGE (GFS) trace (see Chapter 14, “GETMAIN, FREEMAIN, STORAGE (GFS) trace,” on page 545)</td>
</tr>
</tbody>
</table>

Table 3 provides guidance on how to select the service aid to use for a specific problem.

Table 3. Selecting a service aid

<table>
<thead>
<tr>
<th>What is the problem or need?</th>
<th>Type of service aid to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>System or hardware problem: need a starting point for diagnosis or when diagnosis requires an overview of system and hardware events in chronological order.</td>
<td>Logrec data set (see Chapter 15, “Recording logrec error records,” on page 551)</td>
</tr>
<tr>
<td>Information about the content of load modules and program objects or problem with modules on system.</td>
<td>AMBLIST (see Chapter 16, “AMBLIST: Map load modules and program objects,” on page 577)</td>
</tr>
<tr>
<td>Diagnosis requires dynamic change to a program, such as fixing program errors, inserting a SLIP trap match, or altering a program to start component trace.</td>
<td>SPZAP (see Chapter 17, “SPZAP: Modify data in programs and VTOCs,” on page 633)</td>
</tr>
<tr>
<td>Need to pack the diagnostic materials for transmission to another site, and create similar data sets at the receiving site.</td>
<td>AMATERSE (see Chapter 18, “AMATERSE: Pack and unpack a data set,” on page 667)</td>
</tr>
<tr>
<td>Need to eliminate duplicate or unneeded dumps.</td>
<td>DAE (see Chapter 20, “Dump suppression,” on page 687)</td>
</tr>
<tr>
<td>Diagnosis requires a trap to catch problem data while a program is running.</td>
<td>SLIP (see z/OS MVS System Commands)</td>
</tr>
<tr>
<td>Diagnosis requires formatted output of problem data, such as a dump or trace.</td>
<td>IPCS (see z/OS MVS IPCS User’s Guide)</td>
</tr>
</tbody>
</table>

What tools and service aids are available?

This topic provides an overview of the tools and service aids in more detail. The tables that follow contain a brief description of each tool or service aid, some reasons why you would use it, and a reference to the topic or document that covers the tool or service aid in detail. (Most of the detailed information on tools and service aids is in this document.) The tools and service aids are covered in three tables; the dumps, traces, or service aids are listed in order by frequency of use.

Table 4 on page 3 lists each type of dump and gives an overview of how they can be used.
## Table 4. Description of dumps

<table>
<thead>
<tr>
<th>Type of dump</th>
<th>Description</th>
</tr>
</thead>
</table>
| **ABEND Dump** | Use an ABEND dump when ending an authorized program or a problem program because of an uncorrectable error. These dumps show:  
  - The virtual storage for the program requesting the dump.  
  - System data associated with the program.  
  The system can produce three types of ABEND dumps, SYSABEND, SYSMDUMP, and SYSUDUMP. Each one dumps different areas. Select the dump that gives the areas needed for diagnosing your problem. The IBM supplied defaults for each dump are:  
  - SYSABEND dumps - The largest of the ABEND dumps, containing a summary dump for the failing program plus many other areas useful for analyzing processing in the failing program.  
  - SYSMDUMP dumps - Contains a summary dump for the failing program, plus some system data for the failing task. SYSMDUMP dumps are the only ABEND dumps that you can format with IPCS.  
  - SYSUDUMP dumps - The smallest of the ABEND dumps, containing data and areas only about the failing program.  
| **Transaction Dump (IEATDUMP)** | Similar to SNAP dumps, an application can issue an IEATDUMP macro to dump virtual storage areas of interest if the application is running. However, the result is an unformatted dump that must be analyzed using IPCS. See Chapter 3, “Transaction dump,” on page 55 for details. |
| **SNAP Dump** | Use a SNAP dump when testing a problem program. A SNAP dump shows one or more areas of virtual storage that a program, while running, requests the system to dump. A series of SNAP dumps can show an area at different stages in order to picture a program’s processing, dumping one or more fields repeatedly to let the programmer check intermediate steps in calculations. SNAP dumps are preformatted, you cannot use IPCS to format them.  
  Note that a SNAP dump is written while a program runs, rather than during abnormal end.  
| **Stand-Alone Dump** | Use a stand-alone dump when:  
  - The system stops processing.  
  - The system enters a wait state with or without a wait state code.  
  - The system enters an instruction loop.  
  - The system is processing slowly.  
  These dumps show central storage and some paged-out virtual storage occupied by the system or stand-alone dump program that failed. Stand-alone dumps can be analyzed using IPCS.  
### Table 4. Description of dumps (continued)

<table>
<thead>
<tr>
<th>Type of dump</th>
<th>Description</th>
</tr>
</thead>
</table>
| SVC Dumps    | SVC dumps can be used in two different ways:  
|              | • Most commonly, a system component requests an SVC dump when an unexpected  
|              |    system error occurs, but the system can continue processing.  
|              | • An authorized program or the operator can also request an SVC dump when they  
|              |    need diagnostic data to solve a problem.  
|              | SVC dumps contain a summary dump, control blocks and other system code, but the  
|              |    exact areas dumped depend on whether the dump was requested by a macro, command,  
|              |    or SLIP trap. SVC dumps can be analyzed using IPCS.  
|              | Reference: See [Chapter 2, “SVC dump,” on page 7](#) for detailed information. |

### Table 5. Description of traces

<table>
<thead>
<tr>
<th>Trace</th>
<th>Description</th>
</tr>
</thead>
</table>
| Component Trace  | Use a component trace when you need trace data to report an MVS component problem  
|                  | to the IBM Support Center. Component tracing shows processing within an MVS  
|                  | component. Typically, you might use component tracing while recreating a problem.  
|                  | The installation, with advice from the IBM Support Center, controls which events are  
|                  | traced for a component.  
|                  | Reference: See [Chapter 12, “Component trace,” on page 373](#) for detailed information. |
| GFS Trace        | Use GFS trace to collect information about requests for virtual storage through the  
|                  | GETMAIN, FREEMAIN, and STORAGE macro.  
|                  | Reference: See [Chapter 14, “GETMAIN, FREEMAIN, STORAGE (GFS) trace,” on page 545](#)  
|                  | for detailed information. |
| GTF Trace        | Use a GTF trace to show system processing through events occurring in the system over  
|                  | time. The installation controls which events are traced.  
|                  | GTF tracing uses more resources and processor time than a system trace. Use GTF when  
|                  | you are familiar enough with the problem to pinpoint the one or two events required to  
|                  | diagnose your system problem. GTF can be run to an external data set as well as a  
|                  | buffer.  
|                  | Reference: See [Chapter 10, “The Generalized Trace Facility (GTF),” on page 229](#)  
|                  | for detailed information. |
| Master Trace     | Use the master trace to show the messages most recently issued. Master trace is useful  
|                  | because it provides a log of these messages in a dump. These can be more pertinent to  
|                  | your problem than the messages accompanying the dump itself.  
|                  | Reference: See [Chapter 9, “Master trace,” on page 221](#) for detailed information. |
| System Trace     | Use system trace to see system processing through events occurring in the system over  
|                  | time. System tracing is activated at initialization and, typically, runs continuously. It  
|                  | records many system events, with minimal detail about each. The events traced are  
|                  | predetermined, except for branch tracing.  
|                  | This trace uses fewer resources and is faster than a GTF trace.  
|                  | Reference: See [Chapter 8, “System trace,” on page 169](#) for detailed information. |

Table 6 on page 5 describes the service aids and how they can be used.
### Table 6. Description of service aids

<table>
<thead>
<tr>
<th>Service Aid</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AMATERSE</strong></td>
<td>Use the AMATERSE service aid to create a compact image of diagnostic data sets. The compact image helps to use less space while retaining materials and prepare for efficient transmission of materials from one site to another, such as to send the materials to IBM support.</td>
</tr>
</tbody>
</table>
| **AMBLIST** | Use AMBLIST when you need information about the content of load modules and program objects or you have a problem related to the modules on your system. AMBLIST is a program that provides lots of data about modules in the system, such as a listing of the load modules, map of the CSECTs in a load module or program object, list of modifications in a CSECT, map of modules in the LPA (link pack area), and a map of the contents of the DAT-on nucleus.  
**Reference:** See Chapter 16, “AMBLIST: Map load modules and program objects,” on page 577 for detailed information. |
| **Common Storage Tracking** | Use common storage tracking to collect data about requests to obtain or free storage in CSA, ECSA, SQA, and ESQA. This is useful to identify jobs or address spaces using an excessive amount of common storage or ending without freeing storage.  
Use RMF™ or the IPCS VERBEXIT VSMDATA subcommand to display common storage tracking data.  
**References:**  
- See [z/OS MVS Initialization and Tuning Guide](#) for detailed information on requesting common storage tracking.  
- See the VSM chapter of [z/OS MVS Diagnosis: Reference](#) for information on the IPCS VERBEXIT VSMDATA subcommand. |
| **DAE** | Use dump analysis and elimination (DAE) to eliminate duplicate or unneeded dumps. This can help save system resources and improve system performance.  
**Reference:** See Chapter 20, “Dump suppression,” on page 687 for detailed information. |
| **IPCS** | Use IPCS to format and analyze dumps, traces, and other data. IPCS produces reports that can help in diagnosing a problem. Some dumps, such as SNAP and SYSABEND and SYSUDUMP ABEND dumps, are preformatted, and are not formatted using IPCS.  
**Reference:** See [z/OS MVS IPCS User’s Guide](#) for detailed information. |
| **Logrec Data Set** | Use the logrec data set as a starting point for problem determination. The system records hardware errors, selected software errors, and selected system conditions in the logrec data set. Logrec information gives you an idea of where to look for a problem, supplies symptom data about the failure, and shows the order in which the errors occurred.  
**Reference:** See Chapter 15, “Recording logrec error records,” on page 551 for detailed information. |
| **SLIP Traps** | Use serviceability level indication processing (SLIP) to set a trap to catch problem data. SLIP can intercept program event recording (PER) or error events. When an event that matches a trap occurs, SLIP performs the problem determination action that you specify:  
- Requesting or suppressing a dump.  
- Writing a trace or a logrec data set record.  
- Giving control to a recovery routine.  
- Putting the system in a wait state.  
- Issuing system commands  
**Reference:** See the SLIP command in [z/OS MVS System Commands](#) for detailed information. |
## Selecting tools and service aids

### Table 6. Description of service aids (continued)

<table>
<thead>
<tr>
<th>Service Aid</th>
<th>Description</th>
</tr>
</thead>
</table>
| **SPZAP**   | Use the SPZAP service aid to dynamically update and maintain programs and data sets. For problem determination, you can use SPZAP to:  
  • Fix program errors by replacing a few instructions in a load module or member of a partitioned data set (PDS).  
  • Insert an incorrect instruction in a program to force an ABEND or make a SLIP trap work.  
  • Alter instructions in a load module to start component trace.  
  • Replace data directly on a direct access device to reconstruct a volume table of contents (VTOC) or data records that were damaged by an input/output (I/O) error or program error. |

**Reference:** See Chapter 17, “SPZAP: Modify data in programs and VTOCs,” on page 633 for detailed information.
Chapter 2. SVC dump

An SVC dump provides a representation of the virtual storage for the system when an error occurs. Typically, a system component requests the dump from a recovery routine when an unexpected error occurs. However, an authorized program or the operator can also request an SVC dump when diagnostic dump data is needed to solve a problem.

An SVC dump comes in the following types, depending on how it was requested. Note that the type of dump requested determines its contents.

- **Asynchronous SVC dump (scheduled SVC dump):**
  The system issues an instruction or the caller uses a combination of parameters on the SVC dump macro invocation. SVC dump captures all of the dump data into a set of data spaces then writes the dump data from the data spaces into a dump data set. The system is available for another SVC dump upon completion of the capture phase of the dump. In an asynchronous SVC dump, the summary dump data is captured first and can be considered more useful for diagnosis.

- **Synchronous SVC dump:**
  The requester's SVC dump macro invocation issues an instruction to obtain the dump under the current task. The system returns control to the requester once the dump data has been captured into a set of data spaces. SVC dump processing then writes the dump data from the data spaces into a dump data set. The system is available for another SVC dump upon completion of the capture phase of the dump. In a synchronous SVC dump, the summary dump data is captured last.

Each SVC dump also contains a summary dump, if requested. Because dumps requested from disabled, locked, or SRB-mode routines cannot be handled by SVC dump immediately, system activity overwrites much useful diagnostic data. The summary dump supplies copies of selected data areas taken at the time of the request. Specifying a summary dump also provides a means of dumping many predefined data areas simply by specifying one option. Summary dump data is dumped using ASID(X'aaaa') SUMDUMP records and ASID(X'aaaa') DSPNAME(dddddddd) SUMDUMP records. The IPCS user has the option of causing storage dumped in these records also to be mapped as ASID(X'aaaa') or ASID(X'aaaa') DSPNAME(dddddddd) storage. Message BLS18160D is displayed during dump initialization when TSO prompting and IPCS confirmation options permit. If the TSO prompting and the IPCS confirmation options don't permit, the additional mapping is performed. Selective display of ASID(X'aaaa') SUMDUMP or ASID(X'aaaa') DSPNAME(dddddddd) SUMDUMP storage might be requested by referring to those address spaces.

This section includes information system programmers need to know about SVC dump and SVC dump processing:

- “Using automatically allocated dump data sets” on page 8
- “Using pre-allocated dump data sets” on page 13
- “Choosing SVC dump data sets” on page 15
- “Obtaining an SVC dump” on page 17
- “Printing, viewing, copying, and clearing a pre-allocated or SYS1.DUMPxx data set” on page 24
SVC dump

- “Contents of SVC dumps” on page 25
- “Analyzing summary SVC dumps” on page 35
- “Analyzing an SVC dump” on page 39

See z/OS MVS Programming: Authorized Assembler Services Guide for information any programmer needs to know about programming the SDUMP or SDUMPX macros to obtain an SVC dump:

- Deciding when to request an SVC dump
- Understanding the types of SVC dumps that MVS produces
- Designing an application program to handle a specific type of SVC dump
- Identifying the data set to contain the dump
- Defining the contents of the dump
- Suppressing duplicate SVC dumps using dump analysis and elimination (DAE)

Planning data set management for SVC dumps

SVC dump processing stores data in dump data sets that the system allocates automatically, as needed, or that you pre-allocate manually. IBM recommends the use of automatically allocated dump data sets whenever possible. Only the space required for the dump being written is allocated. The dump is written using a system-determined block size, so write time is reduced. SMS extended attributes, such as compression and striping, can be assigned to further reduce the amount of space required and the time to write.

IBM recommends using extended format sequential data sets as dump data sets for SVC Dumps. For the reasons why, see “Choosing SVC dump data sets” on page 15.

Use pre-allocated dump data sets only as a back up, in case the system is not able to automatically allocate a data set. Otherwise, the dump can become truncated, making error diagnosis difficult.

Using automatically allocated dump data sets

SVC dump processing supports automatic allocation of dump data sets at the time the system writes the dump to DASD. Automatically allocated dumps will be written using the system-determined block size. The dump data sets can be allocated as SMS-managed or non-SMS-managed, depending on the VOLSER or SMS classes defined on the DUMPDS ADD command. When the system captures a dump, it allocates a data set of the correct size from the resources you specify. See “Choosing SVC dump data sets” on page 15 for DFSMS support of extended format sequential data sets. Using Extended Format Sequential data sets, the maximum size of the dump can exceed the size allowed for non-SMS managed data sets.

If automatic allocation fails, pre-allocated dump data sets are used. If no pre-allocated SYS1.DUMPnn data sets are available, message IEA793A is issued, and the dump remains in virtual storage. SVC Dump periodically retries both automatic allocation and writing to a pre-allocated dump dataset until successful or until the captured dump is deleted either by operator intervention or by the expiration of the CHNGDUMP MSGTIME parameter governing message IEA793A. If you set the MSGTIME value to 0, the system will not issue the message, and it deletes the captured dump immediately.
Naming automatically allocated dump data sets
The installation has control of the name of the data sets created by the automatic allocation function, and you can select a name-pattern to allow for dump data set organization according to your needs. The name is determined through an installation-supplied pattern on the DUMPDS command. A set of symbols is available so that you can include the following kinds of information in the names of your automatically allocated dump data sets:
- System name
- Sysplex name
- Job name
- Local and GMT time and date
- Sequence number

You can specify a name-pattern to generate any name acceptable under normal MVS data set name standards. The only requirement is that you include the sequence number symbol to guarantee each automatically allocated dump data set has a unique name.

Using automatic allocation of SVC dump data sets
You can specify the command instructions to enable or disable automatic allocation either in the COMMNDxx parmlib member, to take effect at IPL, or from the operator console at any time after the IPL, to dynamically modify automatic allocation settings. The DUMPDS command provides the following flexibility:
- Activate automatic allocation of dump data sets
- Add or delete allocation resources
- Direct automatic allocation to SMS or non-SMS managed storage
- Deactivate automatic allocation of dump data sets
- Reactivate automatic allocation of dump data sets
- Change the dump data set naming convention

Set up automatic allocation with the following steps:
- Set up allocation authority
- Establish a name pattern for the data sets
- Define resources for storing the data sets
- Activate automatic allocation

After automatic allocation of these SVC dump data sets is active, allocation to a DASD volume is done starting with the first resource allocated via the DUMPDS ADD command. When allocation to that volume is no longer successful, the next resource is then used.

SVC Dump data sets can be SMS-managed or non-SMS-managed. If the DUMPDS ADD command defined SMS classes, then the allocation will first pass these classes to the ACS routines to try to allocate the SVC dump data set as SMS-managed. If this allocation is not successful for any reason, or if no SMS classes are defined, then the data set allocation will use the DASD volumes that were defined on the DUMPDS ADD command, and the SVC Dump data set will be allocated as non-SMS-managed.

SVC Dump data sets allocated as non-SMS-managed must be single volume; they can have multiple extents but they cannot span multiple volumes. Non-SMS-managed DASD does not support striping. SVC Dump data sets
allocated as SMS-managed can be multi-volume only if they are allocated as striped data sets. Striping is an attribute that must be defined in the SMS classes. Striping and compression, another SMS attribute, can be used to allocate datasets that are larger than those allowed for a pre-allocated or non-SMS managed dataset.

**Note:** You must update automatic class selection (ACS) routines to route the intended data set into SMS-management so that it is assigned a storage class.

### Setting up allocation authority

To allocate dump data sets automatically, the DUMPSRV address space must have authority to allocate new data sets. Do the following:

1. **Associate the DUMPSRV address space with a user ID.**
   
   Use the RACF® STARTED general resource class or the RACF Started Procedures Table, ICHRIN03, to associate DUMPSRV with a user id.

2. **Authorize DUMPSRV's user ID to create new dump data sets using the naming convention in the following topic.**
   
   With the high-level qualifier of SYS1, the data sets are considered group data sets. You can assign CREATE group authority to the DUMPSRV user ID within that group.

See [z/OS Security Server RACF System Programmer's Guide](#) for information about the RACF STARTED general resource class, and the RACF Started Procedures Table. See [z/OS Security Server RACF Security Administrator's Guide](#) for information on using the RACF STARTED general resource class, and on controlling creation of new data sets.

### Establishing a name pattern

Establishing the name pattern for the dump data sets is accomplished by the DUMPDS NAME= command. Names must conform to standard data set naming conventions and are limited to 44 characters, including periods used as delimiters between qualifiers. For a complete description, see [z/OS DFSMS Using Data Sets](#). To allow meaningful names for the dump data sets, several symbols are provided that are resolved when the dump data is captured in virtual storage. For a complete list of the symbols you can use, see the explanation of DUMPDS NAME= in [z/OS MVS System Commands](#).

When determining the pattern for the dump data set names, consider any automation tools you may have at your installation that work on dump data sets. Also, the automatic allocation function requires you to include the &SEQ. sequence number symbol in your data set name pattern to guarantee unique data set names. If you do not use the sequence number, the system rejects the name pattern with message IEE855I and the previous name pattern remains in effect.

By default, the system uses one of three name patterns. The system typically uses the normal pattern SYS1.DUMP.D&DATE..T&TIME..&SYSNAME..S&SEQ; The system automatically uses S&SYSNAME convention when the system name begins with numeric and is less than 8 characters long. For example:

- **SYS1.DUMP.D&YYMMDD..T&HHMMSS..S&SYSNAME..S&SEQ.** or
- **SYS1.DUMP.D&YYMMDD..T&HHMMSS..S&SYSNAME(2&colon.8)..S&SEQ**

  when the system name begins with numeric and is 8 characters long.

[Figure 1 on page 11](#) describes the default name pattern.
While the default data sets begin with a high-level qualifier of SYS1, this convention is no longer a requirement for data sets named by your installation.

Notice that the symbols are resolved into date, time, and sequence numerics, so they are preceded by an alphabetic character to conform to MVS data set name requirements. Also, the symbol starts with an ampersand (&) and ends with a period (.), resulting in a name pattern that has double periods when a symbol finishes a qualifier. One period ends the symbol, and the second serves as the delimiter between qualifiers of the generated data set name.

**Defining resources for dump data sets**

If allocation is active, SVC dump data sets can be automatically allocated as soon as resources are defined to store them. If you have not changed the name pattern, then the system default is used. See [“Establishing a name pattern” on page 10](#). You can define dump data set resources using the `DUMPDS ADD,VOL=volser` (for DASD volumes) and `DUMPDS ADD,SMS=class` (for SMS classes) commands. You can remove resources using the `DUMPDS DEL,VOL=volser` and `DUMPDS DEL,SMS=class` commands. Automatic allocation is directed to SMS classes in preference to DASD volumes.

When automatic allocation is inactive, dumps are written to pre-allocated SYS1.DUMPxx data sets. Deactivating automatic allocation does not result in the loss of resource definitions, however. So, if automatic allocation is reactivated, all the previous resources remain available for receiving automatically allocated dump data sets. Similarly, removing the last allocation resource will not cause automatic allocation to be inactive. Removing the last allocation resource effectively ‘turns off’ the function, though, just as if all the defined resources were full. In both cases the system responds with message IEA799I and dumps are written to pre-allocated SYS1.DUMPxx data sets if they exist. Otherwise the dump remains captured until:

- You create a place for it
- The established time limit, as indicated by the `CHNGDUMP MSGTIME` parameter, expires
- The operator deletes the dump.

**Activating automatic allocation**

By default, automatic allocation is inactive after IPLing the system. However, you can add to your `COMMNDxx` parmlib member the `DUMPDS NAME= command, any DUMPDS ADD commands, and the DUMPDS ALLOC=ACTIVE command to activate automatic allocation during IPL.
If you have turned off automatic allocation using ALLOC=INACTIVE, reactivate it by entering the DUMPDS ALLOC=ACTIVE operator command.

**Verifying dump status**

To verify dump status issue the DISPLAY DUMP,STATUS command. For example, after IPLing SYSTEM1 specifying DUMP=NO as a system parameter, and without requesting any dumps or specifying any DUMPDS or CHNGDUMP commands, the output shown in [Figure 2](#) would be expected as a result of the DISPLAY DUMP,STATUS command:

```
IEE852I 10.56.03 SYS1.DUMP STATUS
SYS1.DUMP DATA SETS AVAILABLE=000 AND FULL=000
CAPTURED DUMPS=0000, SPACE USED=00000000M, SPACE FREE=00000500M
AUTOMATIC ALLOCATION IS: INACTIVE
NO SMS CLASSES DEFINED
NO DASD VOLUMES DEFINED
NAME=SYS1.DUMP.D&D&DATE..T&D&TIME..&SYSNAME..S&SEQ.
EXAMPLE=SYS1.DUMP.D930324.T105603.SYSTEM1.S00000
```

**Figure 2. Example: verifying dump status**

Now assume that the following steps are performed to establish the automatic allocation function:

1. Set up your installation data set name pattern using the DUMPDS command:
   ```
   ```
   **Note:** This step is only required if you are not using the default name pattern as shown in [Figure 1 on page 11](#).

2. Add dump data set resources that can be used by the automatic allocation function:
   ```
   DUMPDS ADD,VOL=(SCRTH1,HSM111)
   DUMPDS ADD,SMS=(DUMPDA)
   ```

3. Activate automatic dump data set allocation using the DUMPDS command:
   ```
   DUMPDS ALLOC=ACTIVE
   ```
   **Note:** These steps can be performed after IPL using the DUMPDS command from an operator console, or early in IPL by putting the commands in the COMMNDxx parmlib member and pointing to the member from the IEASYSxx parmlib member using CMD=xx.

If you use COMMNDxx, you may want to specify DUMP=NO in the IEASYSxx parmlib member to prevent dumps taken during IPL from being written to SYS1.DUMPxx data sets.

After issuing the DUMPDS commands shown in steps 1 through 3 requesting dump status would be as shown [Figure 3 on page 13](#).
Managing automatically allocated dump data sets

Automatic allocation of dump data sets is managed through the DUMPDS command. Placing appropriate commands into the COMMNDxx parmlib member allows the function to be established at IPL.

The DISPLAY DUMP command can display information about the last 100 data sets that were automatically allocated during the current IPL. Typical dump inventory management should be done using the Sysplex Dump Directory. The System Dump Directory provides access to all of the cataloged and added dump data sets created across system IPLs. Details about using the User and Sysplex Dump Directory can be found in the z/OS MVS IPCS User’s Guide.

The installation must manage the space allocated to dump data sets by limiting the volumes (non-SMS) or the classes (SMS) available for automatic allocation of dump data sets. z/OS MVS System Commands contains the syntax of the DUMPDS ADD, DEL, and ALLOC=ACTIVE commands.

For more information about SMS, see z/OS DFSMSdfp Storage Administration.

Using pre-allocated dump data sets

Pre-allocated dump data sets should be used as a backup method to automatic allocation. Like the automatically allocated dump data sets, pre-allocated dump data sets will hold SVC dump information for later review and analysis, but have size and performance limitations that automatically allocated dump data sets do not have. This section describes how to set up pre-allocated data sets for SVC dump, including:

- Allocating SYS1.DUMPxx data sets with secondary extents
- Specifying SYS1.DUMPxx data sets
- Controlling SYS1.DUMPxx data sets

Allocating SYS1.DUMPxx data sets with secondary extents

Allocate SYS1.DUMPxx data sets using the following requirements:

- Name the data set SYS1.DUMPxx, where xx is decimal 00 through 99.
- Select a device with a track size of 4160 bytes. The system writes the dump in blocked records of 4160 bytes.
- Initialize with an end-of-file (EOF) record as the first record. (If you use ISPF 3.2 to allocate SYS1.DUMPxx data sets, the EOF record will be automatically written on the first track.)
- Allocate the data set before requesting a dump. Allocation requirements are:
  - UNIT: A permanently resident volume on a direct access device.
  - DISP: Catalog the data set (CATLG). Do not specify SHR.
SVC dump

- VOLUME: Place the data set on only one volume. Allocating the dump data set on the same volume as the page data set could cause contention problems during dumping, as pages for the dumped address space are read from the page data set and written to the dump data set.

- SPACE: An installation must consider the size of the page data set that will contain the dump data. The data set must be large enough to hold the amount of data as defined by the MAXSPACE parameter on the CHNGDUMP command, VIO pages, and pageable private area pages.

SVC dump processing improves service by allowing secondary extents to be specified when large dump data sets are too large for the amount of DASD previously allocated. An installation can protect itself against truncated dumps by specifying secondary extents and by leaving sufficient space on volumes to allow for the expansion of the dump data sets.

For the SPACE keyword, you can specify CONTIG to make reading and writing the data set faster. Request enough space in the primary extent to hold the smallest SVC dump expected. Request enough space in the secondary extent so that the primary plus the secondary extents can hold the largest SVC dump. The maximum size of a data set is 65,535 tracks. For a 3390 this is 4369 cylinders, and will hold about 2.8 gigabytes of data. The actual size of the dump depends on the dump options in effect when the system writes the dump.

Estimate the largest dump size as follows:

\[
\text{Bytes of SDATA options + bytes in largest region size} = \text{Result1}
\]

\[
\text{Result1} \times \text{number of address spaces in dump} = \text{Result2}
\]

\[
\text{PLPA + 20} = \text{Result3}
\]

\[
\text{Bytes of requested data space storage} = \text{Result4}
\]

\[
\text{Result2 + Result3 + Result4} = \text{Bytes in SVC dump}
\]

Where:

- Result1, Result2, Result3, Result 4: Intermediate results
- SDATA options: Described in "Contents of SVC dumps" on page 25
- PLPA: Pageable link pack area

For the size of the smallest dump, use the default options for the SDUMPX macro. The difference between the largest dump and the smallest dump will be the size of the secondary extent.

For example, to calculate the largest amount of storage required for a 3390 DASD, assume that, from the preceding calculations, the records needed for the SVC dump amount to 43200 kilobytes. There are 11 records per track and 15 tracks per cylinder. To determine the number of cylinders needed to allocate a data set of this size, do the following:

- For 43200 kilobytes of storage, you will need space for 10800 SVC dump records (43200 / 4 kilobytes per record).
- With 11 records per track, you will require 982 (10800 / 11 records) tracks.
- Therefore, the data set would require 66 cylinders (982 / 15 tracks per cylinder) for allocation.

Note: If you are not receiving the dump data you require, increase the size of the dump data set. You will receive system message IEA911E.

The system writes only one dump in each SYS1.DUMPxx data set. Before the data set can be used for another dump, clear it using the DUMPDS command with the CLEAR keyword.
Choosing SVC dump data sets

IBM recommends using extended format sequential data sets as dump data sets for SVC Dumps. Extended format sequential data sets:

- Have a greater capacity than sequential data sets
- Support striping
- Support compression

Greater capacity: Some dump data sets are quite large compared with other data sets generated by a system. An extended format sequential data set can hold the largest SVC dumps, as much as 128 gigabytes.

Support for striping: Striping spreads sections, or stripes, of a data set across multiple volumes and uses independent paths, if available, to those volumes. The multiple volumes and independent paths accelerate sequential reading and writing of the data set, reducing the time during which dump I/O competes with production I/O.

It is recommended that the number of stripes match the number of volumes you use. This combination will yield the best performance because MVS data management allows random access to any record as though it appeared on a single volume. This is particularly useful during an IPCS analysis of a dump. The savings when loading the data set are real but smaller, the result of reducing the number of times end of volume processing comes into play.

In a striped data set, when the last volume receives a stripe, the next stripes are placed on the first volume, the second volume, the third, and so on to the last volume, then back to the first volume.

If you use more than six dozen stripes, the performance benefit of each additional stripe is much less than the performance benefit of adding the earlier stripes. Keep in mind that this is talking about the original data set definition. You can not add stripes to an existing striped data set. You must plan ahead. The faster processing speeds up moving dump data from relatively expensive data space storage to less expensive DASD.

Support for compression: Compression allows dump data sets to use less DASD space. Before using compression, consider the following:

- Compression and decompression trade off processing cycles for more efficient use of DASD. If hardware compression is not available, the number of processing cycles is significantly higher.

Using DSNTYPE=LARGE: In z/OS V1R7 and later releases, sequential data sets that use DSNTYPE=LARGE are allowable for SVC dumps when the systems that are involved in processing a DSNTYPE=LARGE data set are migrated to V1R7 prior to their use. If analysis using an earlier release is required, use z/OS V1R7 to transcribe the dump into a data set supported by the earlier release.

Placing dump data sets in cylinder-managed space: In z/OS V1R11 and later releases, extended format sequential data sets can be placed in either
track-managed space or cylinder-managed space. SVC dump fully supports placement of dump data sets in cylinder-managed space.

**Finding automatically allocated dump data sets**

The AUTODSN= parameter of the DISPLAY DUMP,TITLE operator command enables you to list up to 100 of the most recent dump data sets that were automatically allocated during this IPL. No information is preserved about data sets that were automatically allocated before the last 100. As an example, if you wanted to see the titles of the last 5 automatically allocated dump data sets, you would issue:

```
DISPLAY DUMP,TITLE,AUTODSN=5
```

For complete information on the use of the DISPLAY DUMP command, see DISPLAY DUMP in [z/OS MVS System Commands](https://www.ibm.com/support/knowledgecenter/SSSHV8_2.2.0/com.ibm.zos.v2r2.mp5.iod/iod_mmz_pcommands.html).

**Communication from the system**

The system communicates about automatic allocation of dump data sets using two messages:

- IEA611I is issued when a complete or partial dump is taken to an automatically allocated dump dataset. IEA611I is an informational message, it will not be issued highlighted.
- IEA799I is issued once per captured dump when automatic allocation fails; it will not be re-issued as a result of automatic allocation failing for subsequent attempts to allocate the same dump data set unless the reason text is different.

**Specifying SYS1.DUMPxx data sets**

When planning SYS1.DUMPxx data sets, remember that the data sets frequently contain sensitive data (user or installation confidential information, logon passwords, encryption keys, etc.). Protect these data sets with RACF to limit access to them.

The installation can specify SYS1.DUMPxx data sets in two ways:

- IBM recommends that you use the DUMPDS operator command through the COMMNDxx parmlib member. Use the DUMPDS ADD command within the COMMNDxx parmlib member to ensure that all interaction with the dump data set occurs through the DUMPDS command.
  
  For example, to specifically add data set SYS1.DUMP05, enter:
  
  ```
  COM='DUMPDS ADD,DSN=05'
  ```

- During system initialization, in the DUMP parameter in the IEASYSxx parmlib member.
  
  Specify DUMP=NO in the IEASYSxx parmlib member. Otherwise, all available data sets will be allocated before the COMMNDxx parmlib member is processed.

The data sets are on direct access only. The maximum number of SYS1.DUMPxx data sets an installation can have is 100. The direct access data set must be on a permanently resident volume; that is, the data set must be allocated and cataloged. These dump data sets cannot be shared by more than one system.

All dump data sets should not be on the same pack. A pack should contain enough storage to allow the dump data sets to allocate secondary extent space, if needed.

For more information, see the following references:
Controlling SYS1.DUMPxx data sets

After system initialization, use the following to change and control these data sets:

- Copy the dump from the SYS1.DUMPxx data set to another data set; then clear the SYS1.DUMPxx data set, so that it can be sued for another dump. You can use IPCS to format and view or print the copied dump, as described in the following topic.
- Use the DUMPDS operator command to:
  - Add more SYS1.DUMPxx data sets on direct access for SVC dumps.
  - Delete SYS1.DUMPxx data sets for SVC dumps.
  - Clear a SYS1.DUMPxx data set containing a dump by writing an EOF mark as the first record. An EOF mark as the first record makes the data set available for another dump.

A rel IPL is not necessary when adding, deleting, or clearing a data set with the DUMPDS operator command.
- Use the REPLY command to system message IEA793A to cancel a dump.
- Use a post dump exit routine to copy the dump to another data set. IEAVTSEL is an SVC dump post dump exit name list that lists the module names of installation exit routines to be given control when dump processing ends.

For more information, see the following references:

- See [z/OS MVS Installation Exits](#) for more information about the IEAVTSEL post dump exit name list.
- See [z/OS MVS IPCS Commands](#) for the IPCS COPYDUMP subcommand.
- See [z/OS MVS System Commands](#) for the DUMPDS and REPLY operator commands.

Obtaining an SVC dump

Obtain an SVC dump by:

- Using a SDUMP or SDUMPX macro in an authorized program.
- Entering the DUMP or SLIP operator command
- Setting a SLIP in the IEASLPxx parmlib member.

When z/OS takes an SVC dump, it copies data into the DUMPSRV owned data spaces and high virtual storage. The collection of data can introduce an unusually heavy burden on storage resources. The virtual storage load remains until the dump is written out to a target data set on DASD. The data set can be:

- SVC dump data set that is specified on the DCB parameter of the SDUMP or SDUMPX macro.
- Pre-allocated SYS1.DUMPxx data set, or automatically allocated dump data set.

Use the DUMPDS operator command to manage the pre-allocated and automatically allocated data sets.

When an SVC dump occurs, if normal auxiliary storage use rises above 30%, the system might experience severe performance problems. The system might also experience an 03C wait state, which indicates that the system ran out of available
SVC dump

paging slots. You can use the MAXSPACE and AUXMGMT options on the CHNGDUMP SET, SDUMP command to manage the burden of taking SVC dumps on a system. Using the options might not be sufficient to eliminate the problems that are associated with the restricted auxiliary (paging) storage.

- The MAXSPACE value restricts the virtual storage available to the DUMPSRV address space. When you use MAXSPACE, the installation must tune for the worst case usage of real and auxiliary storage. The following rules apply:
  - If the installation does not have a history that can be drawn upon, see “Allocating SYS1.DUMPxx data sets with secondary extents” on page 13 for help in determining a maximum data set size. Use a multiple of the data set size to determine a MAXSPACE value. The installation must predict the size of the largest dump that it can configure.
  - The paging resources for the affected systems must also be increased to accommodate the additional load that is represented by the MAXSPACE value. The minimum value for defining the additional auxiliary storage capacity must be three times the MAXSPACE value. Adhering to the MAXSPACE guideline can maintain utilization within 30%.

- When you specify the AUXMGMT=ON parameter, the installation disregards first failure data capture (FFDC) to maintain system availability. The following rules apply:
  - No new dumps are allowed when auxiliary storage usage reaches 50%. New dumps can be initiated after the auxiliary storage usage drops below 35%.
  - Current SDUMP data capture stops when auxiliary storage usage exceeds 68%, and generates a partial dump.

For more information about setting the MAXSPACE or AUXMGMT value, see the CHNGDUMP Command in z/OS MVS System Commands.

Requesting dumps from multiple systems

In a sysplex, you probably need dumps from more than one system to collect all of the problem data. These dumps must be requested at the same time. To request multiple dumps, use the following procedures on any of the systems that might be part of the problem:

- Enter a DUMP command with a REMOTE parameter.
- Issue a SDUMPX macro with a REMOTE parameter.
- Create a SLIP trap in an IEASLPxx parmlib member in the shared SYS1.PARMLIB or in the parmlib on each system.
  - Sometimes you cannot predict which system has the problem. Use a ROUTE operator command to activate the traps on all systems with similar configurations. Each trap must include a REMOTE parameter to dump all the other systems that might be involved.

To help you set the requests, the commands and macro can contain the wildcard characters * and ?. Use wildcard characters when an installation has names that form patterns to the systems in the sysplex and to the jobs for associated work.

For example, use wildcard characters * and ? to specify job names. Use TRANS? for the job names TRANS1, TRANS2, and TRANS3 and TRANS* for TRANS1, TRANS12, and TRANS123.
Issuing a macro for SVC dump

To request an SVC dump in an authorized program, use an SDUMP or SDUMPX macro. The system writes the dump in a SYS1.DUMPxx data set or, if specified in the macro, in a user-supplied data set.

For example, to dump the default contents listed in “Contents of SVC dumps” on page 25 to a SYS1.DUMPxx data set, enter the following command:

```
SDUMPX
```

If the dump is written to a user-supplied SVC dump data set, the program provides a data control block (DCB) for the data set, opens the DCB before issuing the SDUMP or SDUMPX macro, and closes the DCB after the dump is written. For a synchronous dump, the close should occur when the system returns control to the requester. For a scheduled dump, the close should occur when the ECB is posted or when the SRB is scheduled.

As another example, to write a synchronous dump to a data set whose DCB address is in register 3, you would specify the following command:

```
SDUMPX DCB=(3)
```

See z/OS MVS Programming: Authorized Assembler Services Guide for information about requesting a scheduled SVC dump and a synchronous SVC dump.

Operator activities

From a console with master authority, the operator can enter either of the following commands:

- **DUMP operator command.**
  The following DUMP operator command will write an SVC dump:

  ```
  DUMP COMM=(MYDUMP1 5-9-88)
  - To a SYS1.DUMPxx data set
  - With a dump title of “MYDUMP1 5-9-88”
  - With the default contents listed in “Contents of SVC dumps” on page 25
  - For a job named MYJOB1
  ```

  The system will respond with the message:

  ```
  * 23 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
  ```

  Ask the operator to reply:

  ```
  REPLY 23,JOBNAME=MYJOB1
  ```

  Note that if the operator replies REPLY 23,U to IEE094D, the system dumps the current address space, which is the master scheduler address space. The operator must use an ASID, JOBNAME, or TSONAME parameter in the reply to obtain other dumps.

  Use the DUMPDS command to produce a scheduled SVC dump.

- **SLIP operator command with an ACTION option of STDUMP, SVCD, SYNCIVC, or TRDUMP.**

  For example, the following SLIP operator command will write an SVC dump:

  ```
  SLIP SET,ACTION=SVCD,ERTYP=PRG,JOBNAME=MYJOB1,END
  ```

  The SLIP command produces a scheduled SVC dump.
### Operator command in an IEASLPxx parmlib member

The installation can also place SLIP operator commands in IEASLPxx parmlib members to produce an SVC dump. When a command is needed, the operator dynamically sets the IEASLPxx member containing the needed SLIP command. The installation can place SLIP commands that request different types of SLIP traps in different IEASLPxx members.

See [z/OS MVS System Commands](z/OS MVS System Commands) for details about the DUMP and SLIP operator commands. See [z/OS MVS Initialization and Tuning Reference](z/OS MVS Initialization and Tuning Reference) for information about the IEASLPxx member.

### Operator command in an IEADMCxx parmlib member

IEADMCxx enables you to supply DUMP command parameters through a parmlib member. IEADMCxx enables the operator to specify the collection of dump data by issuing a DUMP command, indicating the name of the parmlib member and any symbolic substitution variables.

Since z/OS Release 2, a number of sample DUMP command parmlib members are delivered in SYS1.SAMPLIB. Modifications to, for example, system names, address space names, and so on, are potentially required to deal with each installation's specific requirements. In some cases, substitution variables are supplied to provide an example of how an installation could use them.

**Note:** The length of the sample substitution variable name might not suffice for the values that are actually used at a particular site. For example, &job in IEADMCAS can only accommodate job names that are 4 characters or less. To accommodate up to eight characters, change the variable to something like &thisjob. Therefore, to take advantage of these members, save a modified copy into a data set in your parmlib concatenation.

Table 7 summarizes the sample dump commands that z/OS supplies in SYS1.SAMPLIB.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Suspected Problem Area</th>
<th>Areas Dumped</th>
<th>Symbolics Used</th>
<th>Remote Option Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEADMCAR</td>
<td>APPC</td>
<td>APPC transaction environment, including RRS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEADMCAS</td>
<td>Shared Tape</td>
<td>Allocation Autoswitch and XCF, with affected job</td>
<td>&amp;job</td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCCA</td>
<td>Catalog</td>
<td>Catalog address space and associated areas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEADMCCN</td>
<td>Console</td>
<td>CONSOLE address space and its data spaces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEADMCCP</td>
<td>CP/SM</td>
<td>CICSplex SM environment on all systems in the sysplex. This includes the CAS, CMAS and EYU address spaces.</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCD2</td>
<td>DB2® distributed transactions</td>
<td>DB2/RRS environment</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCJ2</td>
<td>JES2</td>
<td>JES2/XCF environment on current and specified system</td>
<td>&amp;SYSTM</td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCLC</td>
<td>Logger/CICS</td>
<td>System Logger, RLS and CICS®</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 7. Sample operator DUMP command members in SYS1.SAMPLIB (continued)

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Suspected Problem Area</th>
<th>Areas Dumped</th>
<th>Symbolics Used</th>
<th>Remote Option Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEADMCLG</td>
<td>Logger/GRS</td>
<td>System Logger and GRS on all systems in the sysplex</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCLS</td>
<td>General Logger Problem</td>
<td>Logger, XCF, ALLOC, CATALOG, GRS, DFHSM, and SMS along with specified structure, on all systems in the sysplex.</td>
<td>&amp;STRNAME &amp;STRNAME2</td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCLX</td>
<td>Logger/XCF</td>
<td>System Logger and XCF on all systems in the sysplex</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCOE</td>
<td>OMVS</td>
<td>OMVS</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCRL</td>
<td>RRS</td>
<td>RRS and the System Logger on all systems in the sysplex</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCRRL</td>
<td>RRS</td>
<td>RRS and its data spaces</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCSQ</td>
<td>IMS™</td>
<td>IMS Shared Queues environment (IMS Control region, CL/I SAS Region, DBRC Region, and all of the CQS address spaces connected to the shared queues</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCCTA</td>
<td>TCP/IP</td>
<td>TCP/IP, along with the specified application</td>
<td>&amp;tcp &amp;appl</td>
<td></td>
</tr>
<tr>
<td>IEADMCCTC</td>
<td>TCP/IP</td>
<td>TCP/IP, along with the Comm Server address space</td>
<td>&amp;tcp</td>
<td></td>
</tr>
<tr>
<td>IEADMCCTI</td>
<td>TCP/IP</td>
<td>TCP/IP and its data space</td>
<td>&amp;tcp</td>
<td></td>
</tr>
<tr>
<td>IEADMCTO</td>
<td>TCP/IP</td>
<td>TCP/IP and OMVS</td>
<td>&amp;tcp</td>
<td></td>
</tr>
<tr>
<td>IEADMCVC</td>
<td>Comm Server</td>
<td>VTAM® and TCP/IP, with the TCPIP and VTAM data spaces</td>
<td>&amp;tcp &amp;net</td>
<td></td>
</tr>
<tr>
<td>IEADMCVG</td>
<td>VTAM GR</td>
<td>VTAM Generic Resources environment, with its CF structure</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCVT</td>
<td>Comm Server</td>
<td>VTAM and TCP/IP (address spaces only)</td>
<td>&amp;tcp &amp;net</td>
<td></td>
</tr>
<tr>
<td>IEADMCVV</td>
<td>VTAM</td>
<td>VTAM and the VIT data space</td>
<td>&amp;net</td>
<td></td>
</tr>
<tr>
<td>IEADMCLWL</td>
<td>WLM</td>
<td>WLM on all systems in the sysplex</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCLWS</td>
<td>Web server</td>
<td>HTTP web server with OMVS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEADMCLWT</td>
<td>Web server</td>
<td>HTTP web server and TCP/IP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEADMCXI</td>
<td>IRLM</td>
<td>XCF and IRLM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEADMCXI1</td>
<td>IRLM</td>
<td>XCF and IRLM on all systems in the sysplex</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>IEADMCZM</td>
<td>CEA</td>
<td>CEA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

1. When specifying parmlib members containing symbolic parameters, you must specify the symbolic and substitution value using the SYMDEF keyword.
2. The dump command indicated by each row with a “Y” in the “Remote Option Used” column results in a multi-system dump.

**Making a dump data set available**

An SVC dump is taken to an SVC dump data set, either specified on the DCB parameter of the SDUMP or SDUMPX macro, available as SYS1.DUMPxx, or automatically allocated. SVC dump processing issues message IEA793A when the...
dump has been captured but there are no available dump data sets. When a SYS1.DUMPxx data set is not available, the operator has the option either of deleting the captured dump by replying D or making another dump data set available to SVC dump processing. To make another dump data set available, the operator uses the DUMPDS command.

For example, you can use a DUMPDS command to make a dump data set available to SVC dump.

- System message:
  * 16 IEA793A NO SVC DUMP DATA SETS AVAILABLE FOR DUMPID=dumpid FOR JOB (*MASTER*).
  * 16 IEA793A USE THE DUMPDS COMMAND OR REPLY D TO DELETE THE CAPTURED DUMP

- Operator reply:
  DUMPDS ADD, DSN=02

See z/OS MVS System Commands for information about the DUMPDS command. See z/OS MVS System Messages, Vol 6 (GOS-IEA) for information about message IEA793A.

### Determining current SVC dump options and status

An operator can determine the current dump options and the SYS1.DUMPxx data sets that contain SVC dumps.

#### Dump mode and options

Use a DISPLAY DUMP operator command to get the dump mode and options in effect for SVC dumps and SYSABEND, SYSDUMP, and SYSUDUMP dumps. The system displays the mode and options in message IEE857I.

For example, to determine the mode and options, enter:

```
DISPLAY DUMP, OPTIONS
```

If the options listed are not the ones desired, have the operator use a CHNGDUMP operator command to change them.

See z/OS MVS System Commands for the DISPLAY for more information about dump modes. See z/OS MVS System Messages, Vol 7 (IEB-IEE) for more information about IEE857I.

#### Status of SYS1.DUMPxx data sets

Use a DISPLAY DUMP operator command to get the status of all defined SYS1.DUMPxx data sets on direct access. The system displays the status in message IEE852I or IEE856I. The message indicates the full and available data sets.

For example, to determine the status of SYS1.DUMPxx data sets, enter:

```
DISPLAY DUMP, STATUS
```

See z/OS MVS System Commands for information about the CHNGDUMP and DISPLAY commands. For a description of these messages, use LookAt or see MVS System Messages.

### Finding SVC dumps

An operator can search the current SYS1.DUMPxx data sets for the SVC dump for a particular problem. To select the dump, use the title and time or use the dump symptoms. The operator can also find a dump that has been captured in virtual storage but has not been written to a data set.
Title and time of SVC dump(s)

Use one of the following to get the titles and times for SVC dumps:

- A DISPLAY DUMP operator command. The system displays the titles and times in message IEE853I.

  You can use the DISPLAY command to find title and time information.

  - To see the titles and times for the dumps in SYS1.DUMP08 and SYS1.DUMP23, without displaying any automatically allocated dump data sets, enter:
    
    ```
    DISPLAY DUMP,TITLE,DSN=(08,23)
    ```

  - To display the titles of the most recently automatically allocated dump data set and all pre-allocated dump data sets, enter:
    
    ```
    DISPLAY DUMP,TITLE
    ```

    or:

    ```
    DISPLAY DUMP,TITLE,DSN=ALL
    ```

  - To display the titles of the last 5 most recently allocated dump data sets, enter:
    
    ```
    DISPLAY DUMP,TITLE,AUTODSN=5
    ```

  - To see the dump titles for all captured dumps, enter:
    
    ```
    DISPLAY DUMP,TITLE,DUMPID=ALL
    ```

- An IPCS SYSDSCAN command entered at a terminal by a TSO/E user. IPCS displays the titles and times at the terminal.

  For example, to use IPCS to find the dump titles and times for the dumps in SYS1.DUMP08 and SYS1.DUMP23, enter the following IPCS command:

  ```
  SYSDSCAN 08
  SYSDSCAN 23
  ```

  See z/OS MVS IPCS Commands for information about SYSDSCAN.

If a data set listed in either command is empty or undefined, the system issues a message to tell why the title is not available.

Symptoms from SVC dumps

Use a DISPLAY DUMP operator command to get the symptoms from SVC dumps in SYS1.DUMPxx data sets on direct access or from SVC dumps that have been captured in virtual storage. The system displays the following symptoms in message IEE854I:

- Dump title or a message telling why the title is not available
- Error id consisting of a sequence number, the processor id, the ASID for the failing task, and the time stamp
- System abend code
- User abend code
- Reason code
- Module name
- Failing CSECT name
- Program status word (PSW) at the time of the error
- Interrupt length code in the system diagnostic work area (SDWA)
- Interrupt code in the SDWA
- Translation exception address in the SDWA
- Address of the failing program in the SDWA
- Address of the recovery routine in the SDWA
SVC dump

- Registers at the time of the error saved in the SDWA

For example, you can issue DISPLAY DUMP to view various types of symptoms:
- To see symptoms from the dump in the SYS1.DUMP03 data set without displaying any automatically allocated data sets, enter:
  
  ```
  DISPLAY DUMP,ERRDATA,DSN=03
  ```
- To see symptoms from the most recently automatically allocated dump data set and all pre-allocated dump data sets, enter:
  
  ```
  DISPLAY DUMP,ERRDATA
  ```
  or:
  
  ```
  DISPLAY DUMP,ERRDATA,DSN=ALL
  ```
- To see symptoms from the last 5 most recently allocated dump data sets, enter:
  
  ```
  DISPLAY DUMP,ERRDATA,AUTODSN=5
  ```
- To see symptoms from the captured dump identified by DUMPID=005, enter:
  
  ```
  DISPLAY DUMP,ERRDATA,DUMPID=005
  ```

Using the DISPLAY DUMP,ERRDATA command (see Figure 4), you can retrieve basic information about the dump without having to format the dump or read through the system log. Message IEE854I displays the error data information, including the PSW at the time of the error, the system abend code and reason code, and the module and CSECT involved.

```
  d d,errdata
  IEE854I 13.01.25 SYS1.DUMP ERRDATA 745
  SYS1.DUMP DATA SETS AVAILABLE=001 AND FULL=001
  CAPTURED DUMPS=0000, SPACE USED=00000000M, SPACE FREE=00000000M
  DUMP00 TITLE=ABDUMP ERROR,COMPON=ABDUMP,COMPID=5752-SCDMP, 
  ISSUER=IEAVTABD
  DUMP TAKEN TIME=13.01.02 DATE=09/27/1996
  ERRORID=SEQ00010 CPU0000 ASID0010 TIME=13.00.55
  SYSTEM ABEND CODE=0C1 REASON CODE=00000001
  MODULE=IGC0101C CSECT=IEAVTABD
  PSW AT TIME OF ERROR=070C0000 823FE0F6 ILC=2 INT=01
  TRANSLATION EXCEPTION ADDR=008E1094
  ABENDING PROGRAM ADDR=******** RECOVERY ROUTINE=ADRECOV
  GPR 0-3 00000000 7F6EBAE4 023FFFF6 023F2BFF
  GPR 4-7 00000000 7F6EBAE4 023FFFF6 023F2BFF
  GPR 8-11 00000000 7F6E9338 7F6EBAE8 7F6EBAE8
  GPR 12-15 00000000 7F6EBAE8 7F6EBAE8 00000000
  NO DUMP DATA AVAILABLE FOR THE FOLLOWING EMPTY SYS1.DUMP DATA SETS: 01
```

Figure 4. Example: Output from DISPLAY,DUMP ERRDATA command

See [z/OS MVS System Commands](http://www.ibm.com/support/docview.wss?uid=swg21311717) for information about the DISPLAY operator command.

Printing, viewing, copying, and clearing a pre-allocated or SYS1.DUMPxx data set

SVC dumps are unformatted when created. Use IPCS to format a dump and then view it at a terminal or print it.

After the dump has been copied to a permanent data set, use a DUMPDS operator command to clear the data set so that the system can use the data set for another dump. Then use IPCS to view the copy.
SVC dump

You can copy a dump that was written to tape so that you can view the dump through IPCS more efficiently.

For a pre-allocated data set or a SYS1.DUMPxx data set, the JCL shown in Figure 5 does the following:

- Uses the SVC dump in the SYS1.DUMP00 data set. The IPCSDUMP DD statement identifies this data set.
- Copies the dump from the SYS1.DUMP00 data set to the data set identified in the DUMPOUT DD statement. To use this example, change the DUMPOUT DD statement to give the DSNAMES for the desired location.
- Clears the SYS1.DUMP00 data set so that it can be used for a new dump.
- Deletes the IPCS dump directory in the DELETE(DDIR) statement. This statement uses the USERID of the batch job in the directory identification.
- Allocates the dump directory through the BLSCDDIR statement. The default is volume VSAM01. The example shows VSAM11. Override the default volume with the desired volume.
- Formats the dump using the IPCS subcommands in LIST 0. To use this example, replace the LIST 0 command with the desired IPCS subcommands or a CLIST. See z/OS MVS IPCS User’s Guide for CLISTS.

```
//IPCSJOB JOB
//IPCS EXEC PGM=IKJEFT01,DYNAMNBR=75,REGION=1500K
//SYSPROC DD DSN=SYS1.SBLSCLI0,DISP=SHR
//IPCSDUMP DD DSN=SYS1.DUMP00,DISP=SHR
//DUMPOUT DD DSN=GDG.DATA.SET(+1),DISP=SHR
//SYSUDUMP DD SYSOUT=*
//IPCSTOC DD SYSOUT=*
//IPCSPRNT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DELETE(DDIR) PURGE CLUSTER
BLSCDDIR VOLUME(VSAM11)
IPCS NOPARM
SETDEF DD(IPCSDUMP) LIST NOCONFIRM
LIST 0
COPYDUMP INFILE(IPCSDUMP) OUTFILE(DUMPOUT) CLEAR NOPRINT NOCONFIRM
END
/*
```

Figure 5. Example: JCL to print, copy, and clear an SVC dump data set

Contents of SVC dumps

Unlike ABEND dumps, SVC dumps do not have a parmlib member that establishes the dump options list at system initialization. The IBM-supplied IEACMD00 parmlib member contains a CHNGDUMP operator command that adds the local system queue area (LSQA) and trace data (TRT) to every SVC dump requested by an SDUMP or SDUMPX macro or a DUMP operator command, but not for SVC dumps requested by SLIP operator commands.

The contents of areas in an SVC dump depend on the dump type:

- Scheduled SVC dump: The current task control block (TCB) and request block (RB) in the dump are for the dump task, rather than for the failing task. For additional address spaces in the dump, the TCB and RB are for the dump task.
- Synchronous SVC dump: The current TCB and RB in the dump are for the failing task.
Customizing SVC dump contents

You can customize the contents of an SVC dump to meet the needs of your installation. For example, you might want to add areas to be dumped, reduce the dump size, or dump Hiperspaces. In most cases, you will customize the contents of an SVC dump or summary dump through the SDATA parameter of the SDUMP or SDUMPX macro or through operator commands.

Reducing dump size: To obtain a smaller dump that does not have all the usual defaults, code a NODEFAULTS option in the SDATA parameter of the SDUMP or SDUMPX macro. With the NODEFAULTS option, the dump contains:

• Certain default system areas needed by IPCS for dump analysis
• Areas requested on the SDUMP or SDUMPX macro

Hiperspaces: SVC dumps do not include Hiperspaces. To include Hiperspace™ data in an SVC dump, you have to write a program to copy data from the Hiperspace into address space storage that is being dumped.

Adding areas: If the dump, as requested, will not contain all the needed areas, see one of the following for ways to add the areas:

• “Customized contents using the SDATA parameter”
• “Contents of summary dumps in SVC dumps” on page 30
• “Customizing contents through operator commands” on page 33

Customized contents using the SDATA parameter

The IBM-supplied default contents and the contents available through customization are detailed in Table 8 on page 27. The tables show dump contents alphabetically by the parameters that specify the areas in the dumps. Before requesting a dump, decide what areas will be used to diagnose potential errors. Find the areas in the tables. The symbols in columns under the dump indicate how the area can be obtained in that dump. The symbols are:

C Available on the command that requests the dump
D IBM-supplied default contents
M Available on the macro that requests the dump
P Available in the parmlib member that controls the dump options
X Available on the CHNGDUMP operator command that changes the options for the dump type

blank No symbol indicates that the area cannot be obtained.

Note: System operator commands and assembler macros use the parameters in the table to specify dump contents.

The order of the symbols in the following table is not important.
## Table 8. Customizing SVC dump contents through the SDATA parameter

<table>
<thead>
<tr>
<th>SDATA Parameter Option</th>
<th>Dump Contents</th>
<th>SVC Dump for:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SDUMPX Macro or DUMP Command with SDATA Parameter</td>
</tr>
<tr>
<td>ALLNUC</td>
<td>The DAT-on and DAT-off nucleuses</td>
<td>M C X</td>
</tr>
<tr>
<td>ALLPSA</td>
<td>Prefixed save area (PSA) for all processors</td>
<td>D M X</td>
</tr>
<tr>
<td>COUPLE</td>
<td>Data on cross-system coupling</td>
<td>M C X</td>
</tr>
<tr>
<td>CSA</td>
<td>Common service area (CSA) (that is, subpools 227, 228, 231, 241) and virtual storage for 64-bit addressable memory objects created using one of the following services: • IAR64 REQUEST=GETCOMMON,DUMP=LIKECSA • IARCP64 COMMON=YES,DUMP=LIKECSA • IARST64 COMMON=YES,TYP=PAGEABLE</td>
<td>M C X</td>
</tr>
<tr>
<td>DEFAULTS</td>
<td>Default areas</td>
<td>M X</td>
</tr>
<tr>
<td>GRSQ</td>
<td>Global resource serialization control blocks for the task being dumped: • Global queue control blocks • Local queue control blocks</td>
<td>M C X</td>
</tr>
<tr>
<td>HCAS (HCSAByASID)</td>
<td>High CSA area by ASID</td>
<td>M C X</td>
</tr>
<tr>
<td>HCNO (HCSANoOwner)</td>
<td>High CSA area for which the owner has ended</td>
<td>M C X</td>
</tr>
<tr>
<td>HCSY (HCSASysOwner)</td>
<td>High CSA area that belongs to the SYSTEM</td>
<td>M C X</td>
</tr>
<tr>
<td>IO</td>
<td>Input/output supervisor (IOS) control blocks for the task being dumped: • EXCPD • UCB</td>
<td>D</td>
</tr>
<tr>
<td>LPA</td>
<td>Active link pack area (LPA): module names and contents</td>
<td>M C X</td>
</tr>
<tr>
<td>LSQA</td>
<td>Local system queue area (LSQA) allocated for the address space (that is, subpools 203 - 205, 213 - 215, 223 - 225, 229, 230, 233 - 235, 249, 253 - 255), and virtual storage for 64-bit addressable memory objects created using one of the following services: • IAR64 REQUEST=GETSTOR,DUMP=LIKELSQA • IARCP64 COMMON=NO,DUMP=LIKELSQA • IARST64 COMMON=NO</td>
<td>D M C X</td>
</tr>
<tr>
<td>NOALL</td>
<td>No ALLPSA</td>
<td>M X</td>
</tr>
<tr>
<td>NOALLPSA</td>
<td>No ALLPSA</td>
<td>M X</td>
</tr>
</tbody>
</table>
### Table 8. Customizing SVC dump contents through the SDATA parameter (continued)

<table>
<thead>
<tr>
<th>SDATA Parameter Option</th>
<th>Dump Contents</th>
<th>SVC Dump for:</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SDUMPX Macro or DUMP Command with SDATA Parameter</td>
<td>DUMP Command without SDATA Parameter</td>
<td>SLIP Command with ACTION=SVCD or SYNSVCD</td>
<td>SLIP Command with ACTION=STDUMP</td>
<td>SLIP Command with ACTION=TRDUMP</td>
</tr>
<tr>
<td>NODEFAULTS</td>
<td></td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOPSA</td>
<td>No PSA</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOSQA</td>
<td>No SQA</td>
<td>M C X</td>
<td>X</td>
<td>C</td>
<td>D C</td>
<td>D C</td>
</tr>
<tr>
<td>NOSUM</td>
<td>No SUM</td>
<td>M C X</td>
<td>X</td>
<td>C</td>
<td>D C</td>
<td>D C</td>
</tr>
<tr>
<td>NUC</td>
<td>Minimum default areas needed for IPCS dump analysis and Areas requested on the SDUMP or SDUMPX macro</td>
<td>M C X</td>
<td>X</td>
<td>D C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>PSA</td>
<td>Prefixed save areas (PSA) for the processor at the time of the error or the processor at the time of the dump</td>
<td>D M C X</td>
<td>D X</td>
<td>D C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>RGN</td>
<td>Allocated pages in the private area of each address space being dumped, including subpools 0 - 127, 129 - 132, 203 - 205, 213 - 215, 223 - 225, 229, 236, 237, 244, 249, 251 - 255, and virtual storage for 64-bit addressable memory objects created using one of the following services:</td>
<td>M C X</td>
<td>X</td>
<td>D C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERVERS</td>
<td>Areas added by IEASDUMP SERVERS exits</td>
<td>M C X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQA</td>
<td>System queue area (SQA) allocated (that is, subpools 226, 239, 245, 247, 249) and virtual storage for 64-bit addressable memory objects created using one of the following services:</td>
<td>D M C X</td>
<td>D X</td>
<td>D C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>Summary dump (See &quot;Contents of summary dumps in SVC dumps&quot; on page 10)</td>
<td>D M C X</td>
<td>D X</td>
<td>D C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>SWA</td>
<td>Scheduler work area (SWA) (that is, subpools 236 and 237)</td>
<td>M C X</td>
<td>D X</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>TRT</td>
<td>System trace, generalized trace facility (GTF) trace, and master trace, as available</td>
<td>D M C X</td>
<td>D X</td>
<td>D C</td>
<td>D C</td>
<td>D C</td>
</tr>
</tbody>
</table>
Table 8. Customizing SVC dump contents through the SDATA parameter (continued)

<table>
<thead>
<tr>
<th>SDATA Parameter Option</th>
<th>Dump Contents</th>
<th>SDUMPX Macro or DUMP Command with SDATA Parameter</th>
<th>DUMP Command without SDATA Parameter</th>
<th>SLIP Command ACTION= SVCD or SYNC(SVCD)</th>
<th>SLIP Command ACTION= STDUMP</th>
<th>SLIP Command ACTION= TRDUMP</th>
<th>DUMP Command SVCDEMPRN = YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default system data</td>
<td>Instruction address trace, if available</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Default system data</td>
<td>Nucleus map and system control blocks, including:</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• ASCB for each address space being dumped</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• ASVT</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• Authorization table for each address space</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• CVT, CVT prefix, and secondary CVT (SCVT)</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• Entry tables for each address space</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• GDA</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• JSAB of each address space being dumped</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• Linkage stack</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• Linkage table for each address space</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• PCCA and the PCCA vector table</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• TOT</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• TRVT</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>• UCB</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Default system data</td>
<td>DFP problem data, if DFP Release 3.1.0 or a later release is installed</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Default system data</td>
<td>Storage for the task being dumped and program data for all of its subtasks</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Default system data</td>
<td>Storage: 4 kilobytes before and 4 kilobytes after the address in the PSW at the time of the error</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Default system data</td>
<td>SUBTASKS: Storage for the task being dumped and program data for all of its subtasks</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

The following table describes how HCSAByASID, HCSANoOwner, and HCSASysOwner affect the CSA storage that is captured in an SVC dump:

Table 9. Affects on the CSA storage captured in an SVC dump

<table>
<thead>
<tr>
<th>Specified SDATA option or options</th>
<th>CSA storage that is included in the dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSA</td>
<td>All above the bar and below the bar CSA storage</td>
</tr>
<tr>
<td>CSA, HCSAByASID, HCSANoOwner, HCSASysOwner</td>
<td>All below the bar CSA storage, high virtual CSA storage that is owned by the ASIDs that are included in the dump, high virtual CSA storage for which the owner has ended, and high virtual CSA storage that belongs to the SYSTEM. The dump does not include high virtual CSA storage that is owned by the ASIDs that are excluded from the dump.</td>
</tr>
</tbody>
</table>
Table 9. Affects on the CSA storage captured in an SVC dump (continued)

<table>
<thead>
<tr>
<th>Specified SDATA option or options</th>
<th>CSA storage that is included in the dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCSAByASID, HCSANoOwner, HCSASysOwner</td>
<td>All high virtual CSA storage that is owned by the ASIDs that are included in the dump, high virtual CSA storage for which the owner has ended, and high virtual CSA storage that belongs to the SYSTEM</td>
</tr>
<tr>
<td>(Neither CSA nor any of the HCSAxxx options)</td>
<td>No below the bar CSA storage is included in the dump.</td>
</tr>
<tr>
<td>None of the CSA storage is included in the dump.</td>
<td>None of the CSA storage is included in the dump.</td>
</tr>
</tbody>
</table>

Contents of summary dumps in SVC dumps

Request a summary dump for two reasons:
1. The SUM or SUMDUMP parameters request many useful, predefined areas with one parameter.
2. The system does not write dumps immediately for requests from disabled, locked, or SRB-mode programs. Therefore, system activity destroys much needed diagnostic data. When SUM or SUMDUMP is specified, the system saves copies of selected data areas at the time of the request, then includes the areas in the SVC dump when it is written.

Use SDUMP or SDUMPX macro parameters to request different types of summary dumps, as follows:

- **Disabled summary dump**: This summary dump saves data that is subject to rapid and frequent change before returning control to the scheduled dump requester. Because the system is disabled for this dump, the dump includes only data that is paged in or in DREF storage. Specify BRANCH=YES and SUSPEND=NO on an SDUMP or SDUMPX macro to obtain a disabled summary dump.

- **Suspend summary dump**: This summary dump also saves data that is subject to rapid and frequent change before returning control to the scheduled dump requester. This dump, however, can save pageable data. To obtain a suspend summary dump, do the following:
  - For an SDUMP or SDUMPX macro, specify BRANCH=YES and SUSPEND=YES
  - For an SDUMPX macro, specify BRANCH=NO for a scheduled dump with SUMLSTL parameter

- **Enabled summary dump**: This summary dump does not contain volatile system information. The system writes this summary dump before returning control to the dump requester; the summary information is saved for each address space being dumped. To obtain an enabled summary dump, do the following:
  - For an SDUMP or SDUMPX macro, specify BRANCH=NO.
  - For a SLIP operator command, do not specify an SDATA parameter or specify SUM in an SDATA parameter.
  - For a DUMP operator command, do not specify an SDATA parameter or specify SUM in an SDATA parameter. Note that this dump does not contain data that the system creates when it detects a problem; for example, this dump would not contain a system diagnostic work area (SDWA).
In Table 10, an S indicates that the area is included in the summary dump for the dump type.

**Table 10. Customizing SVC dump contents through summary dumps**

<table>
<thead>
<tr>
<th>Summary dump contents</th>
<th>Disabled</th>
<th>Suspend</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address space identifier (ASID) record for the address space of the dump task</td>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td><strong>Control blocks</strong> for the failing task, including:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• For task-mode dump requesters:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Address space control block (ASCB)</td>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>– Request blocks (RB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>– System diagnostic work areas (SDWA) pointed to by the recovery termination management 2 work areas (RTM2WA) associated with the task control block (TCB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>– TCB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Extended status block (XSB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• For service request block (SRB)-mode dump requesters:</td>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>– ASCB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Suspended SRB save area (SSRB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>– SDWA used for dump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>– XSB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Control blocks</strong> for the recovery termination manager (RTM):</td>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>• RTM2WA associated with all TCBs in the dumped address space</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• RTM2WA associated with the TCB for the dump requester</td>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td><strong>Cross memory status record</strong> and, if the dump requester held a cross memory local (CML) lock, the address of the ASCB for the address space whose local lock is held</td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td><strong>Dump header</strong>, mapped by AMDDATA</td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>For the AMDDATA mapping, see z/OS MVS Data Areas in <a href="http://www.ibm.com/systems/z/os/zos/bkserv/">z/OS Internet library</a></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Functional recovery routine (FRR) stack</strong> for the current processor</td>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td><strong>Interrupt handler save area (IHSA)</strong> for the home address space or, if a CML is held, for the address space whose local lock is held</td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td><strong>Logical communication area (LCCA)</strong> for each active processor</td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>In dumps requested by AR-mode callers, the LCCA includes the AR mode control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Physical configuration communication area (PCCA)</strong> for each active processor</td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td><strong>Program call link stack elements (PCLINK) stack elements:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Pointed to by PSASEL</td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>• Pointed to by the XSB associated with the IHSA in the dump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Pointed to by the SSRB and XSB for the SRB-mode dump requester</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Associated with the suspended unit of work</td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td><strong>Prefixed save area (PSA)</strong> for each active processor</td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>
### SVC dump

#### Table 10. Customizing SVC dump contents through summary dumps (continued)

<table>
<thead>
<tr>
<th>Summary dump contents</th>
<th>Disabled</th>
<th>Suspend</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Save areas</strong> of register contents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDWA associated with the failure of a system routine</td>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td><strong>Storage</strong>: The storage ranges and ASIDs requested in parameters on the SDUMP or SDUMPX macro</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td><strong>Storage</strong>: 4 kilobytes before and 4 kilobytes after:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The address in the program status word (PSW)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• All valid unique addresses in the registers saved in the IHSA shown in the dump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• All valid unique addresses in the registers saved in the SDWA shown in the dump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Instruction counter values of the external old PSW, program check old PSW, I/O old PSW, and restart old PSW saved in the PSAs of each active processor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Storage</strong>: 4 kilobytes before and 4 kilobytes after:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• All valid unique addresses in the registers saved in the SDWA shown in the dump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• All valid unique addresses in the registers in the dump requester's register save area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• All valid unique addresses in the PSWs in all SDWAs shown in the dump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Storage</strong>: 4 kilobytes before and 4 kilobytes after:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• All valid unique addresses in the PSWs in the RTM2WAs shown in the dump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• All valid unique addresses in the registers in the RTM2WAs shown in the dump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Storage</strong>: When a PSWREGS parameter is specified on the SDUMP or SDUMPX macro, 4 kilobytes before and 4 kilobytes after:</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>• The address in the PSW, if supplied in the PSWREGS parameter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The address in the general purpose registers, if supplied in the PSWREGS parameter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The storage dumped is from the primary and secondary address spaces of the program issuing the SDUMP or SDUMPX macro. The control registers, if supplied in the PSWREGS parameter, are used to determine the primary and second address spaces.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If access registers are also provided and the PSW indicates AR ASC mode, the access registers will also be used to locate the data.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Supervisor control blocks</strong>:</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>• Current linkage stack</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Primary address space number (PASN) access list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Work unit access list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Vector Facility control blocks</strong>:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global, CPU, and local work/save area vector tables (WSAVTG, WSAVTC, and WSAVTL) and work/save areas pointed to by addresses in the tables</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>XSB associated with the IHSA in the dump</strong>:</td>
<td>S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For information about control blocks listed in the above table, see [z/OS MVS Data Areas in z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/).
Customizing contents through operator commands

The dump options list for SVC dumps can be customized through a DUMP operator command by all the ways shown in Table 11.

Note: The contents of SVC dumps requested by SLIP operator commands are controlled only by the SLIP operator command. They are not affected by the IEACMD00 parmlib member or the CHNGDUMP command.

Nucleus areas in dumps: Dump options control the parts of the nucleus that appear in a dump. A diagnostician seldom needs to analyze all the nucleus. An installation can eliminate nucleus areas from dumps. If the IBM-supplied defaults are used:

- SVC dump for a SLIP operator command with ACTION=SVCD contains the read/write DAT-on nucleus
- SVC dump for an SDUMP or SDUMPX macro contains the nucleus map and certain control blocks

If no nucleus changes have been made, an installation should obtain one copy of the DAT-off nucleus to use with all dumps. To obtain this nucleus, enter a DUMP operator command with SDATA=ALLNUC and no other SDATA options. The nucleus does not change from one IPL to another, so one dump can be used again and again.

DAT, dynamic address translation, is the hardware feature that enables virtual storage. In the DAT-on part of the nucleus, the addresses are in virtual storage; in the DAT-off part of the nucleus, the addresses are in central storage.

### Table 11. Customizing SVC dump contents through operator commands

<table>
<thead>
<tr>
<th>Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
</table>
| Use SDATA=NODEFAULTS on SDUMP or SDUMPX macro | Change occurs: At dump request  
What changes: Excludes the following SDATA default options currently in effect:  
• ALLPSA  
• SQA  
• SUMDUMP  
• IO  
• From all CHNGDUMP operator commands entered through the IEACMD00 parmlib member or through the console  
Exclusion is only for the dump being requested.  
Note that certain default system areas are not excluded; these areas are required for IPCS dump analysis.  
The CHNGDUMP operator command can override the NODEFAULTS option. | To minimize the amount of default data in the dump, code in the program:  
SDUMPX SDATA=NODEFAULTS |
### Table 11. Customizing SVC dump contents through operator commands (continued)

<table>
<thead>
<tr>
<th>Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
</table>
| **Replacing CHNGDUMP operator command in IEACMD00 parmlib member** | Change occurs: At system initialization  
What changes: This command establishes the IBM-supplied dump options for SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands; see "Contents of SVC dumps" on page 25 for the list. | To add the link pack area (LPA) to all SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands, while keeping the local system queue area (LSQA) and trace data, add the following command to IEACMD00:  
CHNGDUMP SET,SDUMP=(LPA) |
| **Entering CHNGDUMP operator command with SDUMP parameter on a console with master authority** | Change occurs: Immediately when command is processed  
What changes:  
For the ADD mode: CHNGDUMP options are added to the current SVC dump options list and to any options specified in the macro or operator command that requested the dump. The options are added to all SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands until another CHNGDUMP SDUMPX operator command is entered.  
For the OVER mode: CHNGDUMP options are added to the current SVC dump options list. The system ignores any options specified in the macro or operator command that requested the dump. The options override all SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands until a CHNGDUMP SDUMPX,ADD operator command is entered.  
For the DEL option: CHNGDUMP options are deleted from the SVC dump options list.  
When more than one CHNGDUMP operator command with SDUMPX is entered, the effect is cumulative. | To add the LPA to all SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands until changed by another CHNGDUMP SDUMP, enter:  
CHNGDUMP SET,SDUMP=(LPA)  
To add the CHNGDUMP SDUMPX options list to all SVC dumps:  
CHNGDUMP SET,SDUMP,ADD  
To override all SVC dumps with the CHNGDUMP SDUMPX options list:  
CHNGDUMP SET,SDUMP,OVER  
To remove LPA from the SDUMPX options list:  
CHNGDUMP DEL,SDUMP=(LPA) |
| **Using an operator command parameter.**  
Parameters on the DUMP operator command specify the contents for the dump being requested. | Change occurs: At dump request  
What changes: The DUMP operator command parameter options are added to the dump options list, but only for the dump being requested. | To add ALLNUC to this SVC dump, enter:  
DUMP COMM=(MYDUMP1 5-9-88)  
The system issues a message:  
* 23 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND  
Enter in reply:  
REPLY 23,JOBNAME=MYJOB1,SDATA=(ALLNUC),END |
Tailoring SVC dumps

Sometimes servers retain client-related data in address spaces and dataspaces other than the client's, which means this data will not be in the dump. For this reason, server code can modify the contents of an SVC dump to provide additional problem determination data by creating a tailored SVC dump exit. This feature allows a requestor to specify a dump request without identifying related server address, dataspaces, and storage areas, which could be unknown and dynamic in nature.

The server code provider can create these SVC dump exits without modifying module IEAVTSXT. The CSVDYNEX macro identifies the exit load module and associates it with the IEASDUMP:SERVER resource. The exit is allowed to scan the current dump request and determines if data should be added to the dump. The data is added to the dump by identifying it in the appropriate SDMSE_OUTPUT area. For additional details, see IEASDUMP:SERVER Dynamic Exit Processing in z/OS MVS Programming: Authorized Assembler Services Guide.

The tailored SVC dump exits are not called in any particular order. To ensure that the current requests are presented to an exit, the dump request is updated between exit invocations. If an exit adds data to the dump, every exit is re-invoked until no additional changes are made. Because of the additional processing required, tailored SVC dump exits do not receive control by default for SDUMPX macro requests. To cause the exit processing to take place, you must specify SDATA=SERVERS in the SDUMPX macro.

SDATA=SERVERS is in force for all operator Dump and SLIP SVC dump requests.

Analyzing summary SVC dumps

The SUMDUMP or SUM option on the SDUMP or SDUMPX macro causes SVC dump to capture a summary dump. Two types of information are captured in summary dumps. First, index data for storage is captured. This index data can be formatted using the IPCS VERBX SUMDUMP command. The second type of information captured is the storage itself. Storage captured by summary dump processing can be viewed using IPCS by specifying the SUMDUMP option (for example, IPCS LIST 00003000 SUMDUMP). IBM strongly recommends that you view the SUMDUMP output prior to investigating the usual portions of the dump. The SUMDUMP option provides different output to SDUMPX branch entries and SVC entries to SDUMP. For example, data included for branch entries to SDUMPX include PSA, LCCA, and PCCA control blocks, and data recorded for SVC entries to SDUMPX include RTM2WA control blocks. Each summary dump index record, when formatted using the IPCS VERBX SUMDUMP command, is displayed as “---tttt--- range-start range-end range-iasid range-attributes”. The range-attributes include a value of INCOMP, which means that some or all of the areas represented by the specified range may not be in the dump.

Figure 6 on page 36 is an example format using the IPCS VERBX SUMDUMP command. The summary dump is formatted by the IPCS VERBEXIT SUMDUMP subcommand and has an index which describes what the summary contains. Summary dumps are not created for dumps taken with the DUMP command. Only dumps created by the SDUMP or SDUMPX macro contain summary dumps.
During SVC dump processing, the system sets some tasks in the requested address space non-dispatchable; non-dispatchable tasks in the dump may have been dispatchable at the time of the problem.

Figure 7 is a partial example of a summary dump using the IPCS VERBX SUMDUMP command.

For more information about the record ID values, see the SMDLR and SMDXR control blocks in z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/)
SUMDUMP output for SVC-Entry SDUMPX

For an SVC entry, the storage captured in a summary dump can contain information that is not available in the remainder of the SVC dump if options such as region, LSQA, nucleus, and LPA were not specified in the dump parameters.

For each address space dumped, a summary dump index record is written with the ASID, plus the jobname and stepname for the last task created in the address space. The SUMDUMP output contains RTM2 work areas for tasks in address spaces that are dumped. Many of the fields in the RTM2WA provide valuable debugging information.

The summary dump data is dumped in the following sequence:
1. The ASID record is dumped for the address space.
2. The SUMLIST/SUMLSTA/SUMLSTL/SUMLIST64 ranges and the PSWREGS, parameter list and data ID, data are dumped next. These contain information that is helpful in debugging the problem, and should be examined carefully.
3. All RTM2 work areas pointed to by all TCBs in this address space are dumped.
4. An address range table is built containing the following ranges, pointed to by the RTM2WA:
   - 4K before and after the PSW at the time of error (RTM2NXT1)
   - 4K before and after each register at the time of error (RTM2EREG).
Duplicate storage is eliminated from this address range table to reduce the amount of storage dumped.

SUMDUMP output for branch-entry SDUMPX

For branch entry to SDUMP, there are two types of summary dumps:
- Disabled summary dump - which performs the summary dump with the system disabled for interruptions. This means that all data to be dumped must be paged in at the time of the summary dump.
- Suspend summary dump - which is taken in two parts. The first part is similar to the disabled summary dump and dumps some of the global system control blocks. The second part runs with the system enabled for interruptions. This allows data to be dumped that is currently paged out, but was going to be modified by the recovery routine that requested SVC dump processing.

The SUMDUMP output for a branch entry to SVC dump might not match the data that is at the same address in the remainder of the dump. The reason for this is that SUMDUMP is taken at the entry to SVC dump while the processor is disabled for interruptions. The system data in the remainder of the dump is often changed because other system activity occurs before the dump is complete. The SUMDUMP output follows a header that contains the ASID of the address space from which the data was obtained.

The following conditions can occur that prevent SDUMPX from taking a disabled or suspend summary dump.
- The system is not able to obtain the necessary locks to serialize the real storage buffer (RSB).
- The system is in the process of modifying the storage queues and cannot satisfy the request for a RSB.
- No frames are available for a RSB.
- SVC dump encounters an error while holding serialization for the RSB.
A critical frame shortage causes the system to steal the pages of the RSB.

The SVC dump timer disabled interruption exit determines that SVC dump has failed and frees the RSB.

Analyzing disabled summary dumps

For disabled summary dumps, records are dumped in the following order:

1. If a suspend summary dump was requested but could not be taken, the system attempts to obtain a disabled summary dump. If this occurs, an error record is written to that effect. If the system is unable to obtain a suspend summary dump and a disabled summary dump, then no summary data is available for the dump.

2. The XMEM ASID record is written that gives the ASID that is home, primary, secondary, and CML (if the CML lock is held).

3. The SUMLIST/SUMLSTA/SUMLSTL/SUMLIST64 address ranges and the PSWREGS data are dumped.

4. The PSA, PCCA, LCCA, and LCCX for each processor are dumped.

5. The current PCLINK stack (pointed to by PSASEL) is dumped (if it exists).

6. If this is a SLIP request for a dump (ACTION=SVCD), then the SLIP reg/PSW area (pointed to by the SUMDUMP parameter list SDURGPSA) is dumped. The following address ranges are added to the address range table:
   - 4K before and after the PSW address at the time of the SLIP trap.
   - 4K before and after each address in the registers at the time of the SLIP trap.

Duplicate storage is eliminated from this address range table to reduce the amount of data written to the dump data set.

Note that if the primary and secondary ASIDs are different, the above address ranges are added to the table for both ASIDs.

7. The IHSA is dumped along with its associated XSB and PCLINK stack. The PSW and register addresses from the IHSA are added to the range table. This causes 4K of storage to be dumped around each address.

8. The caller’s SDWA is dumped, if one exists. The PSW and register addresses from the SDWA are added to the range table. This causes 4K of storage to be dumped around each address.

9. The addresses in the address range table are dumped.

10. The super FRR stacks are dumped.

11. The global, local, and CPU work save area (WSA) vector tables are dumped. The save areas pointed to by each of these WSA vector tables are also dumped.

12. 4K of storage on either side of the address portion of the I/O old PSW, the program check old PSW, the external old PSW, and the restart old PSW saved in the PSA for all processors, are dumped.

Analyzing suspend summary dumps

For suspend summary dumps, records are dumped in the following order:

1. The ASID: the PSA, PCCA, LCCA records, the IHSA, XSB, and the PCLINK stack, are all dumped with the system disabled in the same way they are dumped in steps 2, 4, and 5 for the disabled summary dump.

At this point, an SRB is scheduled to the DUMPSRV address space and the current unit of work (SDUMP’s caller) is suspended by using the STOP service.
SVC dump

Data dumped at this point does not have to be paged in because the system is enabled. Cross memory functions are used to gain access to data in the caller's address space.

2. The SUMLIST/SUMLSTA/SUMLSTL/SUMLIST64 address ranges and the PSWREGS data are dumped.

3. The caller's ASCB is dumped.

4. The suspended unit of work (SVC dump's caller) is dumped. This is either a TCB or an SSRB. The related PCLINK stacks are also dumped.

5. For TCB mode callers, the caller's SDWA is dumped. The PSW and register addresses from the SDWA are added to the range table. This causes 4K of storage to be dumped around each address. All RTM2 work areas pointed to by this TCB and any associated SDWAs are all dumped.

   For SRB mode callers, the SDWA is dumped. The PSW and register addresses from the SDWA are added to the range table. This causes 4K of storage to be dumped around each address. Also, the caller's register save area is added to the range table and the storage dumped.

   Duplicate storage is eliminated from the address range table to reduce the amount of storage dumped.

6. After all the storage is saved in a virtual buffer in the DUMPSRV address space, the caller's unit of work is reset by using the RESET service. This allows SVC dump to complete and return to the caller. When SVC dump processing completes in the address space to be dumped, whatever processing was taking place in that address space when it was interrupted by SVC dump resumes.

   The rest of the dump is then scheduled from the DUMPSRV address space.

Analyzing an SVC dump

This section shows you how to use IPCS to analyze an SVC dump. You would analyze an SVC dump because of one of the following:

- Dump output from the IPCS STATUS FAILDATA subcommand did not contain data for the abend being diagnosed.
- The problem involved multiple abends.
- The dump was taken but does not contain abend-related information.

This section contains the following topics, which, if followed in order, represent the procedure for analyzing an SVC dump:

- "Formatting the SVC dump header" on page 40
- "Looking at the dump title" on page 41
- "Displaying the incident token, time and type of dump" on page 42
- "Locating error information" on page 43
- "Analyze TCB structure" on page 46
- "Examining the LOGREC buffer" on page 48
- "Examining the system trace" on page 50
- "Looking at the registers" on page 50
- "Other useful reports for SVC dump analysis" on page 52
- "Reading the SDUMPX 4K SQA buffer" on page 53

Specifying the source of the dump

The first step in analyzing the dump is to specify the source of the dump that IPCS should format. In the IPCS dialog (see Figure 9 on page 40), choose option 0
SVC dump

(Defaults) and specify the name of the SVC dump data set on the “Source” line.

------------- IPCS Default Values -------------
COMMAND ==> 0

You may change any of the defaults listed below.
If you change the Source default, IPCS will display the current default
Address Space for the new source and will ignore any data entered in
the Address Space field.

Source ==> DSN('D46IPCS.SVC.CSVLLA.DUMP002')
Address Space ==> Ignored if Source is changed.
Message Routing ==> NOPRINT TERMINAL
Message Control ==> FLAG(WARNING)
Display Content ==> NOMACHINE REMARK REQUEST NOSTORAGE SYMBOL

Press ENTER to update defaults.
Use the END command to exit without an update.

Figure 9. IPCS Default Values menu

Press Enter to register the new default source name. Then, press PF3 to exit the
panel.

You can also use the SETDEF subcommand to specify the source. For the dump in
the preceding example, enter:
SETDEF DNAME('D46IPCS.SVC.CSVLLA.DUMP002')

IPCS does not initialize the dump until you enter the first subcommand or IPCS
dialog option that performs formatting or analysis. At that time IPCS issues
message BLS18160D to ask you if summary dump data can be used by IPCS. The
summary dump data should always be used for an SVC dump because it is the
data captured closest to the time of the failure. If you do not allow IPCS to use
summary dump data, other data captured later for the same locations will be
displayed, if available. Such data is less likely to be representative of the actual
data at these storage locations at the time of the failure.

Formatting the SVC dump header

The SVC dump header contains the following information:
• SDWA or SLIP data
• Dump title, error identifier, and time of the dump
• Requestor of dump

This information describes the type of SVC dump and can tell you if the dump is a
CONSOLE dump or a dump caused by the SLIP command. You would analyze
these dumps differently.

Format data in the header of an SVC dump using the following IPCS
subcommands:
• LIST TITLE
• STATUS FAILDATA
• STATUS REGISTERS
• STATUS WORKSHEET

The following sections give examples of how to use these IPCS subcommands (or
IPCS dialog options, where applicable) to obtain the desired information.
Looking at the dump title

The dump title tells you the component name, component identifier and module name. You can find the dump title using the following IPCS subcommands:

- LIST TITLE
- STATUS WORKSHEET

You can also obtain the STATUS WORKSHEET report through option 2.3 of the IPCS dialog. First, choose option 2 (ANALYSIS) from the primary option menu, as shown in [Figure 10].

![Figure 10. IPCS primary option menu](image1)

Then, choose option 3 (WORKSHEET) from the analysis of dump contents menu, as shown in [Figure 11].

![Figure 11. IPCS MVS analysis of dump contents menu](image2)

IPCS displays a new panel with information similar to that in [Figure 12 on page 42]. The dump title is labelled at the top of the STATUS WORKSHEET report. The dump title is “Compon=Program Manager Library-Lookaside, Compid=SC1CJ, Issuer=CSVLLBLD.” See [z/OS MVS Diagnosis: Reference] for an explanation of dump titles.
SVC dump

STATUS WORKSHEET also displays the error ID. In Figure 12, the dump ID is 001, error ID is sequence number 00051, ASID=X'001B', and processor 0000. Use this dump ID to match messages in SYSLOG and LOGREC records to the dump.

Displaying the incident token, time and type of dump

The IPCS subcommand STATUS SYSTEM identifies the following types of information. The IPCS dialog does not have a menu option for STATUS SYSTEM. Instead you must enter the subcommand.

- The time of the dump
- The program requesting the dump
- An incident token that associates one or more SVC dumps requested for a problem on a single system or on several systems in a sysplex

Figure 13 on page 43 is an example of a STATUS SYSTEM report. For a scheduled SVC dump, the following identifies the dump:

Program Producing Dump: SVCDUMP
Program Requesting Dump: IEAVTSDT

A dump requested by a SLIP or DUMP operator command is always a scheduled SVC dump.

For a synchronous SVC dump, the following identifies the dump:

Program Producing Dump: SVCDUMP
Program Requesting Dump: cccccccc

Where cccccccc is one of the following:

- The name of the program running when the system detected the problem
- SVCDUMP, if the system could not determine the failing task

A SYSMDUMP ABEND dump is always a synchronous SVC dump.
SYSTEM STATUS for an SVC dump contains an incident token. The request for the dump specifies the incident token or the system requesting the dumps provides it. The incident token consists of:

- The name of the sysplex
- The name of the system requesting the multiple dumps
- The date in Greenwich Mean Time (GMT)
- The time in GMT

**Locating error information**

Use the IPCS subcommand STATUS FAILDATA to locate the specific instruction that failed and to format all the data in an SVC dump related to the software failure. This report gives information about the CSECT involved in the failure, the component identifier, and the PSW address at the time of the error.

**Note:** For SLIP dumps or CONSOLE dumps, use SUMMARY FORMAT or VERBEXIT LOGDATA instead of STATUS FAILDATA.

Choose option 4 (COMMAND) from the IPCS primary option menu (see Figure 14) and enter the following command. Use the PF keys to scroll up and down through the report. The following sections describe parts of the report.

```
------------------------- IPCS Subcommand Entry -------------------------------
Enter a free-form IPCS subcommand, CLIST, or REXX exec invocation below:

====> STATUS FAILDATA
```

**Figure 14. IPCS Subcommand Entry menu**

**Identifying the abend and reason codes:** As Figure 15 on page 44 shows, under the heading “SEARCH ARGUMENT ABSTRACT”, you will find the abend code and, if provided, an abend reason code.
In Figure 15, the abend code is X'FF0' with no reason code. See z/OS MVS System Codes for a description of the abend code and reason code.

The following IPCS reports also provide the abend and reason codes:
- VERBEXIT LOGDATA
- STATUS WORKSHEET
- VERBEXIT SYMPTOMS

**Finding the system mode:** As Figure 16 shows, below the “SEARCH ARGUMENT ABSTRACT” section is information describing the system mode at the time of the error.

---

**Figure 15. Search argument abstract in the STATUS FAILDATA report**

In Figure 15, the abend code is X'FF0' with no reason code. See z/OS MVS System Codes for a description of the abend code and reason code.

The following IPCS reports also provide the abend and reason codes:
- VERBEXIT LOGDATA
- STATUS WORKSHEET
- VERBEXIT SYMPTOMS

**Finding the system mode:** As Figure 16 shows, below the “SEARCH ARGUMENT ABSTRACT” section is information describing the system mode at the time of the error.

---

**Figure 16. System mode information in the STATUS FAILDATA report**

The line that starts with “The error occurred...” tells you if the failure occurred in an SRB or TCB. In the example in Figure 16, the error occurred while an SRB was in control, which means you need to look under the heading SEARCH ARGUMENT ABSTRACT (see Figure 15) to find the CSECT and load module names. This is the module in which the abend occurred.

If an SRB service routine was in control, look under the heading SEARCH ARGUMENT ABSTRACT for the CSECT and load module names. This is the failing module.
In output from a SUMMARY FORMAT subcommand, look for the RB for the abending program. The RB has an RTPSW1 field that is nonzero.

In a dump requested by a SLIP operator command, use a STATUS CPU REGISTERS subcommand to see data from the time of the problem.

If the error had occurred while a TCB was in control, you would find the failing TCB by formatting the dump using the IPCS subcommand SUMMARY TCBERROR. See “Analyze TCB structure” on page 46.

**Identifying the failing instruction:** The STATUS FAILDATA report also helps you find the exact instruction that failed. This report provides the PSW address at the time of the error and the failing instruction text. Note that the text on this screen is not always the failing instruction text. Sometimes the PSW points to the place where the dump was taken and not the place where the error occurred.

In Figure 17, the PSW at the time of the error is X'11E6A3C' and the instruction length is 4-bytes; therefore, the failing instruction address is X'11E6A38'. The failing instruction is 927670FB.

---

**Figure 17. Time of error information in the STATUS FAILDATA report**

The failing instruction text displayed in this report is always 12 bytes, 6 bytes before and 6 bytes after the PSW address. In this example, the failing instruction, 927670FB, is an MVI of X'76' to the location specified by register 7 + X'FB'.

Register 7 at the time of the error, shown under **Registers 0-7** above, contained a X'000000017'. The attempted move was to storage location X'112'. The first 512 bytes of storage are hardware protected. Any software program that tries to store into that area without authorization will receive a protection exception error and a storage protection exception error.

See z/Architecture Principles of Operation for information about machine language operation codes, operands, and interruption codes.
To find the module that abnormally terminated and the offset to the failing instruction, use the WHERE command. WHERE can identify the module or CSECT that the failing PSW points to.

### Analyze TCB structure

If a TCB was in control at the time of the error, use the IPCS subcommand SUMMARY TCBERROR to look at the TCB information and find the failing component. SUMMARY TCBERROR summarizes the control blocks for the failing address space. (To see all the fields in the control blocks, use SUMMARY FORMAT.) Scan the completion codes (field CMP) for each TCB to find the correct TCB. This report displays RBs from newest to oldest.

[Figure 18 on page 47](#) is an example of SUMMARY TCBERROR output. In this example the TCB at address 008E9A18 has a completion code of X'0C1.' The error occurred under this TCB. Once you have identified the failing TCB, you can follow the RB chain to the failing program.
In this example, the most current RB is the SVRB at address 008FD7A8. This is the SVC dump’s RB. The ESTAE’s RB is the PRB at 008E9750. The ESTAE issued an SVC 33. The RB for the recovery termination manager (RTM) is the SVRB at 008FD638. RTM issued an SVC C to attach the ESTAE. The X’0C1’ abend occurred under the SVRB at 008FD4C8. The last interrupt was a 1 at the address indicated in...
the old PSW field (OPSW). The next RB in the chain shows an SVC X'53' (SMFWTM) had been issued. This is the code the X'0C1' occurred in.

For a scheduled dump, the abnormally terminating TCB can generally be found by scanning for a nonzero completion code. If there is no code, scan the system trace for the abend. The trace identifies the ASID number and TCB address for each entry. See "Examining the system trace" on page 50.

Use the STATUS or the STATUS REGS subcommand to find the data set name and the module name of the SVC dump requester.

**Examining the LOGREC buffer**

Use the IPCS subcommand VERBEXIT LOGDATA to view the LOGREC buffer in a dump. This report might repeat much of the information contained in the STATUS FAILDATA report, but it helps to identify occasions when multiple error events caused the software failure.

The following sample output, from the VERBEXIT LOGDATA subcommand, shows how multiple errors can appear in the LOGREC buffer. Abend X'0D5' is the first abend and X'058' is the second. Always check for multiple errors in the VERBEXIT LOGDATA report that are in the same address space or a related address space and are coincident with or precede the SVC dump.
When viewing the VERBEXIT LOGDATA report, skip the hardware records to view the software records. Search for the first software record.

The field “ERRORID=” gives the error identifier for the software failure. The error identifier consists of the sequence number, ASID, and time of the abend. By matching this identifier with error identifiers from other reports, you can tell if this is the same abend you have been analyzing or if it is a different abend. See “Interpreting software records” on page 569 for more information.
Examining the system trace

The system trace table describes the events in the system leading up to the error. The trace table is helpful when the PSW does not point to the failing instruction, and to indicate what sequence of events preceded the abend.

IPCS option 2.7.4 formats the system trace. The report is long. IBM recommends scrolling to the end of the report, then backing up to find the trace entry for the abend. Type an M on the command line and press F8 to scroll to the bottom of the report.

After you find the entry for the abend, start at the PSW where the dump was taken and track the events in the table to find where the failing instruction is in the code.

The system trace report marks important or significant entries with an asterisk. In Figure 19, *SVC D in the Ident CD/D column identifies the PSW where the program took the dump. Prior to the SVC D are three PGM (program check) entries. PGM 001 has an asterisk next to it, indicating that the program check was unresolved. The next entry, RCVY PRG6, identifies a recovery program that failed because it issued the SVC D a few entries later. See Chapter 8, “System trace,” on page 169 to recognize significant entries in the system trace table.

Looking at the registers

Use the IPCS subcommand STATUS REGISTERS to display the registers for the TCBs and RBs. SUMMARY REGS gives the same information in a different format. This report identifies the PSW, ASID and register values just as the STATUS FAILDATA report, but STATUS REGISTERS also gives the control register values.
The example output in Figure 20 shows the address in the PSW is X'0FE5CFC', the ASID is X'1B', and the failing instruction is located in offset X'5FC' in the CSECT IEAVESVC in the module IEANUC01 in the nucleus. You can now browse the dump at this location and look at the specific failing instruction. You could also use the information about the registers to find out more about the error if the address in the PSW does not point to the failing instruction.

This report identifies the PSW, ASID and register values just as the STATUS FAILDATA report. However, as Figure 21 on page 52 shows, STATUS REGISTERS also gives the control register values.
Other useful reports for SVC dump analysis

To collect further SVC dump data, use any of the following commands.

<table>
<thead>
<tr>
<th>IPCS subcommand</th>
<th>Information in the report</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS CPU REGISTERS</td>
<td>Data about the abend, current ASID, and task.</td>
</tr>
<tr>
<td>DATA CONTENTION</td>
<td>All fields in the TCBs and the current ASID.</td>
</tr>
<tr>
<td>SUMMARY FORMAT</td>
<td>The current FRR stack.</td>
</tr>
<tr>
<td>TCBEXIT IEAVTFMT 21C.%</td>
<td>The entry points in the active LPA and PLPA.</td>
</tr>
<tr>
<td>LPAMAP</td>
<td>A map of the modules in the nucleus when the dump was taken.</td>
</tr>
<tr>
<td>VERBEXIT NUCMAP</td>
<td>The data dumped by the SUMDUMP option on the SDUMPX macro.</td>
</tr>
<tr>
<td>VERBEXIT MTRACE</td>
<td>The master trace table.</td>
</tr>
<tr>
<td>VERBEXIT SYMPTOMS</td>
<td>The primary and secondary symptoms if available.</td>
</tr>
</tbody>
</table>

**Note:** Use the VERBEXIT SYMPTOMS subcommand last in your SVC dump analysis. Other subcommands can add symptoms to the dump header record. This ensures VERBEXIT SYMPTOMS provides all symptoms available from the dump.
Reading the SDUMPX 4K SQA buffer

The following SVC dumps contain problem data in an SDUMPX 4K system queue area (SQA) buffer:

- An SVC dump requested by a SLIP operator command
- Other SVC dumps, when indicated in the explanation of the dump title.
- An SVC dump requested by an SDUMP or SDUMPX macro with a BUFFER=YES parameter

To obtain the buffer, use the following IPCS subcommand:

```
LIST 0 DOMAIN(SDUMPBUFFER) LENGTH(4096)
```

Table 12 describes the fields in the SQA buffer and should be used for diagnosis.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>The characters, TYPE</td>
</tr>
<tr>
<td>4 (4)</td>
<td>4</td>
<td>RTM/SLIP processing environment indicator:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- X'00000001': RTM1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- X'00000002': RTM2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- X'00000003': MEMTERM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- X'00000004': PER</td>
</tr>
<tr>
<td>8 (8)</td>
<td>4</td>
<td>The characters, CPU</td>
</tr>
<tr>
<td>12 (C)</td>
<td>4</td>
<td>Logical processor identifier (CPUID)</td>
</tr>
<tr>
<td>16 (10)</td>
<td>4</td>
<td>The characters, REGS</td>
</tr>
<tr>
<td>20 (14)</td>
<td>64</td>
<td>General purpose registers 0 through 15 at the time of the event</td>
</tr>
<tr>
<td>84 (54)</td>
<td>4</td>
<td>The characters, PSW</td>
</tr>
<tr>
<td>88 (58)</td>
<td>8</td>
<td>The program status word (PSW) at the time of the event</td>
</tr>
<tr>
<td>96 (60)</td>
<td>4</td>
<td>The characters, PASD</td>
</tr>
<tr>
<td>100 (64)</td>
<td>2</td>
<td>The primary address space identifier (ASID) at the time of the event</td>
</tr>
<tr>
<td>102 (66)</td>
<td>4</td>
<td>The characters, SASD</td>
</tr>
<tr>
<td>106 (6A)</td>
<td>2</td>
<td>The secondary ASID at the time of the event</td>
</tr>
<tr>
<td>108 (6C)</td>
<td>4</td>
<td>The characters, ARS</td>
</tr>
<tr>
<td>112 (70)</td>
<td>64</td>
<td>Access registers 0 through 15 at the time of the event.</td>
</tr>
<tr>
<td>176 (80)</td>
<td>4</td>
<td>The characters, G64H</td>
</tr>
<tr>
<td>180 (84)</td>
<td>64</td>
<td>High halves of general purpose registers 0 through 15 at the time of the event.</td>
</tr>
<tr>
<td>244 (F4)</td>
<td>variable</td>
<td>One of the following, as indicated by the RTM/SLIP processing environment indicator at offset 4 of the buffer:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The system diagnostic work area (SDWA), if offset 4 is 1 (RTM1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The recovery termination manager 2 (RTM2) work area (RTM2WA), if offset 4 is 2 (RTM2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The address space control block (ASCB), if offset 4 is 3 (MEMTERM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The PER interrupt code and PER address, if offset 4 is 4 (PER)</td>
</tr>
<tr>
<td>4076 (FEC)</td>
<td>4</td>
<td>The characters, P16</td>
</tr>
<tr>
<td>4080 (FF0)</td>
<td>16</td>
<td>The 16-byte program status word (PSW) at the time of the event.</td>
</tr>
</tbody>
</table>
SVC dump
Chapter 3. Transaction dump

A transaction dump provides a representation of the virtual storage for an address space when an error occurs. Typically, an application requests the dump from a recovery routine when an unexpected error occurs. Transaction dumps are requested as follows:

• **Synchronous transaction dump:**
  The requester’s IEATDUMP macro invocation issues an instruction to obtain the dump under the current task. IEATDUMP returns control to the requester and is available once the dump data has been written into a dump data set.

Each Transaction dump also contains a summary dump, if requested. The summary dump supplies copies of selected data areas taken at the time of the request. Specifying a summary dump also provides a means of dumping many predefined data areas simply by specifying one option. This summary dump data is not mixed with the Transaction dump because in most cases it is chronologically out of step. Instead, each data area selected in the summary dump is separately formatted and identified. IBM recommends that you request summary dump data.

This section includes information system programmers need to know about Transaction dump and Transaction dump processing:

• “Choices for IEATDUMP Data Sets” on page 56
• “Obtaining transaction dumps” on page 58
• “Printing, viewing, copying, and clearing a dump data set” on page 59
• “Contents of transaction dumps” on page 59

See *z/OS MVS Programming: Authorized Assembler Services Guide* for information any programmer needs to know about programming the IEATDUMP macro to obtain a Transaction Dump:

• Deciding when to request a Transaction dump
• Understanding the types of Transaction Dumps that MVS produces
• Designing an application program to handle a specific type of Transaction dump
• Identifying the data set to contain the dump
• Defining the contents of the dump
• Suppressing duplicate Transaction dumps using dump analysis and elimination (DAE)

**Planning data sets for transaction dumps**

Transaction dump processing stores data in dump data sets that you preallocate manually, or that are allocated automatically, as needed. The output dump data set has the attributes of RECFM=FB and LRECL=4160.

**Planning data set management for transaction dumps**

For transaction dumps, use extended format sequential data sets because they have the following characteristics:

• Greater capacity than sequential data sets
• Striping and compression support.
For more information on extended format sequential data sets, see "Choosing SVC dump data sets" on page 15.

Sequential data sets can use large format data sets (DSNTYPE=LARGE).

Extended format sequential data sets can be placed in either track-managed space or cylinder-managed space. Transaction dump fully supports placing dump data sets in cylinder-managed space.

Using preallocated dump data sets
To specify a pre-allocated data set, specify the DDNAME parameter that identifies a data set. The data set must contain sufficient space in one for more extents for the entire dump to be written. DDNAME does not have a 2GB size restriction for the size of the dump. If the data set does not contain sufficient space, a partial dump is returned.

Setting up allocation authority
To allocate dump data sets automatically, the caller’s and/or DUMPSRV address space must have authority to allocate new data sets. Do the following:

1. Associate the caller’s and/or DUMPSRV address space with a user ID.
   - If you have RACF Version 2 Release 1 installed, use the STARTED general resource class to associate the caller or DUMPSRV with a user ID. For this step, the RACF started procedures table, ICHRIN03, must have a generic entry.
   - If you have an earlier version of RACF, use the RACF started procedures table, ICHRIN03.

2. Authorize caller’s or DUMPSRV user ID to create new dump data sets using the naming convention in the following topic.
   - With the high-level qualifier of SYS1, the data sets are considered group data sets. You can assign CREATE group authority to the caller’s user ID within that group.

See the following references for more information:
- z/OS Security Server RACF Security Administrator’s Guide for information on using the STARTED general resource class and on controlling creation of new data sets.

Choices for IEATDUMP Data Sets
Transaction dump processing supports both pre-allocated and automatically allocated dump data sets. The dump is allocated from the generic resource SYSALLDA. IEATDUMP processes the dump data sets in the following ways:

- For pre-allocated data sets, IEATDUMP writes to the data set without first capturing the dump into a data space. The dump can contain more than 2 GB if the data set capacity permits.
- For automatically allocated data sets, IEATDUMP processes the dump data sets depending on whether the dump section number symbol &DS. is used on the end of the data set name pattern:
  - If &DS. is not used on the end of the data set name pattern, IEATDUMP captures the dump and stores it in a data space; the data set is then allocated with the space required to contain the captured data; and the dump is written to disk. The dump cannot exceed 2 GB.
If dynamic allocation fails, message IEA820I is issued, and the dump is deleted.

– If &DS. is used on the end of the data set name pattern, IEATDUMP does not first capture the dump to a data space. Instead, IEATDUMP writes the dump directly to disk. If the size limit of the date set is reached, IEATDUMP allocates another dump with a higher value for &DS.. Each data set has an extent size of 500 M that can be changed using ACS routines. These extents are written to until the disk runs out of space or no more extents can be created. At that time, a new data set in the sequence is created. Multi-data set IEATDUMPs utilize up to 999 data sets. The maximum size depends on the amount of space on the volumes where these data sets get allocated. Before IPCS can process the data, you must combine all the data sets into one data set using IPCS COPYDUMP.

Naming automatically allocated dump data sets
The application has control of the name of the data sets created by the automatic allocation function, and you can select a name-pattern to allow for dump data set organization according to your needs. The name is determined through an installation-supplied pattern on the DSN(AD) keyword in the IEATDUMP macro.

Names must conform to standard MVS data set naming conventions and are limited to 44 characters, including periods used as delimiters between qualifiers. A set of symbols is available so that you can include the following kinds of information in the names of your automatically allocated dump data sets:

- System name
- Sysplex name
- Job name
- Local and GMT time and date
- Dump section number

For a complete list of the symbols you can use, see the explanation of DUMPDS NAME= in z/OS MVS System Commands.

Note:
1. The &SEQ. symbol is not supported for IEATDUMPs.
2. You can use the &DS. symbol for splitting the dump between several data sets. When the &DS. symbol is added to the end of the DSN name pattern, the transaction dump data can be placed into as many as 999 automatically-allocated 500M-extent data sets. Note that you must combine all the data sets into one data set using IPCS COPYDUMP before IPCS can process the data.

When determining the pattern for the dump data set names, consider any automation tools you may have at your installation that work on dump data sets.

Figure 22 on page 58 describes a SPFUSER name pattern. Note that the symbols are resolved into date and time, so they are preceded by an alphabetic character to conform to MVS data set name requirements. Also, the symbol starts with an ampersand (&) and ends with a period (.), resulting in a name pattern that has double periods when a symbol finishes a qualifier. One period ends the symbol, and the second serves as the delimiter between qualifiers of the generated data set name.
Automatically allocated dump data sets are not added to the system's sysplex dump directory, as it is for SVC dumps.

**Communication from the system**

The system communicates about automatic allocation of dump data sets using three messages:

- IEA827I is issued when a complete or partial dump multi-data set dump is taken. IEA827I is an informational message, it will not be issued highlighted.

- IEA822I is issued when a complete or partial dump is taken. IEA822I is an informational message, it is not issued highlighted.

- IEA820I is issued once per Transaction dump when the dump cannot be taken or allocation fails. IEA820I is an informational message, it will not be issued highlighted.

**Obtaining transaction dumps**

Obtain a Transaction dump by issuing a IEATDUMP macro in an authorized or unauthorized program.

In a sysplex, authorized applications might need dumps from more than one address space to collect all of the problem data. These dumps need to be requested at the same time. To request these multiple dumps, issue a IEATDUMP macro with a REMOTE parameter specifying the other address spaces involved in the problem. To help you set up these requests, the parameter can contain wildcards. If the installation gives names that form patterns to the systems in the sysplex and to jobs for associated work, you can use wildcards, * and ?, to specify the names. For example, use the name TRANS? for the jobnames TRANS1, TRANS2, and TRANS3 and the name TRANS* for TRANS1, TRANS12, and TRANS123.

**Note:** If a Transaction dump uses the REMOTE parameter to dump one or more address spaces on a pre-release 4 system, the result will be a single SVC dump containing the requested data, instead of one or more Transaction dumps written to data set names specified with the DSN parameter. Issue the DISPLAY DUMP,STATUS command to determine the name of this SVC dump.
Printing, viewing, copying, and clearing a dump data set

Transaction Dumps are unformatted when created. Use IPCS to format a dump and then view it at a terminal or print it. For example, for a pre-allocated data set or a dump data set, the JCL shown in Figure 23 does the following:

- Uses the Transaction dump in the APPL.TDUMP00 data set. The IPCSTDMP DD statement identifies this data set.
- Deletes the IPCS dump directory in the DELETE(DDIR) statement. This statement uses the USERID of the batch job in the directory identification.
- Allocates the dump directory through the BLSCDDIR statement. The default is volume VSAM01. The example shows VSAM11. Override the default volume with the desired volume.
- Formats the dump using the IPCS subcommands in LIST 0. To use this example, replace the LIST 0 command with the desired IPCS subcommands or a CLIST. See z/OS MVS IPCS User’s Guide for CLISTs.

```
//IPCSJOB JOB
//IPCS EXEC PGM=IKJEFT01,DYNAMNBR=75,REGION=1500K
//SYSPROC DD DSN=SYS1.SBLSCLI0,DISP=SHR
//IPCSTDMP DD DSN=APPL.TDUMP00,DISP=SHR
//SYSDUMP DD SYSOUT=*  
//IPCSTOC DD SYSOUT=*  
//IPCSPTO DD SYSOUT=*  
//SYSTSPRT DD SYSOUT=*  
//SYSTSN DD *  
//DELETE(DDIR) PURGE CLUSTER 
//BLSCDDIR VOLUME(VSAM11)
//IPCS NOPARM
//SETDEF DD((IPCSTDMP) LIST NOCONFIRM
LIST 0
END 
/*
```

Figure 23. Example: JCL to Print, Copy, and Clear the Dump Data Set

Contents of transaction dumps

Transaction Dumps share parmlib member IEADMR00 to establish the dump options list at system initialization. The IBM-supplied IEADMR00 parmlib member specifies dump options NUC, SQA, LSQA, SWA, TRT, RGN, and SUM.

See z/OS MVS IPCS Commands for examples of IPCS output formatted from Transaction Dumps.

Customizing transaction dump contents

You can customize the contents of a Transaction dump to meet the needs of your installation. For example, you might want to add areas to be dumped, reduce the dump size, or dump Hiperspaces. In most cases, you will customize the contents of a Transaction dump through the SDATA parameter of the IEATDUMP macro.

Hiperspaces: Transaction Dumps do not include Hiperspaces. To include Hiperspace data in a Transaction Dump, you have to write a program to copy data from the Hiperspace into address space storage that is being dumped.

Adding areas: If the dump, as requested, will not contain all the needed areas, see one of the following for ways to add the areas:
Customized contents using the SDATA parameter

The IBM-supplied default contents and the contents available through customization are detailed in Table 13. The tables show dump contents alphabetically by the parameters that specify the areas in the dumps. Before requesting a dump, decide what areas will be used to diagnose potential errors. Find the areas in the tables. The symbols in columns under the dump indicate how the area can be obtained in that dump; the order of the symbols is not important.

- **D** IBM-supplied default contents
- **M** Available on the macro that requests the dump
- **P** Available in the parmlib member that controls the dump options
- **X** Available on the CHNGDUMP operator command that changes the options for the dump type
- **Blank** No symbol indicates that the area cannot be obtained.

**Note**: System operator commands and assembler macros use the parameters in the table to specify dump contents.

<table>
<thead>
<tr>
<th>SDATA Parameter Option</th>
<th>Dump Contents</th>
<th>Transaction dump for IEATDUMP Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLNUC</td>
<td>The DAT-on and DAT-off nucleuses</td>
<td>M P X</td>
</tr>
</tbody>
</table>
| CSA                   | Common service area (CSA) (that is, subpools 227, 228, 231, 241) and virtual storage for 64-bit addressable memory objects created using one of the following services:  
  - IARV64 REQUEST=GETCOMMON,DUMP=LIKECSA  
  - IARCP64 COMMON=YES,DUMP=LIKECSA  
  - IARST64 COMMON=YES,TYPE=PAGEABLE | M P X                              |
| DEFS                   | Default areas LSQA, NUC, PSA, RGN, SQA, SUM, SWA, TRT                        | M                                   |
| ALL                   | CSA, GRSQ, LPA, NUC, RGN, SQA, SUM, SWA, TRT                                 | X                                   |
| GRSQ                  | Global resource serialization control blocks for the task being dumped:  
  - Global queue control blocks  
  - Local queue control blocks | M P X                              |
| IO                    | Input/output supervisor (IOS) control blocks for the task being dumped:  
  - EXCPD  
  - UCB | D                                   |
| LPA                   | Active link pack area (LPA): module names and contents                       | M P X                              |
| LSQA                  | Local system queue area (LSQA) allocated for the address space (that is, subpools 203 - 205, 213 - 215, 223 - 225, 229, 230, 233 - 235, 249, 253 - 255), and virtual storage for 64-bit addressable memory objects created using one of the following services:  
  - IARV64 REQUEST=GETSTOR,DUMP=LIKELSQA  
  - IARCP64 COMMON=NO,DUMP=LIKELSQA  
  - IARST64 COMMON=NO | D M P X                            |
### Table 13. Customizing transaction dump contents through the SDATA Parameter (continued)

<table>
<thead>
<tr>
<th>SDATA Parameter Option</th>
<th>Dump Contents</th>
<th>Transaction dump for IEATDUMP Macro</th>
</tr>
</thead>
</table>
| NUC                    | Read/write portion of the control program nucleus (that is, only the non-page-protected areas of the DAT-on nucleus), including:  
  - CVT  
  - LSQA  
  - PSA  
  - SQA | M P X |
| PSA                    | Prefixed save areas (PSA) for the processor at the time of the error or the processor at the time of the dump | D M P |
| RGN                    | Allocated pages in the private area of each address space being dumped, including subpools 0 - 127, 129 - 132, 203 - 205, 213 - 215, 223 - 225, 229, 230, 236, 237, 244, 249, 251 - 255, and virtual storage for 64-bit addressable memory objects created using one of the following services:  
  - IARV64 REQUEST=GETSTOR,DUMP=LIERGN  
  - IARV64 REQUEST=GETSTOR,SVCDUMPRGN=YES  
  - IARCP64 COMMON=NO,DUMP=LIERGN  
  - IARST64 COMMON=NO  
  . Also, allocated eligible storage above the 2–gigabyte address. | M P X |
| SQA                    | System queue area (SQA) allocated (that is, subpools 226, 239, 245, 247, 248) and virtual storage for 64-bit addressable memory objects created using one of the following services:  
  - IARV64 REQUEST=GETCOMMON,DUMP=LIESQA  
  - IARCP64 COMMON=YES,DUMP=LIESQA  
  - IARST64 COMMON=YES,TYPE=FIXED  
  - IARST64 COMMON=YES,TYPE=DREF | D M P X |
| SUM                    | Summary dump (See “Contents of summary dumps in transaction dumps” on page 62.) | D M P X |
| SWA                    | Scheduler work area (SWA) (that is, subpools 236 and 237) | M P X |
| TRT                    | System trace, generalized trace facility (GTF) trace, and master trace, as available | D M P X |
| Default system data    | Instruction address trace, if available | D |
| Default system data    | Nucleus map and system control blocks, including:  
  - ASCB for each address space being dumped  
  - ASVT  
  - Authorization table for each address space  
  - CVT, CVT prefix, and secondary CVT (SCVT)  
  - Entry tables for each address space  
  - GDA  
  - JSAB of each address space being dumped  
  - Linkage stack  
  - Linkage table for each address space  
  - PCCA and the PCCA vector table  
  - TOT  
  - TRVT  
  - UCB | D |
| Default system data    | DFP problem data, if DFP Release 3.1.0 or a later release is installed | D |
### Transaction dump

Table 13. Customizing transaction dump contents through the SDATA Parameter (continued)

<table>
<thead>
<tr>
<th>SDATA Parameter Option</th>
<th>Dump Contents</th>
<th>Transaction dump for IEATDUMP Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default system data</td>
<td>Storage for the task being dumped and program data for all of its subtasks</td>
<td>D</td>
</tr>
<tr>
<td>Default system data</td>
<td>Storage: 4 kilobytes before and 4 kilobytes after the address in the PSW at the time of the error</td>
<td>D</td>
</tr>
</tbody>
</table>

### Contents of summary dumps in transaction dumps

When you request a summary dump, the **SUM** parameter requests many useful, predefined areas with one parameter.

Summary dump does not contain volatile system information. The system writes the summary dump before it returns control to the dump requester; the summary information is saved for each address space that is being dumped.

The Summary Dump contains:

1. ASID record for the address space of the dump task
2. Control blocks for the recovery termination manager (RTM):
   - RTM2WA associated with all TCBs in the dumped address space
3. Dump header, which is mapped by AMDDATA.
   For the AMDDATA mapping, see [z/OS MVS Data Areas](http://www.ibm.com/systems/z/os/zos/bkserv/) in z/OS Internet library
4. 4 kb before and 4 kb after:
   - All valid unique addresses in the PSWs in the RTM2WAs shown in the dump
   - All valid unique addresses in the registers in the RTM2WAs shown in the dump
5. Supervisor control blocks:
   - Current linkage stack
   - Primary address space number (PASN) access list
   - Work unit access list

For information about control blocks that are listed in Table 13 on page 60, see [z/OS MVS Data Areas](http://www.ibm.com/systems/z/os/zos/bkserv/) in z/OS Internet library

### Customizing contents through operator commands

The dump options list for Transaction Dumps can be customized through a CHNGDUMP operator command by all the ways shown in Table 14 on page 63.

### Nucleus areas in dumps

Dump options control the parts of the nucleus that appear in a dump. A diagnostician seldom needs to analyze all the nucleus. An installation can eliminate nucleus areas from dumps. If the IBM-supplied defaults are used, Transaction dump for an IEATDUMP macro contains the nucleus map and certain control blocks.

Most problems can be debugged without dumping the nucleus. If a problem arises that requires the nucleus be dumped, use the CHNGDUMP operator command to add the NUC SDATA option to all IEATDUMPs. This also applies to other options.
DAT, dynamic address translation, is the hardware feature that enables virtual storage. In the DAT-on part of the nucleus, the addresses are in virtual storage; in the DAT-off part of the nucleus, the addresses are in central storage.

Table 14. Customizing transaction dump contents through operator commands

<table>
<thead>
<tr>
<th>Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updating IEADMR00 parmlib member</td>
<td>Change occurs: At system initialization</td>
<td>To add the link pack area (LPA) to all Transaction dumps for IEATDUMP macros and SDATA, while keeping the local system queue area (LSQA) and trace data, change the line in IEADMR00: SDATA=(LSQA,TRT,LPA)</td>
</tr>
<tr>
<td></td>
<td><strong>What changes:</strong> This parmlib member establishes the dump options for Transaction dumps for IEATDUMP macros and SDATA. See &quot;Contents of transaction dumps&quot; on page 59 for the list.</td>
<td></td>
</tr>
<tr>
<td>Adding the CHNGDUMP operator command in IEACMD00 parmlib member</td>
<td>Change occurs: At system initialization</td>
<td>To add the link pack area (LPA) to all Transaction dumps for IEATDUMP macros and SYSMDUMP, while keeping the local system queue area (LSQA) and trace data, add the following command to IEACMD00: CHNGDUMP SET,SYSMDUMP=(LPA)</td>
</tr>
<tr>
<td></td>
<td><strong>What changes:</strong> This command establishes the dump options for Transaction dumps for IEATDUMP macros and SYSMDUMPS. See &quot;Contents of transaction dumps&quot; on page 59 for the list.</td>
<td></td>
</tr>
<tr>
<td>Entering CHNGDUMP operator command with SYSMDUMP parameter on a console with master authority</td>
<td>Change occurs: Immediately when command is processed</td>
<td>To add the LPA to all Transaction dumps for the IEATDUMP macro and SYSMDUMP, until changed by another CHNGDUMP SYSMDUMP, enter: CHNGDUMP SET,SYSMDUMP=(LPA)</td>
</tr>
<tr>
<td></td>
<td><strong>What changes:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>For the ADD mode:</strong> CHNGDUMP options are added to the current Transaction dump options list and to any options specified in the macro or operator command that requested the dump. The options are added to all Transaction dumps for IEATDUMP macros and SYSMDUMP, until another CHNGDUMP SYSMDUMP operator command is entered.</td>
<td>• To add the CHNGDUMP IEATDUMP options list to all Transaction dumps: CHNGDUMP SET,SYSMDUMP,ADD</td>
</tr>
<tr>
<td></td>
<td>• <strong>For the OVER mode:</strong> CHNGDUMP options are added to the current Transaction dump options list. The system ignores any options specified in the macro or operator command that requested the dump. The options override all Transaction dumps for the IEATDUMP macro and SYSMDUMP, until a CHNGDUMP SYSMDUMP,ADD operator command is entered.</td>
<td>• To override all Transaction dumps with the CHNGDUMP IEATDUMP options list: CHNGDUMP SET,SYSMDUMP,OVER</td>
</tr>
<tr>
<td></td>
<td>• <strong>For the DEL option:</strong> CHNGDUMP options are deleted from the Transaction dump options list. When more than one CHNGDUMP operator command with IEATDUMP is entered, the effect is cumulative.</td>
<td>• To remove LPA from the IEATDUMP options list: CHNGDUMP DEL,SYSMDUMP=(LPA)</td>
</tr>
</tbody>
</table>
Transaction dump
Chapter 4. Stand-alone dump

The stand-alone dump program (SADMP) produces a stand-alone dump of storage that is occupied by one of the following:

- A system that failed.
- A stand-alone dump program that failed.

Either the stand-alone dump program dumped itself — a **self-dump** —, or the operator loaded another stand-alone dump program to dump the failed stand-alone dump program.

The stand-alone dump program and the stand-alone dump together form what is known as the stand-alone dump service aid. The term stand-alone means that the dump is performed separately from normal system operations and does not require the system to be in a condition for normal operation.

The stand-alone dump program produces a high-speed, unformatted dump of all of central storage, plus it can include user tailorable parts of paged-out virtual storage. The user generated stand-alone dump program must reside on a storage device that can be used to IPL from.

Produce a stand-alone dump when the failure symptom is a wait state with a wait state code, a wait state with no processing, an instruction loop, or slow processing.

You create the stand-alone dump program that dumps the storage. Use the AMDSADMP macro to produce the following:

- A stand-alone dump program that resides on **DASD**, with output directed to a tape volume or to a DASD dump data set
- A stand-alone dump program that resides on **tape**, with output directed to a tape volume or to a DASD dump data set.

A stand-alone dump supplies information that is needed to determine why the system or the stand-alone dump program failed.

You can create different versions of the stand-alone dump program to dump different types and amounts of storage. To create the different versions, code several AMDSADMP macros by varying the values of keywords on the macros.

Before you begin, also consider reading the following topics:

- For a set of best practices for optimizing stand-alone dump (SADMP) data capture, optimizing problem analysis time, and ensuring that the stand-alone dump is successful at capturing the necessary information for use by IBM Support, see the topic about **Best practices for large stand-alone dump** in [z/OS V2R2 Problem Management](https://www.ibm.com/docs/en/zos-v2r2-problem-management).
- To enable your operators and the system to respond appropriately to disabled wait states, consider activating the AutoIPL function, see the topic about **Using the automatic IPL function** in [z/OS MVS Planning: Operations](https://www.ibm.com/docs/en/zos-v2r2).

This information covers the following topics, which describe how to use stand-alone dump:

- “Planning for stand-alone dump” on page 66
- “Creating the stand-alone dump program” on page 70
Planning for stand-alone dump

There are several decisions you need make when planning for a stand-alone dump. You implement most of these decisions when you create the stand-alone dump program, either when you code the AMDSADMP macro, when you assemble the macro, or when you use the SADMP option on the IPCS Dialog. Some typical questions follow.

**Should I take a stand-alone dump to DASD or to tape?**

When choosing an output device for stand-alone dump, consider the need for operator intervention, the amount of operator intervention involved, and the amount of time the system will be unavailable.

You can reduce the level of operator intervention during stand-alone dump processing by dumping to DASD. With an automation package set up to IPL the stand-alone dump program from DASD, stand-alone dump can be run from a remote site. When you dump to tape, an operator is required to handle other aspects of dumping, such as mounting or changing tapes, unless the tape is in an IBM Virtual Tape Server (VTS).

The system is unavailable when a stand-alone dump is taken. The amount of time the system is unavailable depends upon the size of the dump. See “Dumping to a DASD data set” on page 87 for more information.

**If I do dump to DASD, how much space do I need?**

The maximum size of a single-volume DASD dump data set depends on the type of data set.

- Conventional sequential (DSNTYPE=BASIC) data sets can span 65,535 tracks per volume, and can hold approximately 3 GB per volume.

- Extended format (DSNTYPE=EXTREQ) data sets are supported by z/OS V1R6 and later releases. Extended format sequential data can hold 4,294,967,295 blocks per volume. The maximum size for extended format sequential is approximately 98,304 GB per volume. You cannot use striping or compression options for extended format sequential data sets. You must use the guaranteed free space option to require DFSMS to reserve space at the time that the data set is created.

- Large format data sets are supported by z/OS V1R7 and later releases. Large format (DSNTYPE=LARGE) data sets can span 16,777,215 tracks per volume. The maximum size for large format data sets is 768 GB per volume.

If you require more space than you want to allocate on a single volume, you can define a multi-volume DASD dump data set that can span up to 32 volumes of the same device type.

**Note:** Beginning in z/OS V1R12, SADMP supports placement of dump data sets in cylinder-managed space. In releases prior to z/OS V1R12, stand-alone data sets must remain in track-managed space.
Use the AMDSADDD REXX utility or the SADMP dump data set utility on the
ICCS dialog to allocate and initialize a single volume DASD dump data set or a
multi-volume DASD dump data set. This prepares the data set for use by the
system where initialization is performed and for other systems that have access to
the same data set using the same device numbers. For more information, see:

- “Using the AMDSADDD utility” on page 88
- SADMP option on the IPCS Dialog in [z/OS MVS IPCS User’s Guide]

When using a multi-volume DASD dump data set, the device number of the first
volume is specified. The other volumes are located by stand-alone dump using the
information that is placed in the data set when it is initialized. All volumes are
written concurrently by stand-alone dump. The data set is rejected if stand-alone
dump is unable to access all volumes of the data set or if invalid control
information is read from the data set during initialization.

If you do not allocate enough space in your dump data set, the stand-alone dump
program prompts the operator to continue dumping to another DASD dump data
set or tape volume. You can continue dumping to any stand-alone dump
supported device, however, after a tape device is selected, it must be used to
complete the dump even though multiple volumes might be required.

IBM recommends that you allocate multiple dump data sets to perform a complete
stand-alone dump.

**Can I dump to multiple dump data sets?**

Stand-alone dump does allow you to dump to multiple dump data sets. By coding
the DDSPROMPT=YES keyword on the AMDSADMP macro, you can generate a
stand-alone dump program that allows run-time dump data set prompting.

When the Stand-alone dump program is initiated, message AMD001A is issued to
prompt the operator for an output device. If a DASD device is specified and
run-time dump data set prompting is active, message AMD002A is issued to
prompt the operator for a dump data set name. Providing the dump data set is
validly allocated and initialized on the output device, the stand-alone dump
program uses the dump data set name specified. If message AMD099I is issued
indicating that the dump data set is full, the operator can continue dumping to any
stand-alone dump supported DASD dump data set or tape device by replying to
message AMD001A (and possibly AMD002A) again. After the dump completes,
message AMD104I is issued to indicate the entire set of devices and/or dump data
sets that were used during the taking of the dump.

By coding DDSPROMPT=NO on the AMDSADMP macro, the stand-alone dump
program is generated without run-time dump data set prompting. In this case,
replying to message AMD001A with a DASD device causes the stand-alone dump
program to assume that the output dump data set is named SYS1.SADMP.

**Note:**

1. Use the AMDSADDD REXX or the IPCS SADMP dump data set utilities to
   allocate and initialize the stand-alone dump data sets.

2. The stand-alone dump program must locate the dump data set on the device
   that is specified. Therefore, it is imperative that the necessary data set
   management steps be taken so that the stand-alone dump data sets are not
   placed into a migrated state or moved to a different volume. The dump data
   sets must also be exempt from any space management processing that releases
   unused space.
Stand-Alone dump

3. You can continue a dump to any stand-alone dump supported device, however, after a tape device is selected, it must be used to complete the dump even though multiple tape volumes might be required.

See the following topics for more information:

- For more information on dump data set processing, see the description of the DDSPROMPT keyword in the “Syntax of the AMDSADMP macro” on page 75.
- For more information on how to use multiple dump data sets with IPCS, see “Copying from multiple dump data sets” on page 118.
- For more information on performing tasks associated with creating, clearing, and reallocating SADMP data sets on DASD, see the SADMP option on the IPCS Dialog in z/OS MVS IPCS User’s Guide.

What can I name my DASD dump data sets?

A stand-alone dump data set can be any valid MVS data set name, however, stand-alone dump has two requirements that are checked at both generation time and run-time:

- The data set name must be 44 characters or less
- The data set name must contain the text 'SADMP' as either part of, or as an entire data set qualifier

In addition, because the generation process does not perform any allocation on the output device or dump data set name, it is imperative that you ensure that the data set name specified on the OUTPUT= keyword matches exactly the dump data set name allocated by the AMDSADDD REXX or IPCS SADMP data set utilities. The following are some additional rules to follow when specifying a dump data set name:

- The data set name specified should be fully qualified (without quotation marks)
- The alphabetic characters in the dump data set name should be specified as capital letters

How much of the system should I dump?

The situation dictates the amount of information you need to diagnose the failure. You can use the DUMP keyword to control the amount of storage you want dumped. See “Using the DUMP or ADDSUMM keyword to request additional storage or address spaces” on page 83 for more information.

When should I specify the dump tailoring options?

The most flexible way to specify the dump options for a stand-alone dump is to specify, on the DUMP keyword of the AMDSADMP macro, those areas of storage you normally always want dumped and additionally allow the operator who requests the dump to specify additional options by coding the PROMPT keyword on the AMDSADMP macro. In most cases, to simplify the dumping process, it is best to define any installation specific dump options on the AMDSADMP macro and not use the PROMPT keyword. See “Using the DUMP or ADDSUMM keyword to request additional storage or address spaces” on page 83 for more information.

What type of security does the stand-alone dump program require?

After the stand-alone dump program is properly created on a DASD residence volume, it resides in the SYS1.PAGEDUMP.Vvolser data set. To ensure that the stand-alone dump program is available and processes successfully, do not delete the data set or move it to another volume or pack. To protect the stand-alone dump program in SYS1.PAGEDUMP.Vvolser, use a password or a security product,
such as RACF. If the data set is not protected, unauthorized users can read the dump data in SYS1.PAGEDUMP.Vvolser. Also consider protecting the stand-alone dump output dump data sets from unauthorized reading.


Should I use IEBGENER or the COPYDUMP subcommand to copy a dump to a data set?
The recommended method is IPCS COPYDUMP. IPCS COPYDUMP can run without a dump directory being employed. Use the DEFER option when initiating the IPCS session to tell IPCS to defer accessing a dump directory until one is required. IPCS COPYDUMP has the ability to merge the records from a multi-volume SADMP and recapture the prioritized order used by SADMP to get the most important data into the dump data sets first.

If SADMP is allowed to complete normally, IEBGENER and similar transcription programs can produce a logically complete dump data set that IPCS can process. However, IPCS performance, particularly IPCS dump initialization, degrades as more volumes are added to the SADMP data set.

What is dumped when I run the stand-alone dump program?
The default dump contains all areas of central storage and some areas of virtual storage that are not backed by central storage. The output of the stand-alone dump program includes:
- The prefixed save areas (PSA)
- The nucleus and extended nucleus
- The system queue area (SQA) and the extended SQA
- The common service area (CSA) and the extended CSA
- Subpools 203-205, 213-215, 229, 230, 236, 237, 247, 248, and 249 for all address spaces
- The local system queue area (LSQA) and the extended LSQA for eligible address spaces
- The dump title provided by the operator; otherwise, the dump is untitled
- The processor STORE STATUS information for each processor
- Central storage from address 0 to the top of main storage (some blocks might be missing because of offline storage elements)
- Virtual storage areas selected by the DUMP keyword, or selected by the operator at runtime.
- A message log, normally consisting of all console messages issued by the dump program, including suppressed messages. (To format and print the stand-alone dump message log, use the VERBEXIT SADMPMSG subcommand or the SADMPMSG option of the IPCS dialog.)
- High virtual for the TRACE address space.
- High virtual for the DUMPSRV address space.
- High virtual for the GRS address space.
- Dump records summarizing the zeroed pages in the dump
- The full generalized trace facility (GTF) address space
- Subpool 127 in the GRS address space
- Data spaces whose names begin with ISG for the GRS address space
- All of DUMPSRV’s data spaces
Stand-Alone dump

- The full cross-system coupling facility (XCF) address space
- All of XCF's dataspaces
- XES-related dataspaces for address spaces with an XES connection

Note that this list does not imply an order of the stand-alone dump process. During stand-alone dump processing, several different messages are issued to indicate the progress of the dumping:
- For real dump processing, AMD005I is issued.
- For both real and virtual dump processing, AMD095I is issued every 30 seconds, followed by message AMD056I indicating that dumping of virtual storage has completed and AMD104I to indicate what output devices and/or dump data sets were used by the stand-alone dump program.

Can I use my current version of the stand-alone dump program to dump a new version of z/OS?

Always use the stand-alone dump version that is generated from the same release of z/OS that you want to dump. IBM does not guarantee that a different level of stand-alone dump will successfully dump anything other than the level of z/OS it was designed for. The new version of z/OS might have changed making the stand-alone dump program unable to locate vital information it needs to operate.

When migrating to a new version of z/OS, IBM strongly recommends that you generate a new version of the stand-alone dump program built from the new z/OS system data sets. See "One-stage generation" on page 96 for more information.

Creating the stand-alone dump program

The first step in creating a stand-alone dump program is selecting a tape or DASD as the stand-alone dump IPL volume (residence volume). After you select the residence volume, you can create the stand-alone dump program. To create the stand-alone dump program, you:
1. Code the AMDSADMP macro. See "Coding the AMDSADMP macro" on page 75.
2. Assemble the macro, placing the stand-alone dump program onto the residence volume in ready-to-load form. IBM recommends that you use one-stage generation when building or creating a stand-alone dump program for the currently executing version of MVS. Use the two-stage generation to create multiple stand-alone dump programs and to create a new version of the stand-alone dump program when migrating to a new version of MVS. See "Generating the stand-alone dump program" on page 96.

MNOTES from the AMDSADMP macro

The output listing from the assembly can contain error messages, called MNOTES, that describe errors made while coding the AMDSADMP macro. To respond to one of these messages, check the specification of the macro and run the assembly step again. The meaning of the severity code is as follows:

8  Assembly processing ends
4  Warning
0  Informational
AMDSADMP Messages, Explanations and Severity Codes

AMDSADMP: COMPACT=compact IS NOT ALLOWED. IT MUST BE YES OR NO. COMPACT=YES HAS BEEN USED.

Explanation: The system could not recognize the value specified on the COMPACT keyword. The stand-alone dump program will use the IDRC feature for the output tape if IDRC is installed.

Severity Code: 0.

AMDSADMP: CONSOLE ADDRESS conad IS INVALID. IT MUST BE A DEVICE NUMBER. 001F IS SUBSTITUTED.

Explanation: The console address operand is not a valid device number of 3 or 4 hexadecimal digits.

Severity Code: 0.

AMDSADMP: CONSOLE PARM NOT DETECTED. DEFAULT (001F, 3278) WILL BE USED.

Explanation: Either the console parameter was not specified or it was not specified correctly on the continuation statement. The parameter was probably not continued correctly on the next defined statement. Continue the interrupted parameter or field beginning in any column from 4 through 16.

See Continuing JCL Statements in z/OS MVS JCL Reference

Severity Code: 0.

AMDSADMP: CONSOLE TYPE contp IS INVALID. IT MUST BE A 4 DIGIT NUMBER. 3278 HAS BEEN USED.

Explanation: An incorrect console type was specified. Only 3277, 3278, 3279, or 3290 are acceptable.

Severity Code: 0.

AMDSADMP: DEFAULT OUTPUT DEVICE T0282 WILL BE USED.

Explanation: A device number was incorrectly specified, or was not specified, on the OUTPUT= parameter.

Severity Code: 0.

AMDSADMP: IPL=ipl IS INVALID. FIRST CHARACTER MUST BE D OR T, AND HAS BEEN REPLACED WITH A D.

Explanation: The IPL operand is incorrect. It is not prefixed with a 'D' or a 'T'.

Severity Code: 4.

AMDSADMP: IPL=ipl IS TOO LONG. THE UNIT NAME WILL BE TRUNCATED.

Explanation: The unit name can be at most 8 characters long.

Severity Code: 4.
Stand-Alone dump

AMDSADMP Messages, Explanations and Severity Codes

AMDSADMP: IPLUNIT WAS NOT SPECIFIED OR IPL= TYPE (D OR T) WAS SPECIFIED INCORRECTLY. UNIT WILL BE DEFAULTED TO SYSDA.

Explanation: The IPL parameter should be specified as IPL=duuu, where ‘d’ is D for direct access or T for tape, and ‘uuu’ is a valid unit type or device number for the SADMP IPL volume as described by the UNIT=uuu JCL parameter.

System Programmer Response: A device number consists of 3 or 4 hexadecimal digits.

Severity Code: 0.

AMDSADMP: MSG=msg IS INVALID. IT MUST BE ALL, ACTION, OR ALLASIDS. MSG=ALL HAS BEEN USED.

Explanation: The MSG operand is not ALL, ACTION, or ALLASIDS.

Severity Code: 0.

AMDSADMP: DDSPROMPT=ddsprompt IS NOT ALLOWED. IT MUST BE YES OR NO. DDSPROMPT=YES HAS BEEN USED.

Explanation: The DDSPROMPT operand is incorrect. It must be either ‘YES’ or ‘NO’. DDSPROMPT=YES is assumed.

System Action: The SADMP program will be generated with run-time dump data set prompting active.

Severity Code: 0.

AMDSADMP: OUTPUT=output IS INCORRECT. IT MUST BE EITHER {T|D}UNIT OR (DUNIT, DATA SET NAME).

Explanation: The OUTPUT operand is incorrect. It must be specified in one of the following formats:
• A ‘T’ or a ‘D’ followed by a device number
• A ‘D’ followed by a device number and a data set name pair specified within parentheses.

System Action: Generation continues, using the default for the OUTPUT operand, T0282, regardless of the format used.

System Programmer Response: The output device must be specified as a 3-digit or 4-digit device number. You can change the OUTPUT parameter at run time, if the default is not what you want.

Severity Code: 4.

AMDSADMP: OUTPUT DUMP DATA SET NAME IS INCORRECT. THE DATA SET NAME IS GREATER THAN 44 CHARACTERS.

Explanation: OUTPUT=(Dunit,ddsnname) was specified, however, the data set name (ddsnname) had a length greater than 44 characters.

System Action: Generation continues, however, no default dump data set name will be generated.

System Programmer Response: If a default dump data set name is desired, correct the OUTPUT= specification and regenerate the SADMP program.

Severity Code: 4.
AMDSADMP Messages, Explanations and Severity Codes

AMDSADMP: OUTPUT DUMP DATA SET NAME IS INCORRECT. IT MUST CONTAIN THE TEXT ‘SADMP’.

Explanation: OUTPUT=(Dunit,ddsname) was specified, however, the data set name (ddsname) did not contain the text ‘SADMP’ as either part of, or as an entire data set qualifier.

System Action: Generation continues, however, no default dump data set name will be generated.

System Programmer Response: If a default dump data set name is desired, correct the OUTPUT= specification and regenerate the SADMP program.

Severity Code: 4.

AMDSADMP: REUSEDS=reuseds IS NOT ALLOWED. VALID SPECIFICATIONS ARE NEVER, CHOICE, OR ALWAYS. REUSEDS=CHOICE HAS BEEN USED.

Explanation: The REUSEDS operand is not NEVER, CHOICE, or ALWAYS.

System Action: Generation continues, using the default for the REUSEDS operand, CHOICE.

Severity Code: 0.

AMDSADMP: ULABEL=NOPURGE IS NOT POSSIBLE FOR A TAPE RESIDENCE VOLUME.

Explanation: The ULABEL cannot be NOPURGE when the IPL device is tape. SADMP ignores your ULABEL specification.

Severity Code: 8.

AMDSADMP: keyword IS AN OBSOLETE KEYWORD. IT IS IGNORED. SADMP GENERATION CONTINUES.

Explanation: An obsolete keyword is specified on the AMDSADMP macro. SADMP no longer requires the LOADPT or TYPE keywords to create a stand-alone dump program.

System Action: The system ignores the keyword and continues processing.

System Programmer Response: To eliminate this MNOTE, remove the indicated keyword and its associated parameter from the generation JCL.

Severity Code: 0.

AMDSADMP: ALIB=alib IS NOT VALID. THE REQUIRED SYNTAX IS ALIB=(VOLSER,UNIT).

Explanation: The system could not recognize the parameters specified on the ALIB keyword. The correct syntax is ALIB=(volser,unit), where volser is the volume serial number and unit is the UNIT=value of the device.

System Action: The system ignores this keyword and continues. The second step JCL might be incorrect.

System Programmer Response: Correct the syntax specified on the AMDSADMP macro and resubmit the JCL.

Severity Code: 8.
AMDSADMP Messages, Explanations and Severity Codes

AMDSADMP: NUCLIB=nuclib IS NOT VALID. THE REQUIRED SYNTAX IS NUCLIB=(VOLSER,UNIT).

Explanation: The system could not recognize the parameters specified on the NUCLIB keyword. The correct syntax is NUCLIB=(volser,unit), where volser is the volume serial number and unit is the UNIT=value of the device.

System Action: The system ignores this keyword and continues. The second step JCL might be incorrect.

System Programmer Response: Correct the syntax specified on the AMDSADMP macro and resubmit the JCL.

Severity Code: 8.

AMDSADMP: MODLIB=modlib IS NOT VALID. THE REQUIRED SYNTAX IS MODLIB=(VOLSER,UNIT).

Explanation: The system could not recognize the parameters specified on the MODLIB keyword. The correct syntax is MODLIB=(volser,unit), where volser is the volume serial number and unit is the UNIT=value of the device.

System Action: The system ignores this keyword and continues. The second step JCL might be incorrect.

System Programmer Response: Correct the syntax specified on the AMDSADMP macro and resubmit the JCL.

Severity Code: 8.

AMDSADMP: LNKLIB=lnklib IS NOT VALID. THE REQUIRED SYNTAX IS MODLIB=(VOLSER,UNIT).

Explanation: The system could not recognize the parameters specified on the LNKLIB keyword. The correct syntax is LNKLIB=(volser,unit), where volser is the volume serial number and unit is the UNIT=value of the device.

System Action: The system ignores this keyword and continues. The second step JCL might be incorrect.

System Programmer Response: Correct the syntax specified on the AMDSADMP macro and resubmit the JCL.

Severity Code: 8.

AMDSADMP: CONSOLE TYPE contp IS INVALID. NO VALUE MAY BE SPECIFIED FOR SYSC. IT WILL BE IGNORED.

Explanation: A console type was specified following the console name of SYSC. No console type is allowed for this console.

System Action: The system ignores the specification.

System Programmer Response: None.

Severity Code: 0.
AMDSADMP Messages, Explanations and Severity Codes

AMDSADMP: CONSOLE ADDRESS SYSC MAY ONLY BE SPECIFIED FOR THE FIRST CONSOLE. IT WILL BE IGNORED.

Explanation: The console name SYSC was not specified as the first console in the console list. SYSC can only be specified as the first console.

System Action: The system ignores the specification.

System Programmer Response: None.

Severity Code: 0.

AMDSADMP: ONLY SYSTEM CONSOLE DEFINED. DEFAULT (001F,3278) WILL ALSO BE USED.

Explanation: The console named SYSC was the only console that was defined. At least one 3270 console must also be defined.

System Action: The system defined a default console of (001F,3278).

System Programmer Response: None.

Severity Code: 0.

AMDSADMP: REAL=ALL OR REAL=USED NOT SPECIFIED, THE DEFAULT REAL=ALL WILL BE USED.

Explanation: Either the REAL keyword was not specified or it was not specified correctly. REAL=ALL is the default.

Severity Code: 0.

AMDSADMP: POSITIONAL value IGNORED.

Explanation: A positional value other than PROMPT appeared as the first positional argument to the AMDSADMP macro. It was ignored.

System Programmer Response: None.

Severity Code: 0.

Coding the AMDSADMP macro

This section describes the coding of the AMDSADMP macro, including the following topics:

- “Using the DUMP or ADDSUMM keyword to request additional storage or address spaces” on page 83
- “Dumping to a DASD data set” on page 87

Syntax of the AMDSADMP macro

Figure 24 on page 76 shows the syntax of the AMDSADMP macro and its parameters.
Stand-Alone dump

```
[symbol] AMDSADMP
[,IPL={Tunit|Dunit|DSYSDA}]
[,VOLSER={volser|SADUMP}]
[,ULABEL={PURGE|NOPURGE}]
[,CONSOLE={{cnum}|(cnum,ctype) [,(cnum,ctype)]}...{01F,3278}]
[,SYSUT={unit|SYSDA}]
[,OUTPUT={Tunit|Dunit|(Dunit,ddsname)|T0282}]
[,DUMP=('options')][,PROMPT]
[,MSG={ACTION|ALLASIDS|ALL}]
[,MINASID={ALL|PHYSIN}]
[,COMPACT={YES|NO}]
[,REUSED={CHOICE|ALWAYS|NEVER}]
[,ALIB=(volser,unit)]
[,NUCLIB=(volser,unit)]
[,MODLIB=(volser,unit)]
[,LNKLIB=(volser,unit)]
[,DDSPROMPT={YES|NO}]
[,AMD029={YES|NO}]
[,IPLEXIST={YES|NO}]
[,ADDSUMM=('options')]`

| [symbol] AMDSADMP | [,IPL={Tunit|Dunit|DSYSDA}] | [,VOLSER={volser|SADUMP}] | [,ULABEL={PURGE|NOPURGE}] | [,CONSOLE={{cnum}|(cnum,ctype) [,(cnum,ctype)]}...{01F,3278}] | [,SYSUT={unit|SYSDA}] | [,OUTPUT={Tunit|Dunit|(Dunit,ddsname)|T0282}] | [,DUMP=('options')][,PROMPT] | [,MSG={ACTION|ALLASIDS|ALL}] | [,MINASID={ALL|PHYSIN}] | [,COMPACT={YES|NO}] | [,REUSED={CHOICE|ALWAYS|NEVER}] | [,ALIB=(volser,unit)] | [,NUCLIB=(volser,unit)] | [,MODLIB=(volser,unit)] | [,LNKLIB=(volser,unit)] | [,DDSPROMPT={YES|NO}] | [,AMD029={YES|NO}] | [,IPLEXIST={YES|NO}] | [,ADDSUMM=('options')] | [,REAL={ALL|USED}] |

Figure 24. Format of AMDSADMP Macro Instruction

**symbol**
An arbitrary name you can assign to the AMDSADMP macro. stand-alone dump uses this symbol to create a job name for use in the initialization step.

**AMDSADMP**
The name of the macro.

**IPL={Tunit|Dunit|DSYSDA}**
Indicates the device number, device type, or esoteric name of the stand-alone dump residence volume. The first character indicates the volume type; T for tape, D for DASD. stand-alone dump uses the unit character string as the UNIT=value to allocate the residence volume for initialization.

A device number consists of 1 to 4 hexadecimal digits. To distinguish a device number from a unit type, the device number must be preceded by a slash (/); for example, you could specify IPL=D/410F. Otherwise, the dynamic allocation of the IPL device (IPLDEV DD-statement) may fail with reason code X'021C' (unavailable system resource).

The default is IPL=DSYSDA. When you specify IPL=T, stand-alone dump assumes T3400. When you specify IPL=D, stand-alone dump assumes DSYSDA.
Note:
1. This device also contains a work file used during stand-alone dump processing.
2. It is not recommended to place the IPL text of stand-alone dump on a volume that contains page data sets. A restart of stand-alone dump (see "Running the stand-alone dump program" on page 107) hangs during the real dump phase in this case.

VOLSER={volser|SADUMP}
Indicates the volume serial number the system is to use to allocate the residence volume for initialization. When you specify a tape volume, it must be NL (no labels). VOLSER=SADUMP is the default.

ULABEL={PURGE|NOPURGE}
Indicates whether stand-alone dump deletes (PURGE) or retains (NOPURGE) existing user labels on a DASD residence volume. When you specify NOPURGE, the stand-alone dump program is written on cylinder 0 track 0 of the residence volume, immediately following all user labels. If the user labels occupy so much space that the stand-alone dump program does not fit on track 0, the initialization program issues an error message and ends. ULABEL=NOPURGE is the default.

CONSOLE=((cnum|((cnum,(ctype))...|01F,3278))
Indicates the device numbers and device types of the stand-alone dump consoles that stand-alone dump is to use while taking the dump. When you specify CONSOLE=cnum, stand-alone dump assumes (cnum,3278). You can specify from two to 21 consoles by coding:
CONSOLE=((SYSC)|(cnum,(ctype)),(cnum,(ctype)),,(cnum,(ctype))...)

A device number consists of 3 or 4 hexadecimal digits, optionally preceded by a slash (/). Use a slash preceding a 4-digit device number to distinguish it from a device type.
The 3277, 3278, 3279, and 3290 device types are valid, and are interchangeable. CONSOLE=(01F,3278) is the default.
You can specify CONSOLE=SYSC for the first console only. SYSC is a constant representing the hardware system console.

Note: The specification of CONSOLE does not affect the availability of the system console.

SYSUT={unit|SYSDA}
Specifies the UNIT=value of the device that stand-alone dump uses for work files during stand-alone dump initialization. You can specify the device as a group name (for example, SYSDA), a device type (for example, 3330), or a unit address (for example, 131). SYSUT=SYSDA is the default.

OUTPUT={(Tunit|Dunit),(Dunit,ddsname)|T0282}
Indicates the device type, number, and data set name that stand-alone dump uses as a default value if the operator uses the EXTERNAL INTERRUPT key to bypass console communication, or if the operator provides a null response to message AMD001A during stand-alone dump initialization. OUTPUT=T0282 is the default.
The device type can be specified as either a ‘T’ for tape or ‘D’ for DASD.
Stand-Alone dump

The device number consists of 3 or 4 hexadecimal digits, optionally preceded by a slash (/). Use a slash preceding a 4-digit device number to distinguish it from a device type.

If the default device is a DASD, you can also set up a default dump data set name to use by specifying both the device and the dump data set name on the OUTPUT= parameter. You can specify the first volume of a multi-volume DASD data set. If you specify a default dump data set name it must:

- Have a length that is 44 characters or less.
- Contain the text ‘SADMP’ as either part of, or as an entire data set qualifier.

Note that AMDSADMP processing does not allocate the data set or check to see that a valid MVS data set name has been provided. Therefore, you should insure that:

- The AMDSADDD REXX is used to allocate and initialize the same data set name specified on the OUTPUT= keyword.
- The data set name specified should be fully qualified (without quotation marks).
- The necessary data set management steps are taken so that the stand-alone dump data sets are not placed into a migrated state or moved to a different volume.
- Alphabetic characters appearing in the dump data set name should be specified as capital letters.

If the default DASD device is to be used and no dump data set name is provided, the stand-alone dump program assumes that the default dump data set name is SYS1.SADMP if the DDSPROMPT=NO parameter was also specified. Otherwise, if DDSPROMPT=YES was specified, the stand-alone dump program prompts the operator at runtime for a dump data set name to use.

Note:
1. At run-time, only a null response to message AMD001A causes the stand-alone dump program to use the default device and/or dump data set name.
2. Do not place a data set that is intended to contain a stand-alone dump on a volume that also contains a page data set that the stand-alone dump program might need to dump. When stand-alone dump initializes a page volume for virtual dump processing, it checks to see if the output dump data set also exists on this volume. If it does, the stand-alone dump program issues message AMD100I and does not retrieve any data from page data sets on this volume. Thus, the dump might not contain all of the data that you requested. This lack of data can impair subsequent diagnosis.
3. You cannot direct output to the stand-alone dump residence volume.

**DUMP=’options’**
Indicates additional virtual storage that you want dumped. This storage is described as address ranges, dataspaces, and subpools in address spaces. When you do not specify DUMP, stand-alone dump does not dump any additional storage unless you specify PROMPT. See "Using the DUMP or ADDSUMM keyword to request additional storage or address spaces" on page 83 for more information.

**PROMPT**
Causes stand-alone dump, at run time, to prompt the operator for additional virtual storage to be dumped. The operator can respond with the same
information that can be specified for the DUMP keyword. When you do not specify PROMPT, stand-alone dump does not prompt the operator to specify additional storage. See "Using the DUMP or ADDSUMM keyword to request additional storage or address spaces" on page 83 for more information.

**MSG=(ACTION|ALLASIDS|ALL)**
Indicates the type of stand-alone dump messages that appear on the console. When you specify ACTION, stand-alone dump writes only messages that require operator action. When you specify ALL, stand-alone dump writes most messages to the console. However, messages AMD010I, AMD057I, AMD076I, AMD081I, and AMD102I appear only in the stand-alone dump message log. When you specify ALLASIDS, the stand-alone dump program behaves as if MSG=ALL was specified, except that message AMD010I also appears on the console. ALL is the default.

This keyword has no effect on the stand-alone dump message log; even if you specify MSG=ACTION, the stand-alone dump virtual dump program writes all messages to the message log in the dump.

**MINASID=(ALL|PHYSIN)**
Indicates the status of the address spaces that are to be included in the minimal dump. Specify PHYSIN to dump the minimum virtual storage (LSQA and selected system subpools) for the physically swapped-in address spaces only. Specify ALL to dump the minimum virtual storage (LSQA and selected system subpools) for all of the address spaces. ALL is the default. At run time, if PHYSIN was specified, stand-alone dump writes message AMD082I to the operator's console to warn the operator that some virtual storage might be excluded from the dump.

**COMPACT=(YES|NO)**
COMPACT(YES) compacts the data stored on a tape cartridge if the IDRC hardware feature is available on your tape drive. If the IDRC feature is available and you do not specify the COMPACT keyword, the default is YES, so that IDRC compacts the dump data. Otherwise, the data is handled as usual.

**REUSED=(CHOICE|ALWAYS|NEVER)**
Indicates whether stand-alone dump should reuse the dump data set on the specified output device when it determines that the data set is valid, however, it can contain data from a previous dump. Stand-alone dump determines this by checking to see if the first record in the data set matches the record that is written by the AMDSADDD rexx utility. When you specify ALWAYS, stand-alone dump issues message AMD094I and reuses the specified dump data set. When you specify NEVER, stand-alone dump issues message AMD093I and prompts the operator, through message AMD001A, for an output device. When you specify CHOICE, stand-alone dump informs the operator, with message AMD096A, that the data set is not reinitialized and requests permission to reuse the data set. See for more information about defining, clearing, and reallocating the dump data set.

CHOICE is the default.

**ALIB=(volser,unit)**
Specifies the volume serial number and UNIT=value of the volume that contains all of the following system data sets:

- SYS1.MODGEN
- SYS1.LINKLIB
- SYS1.NUCLEUS
Stand-Alone dump

This parameter is valid only when you are generating the stand-alone dump program using two-stage generation.

**Note:** The specification of the NUCLIB, LNKLIB, or MODLIB parameters overrides the corresponding value specified on the ALIB parameter.

See “Using two-stage generation of stand-alone dump when migrating” on page 103 for information on the use of this parameter.

**NUCLIB=(volser,unit)**
Specifies the volume serial number and UNIT=value of the volume that contains the system data set SYS1.NUCLEUS. If you specify NUCLIB, there is no need to specify IPLTEXT, IPITEXT, DVITEXT, DPLTEXT and PGTEXT DD statements. Beginning with z/OS V1R12, this parameter is valid for one-stage generation JCL. Prior to z/OS V1R12, this parameter is valid only when you generate the stand-alone dump program using two-stage generation. See “One-stage generation” on page 96 for information on the use of this parameter.

**MODLIB=(volser,unit)**
Specifies the volume serial number and UNIT=value of the volume that contains the system data set SYS1.MODGEN. This parameter is valid only when you generate the stand-alone dump program using two-stage generation. See “Using two-stage generation of stand-alone dump when migrating” on page 103 for information on the use of this parameter.

**LNKLIB=(volser,unit)**
Specifies the volume serial number and UNIT=value of the volume that contains the system data set SYS1.LINKLIB. This parameter is valid only when you generate the stand-alone dump program using two-stage generation. See “Using two-stage generation of stand-alone dump when migrating” on page 103 for information on the use of this parameter.

**DDSPROMPT={YES|NO}**
DDSPROMPT=YES allows the stand-alone dump program to prompt the operator for an output dump data set when dumping to a DASD device. When DDSPROMPT=YES is specified, after replying to message AMD001A with a DASD device number, message AMD002A is also issued to prompt the operator for a dump data set name.

DDSPROMPT=NO indicates that the stand-alone dump program should not prompt for a dump data set name when dumping to a DASD device. When DDSPROMPT=NO is specified, after replying to message AMD001A with a DASD device number, the stand-alone dump program uses data set SYS1.SADMP. DDSPROMPT=NO is the default.

Note that regardless of the DDSPROMPT= keyword value, you can always use a default device and dump data set name by specifying the OUTPUT=(Dunit,ddsname) keyword. The stand-alone dump program uses the default values specified on the OUTPUT= keyword when the operator uses the EXTERNAL INTERRUPT key to bypass console communication, or if the operator provides a null response to message AMD001A.

**AMD029={YES|NO}**
If AMD029=NO is specified, SADMP does not issue AMD029D when a 3270 console screen becomes full. SADMP behaves as if the operator had replied NO to AMD029D. This parameter is meaningless when the system console is used, because AMD029D is never issued for the system console. AMD029=YES is the default.
IPLEXIST={YES|NO}

If IPLEXIST=YES is specified, SADMP includes IPLEXIST with the ICKDSF parameters, so that ICKDSF does not prompt the operator with message ICK21836D if there is already IPL text on the volume. IPLEXIST=NO is the default.

ADDSUMM=('[options]')

Indicates additional address spaces that you want dumped during a summary phase. Default summary address spaces are always dumped during a summary phase. If you do not specify ADDSUMM, stand-alone dump dumps only the default summary address spaces unless you specify PROMPT, in which case you have the opportunity to dump additional address spaces at run time. See "Using the DUMP or ADDSUMM keyword to request additional storage or address spaces" on page 83 for more information.

REAL={ALL|USED}

Indicates which real storage to dump. Specify USED to dump real storage that is being used at the time of the dump. Specify ALL to dump all real storage (used and unused). ALL is the default.

Examples of Coding the AMDSADMP Macro

The following examples show how to code the AMDSADMP macro to create various kinds of stand-alone dump programs.

Figure 25 shows the AMDSADMP macro coded without explicitly specified parameters to generate a direct access resident dump program. The defaults are:

| IPL=DSYSDA |
| VOLSER=SADUMP |
| ULABEL=NOPURGE |
| CONSOLE=(01F,3278) |
| SYSUT=SYSDA |
| OUTPUT=T282 |
| MSG=ALL |
| MINASID=ALL |
| COMPACT=YES |
| REUSEDS=CHOICE |
| DDSPROMPT=NO |
| REAL=ALL |

DUMP1 AMDSADMP

Figure 25. Example: Accepting All Defaults

In Figure 26 on page 82 the IPL parameter specifies tape as the residence volume, and the VOLSER parameter identifies that tape. All other parameters are allowed to default. The defaults are:

| ULABEL=NOPURGE |
| CONSOLE=(01F,3278) |
| SYSUT=SYSDA |
| OUTPUT=T282 |
| MSG=ALL |
| MINASID=ALL |
| COMPACT=YES |
| REUSEDS=CHOICE |
| DDSPROMPT=NO |
| REAL=ALL |
In Figure 27, the OUTPUT parameter directs the stand-alone dump output to dump data set SYS1.SADMP on device 450, and the REUSEDS parameter specifies that the operator be prompted about whether to reuse the dump data set. The defaults are:

```
IPL=DSYSDA
VOLSER=SADUMP
ULABEL=NOPURGE
CONSOLE=(01F,3278)
SYSUT=SYSDA
MSG=ALL
MINASID=ALL
COMPACT=YES
DDSPROMPT=NO
REAL=ALL
```

Figure 27. Example: Generating a dump program with output to DASD

In Figure 28, the OUTPUT parameter directs the stand-alone dump output to dump data set SADMP.DDS1 on device 450. Furthermore, the DDSPROMPT=YES keyword allows for run-time dump data set prompting. The defaults are:

```
IPL=DSYSDA
VOLSER=SADUMP
ULABEL=NOPURGE
CONSOLE=(01F,3278)
SYSUT=SYSDA
MSG=ALL
MINASID=ALL
COMPACT=YES
REUSEDS=CHOICE
REAL=ALL
```

Figure 28. Example: Generating a dump program with output to DASD

Recommended specification during the build process is as follows:

```
SP(ALL) IN ASID(1,'JESXCF')
ALSO DATASPACES OF ASID(1,'JESXCF', 'APPC', 'SMSVSAM', 'CONSOLE', 'SYSBMAS')
ALSO PAGETABLES OF DATASPACES
```

If you run JES2, add:

```
ALSO SP(ALL) IN ASID('JES2')
```

Additional subpools and dataspaces might be needed, depending on your installed IBM, vendor, and locally-written products and applications.
Using the DUMP or ADDSUMM keyword to request additional storage or address spaces

You can request that stand-alone dump dump additional storage or additional address spaces in two ways:

- **Specifying DUMP options or ADDSUMM options on the AMDSADMP macro**
  
  As the following example show, specify the dump tailoring options described in "Dump tailoring options" on page 84 within parentheses and single quotation marks as the value of the DUMP keyword on the AMDSADMP macro.

  ```
  DUMP=('SP(5,37,18) IN ASID('JES3')')
  DUMP=('RANGE(0:1000000) IN ASID(1)')
  DUMP=('DATASPACE OF ASID('DUMPSRV')')
  DUMP=('SADMPNO IN ASID('IOSAS')')
  ```

  **Note:** Do not double the quotation marks within the DUMP options. The DUMP options cannot exceed 255 characters in length.

  You can also specify additional summary address space as the value of the ADDSUMM keyword on the AMDSADMP macro.

  ```
  ADDSUMM=('ASID('BPX*')')
  ADDSUMM=('ASID(28),ASID('JOHNDOE')')
  ADDSUMM=('ASID('MYDB',40,46:48)')
  ```

  **Note:** Do not double the quotation marks within the ADDSUMM options. The ADDSUMM options cannot exceed 255 characters in length.

- **Specifying additional dump options at run time**

  By coding the PROMPT keyword on the AMDSADMP macro, you can have Stand-alone dump prompt the operator to dump additional storage or specify additional address spaces to be dumped as part of summary phase. When you code PROMPT, and the virtual storage dump program gets control, stand-alone dump issues the AMD059D message:

  ```
  AMD059D ENTER 'DUMP' OR 'SET' OR 'ADDSUMM' WITH OPTIONS, 'LIST' OR 'END'.
  ```

  The operator can respond with one of the following:

  - **DUMP** followed by dump options. In this case, the ‘=’ after DUMP is optional. See "Dump tailoring options" on page 84 for the possible dump options.
  - **SET** followed by the MINASID or REAL options.
  - **ADDSUMM** followed by address space list to additionally be dumped as part of summary phase. In this case, the ‘=’ after ADDSUMM is optional. See "Dump tailoring options" on page 84 for the possible options.
  - **LIST.** On the console, stand-alone dump displays the current virtual storage areas to be dumped and address space list to be dumped during the summary phase.
  - **END.** Stand-alone dump stops prompting the operator for options and begins processing.

  Figure 29 on page 84 shows a sample exchange between stand-alone dump and the operator. The operator’s replies are in lowercase. Note the operator’s reply to message AMD059D using the DUMP keyword and ADDSUMM keyword.
Stand-Alone dump

When stand-alone dump detects an error in the reply to message AMD059D, it
repeats the incorrect line at the console, underscores the incorrect part with
asterisks, and prompts the operator for replacement text. If the dump options
exceed 255 characters, stand-alone dump marks the whole line in error.

If a system restart occurs during the virtual storage dump program, stand-alone
dump re-prompts the operator for dump options. stand-alone dump does not
use any of the dump options that the operator specified before the system
restart.

Dump tailoring options: You can specify the dump tailoring options in one or all
of the following ways:
• On the DUMP keyword of the AMSADMP macro.
• On the ADDSUMM keyword of the AMSADMP macro.
• By the operator in reply to message AMD059D at run time.

Following is a list of the dump tailoring options you can specify. For a complete
explanation of the options, see “Explanation of dump tailoring options” on page
85.

dump-spec-list is one or more of the following:
• range-spec-list IN ASID(address-space-list) [ALSO...]
• DATASPACE(S OF domain-spec-list[ ...]
• DSP OF domain-spec-list[ ...]
• PAGETABLES OF DATASPACE(S
  range-spec-list is one or more of the following:
  – SP(subpool-list)
  – RANGE(address-range-list)
Stand-Alone dump

- LSQA
- HIGH VIRTUAL
- SADMPNO

  subpool-list is one of the following:
  - subpool-number TO subpool-number [...]
  - ALL

  address-range-list is one of the following:
  - address TO address [...]
  - ALL

  address-space-list is one of the following:
  - asid TO [asid | jobname | SYSKEY | PHYSIN][...]
  - ALL

addsumm-spec-list is the following:

- ASID (address-space-list) [ , ASID(address-space-list)][,...]

  address-space-list is the following:
  - asid[ dlm asid] | jobname[,...]
    dlm is ‘TO’ or ‘.’.

Use the following guidelines when specifying values in your dump tailoring options:

- HIGH VIRTUAL cannot be used by itself; specify additional keywords.
- SADMPNO cannot be used by itself; specify additional keywords.
- address is a hexadecimal number from 0 to X'7FFFFFFF'.
- subpool-number is a decimal number from 0 to 255.
- asid is a hexadecimal number from 0 to X'FFFF'.
- jobname is a valid jobname enclosed in single quotation marks. Including wildcard characters is valid for jobnames. For a description of wildcard characters, see the ASID('jjjj') option in the topic "Explanation of dump tailoring options."
- range-spec-list is a list of subpools, a list of storage ranges, or both.
- domain-spec-list is a list of address spaces.
- ‘TO’ and ‘.’ are synonyms.
- ‘DATASPACES’ and ‘DSP’ are synonyms.

Keywords, such as DATASPACES, can be truncated on the right, provided the truncated form is not ambiguous. You can enter letters in either lower-case or uppercase. Blanks can be inserted between numbers, keywords, and separators; blanks cannot be inserted within numbers or keywords.

Explanation of dump tailoring options: This section provides an explanation for each of the dump tailoring options.

**RANGE(xxxxxxxx:yyyyyyyy,xxxxxxxx:yyyyyyyy...)**

  Specifies one or more ranges of storage that you want dumped. xxx and yyy are hexadecimal addresses from 0 to X'7FFFFFFF'.

**RANGE(ALL)**

  Specifies dumping of all storage from 0 to X'7FFFFFFF'.
Stand-Alone dump

**SP(ddd)**
Causes stand-alone dump to dump subpool *ddd*. *ddd* is a decimal integer from 0 to 255.

**SP(ddd:eee)**
Causes stand-alone dump to dump all subpools from *ddd* to *eee*, inclusive.

**SP(ddd:eee,ddd:eee,...)**
Causes stand-alone dump to dump the combination of subpools that you specify.

**SP(ALL)**
Causes stand-alone dump to dump all subpools, from 0 to 255 inclusive.

**LSQA**
Causes stand-alone dump to dump the LSQA.

**HIGH VIRTUAL**
Causes stand-alone dump to dump all allocated paged out storage above 2G.
This applies to storage that was not being excluded because of a SADMP=NO specification. See "SADMPNO" for more information.

**SADMPNO**
Causes stand-alone dump to dump memory objects created with SADMP=NO in the specified address space(s). See IARV64 in Authorized Assembler Services Reference EDT-IXG for more information.

**ASID(xxx:yyy)**
Causes stand-alone dump to dump storage for the range of address spaces whose ASIDs begin at *xxx* and end at *yyy*, inclusive. *xxx* and *yyy* are hexadecimal numbers from X'1' to X'FFFF'.

**ASID('jjj')**
Causes stand-alone dump to dump storage for the address space that jobname *jjj* identifies. Note that you must enclose the jobname in single quotation marks.
You can use wildcard characters to identify multiple jobnames. The valid wildcard characters are:

* Zero or more characters, up to the maximum length of the string. An asterisk can start the string, end it, appear in the middle of the string, or appear in several places in the string. A single asterisk for the jobname indicates that all jobnames match.

? One character. One or more question marks can start the string, end it, appear in the middle of the string, or appear in several places in the string. A single question mark indicates all jobnames consisting of one character.

**ASID(SYSKEY)**
Causes stand-alone dump to dump storage for all address spaces whose active TCB has an associated storage key of 0 to 7.

**ASID(combination)**
You can combine any of the above specifications. An example of a valid combination is ASID(2,'IMSJOB',SYSKEY).

**ASID(PHYSIN)**
Causes stand-alone dump to dump storage for physically swapped-in address spaces.
ASID(ALL)
Causes stand-alone dump to dump storage for all address spaces. Note that you cannot specify ASID(ALL) in combination with any of the other ASID specifications.

DATASPACES OF ASID(qualifier)
When you specify the DATASPACES OF ASID(qualifier) keyword, stand-alone dump dumps all data spaces owned by the specified address space. For each requested data space, stand-alone dump:
- Dumps pages backed by central storage during the central storage dump
- Copies into central storage and dumps every page that is not a first reference page and not backed by central storage

PAGETABLES OF DATASPACES
When you specify the PAGETABLES OF DATASPACES keyword, stand-alone dump dumps paged-out virtual storage that contains the page tables for all data spaces.

When stand-alone dump dumps the storage that you specify, stand-alone dump dumps all listed subpools and address ranges in all specified address spaces for each specification of dump options. However, stand-alone dump does not merge your specifications across the dump options that you specify. For example, to cause stand-alone dump to dump subpools 0 and 1 in address space A, and subpools 0 and 1 in address space B, enter:

DUMP SP(0,1) IN ASID(A,B)

To cause stand-alone dump to dump subpool 0 in address space A and subpool 1 in address space B, enter:

DUMP SP(0) IN ASID(A) ALSO SP(1) IN ASID(B)

Figure 30 shows other examples of valid specifications.

Figure 30. Example of valid specifications

Dumping to a DASD data set
When you specify DASD on the OUTPUT parameter, you direct the output of the stand-alone dump program to a predefined dump data set on one of the following types of DASD:
- 3380
- 3390
- 9345
Stand-Alone dump

**Note:** The selection of the output device (DASD or tape) can be made at both generation time and at run time. An output device specified at run time overrides an output device specified at generation time.

When preparing to take a stand-alone dump to DASD, you must allocate and initialize the dump data set using the AMDSADDD REXX or IPCS SADMP dump data set utilities.

The following requirements exist for the allocation of the DASD dump data set:

- The dump data set must have the text ‘SADMP’ as either part of, or as an entire data set qualifier.
- Do not place a data set that is intended to contain a stand-alone dump on a volume that also contains a page data set that the stand-alone dump program you might need to dump. When stand-alone dump initializes a page volume for virtual dump processing, it checks to see if the output dump data set also exists on this volume. If it does, the stand-alone dump program issues message AMD100I and does not retrieve any data from page data sets on this volume. Thus, the dump might not contain all of the data that you requested. This lack of data can impair subsequent diagnosis.
- The dump data set cannot be defined on the same volume that contains the IPL text of stand-alone dump.

**Note:** Because the data set does not have to be cataloged, there can be more than one dump data set per system. Furthermore, because the data set can be uniquely named, there can be more than one dump data set per volume.

- IBM recommends that you define the dump data set on a volume that does not contain any other data sets, especially volumes that contain sysplex couple data sets. This ensures maximum capacity when needed and avoid the possibility of other data sets being accessed by another system.
- The dump data set must be both allocated and initialized using the AMDSADDD REXX or IPCS SADMP dump data set utilities.
- Because the stand-alone dump program must be able to locate the dump data set on the output device being used, it is imperative that the necessary data set management steps be taken so that the stand-alone dump data sets are not placed into a migrated state or moved to a different volume. The dump data sets must also be exempt from any space management processing that releases unused space.

When the dump data set is filled, the stand-alone dump program prompts the operator, with message AMD001A, to specify another output device. The stand-alone dump program can continue dumping to any stand-alone dump supported device, however, after a tape device is selected, it must be used to complete the dump even though multiple tape volumes can be required.

**Note:** Dumping to multiple DASD dump data sets requires that each dump data set used has been preformatted by the AMDSADDD REXX or IPCS SADMP dump data set utilities.

**Using the AMDSADDD utility**

The REXX utility AMDSADDD resides in SYS1.SBLSCLl0. This section describes how to use the AMDSADDD REXX utility to:

- Allocate and initialize the data set. See Figure 31 on page 93 for an example of allocating and initializing the dump data set.
Stand-Alone dump

- Clear (reinitialize) the data set. See Figure 33 on page 94 for an example of clearing the dump data set.
- Reallocate and initialize the data set. See Figure 34 on page 95 for an example of reallocating and initializing the dump data set.

The IPCS SADMP dump data set utility performs the same functions as the AMDSADDD REXX utility. See SADMP option on the IPCS Dialog in z/OS MVS IPCS User’s Guide for more information. See z/OS MVS IPCS Customization for more information on the migration tasks involving AMDSADDD.

The data set allocated by the AMDSADDD REXX utility must have these characteristics:

- The data set name (DSNAME) must:
  - be 44 characters or less in length
  - contain the text ‘SADMP’ as either part of, or as an entire data set qualifier.
  For example, valid dump data set names are:
  - SYS1.SADMP
  - SADMP
  - SYSTEMA.SADMPDDS
  Invalid dump data set names are:
  - SYS1.DUMP.DATASET
  - SADUMP
- The block size (BLKSIZE) must be:

  **DASD BLKSIZE**

  | 3380, 9345 |
  | 20800 |
  | 3390 |
  | 24960 |

  **Note:** Stand-alone dump processing can use a 3390 DASD defined with a block size of 20800; however, the allocated space is not fully utilized unless a block size of 24960 is used. The AMDSADDD REXX utility allocates 3390 DASD devices using a block size of 24960.

- The logical record length (LRECL) must be 4160.
- The record format (RECFM) must be FBS.
- The data set must consist of a single extent.
- The data set organization can be PS (DSORG=PS) or non-VSAM extended format (DSORG=PS-E).
- Occupies space on 1-32 volumes.

All stand-alone dump data sets that are SMS managed must have a STORCLAS with the GUARANTEED_SPACE attribute.

All DSORG=PS-E data sets must:

- Be SMS managed
- Have a DATACLAS that specifies no compression
- Have a STORCLAS that specifies a sustained data rate of zero (suppress striping).
For SMS to honor the allocation request, your installation's automatic class selection (ACS) routines must be configured to do so. For instructions on setting up an SMS environment, see the following publications:

- **z/OS DFSMS Using Data Sets**
- **z/OS DFSMSdfp Storage Administration**

You provide the volume, dump data set name, unit, space, and catalog disposition on the invocation of the AMDSADDD REXX utility. If multiple volumes are specified, then a multi-volume data set is allocated and formatted. Up to 16 volumes can be specified, all having the same device type. The amount of space specified for the data set is allocated on each volume.

Special control information is written to multi-volume data sets to allow all of the volumes to be located when the data set is written to. This includes the device number of the volume. The data set is not usable by stand-alone dump if the control information is missing or invalid. If a volume of a multi-volume data set is moved to a new device number, the data set must be re-initialized to update the control information. The data set cannot be used by a system that has the volumes attached at a device number different than the system which writes the control information.

When using multi-volume data sets, it is highly recommended that they be cataloged. This simplifies processing, as IPCS can easily be used to format and copy the dump data in the cataloged data sets.

**Note:** REXX requires that the specified parameters appear in the order listed. If you do not specify a parameter, the AMDSADDD REXX utility prompts for a specification of that parameter.

```
AMDSADDD {DEFINE|CLEAR|REALLOC}
volser{(data set name)}
	(type{,[STORCLAS],,[DATACLAS],[,MGMTCLAS]])} {space}
	{YES|NO} {EXTREQ|LARGE|BASIC} {EATTR(OPT|NO)}
```

or

```
AMDSADDD {DEFINE|CLEAR|REALLOC}
{volumelist}{(data set name)}
	(type{,[STORCLAS],[,DATACLAS],[,MGMTCLAS]}{space}
	{YES|NO} {EXTREQ|LARGE|BASIC} {EATTR(OPT|NO)}
```

**AMDSADDD**

The name of the REXX utility.

**DEFINE|CLEAR|REALLOC**

Indicates the function to be performed by the AMDSADDD REXX utility:

- **DEFINE**
  Allocates and initializes a new dump data set.

- **CLEAR**
  Initializes an existing dump data set again. After you use CLEAR, the data set is ready for use.

- **REALLOC**
  Deletes an existing SYS1.SADMP dump data set, then reallocates and
reinitializes a new SYS1.SADMP dump data set on the same volume. Use REALLOC, for example, to increase the size of the dump data set. If the existing dump data set does not exist, AMDSADDD converts the function to a DEFINE request and continue using DEFINE processing. If the request to reallocate and reinitialize a new SYS1.SADMP dump data set cannot be satisfied (for example, if you attempt to reallocate SYS1.SADMP using more cylinders than are available), AMDSADDD might delete the existing SYS1.SADMP dump data set.

**Note:** When specifying the REALLOC option for an existing multi-volume data set, the same list of volumes must be specified as when the dataset was originally allocated.

### volser{(data set name)}
Indicates the VOL=SER= name of the volume on which the dump data set is to be allocated. Do not use the stand-alone dump residence volume or the volumes containing the system paging data sets. Optionally, also defines the dump data set name to be allocated on the volume. If data set name is specified, it must:

- be fully qualified (without quotation marks)
- have a length of 44 characters or less
- contain the text 'SADMP' as either part of, or as an entire data set qualifier.

**Note:** If no data set name is specified, the AMDSADDD utility will allocate the data set SYS1.SADMP on the specified volume.

### (vollist){(data set name)}
*vollist* is a comma delineated list of volsers to use for the data set. A multi-volume data set will be allocated using the list of volumes. The device number of the first volume is used to specify the data set to stand-alone dump.

**Tip:** When you take a stand-alone dump to a multi-volume data set it will be striped and take significantly less time to capture.

### (type[, [STORCLAS][, [DATACLAS][, [MGMTCLAS]]]]])
Type indicates the device type on which the dump data set should be allocated. Valid DASD types are 3380, 3390, and 9345.

- **STORCLAS**
  The SMS storage class.

- **DATACLAS**
  The SMS data class.

- **MGMTCLAS**
  The SMS management class.

For additional information on these classes, see [z/OS MVS JCL Reference](#).

### space
Indicates the number of cylinders for the dump data set to be allocated. For a multi-volume data set, this amount is allocated on each volume.

The size of your dump output depends on your storage configuration and how much of that storage you choose to dump using the options of stand-alone dump. To estimate how much space, in cylinders, to allocate for your dump data set, use the number of cylinders of DASD that a typical dump to tape consumes when it has been copied to DASD for IPCS processing. If you do not allocate enough space, the stand-alone dump program prompts the operator, through message AMD001A and message AMD002A (if DDSPROMPT=YES.
Stand-Alone dump

was specified on the AMDSADMP macro), to specify a different device and/or a different dump data set so that dumping can continue.

The space option is not required with the CLEAR parameter. The space option is, however, required with the DEFINE and REALLOC parameters.

**YES|NO**

Specifies whether the system is to catalog the dump data set. If you want the data set to be cataloged, specify **YES** or **Y**. If you do not want the data set to be cataloged, specify **NO** or **N**. Specifying **N** allows you to allocate multiple dump data sets with the same name.

The catalog option is not required with the CLEAR parameter. The catalog option is, however, required with the DEFINE and REALLOC parameters.

**EXTREQ|LARGE|BASIC**

Indicates the DSNTYPE of the dump data set to be defined.

EXTREQ requests an extended format dump data set. This data set must have the attribute DSNTYPE=EXTREQ. This attribute allows the system to place the data set in cylinder-managed space on extended access volumes.

LARGE requests a large format dump data set, one with attribute DSNTYPE=LARGE that the system allows to span more than 64K tracks per volume.

BASIC indicates that a large format dump data set is not desired. BASIC can be associated with a conventional dump data set or an extended format sequential dump data set, depending on other options. BASIC is the default.

The dsntype option is not required with CLEAR parameter. The dsntype is optional with DEFINE and REALLOC parameters. The dsntype option with REALLOC must match with the existing dsntype option.

**EATTR(OPT|NO)**

Indicates the extended attributes of a dump data set. The EATTR option is not required with the CLEAR parameter.

OPT requests that extended attributes are optional. The system might store the dump data set in the cylinder-managed space on extended access volumes.

NO request that extended attributes are not required. The default value is NO.

**Figure 31 on page 93** shows an example of using the AMDSADDD REXX utility to allocate and initialize the dump data set with a size of 350 cylinders and a VOL=SER= of SAMPLE. Because no data set name is specified, AMDSADDD allocates the dump data set SYS1.SADMP on the volume SAMPLE.

**Note:** Stand-alone dump does not issue error messages during the processing of AMDSADDD. Stand-alone dump does, however, pass messages to the operator from other sources, such as the TSO/E ALLOC command.
Figure 32 on page 94 shows an example of using the AMDSADDD utility to allocate and initialize an extended format dump data set ‘SADMP.SAMPLE’ with a size of 400 cylinders in the cylinder-managed space. This SMS managed dump data set spans multiple volumes SADPK1 and SADPK2. In an extended address volume environment, some systems might round up the cylinders causing the requested amount and allocated amount to be different. In this case, a message is displayed that indicates the requested amount of cylinders and the allocated amount of cylinders.
Figure 33 shows an example of using the AMDSADDD REXX utility to clear (reinitialize) an existing dump data set called SADMP.DDS1 on VOL=SER=SAMPLE. In this example, the parameters are part of the invocation of the utility; therefore, AMDSADDD does not prompt for values.

Figure 33. Using AMDSADDD to Clear an Existing Dump Data Set
Figure 34 shows an example of using the AMDSADDD REXX utility to allocate a new dump data set called SYSTEM1.SADMPDDS on VOL=SER=SMS001. In this example, the parameters are part of the invocation of the utility; therefore, AMDSADDD does not prompt for values.

**Note:** In an SMS environment, it is possible to have the dump data set cataloged on a different volume than the one specified. If the dump data set is allocated on a different volume, AMDSADDD issues an error message and exits. In Figure 34, the dump data set was not allocated on the specified volume causing AMDSADDD to delete the dump data set, issue an error message and quit.

---

**Examples of running AMDSADDD in batch mode**

The following examples show how to use JCL to allocate and initialize dump data sets.

**Note:** Because users cannot be prompted to enter values when invoking the AMDSADDD REXX utility in batch mode, you must specify all parameters in the order listed.

Figure 35 shows how to use JCL to allocate and initialize the dump data set SYS1.SADMP.A1 on VOL=SER=ZOSSVA with a size of 2653 cylinders. The BASIC type of data set is allocated because the dsntype parameter is not specified.

---

**Figure 35. Example: Using JCL to allocate and initialize a dump data set**

Figure 36 on page 96 shows how to use JCL to allocate and initialize an extended format dump data set named SADMP.DS on VOL=SER=USRDS1 with a size of 2953 cylinders in the cylinder-managed space.
Generating the stand-alone dump program

After coding the AMDSADMP macro, you can generate the stand-alone dump program. There are two ways to generate the stand-alone dump program:

- "One-stage generation"
- "Two-stage generation" on page 103

IBM recommends that you use one-stage generation to create the stand-alone dump program because multiple tasks are performed in one-stage.

**Note:** You can use either two-stage or one-stage JCL when migrating to a new version of MVS.

**One-stage generation**

In one-stage generation, run the AMDSAOSG program as a single job, using the AMDSADMP macro you have coded as input data on the GENPARMS control statement. The stand-alone dump utility program, AMDSAOSG, initializes a stand-alone dump residence volume in one job by dynamically allocating data sets and invoking the appropriate programs. To run the one-stage generation program, indicate one AMDSADMP macro as a control statement for DDNAME GENPARMS.

---

**Figure 36. Example: Using JCL to allocate and initialize an extended format dump data set**

//SAMPLE JOB 'S3031,B707000,S=C', 'BATCH EXAMPLE', RD=R,
// MSGLEVEL=(1,1),CLASS=E,NOTIFY=ASYSUID,MSGCLASS=H
//STEP1 EXEC PGM=IKJEFT01,REGION=64M
//SYSTSPRT DD SYSTSPRT=
//SYSTSIN DD *
EXEC 'SYS1.SBLSCLI0.EXEC'
'DEFINE USRDSI(SADMP.DS)(3390,storclas) 2953 Y EXTREQ OPT'
/*
Figure 36. Example: Using JCL to allocate and initialize an extended format dump data set

---

**Figure 37. Example: One-stage generation**

//SADMPGEN JOB MSGLEVEL=(1,1)
//OSG EXEC PGM=AMDSAOSG REGION=5M
//SYSLIB DD DSN=Sys1.LIB,DISP=SHR
//TRKTEXT DD DSN=TRKTEXT,DISP=(,PASS),
// SPACE=(4096,(2,1)),UNIT=SYSDA
//DSFSYSIN DD DSN=DSFSYSIN,DISP=(,PASS),
// SPACE=(80,(4,1)),UNIT=SYSDA
//GENPRINT DD DSN=SADMP.LIST,DISP=OLD
//GENPARMS DD *
 AMDSADMP IPL=SYSDA, VOLSER=SPOOL2,
 X CONSOLE=(1A0,3277)
END
/*
//PUTIPL EXEC PGM=ICKDSF REGION=5M
//IPLDEV DD DISP=OLD,UNIT=SYSDA,
// VOL=(PRIVATE,RETAIN,SER=SPOOL2)
//TRKTEXT DD DSN=TRKTEXT,DISP=(OLD,DELETE)
//SYSIN DD DSN=DSFSYSIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSPRINT=
//SADMPGEN JOB MSGLEVEL=(1,1)
//OSG EXEC PGM=AMDSAOSG, REGION=5M
//SYSLIB DD DSN=SYS1.LIB,DISP=SHR
//TRKTEXT DD DSN=TRKTEXT,DISP=(,PASS),
// SPACE=(4096,(2,1)),UNIT=SYSDA
//DSFSYSIN DD DSN=DSFSYSIN,DISP=(,PASS),
// SPACE=(80,(4,1)),UNIT=SYSDA
//GENPRINT DD DSN=SADMP.LIST,DISP=OLD
//GENPARMS DD *
 AMDSADMP IPL=SYSDA, VOLSER=SPOOL2,
 X CONSOLE=(1A0,3277)
END
/*

---

**Figure 37. Example: One-stage generation**
Table 15 contains the DDNAMES AMDSAOSG uses, and the defaults for the DDNAMES. Additional related notes follow this table.

<table>
<thead>
<tr>
<th>ddname</th>
<th>Default value</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPLTEXT</td>
<td>DSN=SYS1.NUCLEUS(AMDSADPL),DISP=SHR</td>
<td>Input for AMDSABLD.</td>
</tr>
<tr>
<td>DVTTEXT</td>
<td>DSN=SYS1.NUCLEUS(AMDSADVI),DISP=SHR</td>
<td>Input for AMDSABLD.</td>
</tr>
<tr>
<td>GENPARMS</td>
<td>Must be preallocated.</td>
<td>Input for AMDSAOSG, passed to assembler.</td>
</tr>
<tr>
<td>GENPRINT</td>
<td>SYSOUT=A</td>
<td>Output listing from AMDSAOSG.</td>
</tr>
<tr>
<td>PITEX TEXT</td>
<td>DSN=SYS1.NUCLEUS(AMDSAIP),DISP=SHR</td>
<td>Input for AMDSABLD.</td>
</tr>
<tr>
<td>IPLDEV</td>
<td>DSN=SYS1.PAGEDUMP.Volser,UNIT=iplunit, VOL=(PRIVATE,SER=iplser), DISP=OLD,DCB=(BLKSIZE=12288,RECFM=U, DSORG=PS), LABEL=(),NL</td>
<td>Stand-alone dump program, output from AMDSABLD. ICKDSF uses VOL keywords to describe the residence volume.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tape IPL volume.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DASD IPL volume.</td>
</tr>
<tr>
<td>IPLTEXT</td>
<td>DSN=SYS1.NUCLEUS(AMDSAI),DISP=SHR                                         for DASD</td>
<td>Input for AMDSABLD.</td>
</tr>
<tr>
<td>PGTEXT</td>
<td>DSN=SYS1.NUCLEUS(AMDSAPGE),DISP=SHR                                         for tape</td>
<td></td>
</tr>
<tr>
<td>SYSPRINT</td>
<td>Must not be pre-allocated</td>
<td>Temporary listings from called programs.</td>
</tr>
<tr>
<td>SYSPUNCH</td>
<td>DSN=&amp;OBJ,UNIT=SYSDA,SPACE=(80,(250,50))</td>
<td>Object module passed from assembler to AMDSABLD.</td>
</tr>
<tr>
<td>SYSTERM</td>
<td>None</td>
<td>Assembly messages.</td>
</tr>
<tr>
<td>SYSUT1</td>
<td>UNIT=SYSDA,SPACE=(1700,(400,50))</td>
<td>Work file for assembler.</td>
</tr>
<tr>
<td>TRK0TEXT</td>
<td>Must be preallocated.</td>
<td>Cylinder 0, Track 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPL text from AMDSABLD to ICKDSF (DASD only).</td>
</tr>
<tr>
<td>DSFSYSIN</td>
<td>DSN=&amp;DSFSYSIN, DISP=(,PASS), SPACE=(80,(4,1)), UNIT=SYSDA</td>
<td>SYSIN input for ICKDSF.</td>
</tr>
</tbody>
</table>

**Note:**
1. You **must** specify the GENPARMS DDNAME on the job step.
2. You **cannot** specify the SYSPRINT and SYSIN DD statements in the job step.
3. In GENPARMS, you specify values for UNIT= and VOLSER= on the AMDSADMP macro statement.
4. You **must** specify SYSLIB TRK0TEXT and DFSSYSIN statements.
5. The JCL shown in Figure 38 on page 98 generates a stand-alone dump from DASD 222 using a volume serial of SADMPM. The output is directed to the data set SYS1.SADMP on a DASD 450. Stand-alone dump determines at run-time if that device is usable. If the dump data set on device 450 is not usable, the operator will
Stand-Alone dump

be prompted for another data set. The operator can press enter on any of the
consoles at address 041, 042, 0A0, 3E0, or 3E1. The dump will include the default
storage ranges in those address spaces that are physically-swapped in at the time
of the dump. In addition, all storage in ASID 1 and the JES2 address spaces will be
dumped. Stand-alone dump will also dump the data spaces created by the
DUMPSRV address space.

//OSG EXEC PGM=AMDSAOSG,REGION=5M
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR
// DD DSN=SYS1.MODGEN,DISP=SHR
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(,PASS),
// SPACE=(4096,(2,1)),UNIT=SYSDA
//DFSYIN DD DSN=DSFSYSIN,DISP=(,PASS),
// SPACE=(80,(4,1)),UNIT=SYSDA
//GENPARMS DD *
AMDSADMP CONSOLE=((041,3277),(042,3277),(0A0,3277), X
(3E0,3277),(3E1,3277)), X
DUMP='SP(ALL) IN ASID(1,'JES2') ALSO DATASPACE OF ASID('DUMPSRV')', X
IPL=D222, X
MINASID=PHYSIN, X
OUTPUT=D450, X
REUSED=NEVER, X
PROMP, X
VOLSER=SADMPM
END
/*
//PUTIPL EXEC PGM=ICKDSF,REGION=4M
//IPLDEV DD DISP=OLD,UNIT=SYSDA,
// // VOL=(PRIVATE,RETAIN,SER=SADMPM)
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(OLD,DELETE)
//SYSIN DD DSN=DSFSYSIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=* 
 Figure 38. Example: One-stage generation of stand-alone dump to a DASD

The JCL shown in Figure 39 on page 99 generates a stand-alone dump from tape
333 using a volume serial of TSADMP. The output is directed to the unlabeled
volume on tape 550. Stand-alone dump determines at runtime if that device is
usable. If the dump data set on device 550 is not usable, the operator is prompted
for another data set. The operator can press enter on any of the consoles at address
051,052, 0A0, 3E0, or 3E1. The dump includes the default storage ranges in all
address spaces at the time of the dump. In addition, the data spaces of master,
XCF and OMVS address spaces are also included in the stand-alone dump.
The output from AMDSAOSG contains a listing for the stand-alone dump common communication table (CCT) and device and dump options (DDO) control blocks that contain information specified at generation time. The remainder of the output consists of messages, including message AMD064I, from both stand-alone dump and, when the residence volume is direct access, the device utility ICKDSF.

Table 16 lists the codes AMDSAOSG returns in message AMD064I.

Table 16. AMDSAOSG return codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Residence volume initialized</td>
</tr>
<tr>
<td>4</td>
<td>Residence volume not initialized due to an error, or a warning was issued during AMDSADMP assembly</td>
</tr>
<tr>
<td>8</td>
<td>Residence volume not initialized; GENPRINT could not be opened</td>
</tr>
</tbody>
</table>

See z/OS MVS System Messages, Vol 1 (ABA-AOM) for more information about AMD064I.

Considerations when using one-stage generation

When generating the stand-alone dump program using one-stage generation, do the following:

- Ensure that the SYSLIB DDNAME concatenates SYS1.MODGEN to SYS1.MACLIB. Your installation should catalog the SYS1.MODGEN data set before generating the stand-alone dump program. Otherwise, the JCL that stand-alone dump produces will fail to create the stand-alone dump program.
- If you are generating stand-alone dump for residence on a direct access volume, AMDSAOSG creates and loads a SYS1.PAGEDUMP.Vvolser data set containing the stand-alone dump program and places an IPL text on the volume. If the volume already contains a SYS1.PAGEDUMP.Vvolser data set, AMDSAOSG will fail. While AMDSAOSG is running, the mount attribute of the volume must be PRIVATE.
- When generating the stand-alone dump program from a Magnetic Tape Subsystem, be aware of which tape format you use or you might not be able to IPL the program. Specifically, IPL processing will end abnormally if you:
  - Generate stand-alone dump on a 3490E Magnetic Tape Subsystem and use a tape subsystem other than a 3490E for IPL.
  - Generate stand-alone dump on a tape subsystem other than a 3490E and use a 3490E Magnetic Tape Subsystem for the IPL.

Figure 39. Example: One-stage generation of stand-alone dump to tape

The output from AMDSAOSG contains a listing for the stand-alone dump common communication table (CCT) and device and dump options (DDO) control blocks that contain information specified at generation time. The remainder of the output consists of messages, including message AMD064I, from both stand-alone dump and, when the residence volume is direct access, the device utility ICKDSF. Table 16 lists the codes AMDSAOSG returns in message AMD064I.
Using one-stage generation of stand-alone dump when migrating

When migrating to a new version of MVS, generate a new version of the stand-alone dump program. Use the new MVS system data sets to build the new version of the stand-alone dump program. Always use a stand-alone dump version that is generated from the same release of MVS that you want to dump. IBM does not guarantee that a different level of stand-alone dump can successfully dump anything other than the level of MVS it was designed for. The new version of MVS might have changed making the stand-alone dump program unable to locate vital information it needs to operate.

To generate a new version of the stand-alone dump program, follow the same steps you followed for a normal one-stage generation, then add the following steps:

- Specifying the correct SYSLIB data set to ensure the new version of the AMDSADMP macro is in use.
- Beginning with z/OS V1R12, use the NUCLIB parameters on the AMDSADMP macro invocation to create the correct one-stage JCL.

Use the following JCL examples for DASD when migrating to a new level of MVS:

- **One-stage generation JCL (beginning with z/OS V1R12) for a DASD, see Figure 40 on page 101**
- **One-stage generation JCL (any release prior to z/OS V1R12) for a DASD, see Figure 41 on page 102**

The JCL shown in Figure 40 on page 101 assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. Because the NUCLIB is specified, one-stage JCL uses the SYS1.NUCLEUS system data sets found on the 3390 DASD with volser=NEWSYS. The stand-alone dump is saved on DASD device 560 in the SYS1.SADMP data set.
Figure 41 on page 102 shows JCL that assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. It uses SYS1.NUCLEUS system data set found on 3390 DASD with volser=NEWSYS as suggested by IPLTEXT, IPITEXT, DVITEXT, DPLTEXT and PGTEXT. The stand-alone dump is saved on DASD device 560 in the SYS1.SADMP data set.
Stand-Alone dump

//ASSEMAD JOB MSGLEVEL=(1,1)
//OSG EXEC PGM=AMDSAO,REGION=5M
//STEPLIB DD DSN=SYS1.LINKLIB,DISP=SHR,UNIT=3390,
// VOL=SER=NEWSYS
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR,UNIT=3390,
// VOL=SER=NEWSYS
// DD DSN=SYS1.MODGEN,DISP=SHR,UNIT=3390,
// VOL=SER=NEWSYS
//IPLTEXT DD DSN=SYS1.NUCLEUS(AMDSAIPO),DISP=SHR,
// VOL=SER=NEWSYS,UNIT=3390
//IPITEXT DD DSN=SYS1.NUCLEUS(AMDSAIPI),DISP=SHR,
// VOL=SER=NEWSYS,UNIT=3390
//DVITEXT DD DSN=SYS1.NUCLEUS(AMSDAVI),DISP=SHR,
// VOL=SER=NEWSYS,UNIT=3390
//DPLTEXT DD DSN=SYS1.NUCLEUS(AMSDAPL),DISP=SHR,
// VOL=SER=NEWSYS,UNIT=3390
//PGETEXT DD DSN=SYS1.NUCLEUS(AMDSAPGE),DISP=SHR,
// VOL=SER=NEWSYS,UNIT=3390
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(,PASS),
// SPACE=(4096,(2,1)),UNIT=3390
//DSFSYSIN DD DSN=DSF5YSIN,DISP=(,PASS),
// SPACE=(80,(4,1)),UNIT=3390
//GENPRINT DD SYSOUT=*  
//GENPARMS DD *
AMDSADMP IPL=DSYSDA,VOLSER=SADASD, X
  DUMP=('DASHPACES OF ASID('XCFAS',
  'CTX','APPC')'), X
  MINASID=ALL,PROMPT,MSG=ALL X
  CONSOLE=((020,3277),(030,3277),
  (040,3277),(050,3277)), X
  OUTPUT=O560 X
END  
/*
//PUTIPL EXEC PGM=ICKDSF,REGION=4M
//PLDEV DD DISP=OLD,UNIT=SYSDA, 
// VOL=(PRIVATE,RETAIN,SER=SADASD)
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(OLD,DELETE)
//SYSLIB DD DSN=ADSF5YSIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=*  

Figure 41. Example: One-stage generation JCL (any release) for a DASD (any release)

Use the following JCL examples for tape when migrating to a new level of MVS:

- One-stage JCL (beginning with z/OS V1R12) for tape  (Figure 42 on page 103)
- One-stage JCL (any release prior to z/OS V1R12) for tape (Figure 43 on page 103)

The JCL shown in Figure 42 on page 103 assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. Because NUCLIB is specified, it uses the SYS1.NUCLEUS system data set found on 3390 DASD with volser=NEWSYS. Only one job step is necessary because the stand-alone dump program is saved on a tape 5B0 with a volume serial of SADMPT. The output gets directed to the data set SYS1.SADMP.SAMPLE on DASD 450.
Figure 43 shows JCL that assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. It uses the SYS1.NUCLEUS system data set found on 3390 DASD with volser=NEWSYS. Because the stand alone dump program is being saved on a tape, only one job step is necessary. The stand alone dump program is saved on TAPE 5B0 with volume serial of SADMPT. The output goes to the data set SYS1.SADMP.SAMPLE on DASD 450.

Two-stage generation

In two-stage generation of the stand-alone dump program, you must perform two tasks:

1. Assemble the AMDSADMP macro
2. Initialize the residence volume

After you code the AMDSADMP macro, you can assemble the macro. Use the JCL shown in Figure 44 on page 104 to assemble the AMDSADMP macro. The SYSLIB data set must contain the AMDSADMP macro.
The output of the assembly is a job stream that can be used to initialize the residence volume. The output of the assembly can be directed to a DASD or tape device by coding the SYSPUNCH DD card, as shown in Table 17.

Table 17. Directing the output of assembly

<table>
<thead>
<tr>
<th>Direct assembly output</th>
<th>SYSPUNCH DD statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tape</td>
<td>//SYSPUNCH DD UNIT=tape,LABEL=(,NL),DISP=(NEW,KEEP), VOL=SER=volser</td>
</tr>
<tr>
<td>New direct access data set</td>
<td>//SYSPUNCH DD UNIT=dasd,SPACE=(80,(30,10)),DSN=dsname, DISP=(NEW,KEEP),VOL=SER=volser</td>
</tr>
</tbody>
</table>

Assembling multiple versions of AMDSADMP

You can assemble multiple versions of AMDSADMP at the same time, provided that each version specifies a different residence volume. Differentiate between versions by coding a unique symbol at the beginning of each macro. AMDSADMP uses the symbol you indicate to create unique stage-two job names. The output from a multiple assembly is a single listing and a single object deck, which can be broken into separate jobs if desired. Use the JCL shown in Figure 45 for coding multiple versions of AMDSADMP.

```
//ASMSAD JOB MSGLEVEL=(1,1)  
//ASM EXEC PGM=ASMA90,PARM='DECK'  
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR  
//SYSUT1 DD DSN=SYS1.MODGEN,DISP=SHR  
//SYSUT2 DD UNIT=SYSDA,SPACE=(1700,(400,50))  
//SYSPRINT DD SYSPRINT=*,,STD,HOLD=YES  
//SYSPUNCH DD DSN=DS420.STAGE3.JCL(SADMPST2),DISP=SHR  
//SYSLIB DD SYSIN=*  
//SYSIN DD *  
AMDSADMP MINASID=ALL,IPL=SYSDA, DUMP=('DATASPAACES OF ASID('XCFAS','CTTX','APPC')'), VOLSER=XXXXXX, CONSOLE=((020,3277),(030,3277),(040,3277),(050,3277)), PROMPT,MSG=ALL, OUTPUT=T560  
END
```

Figure 44. Example: Stage-two JCL to assemble the AMDSADMP macro

Initializing the residence volume

When you are generating stand-alone dump for residence on a direct access volume using the stage-two JCL, a SYS1.PAGEDUMP.Vvolser data set containing
Stand-Alone dump

the stand-alone dump program is created, loaded, and IPL text is placed on the volume. If the volume already contains a SYS1.PAGEDUMP.Vvolser data set, the stage-two job fails. While the stage-two job is running, the mount attribute of the volume must be PRIVATE.

Physical output from the assembly part of the initialization step is a listing for the stand-alone dump common communication table (CCT) and devices and dump options (DDO) control blocks that contain information specified at generation time. The remainder of the output consists of informational, error, and action messages from both stand-alone dump and, when the residence volume is direct access, the device utility ICKDSF.

When generating the stand-alone dump program from a Magnetic Tape Subsystem, be aware of which tape format you use or you might not be able to IPL the program. Specifically, IPL processing will end abnormally if you:
- Generate stand-alone dump on a 3490E Magnetic Tape Subsystem and use a tape subsystem other than a 3490E for the IPL.
- Generate stand-alone dump on a tape subsystem other than a 3490E and use a 3490E Magnetic Tape Subsystem for the IPL.

Using two-stage generation of stand-alone dump when migrating
When migrating to a new version of MVS, generate a new version of the stand-alone dump program. Use the new MVS system data sets to build the new version of the stand-alone dump program.

Always use a stand-alone dump version that is generated from the same release of MVS that you want to dump. IBM does not guarantee that a different level of stand-alone dump will successfully dump anything other than the level of MVS it was designed for. The new version of MVS might have changed making the stand-alone dump program unable to locate vital information it needs to operate.

To generate a new version of the stand-alone dump program, follow the same steps you followed for a normal two-stage generation, then add the following steps:
- Ensure that the new version of the AMDSADMP macro is being used by specifying the correct SYSLIB data set.
- Use the NUCLIB, MODLIB, LNKLIB and/or ALIB parameters on the AMDSADMP macro invocation to create the correct stage-two JCL.

The output shown in Figure 46 on page 106 assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. Because the ALIB parameter is specified, the stage-two JCL will use the SYS1.NUCLEUS, SYS1.MODGEN, and SYS1.LINKLIB system data sets, also found on the 3390 DASD with volser=NEWSYS.
Stand-Alone dump

```assembly
//ASMSAD JOB MSGLEVEL=(1,1)
//ASM EXEC PGM=ASMA90,REGION=4096K,PARM='DECK'
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR,
   // UNIT=3390, VOL=SER=NEWSYS
//SYSUT1 DD UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSPRINT DD SYSPUNCH DD DSN=SYS10.SYS430.STAGE3.JCL(SADMPST2),DISP=SHR
//SYSLIN DD SYSPUNCH DD SYSPRINT DD SYSOUT=H
//SYSIN DD *
   * AMDSADMP MINASID=ALL,IPL=DSYSDA,
      DUMP=('DATASPACE OF ASID('XCFAS','CTTX','APPC')'),
      VOLSER=SADUMP,
      CONSOLE=((020,3277),(030,3277),(040,3277),(050,3277)),
      PROMPT,MSG=ALL,
      OUTPUT=T560,
      ALIB=(NEWSYS,3390)
END
/*

Figure 46. Example: Stage-two JCL to assemble the AMDSADMP macro

Note: Using the ALIB parameter is convenient if all of the system data sets used by the stand-alone dump program reside on the same volume. Also, note that the same results could have been achieved by coding the NUCLIB, MODLIB, and LNKLIB keywords separately with each specifying NEWSYS and 3390 for volser and unit.

Figure 47 shows the output that assembles the version of the AMDSADMP macro contained in the SYSLIB data set, SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. Because the MODLIB parameter is specified, the stage-two JCL will use the SYS1.MODGEN system data set found on a 3380 DASD with volser=SYS51A. Because the ALIB parameter is specified, the stage-two JCL will use the SYS1.NUCLEUS and SYS1.LINKLIB system data sets found on a 3390 DASD with volser=SYS51B.

```assembly
//ASMSAD JOB MSGLEVEL=(1,1)
//ASM EXEC PGM=ASMA90,REGION=4096K,PARM='DECK'
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR,
   // UNIT=3390, VOL=SER=NEWSYS
//SYSUT1 DD UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSPRINT DD SYSPUNCH DD DSN=SYS10.SYS430.STAGE3.JCL(SADMPST2),DISP=SHR
//SYSLIN DD SYSPUNCH DD SYSPRINT DD SYSOUT=H
//SYSIN DD *
   * AMDSADMP MINASID=ALL,IPL=DSYSDA,
      DUMP=('DATASPACE OF ASID('XCFAS','CTTX','APPC')'),
      VOLSER=SADUMP,
      CONSOLE=((020,3277),(030,3277),(040,3277),(050,3277)),
      PROMPT,MSG=ALL,
      OUTPUT=T560,
      MODLIB=(SYS51A,3380),
      ALIB=(SYS51B,3390)
END
/*

Figure 47. Example: Stage-two JCL to assemble the AMDSADMP macro

Note that the ALIB parameter has no effect on the SYS1.MODGEN system data set because the MODLIB parameter was specified separately. The stand-alone dump program will be generated using the cataloged system data sets if the NUCLIB, MODLIB, LNKLIB, or ALIB parameters are not specified.
Using two-stage generation for overriding

When overriding to use a different system database, IBM recommends that you generate a new version of the stand-alone dump program. Use the new MVS system data sets to build the new version of the stand-alone dump program.

Although the current version of the stand-alone dump program might be able to dump a new version of MVS successfully, it is not guaranteed. MVS might have changed such that the stand-alone dump program would not be able to locate vital information it needs to operate.

To generate a new version of the stand-alone dump program, follow the same steps you followed for a normal two-stage generation, then add the following steps:

- Ensure that the new version of the AMDSADMP macro is being used by specifying the correct SYSLIB data set.
- Use the NUCLIB, MODLIB, LNKLIB and/or ALIB parameters on the AMDSADMP macro invocation to create the correct stage-two JCL.

The output shown in Figure 48 assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=OVRIDE. Because the ALIB parameter is specified, the stage-two JCL will use the SYS1.NUCLEUS, SYS1.MODGEN, and SYS1.LINKLIB system data sets, also found on the 3390 DASD with volser=OVRIDE.

```hll
// EXEC ASMAC,PARM.C='DECK,NOOBJECT'
//C.SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR,UNIT=3390,VOL=SER=OVRIDE
//C.SYSPUNCH DD DSN=SMITH.TEST.CNTL(SADMP#2),DISP=OLD
//C.SYSIN DD *
AMDSADMP IPL=D3390, VOLSER=SADIPL,
OUTPUT=(D330,SYS1,SADMP), DS=(SYS1,NEWSYS,3390), UNIT=3390, VOL=SER=OVRIDE
REUSEDS=CHOICE, PROMPT FOR DATASET REUSE
DDSPROMPT=NO, OVERRIDE BUILD DATASETS
ALIB=(OVRIDE,3390)
*
END
```

Figure 48. Example: Stage-Two JCL to assemble the AMDSADMP macro with overrides

**Note:** Using the ALIB parameter is convenient if all of the system data sets used by the stand-alone dump program reside on the same volume. Also, note that the same results could have been achieved by coding the NUCLIB, MODLIB, and LNKLIB keywords separately with each specifying NEWSYS and 3390 for volser and unit.

Running the stand-alone dump program

The operator usually takes a stand-alone dump for one of the following types of problems:

- Disabled wait
- Enabled wait
Stand-Alone dump

- Loop
- Partial system hang

When one of these problems occurs, the stand-alone dump program, residing in the SYS1.PAGEDUMP.Vvolser data set, can be run to produce a stand-alone dump. There are several procedures that can be used to run the stand-alone dump program:

- "Procedure A: Initialize and run stand-alone dump"
- "Procedure B: Restart stand-alone dump" on page 112
- "Procedure C: ReIPL stand-alone dump" on page 113
- "Procedure D: Dump the stand-alone dump program” on page 113

When to use which procedure:
- Use Procedure A to initialize the stand-alone dump program and dump storage.
- If you want to run stand-alone dump again, for instance when stand-alone dump fails, use Procedure B, Procedure C, or Procedure D.
- When you want to restart stand-alone dump, try procedure B before you try Procedure C or D.
- Procedures C and D can result in the loss of some central storage from the output, whereas Procedure B usually does not.

Although the stand-alone dump program was created under the operating system, it runs as a stand-alone operation.

Procedure A: Initialize and run stand-alone dump

Use the following procedure to initialize and run a stand-alone dump.

1. Ready the residence device. If it is a tape, mount the volume on a device attached to the selected processor and ensure that the tape cartridge is write-enabled. If it is a DASD volume, ensure that it is write-enabled.
   
   If you mirror the SADMP IPL volume to a device in an alternate subchannel set and have swapped so that the devices in the alternate subchannel set are now in use, prefix the SADMP IPL device number with the subchannel set id when you specify the SADMP IPL device. In this case, DASD SADMP output devices should be defined in the alternate subchannel set as well.

2. If dumping a failed stand-alone dump program, in order to diagnose the stand-alone dump failure, select the Store Status option during the IPL, or perform a manual Store Status. Otherwise, for all other cases, do not perform a Store Status because the machine automatically performs a Store Status when it is necessary.


   Stand-alone dump does not communicate with the operator console. Instead, stand-alone dump loads an enabled wait PSW with wait reason code X'3E0000'. The IPLing of the stand-alone dump program causes absolute storage (X'0'-X'18' and storage beginning at X'FC0') to be overlaid with CCWs. You should be aware of this and not consider it as a low storage overlay.

   **Note**: Stand-alone dump uses the PSW to communicate with the operator or system programmer.

   Stand-alone dump waits for a console I/O interrupt or an external interrupt.

4. When stand-alone dump is IPLed, you can specify a load parm that alters the operation of stand-alone dump. The format of the load parm is Saddddo.
The constant S must be specified as the first character or the load parm will be ignored.

The a specification allows stand-alone dump to start using a console without the operator performing any action on it. It also allows stand-alone dump to bypass the prompts for which output device and default dump title to use. You can specify the following values for a:

- **N** No console communication requested. Use default dump device and title. Execution begins with no console messages. No prompting to the operator is allowed. If a prompt occurs, a wait state will be loaded.

- **O** Use the default console with the default dump device and title. No prompting to the operator is allowed. If a prompt occurs, a wait state will be loaded.

- **M** Use the default console with the default dump device and title. Additional prompts can be made to the operator if they are needed.

- **C** Use the default console. The operator must respond to all prompts.

- **P** Wait for an interrupt from the console device that is to be used. If you do not supply the load parm, this is the default.

The dddd specification is the default console device. It must be one of the devices specified as a console device on the AMDSADMP macro when the stand-alone dump was generated, or the constant SYSC for the hardware system console. If you do not specify a default console device, then the stand-alone dump will use the first console defined on the AMDSADMP macro when the stand-alone dump was generated.

The AMDSADMP macro allows you to specify SYSC as the first console in the console list. If you do this without specifying a console device in the load parm, the hardware system console will be the default console device.

The o field contains flags, and the second bit (bit 1) indicates that SADMP must start an IPL of MVS at the conclusion of its processing. If bit 1 is on, and SADMP locates an AutoIPL policy within MVS storage that specifies a re-IPL of MVS, SADMP uses the information to initiate an IPL of MVS. For details about AutoIPL, see topic about [Using the automatic IPL](https://www.ibm.com/docs/en/zos?topic=using-automatic-ipl) function in z/OS MVS Planning: Operations.

The valid values for the o field are ‘0’, ‘4’ or blank. ‘0’ or blank leaves all bits off. ‘4’ sets bit 1 on. Bit 1 is intended to automate the re-IPL of MVS when SADMP is initiated manually. IBM recommends that it be left off otherwise.

If you do not use the load parm, select the system console or an operator console with a device address that is in the console list that you specified at stand-alone dump generation time (in the CONSOLE keyword of AMDSADMP). At stand-alone dump run time, the operator can choose either a console specified with the CONSOLE= keyword or the system console to control stand-alone dump operation. If an operator console is chosen, press ATTENTION or ENTER on that console. (On some consoles, you might have to press RESET first.) This causes an interruption that informs stand-alone dump of the console’s address. Message AMD001A appears on the console.
Stand-Alone dump

a. Ready an output device. When you dump to devices that have both real and virtual addresses (for example, dumping a VM system), specify the virtual address to the stand-alone dump program. If you are dumping to tape, ensure that the tape cartridge is write-enabled and unlabeled. If you are dumping to DASD, ensure that the DASD data set has been initialized using the IPCS SADMP or AMDSADDI REXX dump data set utilities.

b. Reply with the device number for the output device. Note: A DASD output device must be in the same subchannel set as the device from which the stand-alone dump is IPLed, and only the specification of a 4-digit device number is allowed. So, for example, if in reply to the device number prompt, you enter 4180 and the stand-alone dump has been IPLed from device 15660, device 4180 is assumed to be in subchannel set 1, or in this example, 14180. If you are dumping to a DASD device and DDSPROMPT=YES was specified on the AMDSADMP macro, message AMD002A is issued to prompt the operator for a dump data set. If DDSPROMPT=NO was specified, message AMD002A is not issued and the stand-alone dump program assumes that the dump data set name is SYS1.SADMP.

Note:
1) Pressing ENTER in response to message AMD001A will cause the stand-alone dump program to use the default device specified on the OUTPUT= keyword of the AMDSADMP macro. If the default device is a DASD device, then pressing the ENTER key in response to message AMD001A will cause the stand-alone dump program to use both the default device and the dump data set specified on the OUTPUT= keyword of the AMDSADMP macro. If no dump data set name was provided on the OUTPUT= keyword and the DDSPROMPT=YES keyword was specified, message AMD002A is issued to prompt the operator for a dump data set. If DDSPROMPT=NO was specified, then the stand-alone dump program assumes that the dump data set name is SYS1.SADMP.

2) If you reply with the device number of an attached device that is not of the required device type, or if the device causes certain types of I/O errors, stand-alone dump might load a disabled wait PSW. When this occurs, use procedure B to restart stand-alone dump.

c. Stand-alone dump prompts you, with message AMD001A, for a dump title.

In Figure 49 on page 111 the dump is initialized using a load parm with no console prompts.
5. When no console is available, run stand-alone dump without a console.
   a. Ready the default output device that was specified on the OUTPUT parameter on the AMDSADMP macro. For tapes, ensure that the tape cartridge is write-enabled. For DASD, ensure that the dump data set has been initialized using the AMDSADDD REXX or IPCS SADMP dump data set utilities.
   b. Enter an external interruption on the processor that stand-alone dump was IPLed from. Stand-alone dump proceeds using the default output device and/or the default dump data set. No messages appear on any consoles; stand-alone dump uses PSW wait reason codes to communicate to the operator.

6. Stand-Alone dump first processes the real storage in ASID order. The message AMD005I is issued after each phase to display the status of the dump.
   a. Phase 1 dumps the Page Frame Table and its related structures in virtual order.
   b. The next three phases dump real storage associated with the minimal, summary and swapped-in ASIDs in virtual order.
   c. Phase 5 dumps the In-Use real storage in real order.

7. Stand-Alone dump processes the paged-out storage in virtual order based on customer specifications. Message AMD108I is issued to display the status of the virtual phase of the dump.
   a. Phases 6 to 8 dumps the paged-out storage of minimal, summary and swapped-in ASIDs. At end of phase VIII, all storage associated with the swapped-in ASIDs has been dumped.
   b. Phase 9 dumps the storage of swapped-out ASIDs.

8. Stand-Alone dump proceeds to dump the available real storage in Phase 10. The storage dumped during this phase includes the real frames that were not dumped earlier. At the completion of this phase, message AMD104I is issued to signal the end of the dump.

9. When stand-alone dump begins dumping real storage (Phase 1 to Phase 5 and Phase 10) it issues message AMD005I. Message AMD095I is issued every 30 seconds to illustrate the process of the dump. Message AMD005I will be
Stand-Alone dump

issued as specific portions of real storage have been dumped, as well as upon completion of the real dump. Stand-alone dump can end at this step.

10. When stand-alone dump is dumping virtual storage, it issues message AMD108I as specific portions of virtual storage is dumped. Message AMD056I is issued to signal the end of virtual phase dump.

11. If you specified PROMPT on the AMDSADMP macro, stand-alone dump prompts you for additional storage that you want dumped by issuing message AMD059D.

12. Stand-alone dump dumps paged-out virtual storage, the stand-alone dump message log, and issues message AMD095I every 30 seconds to illustrate the progress of the dump.

13. When stand-alone dump completes processing, stand-alone dump unloads the tape, if there is one, and enters a wait reason code X'410000'.

See z/OS MVS System Codes for more information about the wait state reason codes loaded into the PSW.

Note: Some processor models do not allow selection of a specific processor to IPL from. Normally, the processor previously IPL'ed is selected again for this IPL.

Procedure B: Restart stand-alone dump

A system restart does not always work, either because it occurs at a point when stand-alone dump internal resources are not serialized, or because stand-alone dump has been too heavily damaged to function. If the restart does not work, try procedure C (reIPL).

If a dump to a DASD data set is truncated because there is not enough space on the data set to hold the dump, use a system restart to dump the original data to tape. By causing a system restart, you can reinitialize and restart a failing stand-alone dump program without losing the original data you wanted to dump.

If a permanent error occurs on the output device, the stand-alone dump program will prompt the operator to determine if a restart of the stand-alone dump program should be performed. If the operator indicates that a restart of the stand-alone dump program should be performed, then the stand-alone dump program restarts the dump using the same console and prompts the operator to specify a different output device. Continue procedure A at step 6A; see "Procedure A: Initialize and run stand-alone dump" on page 108.

For other types of stand-alone dump errors and wait states, it might be necessary for the operator to perform a manual restart of the stand-alone dump program. In this case, the operator should perform the following steps:

1. Perform a system restart on the processor that you IPLed stand-alone dump from.

2. If the restart is successful, stand-alone dump dumps central storage. If stand-alone dump abnormally ends while dumping central storage, try to restart stand-alone dump. If the restart succeeds, stand-alone dump reruns the entire dump. It will first enter wait state X'3E0000' to allow you to specify a new console and output device. You can do this to recover from an I/O error on the output device. Stand-alone dump recognizes any console in the console list and starts with the same output device defaults that are used at the IPL of stand-alone dump.

Procedure C: ReIPL stand-alone dump

When you reIPL stand-alone dump, the previous running of stand-alone dump has already overlaid some parts of central storage and modified the page frame table. If the virtual storage dump program was in control, a reIPL might not dump paged-out virtual storage. The number of times that you can IPL stand-alone dump to dump paged-out virtual storage is equal to the number of processors present.

To run procedure C, repeat procedure A. If the previous IPL of stand-alone dump did not load a wait state and reason code of X'250000' or higher and the reIPL succeeds, stand-alone dump usually completes processing as in procedure A. Some storage locations might not reflect the original contents of central storage because, during a previous IPL, stand-alone dump overlaid the contents. These locations include the absolute PSA and possibly other PSAs.

Procedure D: Dump the stand-alone dump program

Use a new IPL of stand-alone dump to debug stand-alone dump if stand-alone dump fails. When you use stand-alone dump to dump itself, the dump program dumps central storage only, because a dump of central storage provides enough information to diagnose a stand-alone dump error. Follow procedure A at step 2 on page 108 by performing a STORE STATUS instruction. Stand-alone dump follows procedure A steps 2 through 6, then issues message AMD088D. This message allows the operator to stop the dump after central storage has been dumped or to continue dumping virtual storage.

Stand-alone self-dump

When running a virtual storage dump and stand-alone dump error recovery detects errors in stand-alone dump, stand-alone dump can take a self-dump before proceeding. At most, stand-alone dump takes twelve self-dumps; after the twelfth request for a self-dump, stand-alone dump stops taking self-dumps, but continues to count the number of self-dump requests and continues to issue the AMD066I message. After a large number of self-dump requests, stand-alone dump terminates. Stand-alone dump places both the self-dump and the operating system dump onto the output tape or DASD.

You can use the LIST subcommand of IPCS to print stand-alone dump self-dumps. The format of the subcommand is as follows, where x = 001 - 012. See z/OS MVS IPCS Commands for more information.

```
LIST address COMPDATA(AMDSAxxx)
```

Running the stand-alone dump program in a sysplex

The operator usually takes a stand-alone dump in a sysplex when an MVS system is not responding. Situations that indicate that stand-alone dump should be run include:

- Consoles do not respond
- MVS is in a WAIT state
- An MVS system is in a “status update missing” condition and has been or is waiting to be removed from the sysplex
- A stand-alone dump has been requested by Level 2.

There are two high-level methods for taking a stand-alone dump of an MVS system that resides in a sysplex. Both methods emphasize the expeditious removal
of the failing MVS system from the sysplex. If the failed MVS system is not
partitioned out of the sysplex promptly, some processing on the surviving MVS
systems might be delayed.

**Method A**

Use this method to take a stand-alone dump of an MVS system that resides in a
sysplex. Assume that the MVS system to be dumped is “SYSA”.

1. IPL the stand-alone dump program on SYSA (see “Running the stand-alone
dump Program”).
2. Issue VARY XCF,SYSA,OFFLINE from another active MVS system in the
   sysplex if message IXC402D or IXC102A is not already present.
   You do not have to wait for the stand-alone dump to complete before issuing
   the VARY XCF,SYSA,OFFLINE command.
3. Reply DOWN to message IXC402D or IXC102A.

Performing steps 2 and 3 immediately after IPLing stand-alone dump will expedite
sysplex recovery actions for SYSA and allow resources held by SYSA to be cleaned
up quickly, thus enabling other systems in the sysplex to continue processing.

After you IPL the stand-alone dump, MVS cannot automatically ISOLATE system
SYSA through SFM. Message IXC402D or IXC102A issues after the VARY
XCF,SYSA,OFFLINE command or after the XCF failure detection interval expires.
You must reply DOWN to IXC402D or IXC102A before sysplex partitioning can
complete.

**Note:** DO NOT perform a SYSTEM RESET in response to IXC402D, IXC102A after
the IPL of stand-alone dump. The SYSTEM RESET is not needed in this case
because the IPL of stand-alone dump causes a SYSTEM RESET. After the IPL of
stand-alone dump is complete, it is safe to reply DOWN to IXC402D or IXC102A.

**Method B**

Use this method if there is a time delay between stopping processors, as part of the
SYSTEM RESET-NORMAL function in step one, and IPLing the stand-alone dump
program.

1. Perform the SYSTEM RESET-NORMAL function on SYSA.
2. Issue VARY XCF,SYSA,OFFLINE from another active MVS system in the
   sysplex if message IXC402D or IXC102A is not present.
3. Reply DOWN to message IXC402D or IXC102A. Performing steps two and
   three immediately after doing the SYSTEM RESET will expedite sysplex
   recovery actions for SYSA. It allows resources that are held by SYSA to be
   cleaned up quickly, and enables other systems in the sysplex to continue
   processing.
4. IPL the stand-alone dump program (see “Running the stand-alone dump
   program” on page 107). While this step can be done earlier, the aim of
   performing steps one through three is to minimize disruption to other systems
   in the sysplex.

After a SYSTEM RESET is performed, MVS cannot automatically ISOLATE system
SYSA through SFM. Message IXC402D or IXC102A is issued after the VARY
XCF,SYSA,OFFLINE command or after the XCF failure detection interval expires.
You must reply DOWN to IXC402D or IXC102A before sysplex partitioning can
complete.
Capturing a stand-alone dump quickly

There are times when you need to process stand-alone dump information quickly to diagnose a problem. It is important to perform the stand-alone dump process quickly, to minimize the time the system is unavailable. Sometimes a stand-alone dump is not captured because of the time that the dumping process takes. Skipping the stand-alone dump, however, can prevent the diagnosis of the system failure. Instead of skipping the stand-alone dump, it is better to spend a short time to get as much of the stand-alone dump as is possible, as quickly as possible. The following are two methods to save time when performing a stand-alone dump.

- Minimize the operator actions
- Get a partial stand-alone dump

Minimize the operator actions

Time spent waiting for the operator to reply to a message or mount a tape is idle time. Minimizing the operator actions turns the idle time into data capture time. It also simplifies the process, so that the stand-alone dump process becomes easier to do. The following are ways to minimize the operator's actions when performing a dump.

- Use the stand-alone dump LOAD parameter SO or SM to skip the prompt for the console to use to avoid other responses to messages.
- Use the default device specified on the OUTPUT= keyword of the AMDSADMP macro. If the default device is a DASD device, then pressing the ENTER key in response to message AMD001A will cause the stand-alone dump program to use both the default device and the dump data set specified on the OUTPUT= keyword of the AMDSADMP macro.
- Use REUSEDS=ALWAYS on the AMDSADMP macro to indicate that stand-alone dump should reuse the dump data set on the specified output device when it determines that the data set is valid, however, it can contain data from a previous dump. Or, you can always clear the dataset.

Note: Be sure you do not overwrite another dump.

- Specify DDSPROMPT=NO, then the stand-alone dump program assumes that the dump data set name is SYS1.SADMP.
- Do not specify PROMPT on the AMDSADMP macro, unless requested by IBM.
- Use "fast" device for output

Get a partial stand-alone dump

While it is always best to get a complete stand-alone dump, sometimes time constraints will not allow this. There is no guarantee that it will be possible to diagnose a failure from a partial stand-alone dump; however, if the choice is between no dump at all or a partial dump, then the partial dump is the best choice.

When taking a partial stand-alone dump:

- Let the stand-alone dump run for as long as you can. If you run out of time, you can stop the dump cleanly.
- Stand-alone dump tries to write out the most important information first.
  - Status information (PSW, registers, and so forth) for all CPUs
  - Critical real storage, including common storage and trace information
  - Real storage for address spaces executing at the time of the dump
Stand-Alone dump

- Any remaining real storage
- Paged out storage for swapped in address spaces
- Paged out storage for swapped out address spaces

- Use the EXTERNAL INTERRUPT key to terminate the dumping process. This causes a clean stop, closing the output dataset properly.

In [Figure 50](#), the dump was ended early using the EXTERNAL INTERRUPT key.

---

**Copying, viewing, and printing stand-alone dump output**

When stand-alone dump processing completes the dump, the output resides on a tape volume, a DASD, or a combination of devices. The easiest way to view the dump is to copy the dump to a DASD data set. When a stand-alone dump resides on multiple devices and/or dump data sets, you can concatenate the dump into one data set. After the dump is available on DASD, it can be viewed online using IPCS.

**Note:** If the dump resides in a DASD dump data set, IBM recommends that you copy the dump to another data set for IPCS processing and clear (reinitialize) the dump data set using the AMDSADDD or IPCS SADMP dump data set utilities. For more information, see "Using the AMDSADDD Utility" on page 88 and "Using the IPCS SADMP Utility" option on the IPCS Dialog in [z/OS MVS IPCS User's Guide](https://www.ibm.com/downloads/ ux/zos-mvs-icp).

**Copying the dump to a data set**

If you want to view the dump online, copy the dump to a data set. There are two tools you can use to copy the dump:

- Use the IPCS COPYDUMP subcommand when the IPCS environment has been set up on your system. This is the only option recommended if the dump was written to a multi-volume DASD data set.
- Use the IEBGENER utility when the IPCS environment has not been set up on your system. Many operators take a stand-alone dump so that the system programmer can view the dump. The operator does not require IPCS on the system because the operator will not be viewing the dump. Therefore, the operator should use the IEBGENER utility to copy the dump to a data set accessible by the system programmer's system.

For more information, see the following references:

- See [z/OS MVS IPCS Commands](https://www.ibm.com/downloads/ux/zos-mvs-icp) for information about COPYDUMP.
- See [z/OS DFSMSdfp Utilities](https://www.ibm.com/downloads/ux/zos-mvs-icp) for information about IEBGENER.
Copying from tape
The example below shows how to use IEBGENER to copy tape output to DASD.
Two advantages of copying stand-alone dump tape output to DASD are:

• When stand-alone dump ends prematurely and does not give the stand-alone dump output (SYSUT1) an end-of-file, the SYSUT2 data set does contain an end-of-file (SYSUT2 is the data set to which stand-alone dump output is copied.) This occurs even when SYSUT2 is another tape. IEBGENER might end with an I/O error on SYSUT1; this is normal if SYSUT1 does not contain an end-of-file.

• Making SYSUT2 a direct access data set to use as input to IPCS saves IPCS processing time.

Use the JCL shown in Figure 51 to invoke the IEBGENER utility, which will copy the stand-alone dump output from tape to a DASD data set.

Note: Specifying AVGREC= requires SMS be running, but the data set does not have to be SMS managed.

Copying from DASD
The example below shows how to use IEBGENER to copy DASD output to a DASD data set. After the dump is successfully copied, use the AMDSADDD REXX utility to clear (reinitialize) the dump data set and ready it for another stand-alone dump. For more information, see:

• SADMP option on the IPCS Dialog in z/OS MVS IPCS User’s Guide
• “Using the AMDSADDD utility” on page 88

Use the JCL shown in Figure 52 to invoke IEBGENER, which will copy the stand-alone dump output from a DASD data set to another DASD data set.
Copying from multiple dump data sets

The stand-alone dump program allows a dump to be contained in multiple dump data sets. Therefore, when you want to view a stand-alone dump using IPCS, it is necessary to concatenate all of the dump data sets onto one DASD data set.

Use the JCL in Figure 53 to invoke the IPCS COPYDUMP subcommand to copy stand-alone dump output from three DASD dump data sets to another data set. Note that two of the dump data sets reside on the volume SADMP1, while the third resides on the volume SADMP2.

```
//SADCOPY JOB MSGLEVEL=(1,1)
//COPY EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=A
//C1 DD DSN=SADMP1.DATA1,DISP=SHR,UNIT=DASD,VOL=SER=SADMP1
//C2 DD DSN=SADMP1.DATA2,DISP=SHR,UNIT=DASD,VOL=SER=SADMP1
//C3 DD DSN=SADMP1.SADMP1.DAT,DISP=SHR,UNIT=DASD,VOL=SER=SADMP2
//COPYTO DD DSN=SADUMP.COPY,UNIT=DASD,
// VOL=SER=SADCOPY,DISP=(NEW,CATLG),
// DCB=(RECFM=VBS,LBLREC=4160,DSORG=PS),
// SPACE=4160,(0000,4000),RLSE)
//SYSTSIN DD *
// IPCS NOPARM DEFER
COPYDUMP OUTFILE(COPYTO) NOCONFIRM INFILE(C1, C2, C3)
END
/*
Figure 53. Example: Copying a stand-alone dump from multiple DASD data sets
```

Use the JCL shown in Figure 54 to invoke the IPCS COPYDUMP subcommand to copy stand-alone dump output from two DASD dump data sets and two tape volumes to a DASD data set.

```
//SADCOPY JOB MSGLEVEL=(1,1)
//COPY EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=A
//C1 DD DSN=SYS1.SADMP.MAIN.DATA1,DISP=SHR,UNIT=DASD, VOL=SER=SADMP1,
// C2 DD DSN=SYS1.SADMP.MAIN.DATA1,DISP=SHR,UNIT=DASD, VOL=SER=SADMP1,
// C3 DD DSN=SYS1.SADMP.ALT.DATA1,DISP=SHR,UNIT=TAPE, VOL=SER=SADMP3,
// C4 DD DSN=SYS1.SADMP.ALT.DATA1,DISP=SHR,UNIT=TAPE, VOL=SER=SADMP4
//COPYTO DD DSN=SADUMP.COPY,UNIT=DASD,
// VOL=SER=SADCOPY,DISP=(NEW,CATLG),
// DCB=(RECFM=VBS,LBLREC=4160,DSORG=PS),
// SPACE=(2080,(1600,800),RLSE)
//SYSTSIN DD *
// IPCS NOPARM
COPYDUMP OUTFILE(COPYTO) NOCONFIRM INFILE((C1,C2,C3,C4))
END
/*
Figure 54. Example: Copying stand-alone dump output from DASD and tape
```

Viewing stand-alone dump output

You can view the stand-alone dump output at a terminal using IPCS. Do the following:
1. Start an IPCS session.
2. On the IPCS Primary Option Menu panel, select the SUBMIT option to copy the dump and do initial dump analysis.
3. Return to the IPCS Primary Option Menu panel. Select the DEFAULTS option.
4. IPCS displays the IPCS Default Values panel. Enter the name of the data set containing the dump on the Source line.
5. Return to the IPCS Primary Option Menu panel. Select the BROWSE, ANALYZE, or COMMAND option to view the dump.

See 
OS MV5 IPCS Commands
for information about the IPCS subcommands.

Printing stand-alone dump output
You can print an analysis of the stand-alone dump or the entire dump using IPCS.

To print an analysis of the dump in batch mode:
1. Start an IPCS session.
2. On the IPCS Primary Option Menu panel, select the SUBMIT option to copy the dump and do initial dump analysis.
3. On the IPCS Dump Batch Job Option Menu panel, enter the requested information.
4. On the next panel, enter the sysout output class. IPCS writes the dump analysis to the specified output class.
5. The system prints the dump in the printout of the output class.

To print the full dump in batch mode:
1. Use IPCS CLIST BLSCBSAP.

See 
OS MV5 IPCS User’s Guide
for IPCS panels and the CLIST BLSCBSAP.

The example in Figure 55 runs an IPCS CLIST that:
• Copies the stand-alone dump from the tape data set defined in an IEFRDER DD statement to a cataloged, direct access data set named SA1DASD.
• Analyzes and formats the dump.
• Writes the formatted dump output to a data set named IPCSPRNT. A TSO/E CLIST used for IPCS should allocate this print output data set to a sysout print class, as follows:
  ALLOCATE DDNAME(IPCSPRNT) SYSOUT(A)

After the CLIST runs, the dump remains available in the SA1DASD data set for supplementary formatting jobs.

```plaintext
//PRINTJOB JOB MSGLEVEL=1,REGION=800M
//IPCS EXEC IPCS,CLIST=BLSCBSAP,DUMP=SA1DASD
//IEFPROC.IEFRDER DD DSN=SA1,DISP=OLD,UNIT=3490
// VOL=SER=12345,LABEL=(1,NL)
/*

Figure 55. Example: Printing an unformatted stand-alone dump

Message output
There are three types of message output from a stand-alone dump program, as follows:
• MNOTES from the AMDSADMP macro
• Messages on the 3480, 3490, or 3590 display
Stand-Alone dump

- Messages on the system console or the operator console

For more information about messages on the system console or the operator console, use LookAt or see *MVS System Messages*.

**Stand-alone dump messages on the 3480, 3490, or 3590 display**

When stand-alone dump output is sent to a 3480, 3490, or 3590 magnetic tape subsystem, stand-alone dump uses the subsystem's eight-character message display to inform and prompt the operator. The leftmost position on the message display indicates a requested operator action. The eighth position (rightmost) gives additional information.

In the messages listed below, alternating indicates that there are two messages which are flashing on the display, one after the other. A blinking message is one message that is repeated on the display.

The stand-alone dump messages that can appear on the display are:

**Dvolser (alternating)**

**MSADMP#U**

Informs the operator that a labeled tape has been rejected and a new tape must be mounted.

**MSADMP#U (blinking)**

Requests that the operator mount a new tape.

**RSADMP#U (blinking)**

Indicates that the stand-alone dump program has finished writing to the tape.

**RSADMP# (alternating)**

**MSADMP#U**

Informs the operator that an end-of-reel condition has occurred and a new tape must be mounted.

**SADMP# (blinking)**

Indicates that the tape is in use by stand-alone dump.

**SADMP# (alternating)**

**NTRDY**

Informs the operator that some type of intervention is required.

The symbols used in the messages are:

- **#** A variable indicating the actual number of cartridges mounted for stand-alone dump. It is a decimal digit starting at 1 and increasing by 1 after each end-of-cartridge condition. When the # value exceeds 9, it is reset to 0.

- **D** Demount the tape and retain it for further system use, for example as a scratch tape. Stand-alone dump does not write on the tape.

- **M** Mount a new tape.

- **R** Demount the tape and retain it for future stand-alone dump use.

- **U** The new tape should not be file-protected.

- **volser** A variable indicating the volume serial number on the existing tape label.
Analyzing stand-alone dump output

The following sections describe how to analyze the output from a stand-alone dump. A stand-alone dump can indicate the following types of problems:

- Enabled wait state
- Disabled wait state
- Enabled loop
- Disabled loop

Use the information in this section to determine the type of problem the system has encountered. After the problem type is determined, see z/OS V2R2 Problem Management for further information about diagnosing the problem type.

Collecting initial data

When an operator takes a stand-alone dump, it is important to determine the conditions of the system at the time the dump was taken. Because a stand-alone dump can be requested for a various number of problem types, the collection of problem data is imperative to determining the cause of the error.

The objectives for analyzing the output of a stand-alone dump are as follows:

- Gather symptom data
- Determine the state of the system
- Analyze the preceding system activity
- Find the failing module and component

Gathering external symptoms

When a stand-alone dump is taken, you must first question the operator or the person who requested the dump. It is important to understand the external symptoms leading up to the system problem. What was noticed before stopping the system? The answer might give you an idea of where the problem lies.

Here are a few questions you should find an answer to before continuing:

- Was the system put into a wait state?
- Were the consoles hung or locked up?
- Were commands being accepted on the operator console without a reply?
- Was a critical job or address space hung?

Gathering IPCS symptoms

After getting a list of symptoms, use IPCS to collect further symptom data. A primary symptom string is usually not available in a stand-alone dump; however, IPCS can add a secondary symptom string. In Figure 56 on page 122, the explanation of the secondary symptom string indicates an enabled wait state condition.
Determining the system state

There are several control blocks that you can view that describe the state of the system when the stand-alone dump was requested.

**CSD**
- Describes the number of active central processors and whether the alternate CPU recovery (ACR) is active.

**PSA**
- Describes the current environment of a central processor, its work unit, FRR stack, an indication of any locks held.

**LCCA**
- Contains save areas and flags of interrupt handlers.

**CVT**
- Contains pointers to other system control blocks.

Use the IPCS subcommand STATUS WORKSHEET to obtain the data that will help you determine the state of the system. For example, in Figure 57 on page 123 look for the following:

- The CPU bit mask, which indicates how many processors are online.
- The PSW at the time of the dump
- The PSATOLD. If the fields are zero, this indicates that an SRB is running and the address in SMPSW indicates the save area of the dispatcher. If the fields are nonzero, the address in PSWSV indicates the save area of the dispatcher.
- The PSAAOLD, which indicates what address space jobs are running in.
You can also obtain the stored status of each central processor using the IPCS
subcommand STATUS CPU REGISTERS. Watch for these bits in the first half of the
PSW:

- Bits 6 and 7 indicate a disabled (04xxxxxx) or enabled (07xxxxxx) condition
- Bit 14 could indicate a wait (000A0000)
- Bits 16 and 17 indicate primary, secondary, access register (AR) or Home mode

Starting in V2R1, the worksheet displays 4-digit CPUIDs. There can be up to eight
CPUs on one line, if allowed. For example:

<table>
<thead>
<tr>
<th>NAME at time of dump</th>
<th>CPU 0001</th>
<th>CPU 0003</th>
<th>CPU 0005</th>
<th>CPU 0006</th>
<th>CPU 0007</th>
<th>CPU 0008</th>
<th>CPU 0009</th>
<th>CPU 000A</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSW at time of dump</td>
<td>00020000</td>
<td>00020000</td>
<td>04047000</td>
<td>00020000</td>
<td>00020000</td>
<td>00000000</td>
<td>08000000</td>
<td>08000000</td>
</tr>
<tr>
<td>CR0 Interrupt mask</td>
<td>00000000</td>
<td>00000001</td>
<td>00000000</td>
<td>00000001</td>
<td>00000001</td>
<td>00000001</td>
<td>00000001</td>
<td>00000001</td>
</tr>
<tr>
<td>CR6 I/O class mask</td>
<td>00000001</td>
<td>00000002</td>
<td>00000002</td>
<td>00000002</td>
<td>00000002</td>
<td>00000002</td>
<td>00000002</td>
<td>00000002</td>
</tr>
</tbody>
</table>

In Figure 58 on page 124, the PSW indicates an enabled wait state condition. The
program is running in primary mode with 24-bit addressing (bits 16 and 17 are 00
Stand-Alone dump

and the second word begins with 0).

CPU(X'0000') STATUS:
PSW=070E0000 00000000 NO WORK WAIT
ASCB1 at FD3280, JOB(+MASTER+), for the home ASID
ASXB1 at FD34F8 for the home ASID. No block is dispatched
CLTE: 01C60068
+0000 BLD..... 00000000 XDS..... 00000000 XES..... 00000000
+000C XD....... 00FD4900 ESET..... 00FD4908 ULUT..... 00FD4910
CURRENT FRR STACK IS: SVC
PREVIOUS FRR STACK(S): NORMAL

GPR VALUES
  0-3  00000000 00000000 00000000 00000000
  4-7  00000000 00000000 00000000 00000000
  8-11 00000000 00000000 00000000 00000000
 12-15 00000000 00000000 00000000 00000000

ACCESS REGISTER VALUES
  0-3 006FB01F 00000000 00000000 00000000
  4-7 00000000 00000000 00000000 00000000
  8-11 00000000 00000000 00000000 00000000
 12-15 00000000 00000000 806FA03C 00000000

Figure 58. Example: STATUS CPU REGISTERS output

To obtain other fields from important control blocks, use the IPCS subcommand CBFORMAT. See [z/OS MVS IPCS Commands](z/OS MVS IPCS Commands) for information about the CBFORMAT subcommand.

You can also use the WHERE subcommand to identify particular areas in the dump. For example, if a general purpose register contains an address, use the WHERE subcommand to determine in what module that address resides. In Figure 59 the WHERE subcommand indicates that the address is part of the READONLY nucleus.

Figure 59. Example: WHERE subcommand output

Analyzing an enabled wait

An enabled wait is also known as a dummy wait or a no work wait. An indication of an enabled wait is a PSW of `070E0000 00000000` or `07060000 00000000 00000000` and GPRs containing all zeroes. An enabled wait occurred when the dispatcher did not find any work to be dispatched. An enabled wait can occur because of resource contention or system non-dispatchability, among other errors.

Reviewing outstanding I/O requests

When analyzing a stand-alone dump for an enabled wait condition, check the status of the input/output requests. A display of the IOS control block and any active UCBs can help determine what was happening when the system entered the wait state.

In Figure 60 on page 125 the HOTIO field indicates that a solicited interrupt has completed other than DCC-3 because the last time HOT-I/O detection was called. Note also that the IOQF and IOQL fields are identical, indicating that the first and last request for this device is the same.
Analyzing for resource contention

You can obtain information related to resource contention by using the IPCS subcommand ANALYZE. This subcommand displays contention information for I/O, ENQs, suspend locks, allocatable devices and real storage. For example, in Figure 61, 61 units of work are waiting to be processed. The top RB is in a wait state.

Obtaining real storage data

Use the IPCS RSMDATA subcommand to obtain information about storage usage and any unusual condition that have occurred prior to requesting the stand-alone dump. In the RSMDATA output, if the percent usage field is 100%, there are no frames left. Also, the percent of available total fixed frames should not be a high number. If it is, there can be a program using too many resources to complete. For example, in Figure 62, the percent of available total fixed frames is at 25%.
You can also check the ASM control blocks to determine the statistics applicable to I/O requests. The I/O requests received and completed should be the same. In

Figure 63 note that the 509577 I/O requests received have all been completed.

---

**Determining dispatchability**

By performing an address space analysis on the major system address space, you can determine if there is any work waiting and if the address space is dispatchable. The major address space you should analyze are:

- Master scheduler, ASID 1
- CONSOLE
- JES2/JES3
- IMS/CICS/VTAM

When you are analyzing an address space for dispatchability, keep in mind these questions:

- Are there any suspended SRBs on the queue?
  
  You will need to run the WEBS on ASCBSAWQ and look for WEBS that have a WEBFLAG1 field of X'000000' to check if there are any SRBs ready to be dispatched.

- Are there any ready TCBs indicated by ASCBTCBS and ASCBTCBL?
  
  ASCBTCBS and ASCBTCBL contain a count of the number of TCBs containing ready work to be dispatched. To find the TCBs for ASCBTCBL, look at the WEBS on the ASCBLTCS and ASCBLTCB queues that belong to the home space.

- If there is ready work, is the ASCB dispatchable (ASCBDSP1)?
  
  ASCBDSP1 is a non-dispatchability flag. For more information about what the values of ASCBDSP1 indicate, see z/OS MVS Data Areas in [z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/).

- If there is no ready work, are the TCBs in a normal wait (TCBFLGS4, TCBFLGS5, TCBNDSP)?
  
  A non-zero value in any of these fields indicates that the TCB is non-dispatchable.

In Figure 64 on page 127 ASCBDSP1 is X'04', indicating that this address space is not eligible for CML lock requests. The ASCBSAWQ, ASCBLTCN, and ASCBTCBS fields all contain zeroes, indicating that there is no ready work available.
For the mapping structure of WEBs under the IHAWEB, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

If your address space analysis indicated that ready work was available to be dispatched, look at ASCBDSP1 to determine if the address space is dispatchable. If your address space analysis indicated that there was no ready work available to be dispatched, look at the TCBs to determine if they are in a normal wait. In Figure 65 on page 128, for example, the TCB fields indicate that the top RB is in a wait.

---

**Figure 64. Example: SUMMARY FORMAT output (determining ready work)**

For the mapping structure of WEBs under the IHAWEB, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).
Analyzing a disabled wait

A disabled wait condition can be analyzed by checking the PSW at the time of the error. If bits 6 and 7 are zero and bit 14 contains a 1, there is a disabled wait. The wait state code is in byte 7, with the reason code in byte 5.

The following examples show how to determine the wait state code:

- In the following PSW, the wait state code is X'014' and the reason code is zero.
  \[\text{PSW} = 000E0000 \text{ X'014'} \]
- In another example, the wait state code is X'064' and the reason code is X'09'.
  \[\text{PSW} = 000A0000 \text{ X'064'} \text{ X'09'} \]
- In z/Architecture® mode, the PSW would look like:
  \[\text{PSW} = 00220000 \text{ X'064'} \text{ X'09'} \]

After you determine the wait state code from the PSW, look at the documentation for the specific wait state code for any action you can take. See [z/OS MVS System Codes](#) for the specific wait state code you encountered.

If you cannot find the wait state code documented, do one of the following:

- Analyze the dump to determine if it is a stand-alone dump wait state.
- Check PSASMPSW and PSAPSWSV to determine if the dispatcher loaded the wait state PSW because of an overlay. See Chapter 7, “The dump grab bag,” on page 163 for more information about storage overlays.
- Use the stored status registers to determine who loaded the wait state into the PSW.

Analyzing an enabled loop

To determine if the stand-alone dump was requested because of an enabled loop, you need to view the system trace table. Repetitive patterns in the system trace...
table indicate an enabled loop condition. An enabled loop, however, does not normally cause a system outage. It will cause an outage in these circumstances:
- There is a non-preemptable loop in SRB mode
- There is a loop in a high priority address space that is in TCB mode

In Figure 66 the CLKC entries indicate an enabled loop, and because column three is all zeroes, this loop is in SRB mode. The PSW addresses on the CLKCs identify the looping program. Use the WHERE subcommand to locate the responsible program.

Because of interrupt processing that occurs during an enabled loop, the stored status data might not point to the module causing the loop. To determine if a first level interrupt handler (FLIH) was active, view the PSASUPER field of the PSA. If the PSASUPER field is non-zero, a FLIH was active at the time of the error. Using the FLIH's save area, find the PSW and registers at the time of the error. The address in the second half of the PSW will point to the module involved in the loop. See “Problem data saved by first level interrupt handlers” on page 130 for more information.

Analyzing a disabled loop
A disabled loop is not visible in the system trace output because disabled routines do not take interrupts. Normally, a disabled loop results in a spin loop in a multiprocessor environment. When analyzing a stand-alone dump for a disabled loop, use the stored status data to determine the module involved in the loop. Also, examine the in-storage logrec buffer for entries that recovery routines have made but which were not written to the logrec data set because of a system problem. Very often it is these records that are the key to the problem solution. See “Obtaining information from the logrec recording control buffer” on page 567 for more information.

SLIP problem data in the SLIP work area
In a stand-alone dump taken after a SLIP ACTION=WAIT trap matches, problem data can be found in a work area pointed to by the PSAWTCOD field in the prefix save area (PSA). Table 18 shows the format of this area.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Content</th>
</tr>
</thead>
</table>
| 0 (0)  | 1      | RTM/SLIP processing environment indicator:  
  - X'01': RTM1  
  - X'02': RTM2  
  - X'03': MEMTERM  
  - X'04': PER |
Table 18. Work area pointed to by the PSAWTCOD field (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td>2</td>
<td>Logical processor identifier (CPUID)</td>
</tr>
<tr>
<td>3 (3)</td>
<td>1</td>
<td>System mask, if offset 0 is 2 (RTM2)</td>
</tr>
<tr>
<td>4 (4)</td>
<td>4</td>
<td>Pointer to general purpose registers 0 through 15 at the time of the event</td>
</tr>
<tr>
<td>8 (8)</td>
<td>4</td>
<td>Pointer to the 16-byte program status word (PSW) at the time of the event</td>
</tr>
</tbody>
</table>
| 12 (C) | 4      | One of the following, as indicated by the RTM/SLIP processing environment indicator at offset 0 of the work area:  
: Pointer to the system diagnostic work area (SDWA), if offset 0 is 1 (RTM1)  
: Pointer to the recovery termination manager 2 (RTM2) work area (RTM2WA), if offset 0 is 2 (RTM2)  
: Pointer to the address space control block (ASCB) being ended, if offset 0 is 3 (MEMTERM)  
: Pointer to the PER code, if offset 0 is 4 (PER) |
| 16 (10)| 4      | Pointer to cross memory information (control registers 3 and 4) at the time of the event |
| 20 (14)| 4      | Pointer to access registers AR0 through AR15 at the time of the event. Pointer to the high 32 bits of the 64-bit GPRs, or 0 if not available. See Wait State 01B in the *z/OS MVS System Codes* for more information. |

Problem data saved by first level interrupt handlers

If processing is stopped or an error occurs in one of the first level interrupt handlers (FLIH), you might need to determine the PSW and registers of the interrupted program. Field PSASUPER has bits to indicate if an FLIH was in control:

- PSAIO for the IO FLIH
- PSASVC for the SVC FLIH
- PSAEXT for the external FLIH
- PSAPI for the program interrupt FLIH

The following tables show where each FLIH saves the PSW and registers for interrupted tasks or SRB:

- Table 19
- Table 20 on page 131
- Table 21 on page 132
- Table 22 on page 132

Table 19. Problem data saved by the SVC FLIH for task and SRB code

<table>
<thead>
<tr>
<th>Code giving up control</th>
<th>Data saved</th>
<th>Field receiving data</th>
<th>Control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>All SVCs, initially</td>
<td>General purpose registers 7-9</td>
<td>PSAGPREG</td>
<td>PSA</td>
</tr>
<tr>
<td></td>
<td>General purpose registers, if a</td>
<td>LCCASGPR</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>problem occurred</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Stand-Alone dump

#### Table 19. Problem data saved by the SVC FLIH for task and SRB code (continued)

<table>
<thead>
<tr>
<th>Code giving up control</th>
<th>Data saved</th>
<th>Field receiving data</th>
<th>Control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>All SVCs</td>
<td>PSW</td>
<td>RBOPSW</td>
<td>Requestor's RB</td>
</tr>
<tr>
<td></td>
<td>Cross memory status</td>
<td>XSBXMCRS</td>
<td>XSB</td>
</tr>
<tr>
<td></td>
<td>PCLINK stack header</td>
<td>XSBSTKE</td>
<td>XSB</td>
</tr>
<tr>
<td></td>
<td>EAX</td>
<td>XSBEAX</td>
<td>XSB</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>STCBARS</td>
<td>STCB</td>
</tr>
<tr>
<td></td>
<td>Current linkage stack entry pointer</td>
<td>STCBLSDP</td>
<td>STCB</td>
</tr>
<tr>
<td>Type 1 and 6 SVCs</td>
<td>General purpose registers 0-15</td>
<td>TCBGRS</td>
<td>TCB</td>
</tr>
<tr>
<td>Type 2, 3, and 4 SVCs</td>
<td>General purpose registers 0-15</td>
<td>RBGRSAVE</td>
<td>SVRB</td>
</tr>
</tbody>
</table>

#### Table 20. Problem data saved for a program check for task and SRB code

<table>
<thead>
<tr>
<th>Code giving up control</th>
<th>Data saved</th>
<th>Field receiving data</th>
<th>Control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially for non-recursive program interruptions</td>
<td>General purpose registers 0-15</td>
<td>LCCAPGR2</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>LCCAPPSW</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>ILC/PINT</td>
<td>LCCAPINT</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>TEA</td>
<td>LCCAPVAD</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>TEA AR number</td>
<td>LCCAPTR2</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>Control registers 0-15</td>
<td>LCCAPCR2</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>LCCAPAR2</td>
<td>LCCA</td>
</tr>
<tr>
<td>Initially for recursive program interruptions</td>
<td>General purpose registers 0-15</td>
<td>LCCAPGR1</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>LCCAPPS1</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>ILC/PINT</td>
<td>LCCAPIC1</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>TEA</td>
<td>LCCAPTE1</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>TEA AR number</td>
<td>LCCAPTR2</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>Control registers 0-15</td>
<td>LCCAPCR1</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>LCCAPAR1</td>
<td>LCCA</td>
</tr>
<tr>
<td>Initially for monitor call interruptions that occur during page fault or segment fault processing</td>
<td>General purpose registers 0-15</td>
<td>LCCAPGR3</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>LCCAPPS3</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>ILC/PINT</td>
<td>LCCAPIC3</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>TEA</td>
<td>LCCAPTE3</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>TEA AR number</td>
<td>LCCAPTR3</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>Control registers 0-15</td>
<td>LCCAPCR3</td>
<td>LCCA</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>LCCAPAR3</td>
<td>LCCA</td>
</tr>
<tr>
<td>Initially for all trace buffer full interruptions</td>
<td>General purpose registers 0-15</td>
<td>LCCAPGR4</td>
<td>LCCA</td>
</tr>
<tr>
<td>For unlocked tasks for page faults or segment faults that require I/O; problem data is moved from the LCCA</td>
<td>Registers</td>
<td>TCB and STCB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>RB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Other status</td>
<td>XSB</td>
<td></td>
</tr>
</tbody>
</table>
### Stand-Alone dump

#### Table 20. Problem data saved for a program check for task and SRB code (continued)

<table>
<thead>
<tr>
<th>Code giving up control</th>
<th>Data saved</th>
<th>Field receiving data</th>
<th>Control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>For locked tasks for page faults or segment faults that require I/O; problem data is moved from the LCCA</td>
<td>Registers</td>
<td>IHSA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>IHSA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Other status</td>
<td>XSB for IHSA</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 21. Problem data saved by the I/O FLIH for task and SRB code

<table>
<thead>
<tr>
<th>Code giving up control</th>
<th>Data saved</th>
<th>Field receiving data</th>
<th>Control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td>General purpose registers 0-15</td>
<td>SCFSIGR1</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>Control registers 0-15</td>
<td>SCFSICR1</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>SCFSIAR1</td>
<td>SCFS</td>
</tr>
<tr>
<td>For unlocked tasks</td>
<td>General purpose registers 0-15</td>
<td>TCBGRS</td>
<td>TCB</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>RBOPSW</td>
<td>RB</td>
</tr>
<tr>
<td></td>
<td>Cross memory status</td>
<td>XSBXMCRS</td>
<td>XSB</td>
</tr>
<tr>
<td></td>
<td>EAX</td>
<td>XSB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>STCBARS</td>
<td>STCB</td>
</tr>
<tr>
<td></td>
<td>Current linkage stack entry pointer</td>
<td>STCBLSDP</td>
<td>STCB</td>
</tr>
<tr>
<td>For locally locked tasks</td>
<td>General purpose registers 0-15</td>
<td>IHSAGPRS</td>
<td>IHS for locked address space</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>IHSACPSW</td>
<td>IHS for locked address space</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>IHSAAARS</td>
<td>IHS for locked address space</td>
</tr>
<tr>
<td></td>
<td>Current linkage stack entry pointer</td>
<td>IHSALSDP</td>
<td>IHS for locked address space</td>
</tr>
<tr>
<td></td>
<td>Cross memory status</td>
<td>XSBXMCRS</td>
<td>XSB for locked address space</td>
</tr>
<tr>
<td></td>
<td>EAX</td>
<td>XSB</td>
<td></td>
</tr>
<tr>
<td>For SRBs and non-preemptive TCBs</td>
<td>General purpose registers 0-15</td>
<td>SCFSIGR1</td>
<td>SCF</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>FLCIOPSW</td>
<td>PSA</td>
</tr>
</tbody>
</table>

#### Table 22. Problem data saved by the external FLIH for task and SRB code

<table>
<thead>
<tr>
<th>Code giving up control</th>
<th>Data saved</th>
<th>Field receiving data</th>
<th>Control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td>General purpose registers 7-10</td>
<td>PSASLSA</td>
<td>PSA</td>
</tr>
</tbody>
</table>
### Table 22. Problem data saved by the external FLIH for task and SRB code (continued)

<table>
<thead>
<tr>
<th>Code giving up control</th>
<th>Data saved</th>
<th>Field receiving data</th>
<th>Control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>For locally locked tasks</td>
<td>General purpose registers 0-15</td>
<td>IHSAGPRS</td>
<td>IHSA</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>IHSACPSW</td>
<td>IHSA</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>IHSAAARS</td>
<td>IHSA</td>
</tr>
<tr>
<td></td>
<td>Current linkage stack entry pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cross memory status</td>
<td>XSBXMCRS</td>
<td>XSB</td>
</tr>
<tr>
<td></td>
<td>EAX</td>
<td>XSB</td>
<td></td>
</tr>
<tr>
<td>Unlocked tasks</td>
<td>General purpose registers 0-15</td>
<td>TCBGRS</td>
<td>TCB</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>RBOPSW</td>
<td>RB</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>STCBARS</td>
<td>STCB</td>
</tr>
<tr>
<td></td>
<td>Current linkage stack entry pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cross memory status</td>
<td>XSBXMCRS</td>
<td>XSB</td>
</tr>
<tr>
<td></td>
<td>EAX</td>
<td>XSB</td>
<td></td>
</tr>
<tr>
<td>For SRBs and non-preemptive TCBs</td>
<td>General purpose registers 0-15</td>
<td>SCFSXGR1</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>SCFSXPS1</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>Control registers 0-15</td>
<td>SCFSXCR1</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>SCFSXAR1</td>
<td>SCFS</td>
</tr>
<tr>
<td>If first recursion</td>
<td>General purpose registers 0-15</td>
<td>SCFSXGR1</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>SCFSXPS2</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>Control registers 0-15</td>
<td>SCFSXCR2</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>SCFSXAR2</td>
<td>SCFS</td>
</tr>
<tr>
<td>If second recursion</td>
<td>General purpose registers 0-15</td>
<td>SCFSXGR3</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
<td>FLCEOOPS</td>
<td>PSA</td>
</tr>
<tr>
<td></td>
<td>Control registers 0-15</td>
<td>SCFSXCR3</td>
<td>SCFS</td>
</tr>
<tr>
<td></td>
<td>Access registers 0-15</td>
<td>SCFSXAR3</td>
<td>SCFS</td>
</tr>
</tbody>
</table>
Stand-Alone dump
Chapter 5. ABEND dump

An ABEND dump shows the virtual storage predominately for an unauthorized program. Typically, a dump is requested when the program cannot continue processing and abnormally ends. An operator can also request an ABEND dump while ending a program or an address space.

The system can produce three types of ABEND dumps, one unformatted dump (SYSMDUMP) and two formatted dumps (SYSABEND and SYSUDUMP). These dumps are produced when a program cannot continue processing and a DD statement for an ABEND dump was included in the JCL for the job step that has ended. The data included is dependent on:

- Parameters supplied in the IEAABD00, IEADMR00, and IEADMP00 parmlib members for SYSABENDs, SYSMDUMPs, and SYSUDUMPs, respectively.
- A determination by the system
- ABEND, CALLRTM, or SETRP macro dump options
- IEAVTABX, IEAVADFM, or IEAVADUS installation exit processing

IBM recommends the use of SYSMDUMP, the unformatted dump. Unformatted dumping is more efficient because only the storage requested is written to the data set, which allows the application to capture diagnostic data and be brought back online faster. Also, pre-formatted dumps force the system to select a single set of reports, too many for the diagnosis of many problems, and too few for others. Unformatted dumps allow the analyst to determine, from a wide variety of reports, what information to use and how it is presented.

Use SYSUDUMP for diagnosis of program problems that need simple problem data. A SYSABEND dump, through the IBM supplied defaults, supplies more of the system information related to the application program’s processing than a SYSUDUMP. The additional information may be better suited for complex problem diagnosis.

This section covers the following topics, which describe how to use ABEND dumps:

- “Synopsis of ABEND dumps” on page 136
- “Obtaining ABEND dumps” on page 137
- “Printing and viewing dumps” on page 141
- “Contents of ABEND dumps” on page 142
- “Customizing ABEND dump contents” on page 147
- “Analyzing an ABEND dump” on page 153
## Synopsis of ABEND dumps

Use Table 23 as a quick reference for the three types of ABEND dumps. If you need further information about ABEND dumps, refer to the sections following this table.

### Table 23. Types of ABEND dumps

<table>
<thead>
<tr>
<th>Obtaining the dump</th>
<th>Receiving the dump</th>
<th>Dump contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SYSABEND</strong>:</td>
<td></td>
<td>Default contents: summary dump for the failing task and other task data. See “Contents of ABEND dumps” on page 142.</td>
</tr>
<tr>
<td>Assembler macro in any program:</td>
<td>Formatted dump in a data set with the ddname of SYSABEND:</td>
<td>Customized by all of the following:</td>
</tr>
<tr>
<td>• ABEND with DUMP</td>
<td>• In SYSOUT; print in the output class or browse at a terminal</td>
<td>• IEAADB00 parmlib member</td>
</tr>
<tr>
<td>• SETRP with DUMP=YES</td>
<td>• On tape or direct access: print in a separate job or browse at a terminal</td>
<td>• Parameter list on the requesting ABEND, CALLRTM, or SETRP macro</td>
</tr>
<tr>
<td></td>
<td>• On a printer (Not recommended; the printer cannot be used for anything else for the duration of the job step.)</td>
<td>• Recovery routines invoked by the recovery termination manager (RTM)</td>
</tr>
<tr>
<td>Assembler macro in an authorized program:</td>
<td>For full information, see “Obtaining ABEND dumps” on page 137.</td>
<td>• Cumulative from all CHNGDUMP operator commands with SYSABEND</td>
</tr>
<tr>
<td>• ABEND with DUMP</td>
<td></td>
<td>• Installation-written routines at the IEAVTABX, IEAVADF, and IEAVADUS exits</td>
</tr>
<tr>
<td>• CALLRTM with DUMP=YES</td>
<td></td>
<td>For full information about customization, see “Customizing ABEND dump contents” on page 147.</td>
</tr>
<tr>
<td>• SETRP with DUMP=YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operator command on a console with master authority:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• CANCEL with DUMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>For full information, see “Obtaining ABEND dumps” on page 137.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **SYSMDUMP**: | Unformatted dump in a data set with the ddname of SYSMDUMP: | Default contents: summary dump and system data for the failing task. See “Contents of ABEND dumps” on page 142. |
| Assembler macro in any program: | • On tape or direct access; use IPCS to format and print/view the dump | Customized by all of the following: |
| • ABEND with DUMP | For full information, see “Obtaining ABEND dumps” on page 137. | • IEADMR00 parmlib member |
| • SETRP with DUMP=YES | | • Parameter list on the requesting ABEND, CALLRTM, or SETRP macro |
| Assembler macro in an authorized program: | | • Recovery routines invoked by the recovery termination manager (RTM) |
| • ABEND with DUMP | | • Cumulative from all CHNGDUMP operator commands with SYSMDUMP |
| • CALLRTM with DUMP=YES | | • Installation-written routines at the IEAVTABX exit |
| • SETRP with DUMP=YES | | For full information about customization, see “Customizing ABEND dump contents” on page 147. |
| Operator command on a console with master authority: | | |
| • CANCEL with DUMP | | |
| For full information, see “Obtaining ABEND dumps” on page 137. | | |
Table 23. Types of ABEND dumps (continued)

<table>
<thead>
<tr>
<th>Obtaining the dump</th>
<th>Receiving the dump</th>
<th>Dump contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSUDUMP:</td>
<td>Formatted dump in a data set with the ddname of SYSUDUMP:</td>
<td>Default contents: summary dump for the failing task. See “Contents of ABEND dumps” on page 142.</td>
</tr>
<tr>
<td>Assembler macro in any program:</td>
<td>• In SYSOUT; print in the output class or browse at a terminal</td>
<td>Customized by all of the following:</td>
</tr>
<tr>
<td>• ABEND with DUMP</td>
<td>• On tape or direct access; print in a separate job or browse at a terminal</td>
<td>• IEADMP00 parmlib member</td>
</tr>
<tr>
<td>• SETRP with DUMP=YES</td>
<td>• On a printer (Not recommended; the printer cannot be used for anything else for the duration of the job step.)</td>
<td>• Parameter list on the requesting ABEND, CALLRTM, or SETRP macro</td>
</tr>
<tr>
<td>Assembler macro in an authorized program:</td>
<td>For full information, see “Obtaining ABEND dumps.”</td>
<td>• Recovery routines invoked by the recovery termination manager (RTM)</td>
</tr>
<tr>
<td>• ABEND with DUMP</td>
<td></td>
<td>• Cumulative from all CHNGDUMP operator commands with SYSUDUMP</td>
</tr>
<tr>
<td>• CALLRTM with DUMP=YES</td>
<td></td>
<td>• Installation-written routines at the IEA V TABX, IEA V ADFM, and IEA V ADUS exits</td>
</tr>
<tr>
<td>• SETRP with DUMP=YES</td>
<td></td>
<td>For full information about customization, see “Customizing ABEND dump contents” on page 147.</td>
</tr>
<tr>
<td>Operator command on a console with master authority:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• CANCEL with DUMP</td>
<td>For full information, see “Obtaining ABEND dumps.”</td>
<td></td>
</tr>
<tr>
<td>For full information, see “Obtaining ABEND dumps.”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Obtaining ABEND dumps

You can obtain SYSABEND, SYSUDUMP, and SYSMDUMP dumps using one process. To obtain a specific type of ABEND dump, specify the correct DD statement in your JCL as shown in Table 24 for more information about these statements, see z/OS MVS JCL Reference.

Table 24. Summary: DD statements to specify for specific ABEND dumps

<table>
<thead>
<tr>
<th>Dump Type</th>
<th>DD statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSABEND</td>
<td>//SYSABEND DD ...</td>
</tr>
<tr>
<td>SYSUDUMP</td>
<td>//SYSUDUMP DD ...</td>
</tr>
<tr>
<td>SYSMDUMP</td>
<td>//SYSMDUMP DD ...</td>
</tr>
</tbody>
</table>

Provide a data set to receive the dump, then arrange to view the dump. If a data set is not provided, the system ignores a request for an ABEND dump. When setting up the data set, determine if it will contain privileged data. If so, protect it with passwords or other security measures to limit access to it.

Because ABEND dumps provide information to debug application programs, the data they have access to is limited. Authorized programs require special processing to allow the information they can access into a dump. ABEND dump processing issues an IEA848I message when violations occur. The primary facility for dumping authorized data is through the SDUMPX macro, however, two security FACILITY classes are provided that allow installations to permit ABEND dumps to contain authorized data:

IEAABD.DMPAUTH

For access to programs that are protected by the PROGRAM facility.
ABEND dumps

IEAABD.DMPAKEY
For programs that execute in authorized keys.

For details on the SDUMPX macro, see
• z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU
• z/OS MVS Programming: Authorized Assembler Services Guide

Data set for dump

Define the data set in either:
• The JCL for the job step, for batch processing
• The logon procedure for a TSO/E userid, for foreground processing

Define the data set in a DD statement with a ddname of SYSABEND, SYSMDUMP, or SYSUDUMP. The ddname for the data set determines how the dump can be printed or viewed, what the dump contains, and how the dump contents can be customized. The first two effects are discussed in the following topics.

The system writes the dump in a sequential data set using the basic sequential access method (BSAM). The dump data set can be on any device supported by BSAM. Note that the system provides a data control block (DCB) for the dump data set and opens and closes the DCB.

You can also use extended format sequential data sets as dump data sets for ABEND dumps. Extended format sequential data sets have the following features:
• Have a greater capacity than sequential data sets
• Support striping
• Support compression

Using DSNTYPE=LARGE: In z/OS V1R7 and later releases, sequential data sets that use DSNTYPE=LARGE are allowable for ABEND dumps when the systems that are involved in processing a DSNTYPE=LARGE data set are migrated to V1R7 prior to their use. If analysis using an earlier release is required, use z/OS V1R7 to transcribe the dump into a data set supported by the earlier release.

Placing dump data sets in cylinder-managed space: In z/OS V1R11 and later releases, extended format sequential data sets can be placed in either track-managed space or cylinder-managed space. Abend dump fully supports placement of dump data sets in cylinder-managed space.

VIO for ADDRSPC=REAL: A SYSMDUMP DD statement must specify a virtual input/output (VIO) data set if the job or step to be dumped is running in non pageable virtual storage, that is, the JCL JOB or EXEC statement specifies ADDRSPC=REAL.

Preallocate data sets for SYSMDUMP dumps
You may use any dataset name you wish for the SYSMDUMP dataset. However, the dataset name SYS1.SYMDPxx will be treated specially. If you use the data set naming convention of SYS1.SYMDPxx for a DISP=SHR data set, the system writes only the first dump, with all subsequent dump requests receiving system message IEA849I. The data set can be either a magnetic tape unit or a direct access storage device (DASD) data set.
When using this naming convention, you must manage the dump data set to use the same data set repeatedly for SYSMDUMP dumps. For subsequent dumps, you must initialize the SYS1.SYSMDPxx data set with an end-of-file (EOF) mark as the first record.

**Naming Convention:** You must use SYS1.SYSMDPxx, where xx is 00 through FF and identifies the exact data set to be used.

**Data Set Disposition:** If you specify DISP=SHR with the SYS1.SYSMDPxx naming convention, the facility that enables the system to write only the first dump becomes active.

If you specify DISP=SHR without the SYS1.SYSMDPxx naming convention, the system writes a new dump over the old dump when the same data set is the target for multiple dumps. This also happens for multiple dumps within the same job if each step does not specify FREE=CLOSE on the SYSMDUMP DD statement.

For dispositions other than DISP=SHR, the system uses the data set as if it were any other MVS data set. If you specify DISP=MOD, the system writes the dump following the previous dump, so that the data set contains more than one dump. If you specify DISP=OLD, the system writes a new dump over the old dump when the same data set is the target for multiple dumps.

**Data Set Management:** To minimize the loss of subsequent dumps, your installation exit should follow these steps for the management of SYS1.SYSMDPxx data sets:

1. Intercept system message IEA993I. The system issues this message when it writes the dump to the SYS1.SYSMDPxx data set.
2. Copy the dump onto another data set.
3. Clear the SYS1.SYSMDPxx data set by writing an EOF mark as the first record, making it available for the next SYSMDUMP dump to be written on the data set.

The installation exit routine can be one of the following:

- IEAVMXIT
- The exit routine specified on the USEREXIT parameter in the MPFLSTxx parmlib member


**Process for obtaining ABEND dumps**

Obtain an ABEND dump by taking the following steps for each job step where you want to code a dump:

1. Code a DD statement in the JCL for every job step where a dump would be needed. The statement can specify one of the following:
   - Direct access
   - SYSOUT
   - Tape
   - Printer (Not recommended; printer cannot be used for anything else for duration of job step.)
### ABEND dumps

The presence of the DCB attributes enables the system-determined block size process to select an efficient block size for the DASD selected. For more information, see [z/OS DFSMS Using Data Sets](https://www.ibm.com/support/knowledgecenter/SSS47U_2.5.0/com.ibm.zos.v2r10.mc1rldata.doc/product/dfsmshsm_fregm.html#dfsmshsm_fregmct). Your installation can make specification of these attributes unnecessary through local SMS class selection routines.

For example, the following SYSOUT SYSABEND DD statement places the dump in sysout output class A. In the example, output class A is a print class. The system prints a dump written to this class when printing the class.

```
//SYSABEND DD SYSOUT=A
```

The following example places a SYSUDUMP dump on a scratch tape. In the example, TAPE is an installation group name. DEFER specifies that the operator is to mount the tape only when the data set is opened; thus, the operator will not mount the tape unless it is needed for a dump. The system deletes the data set if the job step ends normally; in this case, the data set is not needed because no dump was written. The system keeps the data set if the step ends abnormally; the data set contains a dump. A future job step or job can print the dump.

```
//SYSUDUMP DD DSN=DUMPDS,UNIT=(TAPE,,DEFER),DISP=(,DELETE,KEEP)
```

#### 2. Place the DD statement in the JCL for the job step that runs the program to be dumped or in the logon procedure for a TSO/E user ID.

The following example shows a SYSABEND DD statement in the logon procedure for a TSO/E user ID. A dump statement must appear in the logon procedure in order to process a dump in the foreground. The system keeps the data set if the job step ends abnormally.

```
//SYSABEND DD DSN=MYID3.DUMPS,DISP=(OLD,,KEEP)
```

#### 3. If you need to diagnose a program that does not contain code for an ABEND dump, code one of the following:

- **ABEND assembler macro with a DUMP parameter in a problem program or an authorized program**

  The following example shows an ABEND macro that ends a program with a user completion code of 1024 and requests a dump:

  ```
  ABEND 1024,DUMP
  ```

- **SETRP assembler macro with a DUMP=YES parameter in the recovery routine for a problem program or an authorized program**

  The following example shows a SETRP macro in an ESTAE recovery routine for a problem program. The address of the system diagnostic work area (SDWA) is in register 1, which is the default location.

  ```
  SETRP DUMP=YES
  ```

- **CALLRTM assembler macro with a DUMP=YES parameter in an authorized program**

  The following example shows a macro in an authorized program. The CALLRTM macro ends a program and requests a dump. Register 5 contains the address of the task control block (TCB) for the program.

  ```
  CALLRTM TYPE=ABTERM,TCB=(5),DUMP=YES
  ```

#### 4. If you need to diagnose a program that already contains code for an ABEND dump, and that program is already abending, skip step 5.

#### 5. If you need to diagnose a program that already contains code for an ABEND dump, but the program is not currently abending, ask the operator to enter a CANCEL command with a DUMP parameter on the console with master authority.
ABEND dumps

For example, to cancel a job and request a dump, ask the operator to use either of the following:

CANCEL BADJOB,DUMP

CANCEL STARTING,A=1234,DUMP

To cancel a user ID and request a dump, ask the operator to use either of the following commands:

CANCEL U=MYID3,DUMP

CANCEL U=LOGON*,A=5678,DUMP

6. The system writes a formatted dump to the data set defined in step 1.

Printing and viewing dumps

You can print or view the different types of ABEND dumps as follows:

SYSABEND and SYSUDUMP dumps: These two dumps are formatted as they are created. They can be:

- In a SYSOUT data set. The system can print the dump when printing the output class. To view at a terminal, use a facility that allows the viewing of JES SPOOL data sets.
- On a tape or direct access data set. Print the dump in a separate job or job step or view the dump at a terminal by browsing the data set containing the dump.
- A convenient way to print the dump is in a later job step that runs only if an earlier job step abnormally ends and, thus, requests a dump. For this, use the JCL EXEC statement COND parameter.
- Sent directly to a printer. Note this is not recommended; the printer cannot be used for anything else while the job step is running, whether a dump is written or not.

Figure 67 shows JCL that uses the IEBPTPCH facility to print a formatted dump data set. In this example, a SYSABEND dump is printed. The same JCL can be used for a SYSUDUMP. Because the system formats the dump when creating it, the IEBPTPCH utility program can print the dump. The dump is in a data set named DUMPDS on tape.

```plaintext
//PRINT EXEC PGM=IEBPTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=DUMPDS,UNIT=TAPE,DISP=(OLD,DELETE)
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
/*
 PRINT PREFORM=A,TYPORG=PS */
```

Figure 67. Example: Using IEBPTPCH to print a dump

SYSMDUMP dumps: This dump is unformatted when created. The system can write the dump to tape or direct access. Use IPCS to format the dump and then view it at a terminal or print it. SYSMDUMP dumps are especially useful for diagnosing errors because IPCS can produce specific information for specific requests. See z/OS MVS IPCS User’s Guide for more information.
ABEND dumps

Contents of ABEND dumps

You can specify the contents of an ABEND dump by specifying parameters on the ddname in the JCL for the program. This topic discusses the IBM-supplied default contents and contents available through customization.

All three ABEND dumps contain a summary dump, although the SYSMDUMP summary dump contains less information than the SYSABEND and SYSUDUMP summary dumps. The SYSUDUMP consists of only the summary dump. The SYSABEND dump also contains task data, while the SYSMDUMP also contains system data. The SYSMDUMP dump is a synchronous SVC dump and contains data similar to the data in an SVC dump.

Note:
1. ABEND dumps do not include hiperspaces. To include hiperspace in an ABEND dump, read the data from the hiperspace into address space storage that is being dumped.
2. If some needed areas are not included by default, see "Customizing ABEND dump contents" on page 147 for ways to add the areas.

Determining current ABEND dump options

Use a DISPLAY DUMP operator command to get the dump mode and options in effect for SVC dumps and ABEND SYSABEND, SYSMDUMP, and SYSUDUMP dumps. The system displays the mode and options in message IEE857I.

For example, to determine the mode and options, enter the following command:
DISPLAY DUMP,OPTIONS

If the options listed are not the ones desired, use a CHNGDUMP operator command to change them.

See z/OS MVS System Commands for the DISPLAY and CHNGDUMP operator commands. For a description of these messages, use LookAt or see MVS System Messages.

Default contents of ABEND dumps

The contents of the three ABEND dumps are detailed in the following two tables. Table 25 on page 143 shows dump contents alphabetically by the parameters that specify the areas in the dumps. To select a dump, decide what areas will be used to diagnose potential errors. Find the areas in the tables. The symbols in columns under the dump indicate how the area can be obtained in that dump. The symbols are:

- **D** IBM-supplied default contents
- **M** Available on the macro that requests the dump
- **P** Available in the parmlib member that controls the dump options
- **X** Available on the CHNGDUMP operator command that changes the options for the dump type
### Table 25. Summary: dump contents by parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Dump Contents</th>
<th>ABEND Dump to SYSUDUMP</th>
<th>ABEND Dump to SYSABEND</th>
<th>ABEND Dump to SYSMDUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>All the dump options available in a SYSMDUMP dump, except the NOSYM and ALLNUC options</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ALLNUC</td>
<td>The DAT-on and DAT-off nucleuses</td>
<td></td>
<td></td>
<td>P X</td>
</tr>
<tr>
<td>ALLPA</td>
<td>All link pack areas, as follows:</td>
<td>M P X</td>
<td>D M P X</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>• Job pack area (JPA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Link pack area (LPA) active for the task being dumped</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Related Supervisor Call (SVC) modules</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLPDATA</td>
<td>All the program data areas</td>
<td>P X</td>
<td>P X</td>
<td></td>
</tr>
<tr>
<td>ALLSDATA</td>
<td>All the system data areas</td>
<td>P X</td>
<td>P X</td>
<td>P</td>
</tr>
<tr>
<td>ALLVNUC</td>
<td>The entire virtual control program nucleus, including:</td>
<td>M P X</td>
<td>M P X</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>• Prefixed save area (PSA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• System queue area (SQA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Local system queue area (LSQA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CB</td>
<td>Control blocks for the task being dumped</td>
<td>M P X</td>
<td>D M P X</td>
<td>M</td>
</tr>
<tr>
<td>CSA</td>
<td>Common service area (CSA) (that is, subpools 227, 228, 231, 241)</td>
<td></td>
<td></td>
<td>P X</td>
</tr>
<tr>
<td>DM</td>
<td>Data management control blocks for the task being dumped:</td>
<td>M P X</td>
<td>D M P X</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>• Data control block (DCB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Data extent block (DEB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Input/output block (IOB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENQ</td>
<td>Global resource serialization control blocks for the task being dumped:</td>
<td>P X</td>
<td>D P X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Global queue control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Local queue control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERR</td>
<td>Recovery termination manager (RTM) control blocks for the task being dumped:</td>
<td>M P X</td>
<td>D M P X</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>• Extended error descriptor (EED) for RTM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Registers from the system diagnostic work area (SDWA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• RTM2 work area (RTM2WA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set task asynchronous exit (STAE) control block (SCB)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ABEND dumps**

Table 25. Summary: dump contents by parameter (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Dump Contents</th>
<th>ABEND Dump to SYSUDUMP</th>
<th>ABEND Dump to SYSABEND</th>
<th>ABEND Dump to SYSMDUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRSQ</td>
<td>Global resource serialization control blocks for the task being dumped:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Global queue control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Local queue control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>Input/output supervisor (IOS) control blocks for the task being dumped:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Execute channel program debug area (EXCPD)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Unit control block (UCB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPA</td>
<td>Job pack area (JPA): module names and contents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPA</td>
<td>Active link pack area (LPA): module names and contents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSQA</td>
<td>Local system queue area (LSQA) allocated for the address space</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(that is, subpools 203 - 205, 213 - 215, 223 - 225, 229, 230, 233 - 235,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>249, 253 - 255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOSYM</td>
<td>No symptom dump (message IEA995I)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUC</td>
<td>Read/write portion of the control program nucleus (that is, only non-page-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>protected areas of the DAT-on nucleus), including:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Communication vector table (CVT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Local system queue area (LSQA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Prefixed save area (PSA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• System queue area (SQA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCDATA</td>
<td>Program call information for the task</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSW</td>
<td>Program status word (PSW) when the dump was requested</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>Global resource serialization control blocks for the task being dumped:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Global queue control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Local queue control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

144  z/OS V2R2 MVS Diagnosis: Tools and Service Aids
### Table 25. Summary: dump contents by parameter (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Dump Contents</th>
<th>ABEND Dump to SYSUDUMP</th>
<th>ABEND Dump to SYSABEND</th>
<th>ABEND Dump to SYSMDUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGS</td>
<td>Registers at entry to ABEND, that is, when the dump was requested: • Access registers • Floating-point registers • General registers • Vector registers, vector status register, and vector mask register for a task that uses the Vector Facility</td>
<td>M P X</td>
<td>D M P X</td>
<td>M</td>
</tr>
<tr>
<td>RGN</td>
<td>Allocated pages in the private area of each address space being dumped, including subpools 0 - 127, 129 - 132, 203 - 205, 213 - 215, 223 - 225, 229, 230, 236, 237, 244, 249, 251 - 255</td>
<td></td>
<td>D P X</td>
<td></td>
</tr>
<tr>
<td>SA or SAH</td>
<td>Save area linkage information, program call linkage information, and backward trace of save areas</td>
<td>M P X</td>
<td>D M P X</td>
<td>M</td>
</tr>
<tr>
<td>SPLS</td>
<td>Storage allocated in user subpools 0 - 127, 129 - 132, 244, 251, and 252 for the task being dumped Note that SUBPLST in the macro parameter list for a SYSABEND or SYSUDUMP dump overrides SPLS in the dump options list, but only for the dump being requested.</td>
<td>M P X</td>
<td>D M P X</td>
<td>M</td>
</tr>
<tr>
<td>SQA</td>
<td>System queue area (SQA) allocated (that is, subpools 226, 239, 245, 247, 248) The control blocks for the failing task in the SQA include: • Address space control block (ASCB) • Job scheduler address space control block (JSAB)</td>
<td>M P X</td>
<td>M P X</td>
<td>D M P X</td>
</tr>
<tr>
<td>SUBTASKS</td>
<td>Storage for the task being dumped and program data for all of its subtasks</td>
<td>M P X</td>
<td>M P X</td>
<td>D M</td>
</tr>
<tr>
<td>SUM</td>
<td>Summary dump, see “Default contents of summary dumps in ABEND dumps” on page 146</td>
<td>D M P X</td>
<td>D M P X</td>
<td>D M P X</td>
</tr>
<tr>
<td>SWA</td>
<td>Scheduler work area (SWA); that is, subpools 236 and 237</td>
<td>M P X</td>
<td>M P X</td>
<td>D M P X</td>
</tr>
<tr>
<td>TRT</td>
<td>System trace and generalized trace facility (GTF) trace, as available</td>
<td>M P X</td>
<td>D M P X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>System trace, as available</td>
<td></td>
<td></td>
<td>D M P X</td>
</tr>
</tbody>
</table>
ABEND dumps

Default contents of summary dumps in ABEND dumps

If only a summary dump is requested, as in a SYSUDUMP dump that is not customized, the summary information is together, because it forms the entire dump. When a summary dump is combined with other dump options, the summary dump information is scattered throughout the dump. In Table 26 an S indicates that a summary dump is available with the dump type.

Table 26. Default contents of summary dumps in ABEND dumps

<table>
<thead>
<tr>
<th>Summary Dump Contents</th>
<th>ABEND Dump to SYSUDUMP</th>
<th>ABEND Dump to SYSABEND</th>
<th>ABEND Dump to SYSMDUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion code</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASCB (address space control block)</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>CDE (contents directory entry)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LLE (load list element)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RB (request block)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TCB (task control block)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIOT (task input/output table)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XL (extent list)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recovery termination manager (RTM):</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>EED (extended error descriptor) for RTM</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Registers from the system diagnostic work area (SDWA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTM2WA (RTM2 work area)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCB (set task asynchronous exit (STAE) control block)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dump header</td>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Dump index</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Dump title</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load module, if the PSW points to an active load module:</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>Module Contents</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>Offset to the load module of the failing instruction</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>Module pointed to in the last PRB (program request block)</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>PSW (program status word) at entry to ABEND, that is, when the dump was requested. The PSW includes the instruction length code and the interrupt code for the failing instruction.</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Registers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Save areas of register contents</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Storage: 4 kilobytes before and 4 kilobytes after the addresses in the PSW and the registers.</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>System trace table entries for the dumped address space</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>
Table 26. Default contents of summary dumps in ABEND dumps (continued)

<table>
<thead>
<tr>
<th>Summary Dump Contents</th>
<th>ABEND Dump to SYSUDUMP</th>
<th>ABEND Dump to SYSABEND</th>
<th>ABEND Dump to SYSMDUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCB summary: Information from the task control blocks (TCB) in the address space being dumped</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Virtual storage map: The subpools in the address space being dumped:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Subpool number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Subpool key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The owning or sharing task control block (TCB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The beginning address and length of each allocated area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The beginning address and length of each free area</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Customizing ABEND dump contents

The ddname of the data set for the ABEND dump determines how the contents can be customized. The system determines the contents of a particular ABEND dump from the options list the system maintains for the type of dump. The dump options list can be customized, cumulatively, by all the ways shown in the following tables; thus, for example, a SYSMDUMP ABEND dump written for an ABEND macro can be completely different from the default SYSMDUMP ABEND dump described in this document.

- Table 27 on page 148
- Table 28 on page 150
- Table 29 on page 152

For more information about the topics described in this section, see the following references:

- See [z/OS MVS Initialization and Tuning Reference](#) for parmlib members.
- See [z/OS MVS System Commands](#) for the CHNGDUMP operator command.
- See [z/OS MVS Programming: Assembler Services Reference ABE-HSP](#) and [z/OS MVS Programming: Assembler Services Reference IAR-XCT](#) for the ABEND, SETRP, SNAP, SNAPX, ESTAE, ESTAEX, and ATTACH or ATTACHX with ESTAI macros.
- See [z/OS MVS Programming: Authorized Assembler Services Reference SET-WTC](#) and [z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN](#) for the SETRP and CALLRTM macros.
- See [z/OS MVS Installation Exits](#) for the IEAVTABX, IEAVADFMY, and IEAVADUS installation exits.

Recommendations for customizing ABEND dumps: How an installation customizes dumps should depend on the usual use of each type of dump. The IBM-supplied dump options for ABEND dumps are designed for the following uses:

- SYSABEND dumps: For diagnosis of complex errors in any program running under the operating system
- SYSMDUMP dumps: For diagnosis of system problems when the dump is requested in a program
- SYSUDUMP dumps: For diagnosis of program problems needing simple problem data
ABEND dumps

For SYSUDUMP dumps, the IBM-supplied IEADMP00 member specifies the default contents as only a summary dump. An installation should consider using the IEADMP00 member as supplied, because it offers a small dump for simple problems.

Program areas in dumps: To request a meaningful dump for a particular program, code an ABEND macro that points to a macro parameter list. Specify in the list the data areas that are needed to diagnose the abnormally ending program but that are not specified in the parmlib member for the dump. Two examples are:

- If the task that is ending has subtasks and they might cause an error, specify PDATA=SUBTASKS in the macro parameter list to dump the subtasks.
- To see only the subpools used by the program, specify the subpool numbers in a SUBPLST option for a SYSABEND dump. The SPLS option, which is a default for SYSABEND dumps, writes all user subpools. Leaving SPLS in the dump options may make the dump bigger than needed. Note that SUBPLST in the macro parameter list overrides SPLS in the current dump options.

Nucleus areas in dumps: Dump options control the parts of the nucleus that appear in a dump. A diagnostician seldom needs to analyze all of the nucleus. An installation can eliminate nucleus areas from dumps. If the IBM-supplied defaults are used, an SYSMDUMP ABEND dump contains the read/write DAT-on nucleus.

An installation can obtain one copy of the DAT-off nucleus to use in any problem by entering a DUMP operator command.

The ABEND dump options that control dumping of the nucleus areas are:

Option Nucleus aArea
SDATA=NUC
   Read/write DAT-on nucleus

SDATA=ALLNUC
   All of the DAT-on nucleus: read/write and read-only

Customizing SYSABEND dump contents

Table 27 summarizes how to customize the contents of SYSABEND dumps.

<table>
<thead>
<tr>
<th>SYSABEND Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
</table>
| Replacing IEAABD00 parmlib member (by using the IEBUDPTE utility). | Change occurs: At system initialization | To add program call data and the link pack area to all SYSABEND dumps, while retaining the IBM-supplied options, use IEBUDPTE to change the IEAABD00 member to contain:
   SDATA=(LSQA,CB,ENQ,TRT,ERR, DM,1O,SUM,PCDATA)
   PDATA=(PSW,REGS,SPLS,ALLPA, SA,LPA) |
|                                      | What changes: IEAABD00 contains the IBM-supplied default dump options. Replacing IEAABD00 changes the dump options for SYSABEND. |                                                                          |
### Using a macro parameter list

The DUMPOPT or DUMPOPX parameter on the ABEND or CALLRTM macro points to the parameter list. The list is usually created by a list-form SNAP or SNAPX macro.

| Change occurs: | At dump request |
| Change occurs: | Just before dumping |
| Change occurs: | Immediately when entered |

#### Example

To add program call data and the link pack area to this SYSABEND dump, code in the program:

```
ABEND 76,DUMP,
DUMPOPT=PARMS
PARMS SNAP SDATA=PCDATA,
PDATA=LPA,MF=L
```

#### Recovery routines

- **FRRs** (function recovery routines) for a system component
- **ESTAE/ESTAI** recovery routines established by an ESTAE or ESTAEX macro or the ESTAI parameter of an ATTACH or ATTACHX macro
- **ARRs** (associated recovery routines)

These routines issue SETRP macros. To customize the dump contents, the DUMPOPT or DUMPOPX parameter on the SETRP macro points to a parameter list. The list is usually created by a list-form SNAP or SNAPX macro.

#### Entering CHNGDUMP operator command with SYSABEND parameter on a console with master authority.

| Change occurs: | At dump request |
| Change occurs: | Just before dumping |
| Change occurs: | Immediately when entered |

#### Example

To add program call data and the link pack area to this SYSABEND dump, code in the recovery routine:

```
SETRP,DUMP=YES,
DUMPOPT=PARMS
PARMS SNAP SDATA=PCDATA,
PDATA=LPA,MF=L
```

When more than one CHNGDUMP operator command with SYSABEND is entered, the effect is cumulative.
### ABEND dumps

**Table 27. Customizing SYSABEND dump contents (continued)**

<table>
<thead>
<tr>
<th>SYSABEND Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Through IEAVTABX installation exit name list.</td>
<td>Change occurs: Just before dumping</td>
<td>See <a href="https://www-01.ibm.com/support/docview.wss?uid=swg27030700">z/OS MVS Installation Exits</a></td>
</tr>
<tr>
<td></td>
<td>What changes: The routine can add or delete options from the dump options, but only for the current dump.</td>
<td></td>
</tr>
<tr>
<td>Through IEAVADFM or IEAVADUS installation exits. IEAVADFM is a list of installation routines to be run. IEAVADUS is one installation routine.</td>
<td>Change occurs: During dumping. The routine runs during control block formatting of a dump with the CB option.</td>
<td>See <a href="https://www-01.ibm.com/support/docview.wss?uid=swg27030700">z/OS MVS Installation Exits</a></td>
</tr>
<tr>
<td></td>
<td>What changes: The routine can add control blocks to the dump.</td>
<td></td>
</tr>
</tbody>
</table>

### Customizing SYSMDUMP dump contents

**Table 28. Customizing SYSMDUMP dump contents**

**Table 28 contains a summary of how to customize the contents of SYSMDUMP dumps.**

<table>
<thead>
<tr>
<th>SYSMDUMP Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replacing IEADMR00 parmlib member (by using the IEBUPDTE utility).</td>
<td>Change occurs: At system initialization</td>
<td>To add the link pack area to all SYSMDUMP dumps, while retaining all the IBM-supplied defaults, use IEBUPDTE to change the IEADMR00 member to contain: SDATA=(NUC, SQA, LSQA, SWA, TRT, RGN, SUM, LPA)</td>
</tr>
<tr>
<td></td>
<td>What changes: IEADMR00 contains the IBM-supplied default dump options. Replacing IEADMR00 changes the dump options for SYSMDUMP.</td>
<td></td>
</tr>
<tr>
<td>Using a macro parameter list.</td>
<td>Change occurs: At dump request</td>
<td>To add the link pack area to this SYSMDUMP dump, code in the program:</td>
</tr>
<tr>
<td></td>
<td>What changes: The macro parameter list options are added to the dump options list, but only for the dump being requested.</td>
<td>ABEND 76,DUMP, DUMPOPT=PARMS PARMS SNAP PDATA=LPA, MF=L</td>
</tr>
</tbody>
</table>

**Using a macro parameter list.**

The DUMPOPT or DUMPOPX parameter on the ABEND or CALLRTM macro points to the parameter list. The list is usually created by a list-form SNAP or SNAPX macro.
### Table 28. Customizing SYSMDUMP dump contents (continued)

<table>
<thead>
<tr>
<th>SYSMDUMP Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recovery routines</strong> invoked by the recovery termination manager:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• FRRs (function recovery routines) for a system component</td>
<td>Change occurs: Just before dumping</td>
<td>To add the link pack area to this SYSMDUMP dump, code in the recovery routine:</td>
</tr>
<tr>
<td>• ESTAE/ESTAI recovery routines established by an ESTAE or ESTAEX macro or the ESTAI parameter of an ATTACH or ATTACHX macro</td>
<td>What changes: The SETRP macro parameter list options are added to the dump options list, but only for the dump being requested.</td>
<td>SETRP,DUMP=YES, DUMPOPT=PARMS</td>
</tr>
<tr>
<td>• ARRs (associated recovery routines)</td>
<td></td>
<td>PARMS SNAP PDATA=LPA,MF=L</td>
</tr>
<tr>
<td>These routines issue SETRP macros. To customize the dump contents, the DUMPOPT or DUMPOPX parameter on the SETRP macro points to a parameter list. The list is usually created by a list-form SNAP or SNAPX macro.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entering CHNGDUMP operator command with SYSMDUMP parameter on a console with master authority.</strong></td>
<td>Change occurs: Immediately when entered</td>
<td>To add the link pack area to all SYSMDUMP dumps until changed by CHNGDUMP DEL,SYSMDUMP, enter:</td>
</tr>
<tr>
<td></td>
<td>What changes:</td>
<td>CHNGDUMP SET,ADD,SYSMDUMP=(LPA)</td>
</tr>
<tr>
<td>• <strong>For ADD:</strong> CHNGDUMP options are added to the IEADMR00 options, previous CHNGDUMP options, and macro parameter list options. The options remain added until a CHNGDUMP DEL,SYSMDUMP operator command is entered.</td>
<td></td>
<td>To return to the IEADMR00 options, enter:</td>
</tr>
<tr>
<td>• <strong>For OVER:</strong> CHNGDUMP options override all other dump options.</td>
<td></td>
<td>CHNGDUMP DEL,SYSMDUMP</td>
</tr>
<tr>
<td>• <strong>For DEL:</strong> All CHNGDUMP options are deleted and the dump options in IEADMR00 are used again.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>When more than one CHNGDUMP operator command with SYSMDUMP is entered, the effect is cumulative.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Through IEAVTABX installation exit name list.</strong></td>
<td>Change occurs: Just before dumping</td>
<td>See z/OS MVS Installation Exits</td>
</tr>
<tr>
<td></td>
<td>What changes: The routine can add or delete options from the dump options, but only for the current dump.</td>
<td></td>
</tr>
</tbody>
</table>

### Customizing SYSUDUMP dump contents

Table 29 on page 152 contains a summary of how to customize the contents of SYSUDUMP dumps.
### ABEND dumps

Table 29. Customizing SYSUDUMP dump contents

<table>
<thead>
<tr>
<th>SYSUDUMP Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
</table>
| **Replacing IEADMP00 parmlib member (by using the IEBUPDTE utility).** | **Change occurs:** At system initialization  
**What changes:** IEADMP00 contains the IBM-supplied default dump options. Replacing IEADMP00 changes the dump options for SYSUDUMP. | **To add program call data and user subpool storage to all SYSUDUMP dumps, while retaining the summary dump, use IEBUPDTE to change the IEADMP00 member to contain:**  
SDATA=(SUM,PCDATA)  
PDATA=SPLS |

**Using a macro parameter list.**  
The DUMPOPT or DUMPOPX parameter on the ABEND or CALLRTM macro points to the parameter list. The list is usually created by a list-form SNAP or SNAPX macro.  
| **Change occurs:** At dump request  
**What changes:** The macro parameter list options are added to the dump options list, but only for the dump being requested.  
Note that SUBPLST in the macro parameter list overrides SPLS in the dump options list, but only for the dump being requested. | **To add program call data and user subpool storage to this SYSUDUMP dump, code in the program:**  
ABEND 76,DUMP,  
PARMS SNAP SDATA=PCDATA,  
PDATA=SPLS,MF=L | |

**Recovery routines** invoked by the recovery termination manager:  
- FRRs (function recovery routines) for a system component  
- ESTAE/ESTAI recovery routines established by an ESTAE or ESTAEX macro or the ESTAI parameter of an ATTACH or ATTACHX macro  
- ARRs (associated recovery routines)  
These routines issue SETRP macros.  
To customize the dump contents, the DUMPOPT or DUMPOPX parameter on the SETRP macro points to a parameter list. The list is usually created by a list-form SNAP or SNAPX macro.  
| **Change occurs:** Just before dumping  
**What changes:** The SETRP macro parameter list options are added to the dump options list, but only for the dump being requested. | **To add program call data and user subpool storage to this SYSUDUMP dump, code in the recovery routine:**  
SETRP ,DUMP=YES,  
DUMPOPT=PARMS  
PARMS SNAP SDATA=PCDATA,  
PDATA=SPLS,MF=L | |
Table 29. Customizing SYSUDUMP dump contents (continued)

<table>
<thead>
<tr>
<th>SYSUDUMP Customization</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entering CHNGDUMP operator command with SYSUDUMP parameter on a console with master</td>
<td>Change occurs: Immediately when entered</td>
<td>To add program call data and user subpool storage to all SYSUDUMP dumps until changed by CHNGDUMP DEL, SYSUDUMP, enter:</td>
</tr>
<tr>
<td>authority.</td>
<td>What changes:</td>
<td>CHNGDUMP SET, ADD, SYSUDUMP, SDATA=PCDATA, PDATA=SPLS</td>
</tr>
<tr>
<td></td>
<td>• For ADD: CHNGDUMP options are added to the IEADMP00 options, previous CHNGDUMP options, and all macro parameter list options. The options remain added until a CHNGDUMP DEL, SYSUDUMP operator command is entered.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• For OVER: CHNGDUMP options override all other dump options.</td>
<td>To return to the IEADMP00 options, enter:</td>
</tr>
<tr>
<td></td>
<td>• For DEL: All CHNGDUMP options are deleted and the dump options in IEADMP00 are used again.</td>
<td>CHNGDUMP DEL, SYSUDUMP</td>
</tr>
<tr>
<td></td>
<td>When more than one CHNGDUMP operator command with SYSUDUMP is entered, the effect is cumulative.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Through IEAVTABX installation exit name list.</td>
<td>Change occurs: Just before dumping</td>
<td>See z/OS MVS Installation Exits</td>
</tr>
<tr>
<td></td>
<td>What changes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The routine can add or delete options from the dump options, but only for the current dump.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Through IEAVADFM or IEAVADUS installation exits. IEAVADFM is a list of installation</td>
<td>Change occurs: During dumping. The routine runs during control block</td>
<td>See z/OS MVS Installation Exits</td>
</tr>
<tr>
<td>routines to be run and IEAVADUS is one installation routine.</td>
<td>formatting of a dump with the CB option.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>What changes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The routine can add control blocks to the dump.</td>
<td></td>
</tr>
</tbody>
</table>

Analyzing an ABEND dump

Note: A SYSMDUMP ABEND dump is always a synchronous SVC dump. To analyze a SYSMDUMP, see “Analyzing an SVC dump” on page 39.

ABEND dumps written to SYSABEND and SYSUDUMP data sets are useful for analyzing problems in a program running under the operating system. This program can be called any of the following:

• Installation-provided program  
• An application program  
• A non-authorized program  
• A problem program  
• A program in the private area

ABEND dumps are written for problems detected in two ways:

• Software-detected problem, such as:
  • A nonzero return code from a called module
ABEND dumps

- A program check, abend code X'0Cx', that a recovery routine changes to another abend code
- An erroneous control block queue
- Not valid input to a system service

• Hardware-detected problem, which is a program check, abend code X'0Cx', that a recovery routine does not change to another abend code

Analysis Procedure

To analyze a SYSABEND or SYSUDUMP, take the following steps:

1. **Collect and analyze logrec error records.**
   Check all logrec error records related to the abended task. Determine if any records show an earlier system problem; if so, continue diagnosis with that problem. Because of recovery and percolation, a SYSABEND or SYSUDUMP dump can be the end result of an earlier system problem.

2. **Collect and analyze messages about the problem.** Use time stamps to select messages related to the problem:
   - The job log
   - The system log (SYSLOG) or operations log (OPERLOG)
   Check the messages for earlier dumps written while the abended task was running. Determine if these earlier dumps indicate an earlier system problem; if so, continue diagnosis with that problem.

3. **Analyze the dump**, as described in the following steps.

   **Note:** After the problem and before the dump, recovery tried to reconstruct erroneous control block chains before ending the task. If the problem proves to be in a system component, a SYSABEND or SYSUDUMP dump cannot be used to isolate it because of the recovery actions; these dumps are useful only for problems in the private area.

4. **Obtain the abend code, reason code, job name, step name, and program status word (PSW) from the dump title** at the beginning of the dump.
   If the completion code is USER=dddd, an application program issued an ABEND macro to request the dump and to specify the completion code.
   If the completion code is SYSTEM=hhh, a system component ended the application program and a recovery routine in the program requested the dump. The application program probably caused the abend.

   **Reference** See [z/OS MVS System Codes](#) for an explanation of the abend code.

5. **Analyze the RTM2WA**, as follows:
   - In the TCB summary, find the task control block (TCB) for the failing task. This TCB has the abend code as its completion code in the CMP field. In the TCB summary, obtain the address of the recovery termination manager 2 (RTM2) work area (RTM2WA) for the TCB.
   - In the RTM2WA summary, obtain the registers at the time of the error and the name and address of the abending program.
   - If the RTM2WA summary does not give the abending program name and address, probably an SVC instruction abnormally ended.
   - If the RTM2WA summary gives a previous RTM2WA for recursion, the abend for this dump occurred while an ESTAE or other recovery routine was processing another, original abend. In recursive abends, more than one RTM2WA may be created. Use the previous RTM2WA to diagnose the original problem.
ABEND dumps

For information about the RTM2WA, SDWA, and TCB data areas, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

6. Analyze the dump for the program name. Obtain the program name from the RTM2WA summary. If the name field is zero, do the following:
   - Find the control blocks for the task being dumped.
   - The last request blocks are SVRBs. In the WLIC field in an SVRB, find the following SVC interruption codes:
     - X’33’ for a SNAP SVC interruption
     - X’0C’ for a SYNCH SVC interruption
   - The program request block (PRB) for the abending program immediately precedes these SVRBs.
   - When the dump contains more than one CDE, determine the first and last address for each CDE. The entry point address is the first address. Add the length to the entry point address to obtain the last address. Compare these addresses to the address in the right half of the PSW in the dump header; the PSW address falls between the first and last addresses of the correct CDE.
     Note that the leftmost digit in the PSW address denotes addressing mode and is not part of the address.
   - In that CDE, the NAME field gives the program name.

7. Locate the failing program module in the hexadecimal dump.

8. Find the instruction that caused the abend.
   The PSW in the dump header is from the time of the error. Obtain the address in the right half of the PSW. The leftmost digit denotes addressing mode and is not part of the address.
   For most problems, subtract the instruction length in the ILC field of the dump header from the PSW address to obtain the address of the failing instruction. Do not subtract the instruction length in the following cases; the failing instruction is at the PSW address:
   - Page translation exception.
   - Segment translation exception.
   - Vector operation interruption.
   - Other interruptions for which the processing of the instruction identified by the old PSW is nullified. See z/Architecture Principles of Operation for the interruption action.
   - If access registers were being used at the time of the error, so that the access list entry token (ALET) may be incorrect.
   Subtract the failing instruction address from the failing module address. Use this offset to find the matching instruction in the abending program’s assembler listing.

9. For an abend from an SVC or system I/O routine, find the last program instruction.
   If the abend occurred in a system component running on behalf of the dumped program, find the last instruction that ran in the program, as follows:
   - For an abend from an SVC routine, look in the last PRB in the control blocks for the task being dumped. The right half of the PSW in the RTPSW1 field contains the address of the instruction following the SVC instruction.
ABEND dumps

- For an abend from a system I/O routine, look in the save area trace. This trace gives the address of the I/O routine branched to. The return address in that save area is the last instruction that ran in the failing program.

10. **For an abend from an SVC or system I/O routine, determine the cause of the abend**, using the following:
   - For an abend from an SVC, look in the system trace table for SVC entries matching the SVRBs in the control blocks for the task being dumped.
   - For an abend from an I/O routine, look in the system trace table for I/O entries issued from addresses in the failing program. The addresses are in the PSW ADDRESS column.

If SVC entries match the dumped blocks or the I/O entries were issued from the failing program, the system trace table was not overlaid between the problem and the dump.

In this case, start with the most recent entries at the end of the trace. Back up to the last SVC entry with the TCB address of the abending task. Go toward the end of the trace, looking for indications of the problem. See [Chapter 8, “System trace,” on page 169](http://www.ibm.com/systems/z/os/zos/bkserv/) for more information.

11. **For a program interrupt, determine the cause of the abend**, using the registers at the time of the error in the RTM2WA and in the SVRB following the PRB for the abending program.

Also, look at the formatted save area trace for input to the failing module.

12. **For an abend in a cross memory environment**, do the following to analyze the dump.

Many services are requested by use of the Program Call (PC) instruction, rather than by SVCs or SRBs. When an abend is issued by the PC routine, the OPSW field in the RB contains the instruction address of the PC routine that issued the abend. The SVRB contains the registers of the PC routine.

Do the following to look for the registers and PSW at the time the PC instruction was issued:
   - For a stacking PC, find the registers in the linkage stack. Any entries on the linkage stack are before the RBs in the dump.
   - For a basic PC, find the registers in the PCLINK stack. Any entries on the PCLINK stack are after the RBs in the dump.

For a stacking PC, find the linkage stack entry that corresponds to the RB/XSB for the program. The LSED field of the linkage stack entry and the XSBLSSCP field in the corresponding XSB have the same value. From the linkage stack entry, obtain the registers and the PSW at the time the stacking PC was issued. The address in the PSW points to the instruction following the PC instruction in the abending program.

For a basic PC, determine the caller from the PCLINK stack. To locate the PCLINK stack element (STKE):
   - The STKEs appear in the dump following all of the RBs. If the dump contains more than one STKE, the pointer to the STKE for the PC involved in the problem is in the XSBSTKE field of the XSB associated with the RB for the abending program.
   - The RBXSB field in the RB points to the XSB.
   - The XSBSEL field in the XSB points to the current STKE.

In the STKE, the STKERET field contains the return address of the caller of the PCLINK service.

For information about the STKE and XSB data areas, see [z/OS MVS Data Areas](http://www.ibm.com/systems/z/os/zos/bkserv/) in [z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/)
Chapter 6. SNAP dump

This topic (SNAP Dump) contains programming interface information.

A SNAP dump shows virtual storage areas that a program, while running, requests the system to dump. A SNAP dump, therefore, is written while a program runs, rather than during abnormal end. The program can ask for a dump of as little as a one-byte field to as much as all of the storage assigned to the current job step. The program can also ask for some system data in the dump.

A SNAP dump is especially useful when testing a program. A program can dump one or more fields repeatedly to let the programmer check intermediate steps in calculations. For example, if a program being developed produces incorrect results, requests for SNAP dumps can be added to the program to dump individual variables. The first time that incorrect storage is encountered should narrow down the section of code causing the error.

Obtaining SNAP dumps

Provide a data set to receive the dump, then arrange to print the dump. The SNAP or SNAPX macros in a program can place their dumps in the same or different data sets; the DCB parameter in each SNAP or SNAPX macro indicates the data set.

When setting up a dump data set, determine if the data set will contain privileged data. If so, protect it with passwords or other security measures to limit access to it.

You can use extended format sequential data sets as dump data sets for SNAP dumps. Extended format sequential data sets have the following features:

- Have a greater capacity than sequential data sets
- Support striping
- Support compression

Using DSNTYPE=LARGE: In z/OS V1R7 and later releases, sequential data sets that use DSNTYPE=LARGE are allowable for ABEND dumps when the systems that are involved in processing a DSNTYPE=LARGE data set are migrated to V1R7 prior to their use. If analysis using an earlier release is required, use z/OS V1R7 to transcribe the dump into a data set supported by the earlier release.

Placing dump data sets in cylinder-managed space: In z/OS V1R11 and later releases, extended format sequential data sets can be placed in either track-managed space or cylinder-managed space. SNAP dump fully supports placement of dump data sets in cylinder-managed space.

Obtain a SNAP dump by taking the following steps:

1. Code a DD statement in the JCL for the job step that runs the problem program to be dumped with a ddname other than SYSUDUMP, SYSABEND, SYSMDUMP, or another restricted ddname. The statement can specify that the output of the SNAP dump should be written to one of the following:
   - Direct access.
**SNAP dumps**

- Printer. Note that a printer is not recommended, except when running under z/VM®, because the printer cannot be used for anything else while the job step is running, whether a dump is written or not. Under z/VM you can use a virtual printer. This allows you to see or print the partial output on a real printer while the program is running while only using a small amount of system resources.
- SYSOUT. SNAP dumps usually use SYSOUT.
- Tape.

**Example: SYSOUT DD Statement for SNAP Dump:** The following example places a SNAP dump in sysout output class A. In the example, output class A is a print class. When the system prints the output class, the system will print any dumps written to the class.

```
//SNAP1 DD SYSOUT=A
```

**Example: Tape DD Statement for SNAP Dump:** The following example places a SNAP dump on a tape. In the example, TAPE is a group name established by the installation.

```
//SNAP2 DD DSN=DUMPS,UNIT=TAPE,DISP=(,KEEP,KEEP)
```

The system keeps the data set when the job step ends, whether normally or abnormally. In either case, SNAP dumps are taken throughout processing, regardless of the way the step ends.

**Example: Direct Access DD Statement for SNAP Dump:** The following example places a SNAP dump on direct access, for example, the 3350 direct access storage.

```
//SNAP3 DD DSN=SNAPSHOT,UNIT=3350,DISP=(,KEEP,KEEP),
// VOLUME=SER=12345,SPACE=(1680,(160,80))
```

The system writes the dump in a sequential data set using the basic sequential access method (BSAM). The dump data set can be on any device supported by BSAM.

2. In the problem program:
   a. Specify a data control block (DCB) for the data set to receive the dump. For a standard dump, which has 120 characters per line, the DCB must specify:
      - BLKSIZE=882 or 1632
      - DSORG=PS
      - LRECL=125
      - MACRF=(W)
      - RECFM=VBA

      For a high-density dump, which has 204 characters per line and will be printed on an APA 3800 printer, the DCB must specify:
      - BLKSIZE=1470 or 2724
      - DSORG=PS
      - LRECL=209
      - MACRF=(W)
      - RECFM=VBA

   b. Code an OPEN macro to open the DCB.

   Before you issue the SNAP or SNAPX macro, you must open the DCB that you designate on the DCB parameter, and ensure that the DCB is not closed until the macro returns control. To open the DCB, issue the DCB macro with the following parameters, and issue an OPEN macro for the data set:

   ```
   DSORG=PS,RECFM=VBA,MACRF=(W),BLKSIZE=nnn,LRECL=xxx,
   and DDNAME=any name but SYSABEND, SYSDUMP or SYSUDUMP
   ```
If the system loader processes the program, the program must close the DCB after the last SNAP or SNAPX macro is issued.

c. Code a SNAP or SNAPX assembler macro to request the dump.

**Example: Coding the SNAP Macro:** In the following example, the SNAP macro requests a dump of a storage area, with the DCB address in register 3, a dump identifier of 245, the storage area’s starting address in register 4, and the ending address in register 5:

```
SNAP DCB=(3),ID=245, STORAGE=((4),(5))
```

Repeat this macro in the program as many times as wanted, changing the dump identifier for a unique dump. The system writes all the dumps that specify the same DCB to the same data set.

**Example: Two SNAP Dump Requests in a Program:** The following example shows a problem program that requests two SNAP dumps. Both SNAP macros in the example specify the same data control block (DCB) to place both dumps in the same data set. Each dump has a different identifier: PIC3 for the first dump, PIC4 for the second. Both dumps show the same areas: the control blocks. Thus, the programmer can see these areas at two points in the program’s processing.

```
SNAPDCB DCB BLKSIZE=882, DSORG=PS, LRECL=125, MACRF=(W), RECFM=VBA
    .
    .
OPEN SNAPDCB, OUTPUT
LA 3, SNAPDCB
SNAP DCB=(3), ID=PIC3, SDATA=CB
    .
    .
SNAP DCB=(3), ID=PIC4, SDATA=CB
CLOSE SNAPDCB
```

d. Close the DCB with a CLOSE assembler macro.

For more information, see the following references:

- See [z/OS DFSMS Macro Instructions for Data Sets](#) for coding the DCB, OPEN, and CLOSE macros.
- See [z/OS MVS Programming: Assembler Services Reference ABE-HSP](#) and [z/OS MVS Programming: Assembler Services Reference IAR-XCT](#) for required parameters on the DCB macro and for coding the SNAP or SNAPX macro.

3. Print or view the data set. The output of the SNAP or SNAPX macro is a standard EBCDIC data set with ANSI characters in column one. This data set can be edited.

   The dumps are formatted as they are created. Printing depends on the location of the dump when it is created:

   **Location**
   
   **Printing**
   
   **SYSOUt**
   
   The system prints the dump(s) when printing the output class.

   **On a tape or direct access data set**
   
   Print the dump(s) in a separate job or job step.

   **Printer**
   
   The system prints the dump(s) as they are created.

To view SNAP dumps at a terminal, browse the data set containing the dump.
Example: Printing a SNAP Dump

The following JCL prints a SNAP dump. Because the system formats the dump when creating it, the IEBPTPCH utility program can print the dump. The dump is in the SNAPSHOT data set.

```
//PRINT EXEC PGM=IEBPTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SNAPSHOT,UNIT=3350,DISP=(OLD,DELETE),
//   VOLUME=SER=12345
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
//PRINT TYPORG=PS
/*
```

Customizing SNAP dump contents

You can customize the contents of SNAP dumps in one of the following ways:

- Through installation exits.
- Through parameters on the SDUMP or SDUMPX macro.

Customizing through installation exits

An installation can customize the contents of SNAP dumps through the IEAVADFM or IEAVADUS installation exits. IEAVADFM is a list of installation routines to be run and IEAVADUS is one installation routine. The installation exit routine runs during control block formatting of a dump when the CB option is specified on the SNAP or SNAPX macro. The routine can format control blocks and send them to the data set for the dump. See z/OS MVS Installation Exits for more information.

Customizing through the SNAP or SNAPX macro

The parameters on the SNAP or SNAPX macro determine the dump contents. The macro can specify any or all of the areas listed in Table 30.

Note that the parameters cannot request that a Hiperspace be included in the dump. To include Hiperspace data in a SNAP dump, read the data from the Hiperspace into address space storage that is being dumped. See z/OS MVS Programming: Extended Addressability Guide for more information about manipulating data in Hiperspace storage.

Table 30. Customizing dumps using through the SNAP or SNAPX macro

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Dump Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLPA</td>
<td>All link pack areas, as follows:</td>
</tr>
<tr>
<td></td>
<td>• Job pack area (JPA)</td>
</tr>
<tr>
<td></td>
<td>• Link pack area (LPA) active for the task being dumped</td>
</tr>
<tr>
<td></td>
<td>• Related supervisor call (SVC) modules</td>
</tr>
<tr>
<td>ALLVNUC</td>
<td>The entire virtual control program nucleus</td>
</tr>
<tr>
<td>CB</td>
<td>Control blocks for the task being dumped</td>
</tr>
<tr>
<td>DM</td>
<td>Data management control blocks for the task being dumped:</td>
</tr>
<tr>
<td></td>
<td>• Data control block (DCB)</td>
</tr>
<tr>
<td></td>
<td>• Data extent block (DEB)</td>
</tr>
<tr>
<td></td>
<td>• Input/output block (IOB)</td>
</tr>
</tbody>
</table>
Table 30. Customizing dumps using through the SNAP or SNAPX macro (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Dump Contents</th>
</tr>
</thead>
</table>
| ERR       | Recovery termination manager (RTM) control blocks for the task being dumped:  
• Extended error descriptor (EED) for RTM  
• Registers from the system diagnostic work area (SDWA)  
• RTM2 work area (RTM2WA)  
• Set task asynchronous exit (STAE) control block (SCB) |
| IO        | Input/output supervisor (IOS) control blocks for the task being dumped:  
• Execute channel program debug area (EXCPD)  
• Unit control block (UCB) |
| JPA       | Job pack area (JPA): module names and contents |
| LPA       | Link pack area (LPA) active for the task being dumped: module names and contents |
| LSQA      | Local system queue area (LSQA) allocated for the address space (that is, subpools 203 - 205, 213 - 215, 223 - 225, 229, 230, 233 - 235, 249, 253 - 255) |
| NUC       | Read/write portion of the control program nucleus (that is, only non-page-protected areas of the DAT-on nucleus), including:  
• Communication vector table (CVT)  
• Local system queue area (LSQA)  
• Prefixed save area (PSA)  
• System queue area (SQA) |
| PCDATA    | Program call information for the task |
| PSW       | Program status word (PSW) when the dump is requested |
| Q         | Global resource serialization control blocks for the task being dumped:  
• Global queue control blocks  
• Local queue control blocks |
| REGS      | Registers when the dump is requested:  
• Access registers  
• Floating-point registers  
• General registers  
• Vector registers, vector status register, and vector mask register for a task that uses the Vector Facility |
| SA or SAH | Save area linkage information, program call linkage information, and backward trace of save areas |
| SPLS      | Storage allocated in user subpools 0 - 127, 129 -132, 244, 251, and 252 for the task being dumped |
| SQA       | System queue area (SQA) allocated (that is, subpools 226, 239, 245, 247, 248) |
| SUBTASKS  | Storage for the task being dumped and program data for all of its subtasks |
| SWA       | Scheduler work area (SWA) (that is, subpools 236 and 237) |
| TRT       | System trace and generalized trace facility (GTF) trace, as available |
| --        | One or more data spaces identified on the SNAPX macro |
| --        | One or more storage areas, identified by beginning and ending addresses on the SNAP or SNAPX macro |
| --        | One or more subpools, identified by subpool number on the SNAP or SNAPX macro |
SNAP dumps
Chapter 7. The dump grab bag

A dump contains information about an error that can help you identify a problem type. Using interactive problem control system (IPCS), the information about the error is formatted to provide a quick and effective method of retrieval.

The hints that follow apply to processing all kinds of dumps: SVC dumps, stand-alone dumps, and SYSMDUMP dumps. This section covers the following topics:

- “Problem data for storage overlays”
- “Problem data from the linkage stack” on page 164
- “Problem data for modules” on page 165
- “Problem data from recovery work areas” on page 166
- “Problem data for ACR” on page 167
- “Problem data for machine checks” on page 167

Problem data for storage overlays

Always be aware of the possibility of a storage overlay when analyzing a dump. System problems in MVS are often caused by storage overlays that destroy data, control blocks, or executable code. The results of such an overlay vary. For example:

- The system detects an error and issues an abend code, yet the error can be isolated to an address space. Isolating the error is important in discovering whether the overlay is in global or local storage.
- Referencing the data or instructions can cause an immediate error such as a specification exception (abend X'0C4') or operation code exception (abend X'0C1').
- The bad data is used to reference a second location, which then causes another error.

When you recognize that the contents of a storage location are not valid and subsequently recognize the bit pattern as a certain control block or piece of data, you generally can identify the erroneous process/component and start a detailed analysis.

Analyzing the damaged area

Once you determine that storage is bad or overlaid, try to identify the culprit. First, determine the extent of the bad data. Look for EBCDIC data or module addresses in storage to identify the owner. Any type of pattern in storage can indicate an error and identify the program that is using the damaged storage. Look at the data on both sides of the obviously bad areas. See if the length of the bad area is familiar; that is, can you relate the length to a known control block length, data size, MVC length? If so, check various offsets to determine their contents and, if you recognize some, try to determine the exact control block. In Figure 68 on page 164 for example, storage from CSA shows a pattern of allocated blocks.
Even if you do not recognize the pattern, take one more step. Can you determine the offset from some base that would have to be used in order to create the bit pattern? If so, the fact that there is a certain bit pattern at a certain offset can be helpful.

For example, a BALR register value (X'40D21C58') at an offset X'C' can indicate that a program is using this storage for a register save area (perhaps caused by a bad register 13). Another field in the same overlaid area might trigger recognition.

Repetition of a pattern can indicate a bad process. If you can recognize the bad data you might be able to relate that data to the component or module that is causing the error. This provides a starting point for further analysis.

**Common bad addresses**

The following are commonly known as bad addresses. If you recognize these in the code you are diagnosing, focus your problem source identification on these areas:

- X'000C0000', X'040C0000', or X'070C0000', and one of these addresses plus some offset. These are generally the result of some code using 0 as the base register for a control block and subsequently loading a pointer from 0 plus an offset, thereby picking up the first half of a PSW in the PSA.

  Look for storage overlays in code pointed to by an old PSW. These overlays result when 0 plus an offset cause the second half of a PSW to be used as a pointer.

- X'CO00', X'D000', X'D20', X'D28', X'D40', and other pointers to fields in the normal functional recovery routine (FRR) stack. Routines often lose the contents of a register during a SETFRR macro expansion and incorrectly use the address of the 24-byte work area returned from the expansion.

- Register save areas. Storage might be overlaid by code doing a store multiple (STM) instruction with a bad register save area address. In this case, the registers saved are often useful in determining the component or module at fault.

**Problem data from the linkage stack**

The linkage stack is used to identify a program that requested a system service, if the service was entered by a branch instruction.

For example, to see the linkage stack entry that is associated with address space identifier (ASID) X'1A', use the IPCS subcommand:

```
00CFD000 00000000 00000000 05C7E3E3 0800000F  ..........VGTT...
00CFD010 00000000 00000000 00000000 00000000  ..........TQ...
00CFD020 008E03D0 00000000 05C7E3E3 00000000  ..........VGTT...
00CFD030 00000000 00000000 00000000 00000000  ..........S...
00CFD040 00CA6A60 00000000 17000000 05E2C90F  ..........YSI
00CFD050 00CFe01B 00CFD088 CBE2D403 40D43C4A \......).HSML.MC
00CFD060 E24BC4C1 E3C1C440 40040440 40040440 S.DATA
00CFD070 00000000 00000000 00000000 00000000  ..........X'000000010' bytes--All bytes contain X'40', C'
00CFD080 00000020 00000000 00000000 00000000  ..........X'000000019' bytes--All bytes contain X'40', C'
00CFD090 00000000 00000000 00000000 000CA000  ..........        
00CFD0A0 00000000 00000000 00000000 17C90000  ..........        
00CFD0B0 00000000 00000000 00000000 00000000  ..........        
00CFD0C0 00000000 00000000 00000000 00000000  ..........        
00CFD0D0 00000000 00000000 00000000 00000000  ..........        
00CFD0E0 00000000 00000000 00000000 00000000  ..........        
00CFD0F0 00000000 00000000 00000000 00000000  ..........        
00CFD100 00000000 00000000 00000000 00000000  ..........        
00CFD110 00000000 00000000 00000000 00000000  ..........        
00CFD120 00000000 00000000 00000000 00000000  ..........        
00CFD130 00000000 00000000 00000000 00000000  ..........        
00CFD140 00000000 00000000 00000000 00000000  ..........        
00CFD150 00000000 00000000 00000000 00000000  ..........        
```

Figure 68. Example: Recognizing a pattern
The resulting dump for the linkage stack associated with the address space (see Figure 69) shows one entry, as follows:

**BAKR STATE ENTRY**

A Branch and Stack (BAKR) instruction caused this entry.

**SASN..1A and PASN..1A**

At the time of the BAKR, the program was not in cross memory mode. When the branching program is not in cross memory mode, secondary address space number (SASN) and primary address space (PASN) are identical. If the program had been in cross memory mode, SASN and PASN would not have been identical.

**PSW..070C0000 80FD7618**

The return address of the branch caused by the BAKR is FD7618. This address is in the right half of the program status word (PSW).

Many system services are called through branches. For branch entry services, use register 14 to identify the calling program. Look for the problem in the calling program. See z/OS MVS Programming: Extended Addressability Guide for more information about the linkage stack.

**Problem data for modules**

For a module, the system saves and restores status from different locations, depending on the processing mode of the module when it lost control. Use the IPCS STATUS CPU subcommand to find out the mode of the module that had been currently running for each processor. Use the saved status as problem data for diagnosis.

**Processing modes**

The processing modes follow. Code always runs in one or more of these modes. For example, code running in task or service request block (SRB) mode can also be either locally locked or physically disabled.
Task mode is the most common processing mode. All programs given control by ATTACH, ATTACHX, LINK, LINKX, XCTL, and XCTLX macros run in task mode.

SRB mode is code that runs from one of the service request block (SRB) queues.

Physically disabled mode is reserved for high-priority system code that manipulates critical system queues and data areas. This mode is usually combined with supervisor state and key 0 in the PSW. The combination ensures that the routine can complete its function before losing control. The mode is restricted to just a few modules in the system, for example, interrupt handlers, the dispatcher, and programs that are holding a global spin lock.

Locked mode is for code that runs in the system while holding a lock.

Cross memory mode. Cross memory mode is defined by:
- **Primary address space**: Address space identifier (ASID) in control register 3
- **Secondary address space**: ASID in control register 4
- **Home address space**: Address of the address space control block (ASCB) in the PSAAOLD field
- **PSW S-bit** (bit 16 of the PSW): Indicator of current addressability:
  - S-bit=0 - To the primary address space
  - S-bit=1 - To the secondary address space
  When primary addressability and secondary addressability are to the home address space and the S-bit=0, the work is not in cross memory mode.

Access register (AR) mode, where a program can use the full set of assembler instructions (except MVCP and MVCS) to manipulate data in another address space or in a data space. Unlike cross memory, access registers allow full access to data in many address spaces or data spaces.

Problem data from recovery work areas

You can use the recovery work area (RWA) to find the failing module. In most cases, you would use the TCB and RB structure to find the failing module instead of the RWA. Use the RWA in the following situations:

- When an SVC dump is requested in a SLIP trap. In this dump, the current status at the time of the problem is in the recovery save areas or in the SDUMP SQA 4K buffer. See "Reading the SDUMPX 4K SQA buffer" on page 53 for more information.
- When the problem is in the recovery process itself.
- When a stand-alone dump is written because of a suspected loop.

The recovery work areas are:
- Logrec records
- Logrec buffer in the system: obtained by a VERBEXIT LOGDATA subcommand
- System diagnostic work area (SDWA), including the variable recording area (VRA) formatted in logrec records and in the logrec buffer.
- Functional recovery routine (FRR) stacks: described in the next topic.
- Recovery termination manager (RTM) data areas, including the RTM2 work area (RTM2WA): formatted by a SUMMARY FORMAT subcommand or obtained in a formatted ABEND or SNAP dump by the ERR option.

The RTM2WA and SDWA blocks contain registers, PSW, and other time of problem information. Use these blocks in diagnosis when they are associated with a task control block (TCB).
Problem data for ACR

When alternate CPU recovery (ACR) is active at the time of the dump, the search argument in IPCS STATUS WORKSHEET output contains the symptom:

FLDS/CSDACR

Pre-Processing phase data

If ACR is active, problem data for the pre-processing phase are:

- The CSDCPUAL field of the common system data (CSD) indicates which processor failed and which is still running.
- A system trace table entry with ACR in the Ident column indicates that ACR began and identifies the failing processor.
- Use the CSD online mask to determine which CPU’s LCCA to examine. Use the IPCS subcommand CBFORMAT to examine the failing CPU’s LCCA.
- The WSACACR in the CPU work save area vector table (WSAVTC) for both processors’ logical configuration communication areas (LCCA) points to a copy of the PSAs and FRR stacks for both processors.
- The LCCADCPU in both processors points to the LCCA of the failing processor and the LCCARCPU points to the LCCA of the running processor.

Note that a dump shows the PSA of the failed processor when the running processor initiated ACR. The normal FRR stack, pointers to other FRR stacks, locks, PSA super bits, and other data reflect the processor at the time of the failure.

Post-Processing phase data

ACR issues message IEA858E when it completes and resets the CSDACR flag to X’00’.

Data obtained by IPCS

Use the following IPCS subcommand to see all the LCCAs and the CSD:

```
STATUS CPU DATA WORKSHEET
```

Problem data for machine checks

The hardware uses a machine check interruption to tell the control program that it has detected a hardware malfunction. Machine checks vary considerably in their impact on software processing:

- **Soft errors**: Some machine checks notify software that the processor detected and corrected a hardware problem that required no software recovery action.
- **Hard errors**: Other hardware problems detected by a processor require software-initiated action for damage repair. Hard errors also require software recovery to verify the integrity of the process that experienced the failure.
Dump grab bag

The machine check interrupt code (MCIC) in the PSA FLCMCIC field describes the error causing the interrupt. An MCIC can have more than one bit on to indicate more than one failing condition.

For a machine check, the system writes a logrec error record. The error record contains the MCIC, except when:

- The LRBMTCKS bit in field LRBMTERM of the logrec buffer (LRB) is ON to indicate that the machine check old PSW and the MCIC are both zero.
- The LRBMTINV bit in field LRBMTERM is ON to indicate that the machine check old PSW is nonzero but the MCIC is zero.

Hard errors cause FRR and ESTAE processing.

See z/Architecture Principles of Operation for a complete description of the MCIC.
Chapter 8. System trace

System trace provides an ongoing record of hardware events and software events that occur during system initialization and operation. The system activates system tracing at initialization, which runs continuously, unless your installation has changed the IBM-supplied system tracing. After system initialization, you can use the TRACE operator command on a console with master authority to customize system tracing.

System trace writes trace data in system trace tables in the trace address space. System trace maintains a trace table for each processor.

Because system trace usually runs all the time, it is useful for problem determination. While system trace and the generalized trace facility (GTF) lists many of the same system events, system trace also lists events occurring during system initialization, before GTF tracing can be started. System trace also traces branches and cross-memory instructions, which GTF cannot do.

The following topics explain system trace in detail:

- "Customizing system tracing"
- "Receiving system trace data in a dump" on page 171
- "Formatting system trace data in a dump" on page 171
- "Reading system trace output" on page 171.

Customizing system tracing

The system starts system tracing during system initialization and the trace runs continually. There are, however, a few things you can do to alter system tracing:

- "Increasing the size of the system trace table."
- "Tracing branch instructions" on page 170.

Increasing the size of the system trace table

System trace tables reside in fixed storage on each processor. The default trace table size is 1 MB per processor, but you can change it using the TRACE ST command. You might, however, want to increase the size of the system trace table from the default 1 MB when:

- You find that the system trace does not contain tracing from a long enough time period.
- You want to trace branch instructions (using the BR=ON option on the TRACE ST command when you start tracing).

Do the following to increase the size of the trace table:

- Enter the TRACE ST command to change the size of the system trace table. For example, to increase the size of the trace table from the default 1 megabyte per processor to 2 megabytes per processor:

```
TRACE ST,2M
```

- Enter the TRACE ST command with the BUFSIZE (or BUFSIZ) parameter to change the total size of the system trace table allocated. For example, in a system
that has 10 processors, to restart system tracing and increase the size of the trace table from the default of 10 megabytes (1 megabyte per processor) to 20 megabytes (2 megabytes per processor):

TRACE ST,BUFSIZ=20M

**Remember:** Choose a reasonable value for the system trace storage after considering the available central storage and the actual storage required for system trace. Increasing the system trace storage to a large value amount may cause shortage of pageable storage in the system.

**Tracing branch instructions**

System tracing allows you the option of tracing branch instructions, such as BALR, BASR, BASSM and BAKR, along with other system events. The mode tracing option is separate from branch tracing.

**Attention:** Branch tracing ON can affect your system performance and use very large amounts of storage. Do not use branch tracing as the default for system tracing on your system. Only use it for short periods of time to solve a specific problem. The default system tracing does not include branch instructions.

IBM provides two health checks, SYSTRACE_BRANCH and SYSTRACE_MODE, to help identify when branch tracing may be affecting your system performance. For example, if branch tracing has been set on for a long time, you will receive message IEAH801E. For more information, see System trace checks (IBMSYSTRACE) in IBM Health Checker for z/OS User’s Guide. For information about related messages, see z/OS MVS System Messages, Vol 6 (GOS-IEA).

When you want to trace branch instructions such as BALR, BASR, BASSM and BAKR, do the following:

- Turn on system tracing with branch tracing using the TRACE command from a console with master authority:
  
  TRACE ST,BR=ON

  Because tracing branch instructions tends to significantly increase the number of trace entries being generated, you can increase the size of the trace tables when you turn tracing on.

  - To increase the size of the trace tables for each processor from the default 1 MB to 2 MB, issue:
    
    TRACE ST,ZM,BR=ON

  - To increase the size of total storage for trace buffers (that is, the sum of the storage set aside for trace table entries on all the installed processors) to 2 megabytes:
    
    TRACE ST,BUFSIZ=2M,BR=ON

For more information, see the following documentation:

- z/Architecture Principles of Operation describes the branch instruction trace entries and the mode trace entries that MVS combines with them (and are generated by the hardware). MVS enables or disables the production of these entries by manipulating control register bits. The trace table entries that are not 'branch (or mode)' entries that are generated by MVS software through the TRACE or TRACG instructions. See the Tracing topic for information.

- For a description of the TTE from mapping macro IHA TTE, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/)
Receiving system trace data in a dump

System trace writes trace data in system trace tables in the trace address space. System trace maintains a trace table for each processor. Obtain the trace data in a dump that included option SDATA=TRT. Table 31 shows the dumps that have TRT in their default options and how to request trace data for dumps that do not include the data by default.

System trace data that are requested by RTM and ABDUMPs will only receive the most current 64K of data for each CPU when the number of concurrent snapshots could affect system availability. This is referred to as a mini-trace table snapshot.

For more information about working with system trace tables, see the SYSTRACE section in z/OS MVS IPCS Commands.

Table 31. Dumps that have TRT in their default options

<table>
<thead>
<tr>
<th>Dump</th>
<th>How to obtain trace data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABEND dump to SYSABEND</td>
<td>Default</td>
</tr>
<tr>
<td>ABEND dump to SYSMDUMP</td>
<td>Default</td>
</tr>
<tr>
<td>ABEND dump to SYSUDUMP</td>
<td>Default</td>
</tr>
<tr>
<td>SNAP dump</td>
<td>Request SDATA=TRT</td>
</tr>
<tr>
<td>Stand-alone dump</td>
<td>Default</td>
</tr>
<tr>
<td>SVC dump for SDUMP or SDUMPX macro</td>
<td>Default</td>
</tr>
<tr>
<td>SVC dump for DUMP operator command</td>
<td>Default</td>
</tr>
<tr>
<td>SVC dump for SLIP operator command with ACTION=SVCD, ACTION=STDUMP, ACTION=SYNCSVCD, or ACTION=TRDUMP</td>
<td>Default</td>
</tr>
<tr>
<td>Any dump customized to exclude trace data</td>
<td>Request SDATA=TRT</td>
</tr>
</tbody>
</table>

Formatting system trace data in a dump

- For formatted dumps, system trace formats the system trace data and the system prints it directly.
- For unformatted dumps, use the IPCS SYSTRACE subcommand to format and print or view the trace data in the dump.

Reading system trace output

The following topics describe system trace table entries (TTE) as they appear in a dump formatted with the IPCS SYSTRACE subcommand.

- “Example of a system trace in a dump” on page 172
- “Summary of system trace entry identifiers” on page 172 shows a table of the system trace identifiers for each system trace entry in a dump and shows where you can find the format of the entry in this section. If you are looking for a particular entry start with this table, because many of the entries are similar and are grouped together.
- “ACR trace entries” on page 174 through “USRn trace entries” on page 218 shows the format for each type of trace entry. For the detailed format of TTEs, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).
Example of a system trace in a dump

Figure 70 shows system trace entries. IPCS formatted the entries from an example SVC dump. Note that system trace data in an ABEND dump has the same format.

The subcommand issued from the IPCS Subcommand Entry panel was:

SYSTRACE TIME(LOCAL)

The oldest trace entries appear first in the trace; the newest entries are at the end.

An asterisk (*) before an identifier indicates an unusual condition; see the format of the entry for an explanation.

Summary of system trace entry identifiers

This topic summarizes all the system trace entries by identifier. Because many trace entries are similar, they are described together. Use Table 32 on page 173 to locate the format for a particular entry.

For example, in the trace entry shown in Figure 71 on page 173, the system trace identifier is SVC. Look up SVC in Table 32 on page 173 to find where the SVC identifier is located.
trace entry format is described. In this case, the SVC trace entry is described in "SVC, SVCE, and SVCR trace entries" on page 215.

Table 32. References for system trace entry format description

<table>
<thead>
<tr>
<th>Identifier (Iden)</th>
<th>Description</th>
<th>For format, see:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACR</td>
<td>Alternate CPU recovery</td>
<td>“ACR trace entries” on page 174</td>
</tr>
<tr>
<td>AINT</td>
<td>Adapter interruption</td>
<td>“AINT trace entries” on page 176</td>
</tr>
<tr>
<td>ALTR</td>
<td>Alteration of trace option</td>
<td>“ALTR trace entries” on page 177</td>
</tr>
<tr>
<td>BR</td>
<td>Branch through a BAKR, BALR, BASR, or BASSM instruction</td>
<td>“BR trace entries” on page 178</td>
</tr>
<tr>
<td>BSG</td>
<td>Branch on subspace group</td>
<td>“BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries” on page 179</td>
</tr>
<tr>
<td>CSCH</td>
<td>Clear subchannel operation</td>
<td>“CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries” on page 184</td>
</tr>
<tr>
<td>DSP</td>
<td>Task dispatch</td>
<td>“DSP, SRB, SSRB, and WAIT trace entries” on page 186</td>
</tr>
<tr>
<td>EXT</td>
<td>General external interruptions: • CALL - external call interruption • CLKC - clock comparator interruption • EMS - emergency signal interruption • SS - service signal interruption • TIMR - timer interruption • WTI - warning track interruption</td>
<td>“CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries” on page 181</td>
</tr>
<tr>
<td>HSCH</td>
<td>Halt subchannel operation</td>
<td>“CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries” on page 184</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/output interruption</td>
<td>“CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries” on page 181</td>
</tr>
<tr>
<td>MCH</td>
<td>Machine check interruption</td>
<td>“CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries” on page 181</td>
</tr>
<tr>
<td>MOBR</td>
<td>Change of addressing mode along with a change of instruction address</td>
<td>“MODE and MOBR trace entries” on page 188</td>
</tr>
<tr>
<td>MODE</td>
<td>Change of addressing mode</td>
<td>“MODE and MOBR trace entries” on page 188</td>
</tr>
<tr>
<td>MSCH</td>
<td>Modify subchannel operation</td>
<td>“CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries” on page 184</td>
</tr>
<tr>
<td>PC</td>
<td>Program Call control instruction</td>
<td>“BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries” on page 179</td>
</tr>
<tr>
<td>PCIL</td>
<td>PCI load instruction</td>
<td>“PCIL trace entries” on page 189</td>
</tr>
<tr>
<td>PCIS</td>
<td>PCI store instruction</td>
<td>“PCIS trace entries” on page 190</td>
</tr>
<tr>
<td>PDMX</td>
<td>PCIE adapter interruption de-multiplexing event</td>
<td>“PDMX trace entries” on page 191</td>
</tr>
<tr>
<td>PGM</td>
<td>Program interruption</td>
<td>“PGM, SPER and SPR2 trace entries” on page 192</td>
</tr>
<tr>
<td>PR</td>
<td>Program Return control instruction</td>
<td>“BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries” on page 179</td>
</tr>
</tbody>
</table>
### System trace

**Table 32. References for system trace entry format description (continued)**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
<th>For format, see:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT</td>
<td>Program Transfer control instruction</td>
<td>“BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries” on page 179</td>
</tr>
<tr>
<td>RCVY</td>
<td>Recovery event</td>
<td>“RCVY trace entries” on page 194</td>
</tr>
<tr>
<td>RSCH</td>
<td>Resume subchannel operation</td>
<td>“CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries” on page 184</td>
</tr>
<tr>
<td>RST</td>
<td>Restart interruption</td>
<td>“CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries” on page 181</td>
</tr>
<tr>
<td>SIGA</td>
<td>Signal adapter operation</td>
<td>“CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries” on page 184</td>
</tr>
<tr>
<td>SPER</td>
<td>SLIP program event recording</td>
<td>“PGM, SPER and SPR2 trace entries” on page 192</td>
</tr>
<tr>
<td>SPIN</td>
<td>Starting, middle, or stopping of a system spin.</td>
<td>“SPIN trace entries” on page 199</td>
</tr>
<tr>
<td>SPR2</td>
<td>SLIP program event recording, when STDATA is specified</td>
<td>“PGM, SPER and SPR2 trace entries” on page 192</td>
</tr>
<tr>
<td>SRB</td>
<td>Initial service request block dispatch</td>
<td>“DSP, SRB, SSRB, and WAIT trace entries” on page 186</td>
</tr>
<tr>
<td>SSAR</td>
<td>Set Secondary Address Space Number control instruction</td>
<td>“BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries” on page 179</td>
</tr>
<tr>
<td>SSCH</td>
<td>Start subchannel operation</td>
<td>“CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries” on page 184</td>
</tr>
<tr>
<td>SSRB</td>
<td>Suspended service request block dispatch</td>
<td>“DSP, SRB, SSRB, and WAIT trace entries” on page 186</td>
</tr>
<tr>
<td>SSRV</td>
<td>System service entered by a Program Call (PC) instruction or a branch</td>
<td>“SSRV trace entries” on page 204</td>
</tr>
<tr>
<td>SUSP</td>
<td>Lock suspension</td>
<td>“SUSP trace entries” on page 214</td>
</tr>
<tr>
<td>SVC</td>
<td>Supervisor call interruption</td>
<td>“SVC, SVCE, and SVCR trace entries” on page 215</td>
</tr>
<tr>
<td>SVCE</td>
<td>SVC error</td>
<td>“SVC, SVCE, and SVCR trace entries” on page 215</td>
</tr>
<tr>
<td>SVCR</td>
<td>SVC return</td>
<td>“SVC, SVCE, and SVCR trace entries” on page 215</td>
</tr>
<tr>
<td>TIME</td>
<td>Timer services</td>
<td>“TIME trace entries” on page 217</td>
</tr>
<tr>
<td>USRn</td>
<td>User event</td>
<td>“USRn trace entries” on page 218</td>
</tr>
<tr>
<td>WAIT</td>
<td>Wait task dispatch</td>
<td>“DSP, SRB, SSRB, and WAIT trace entries” on page 186</td>
</tr>
<tr>
<td>XSCH</td>
<td>Cancel subchannel operation</td>
<td>“CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries” on page 184</td>
</tr>
<tr>
<td>?EXPL</td>
<td>The SYSTRACE subcommand cannot identify the system trace entry</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**ACR trace entries**

An ACR trace entry represents failure of a processor and subsequent entry into the alternate CPU recovery component.
<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>Unique-4</th>
<th>Unique-5</th>
<th>Unique-6</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>fail</td>
<td>wu-addr-</td>
<td>+ACR</td>
<td>cpu</td>
<td>psaeepsw</td>
<td>flg-crex</td>
<td>pсаstк-</td>
<td>pсаlocal-</td>
<td>psaslс-</td>
<td>psaslс-</td>
<td>psaslс-</td>
<td>psaslс-</td>
<td>psaslс-</td>
<td>psaslс-</td>
<td>psaslс-</td>
<td>timestamp-</td>
<td>CP</td>
<td></td>
</tr>
</tbody>
</table>

**PR**
- pr: Identifier of the processor that produced the TTE.

**ASID**
- fail: Home address space identifier (ASID) of the failing processor

**WU-Addr**
- wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**
- The TTE identifier, as follows:
  - ACR: Alternate CPU recovery.

**CD/D**
- cpu: The failing processor address from the PSACPUPA field of the PSA

**PSW----- Address-**
- Blank

**Unique-1/Unique-2/Unique-3**

**Unique-4/Unique-5/Unique-6**
- flg-crex: LCCACREX field of the logical configuration communication area (LCCA) for the failing processor.
- pсаstк-: PSACSTK field from the prefix save area (PSA) from the failing processor.
- psaeepsw: PSAEEPSW field in the PSA. Bytes 1 and 2 contain the failing processor’s address. Bytes 3 and 4 contain the external interruption code.
- psamodew: PSAMODEW field in the PSA
- psasuper: PSASUPER field in the PSA

**PSACLHS-**
- psaclhs-: String for the current lock held, from the PSACLHS field of the PSA from the failing processor.

**PSACLHSE**
- psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA from the failing processor.

**PSALOCAL**
- psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA from the failing processor.

**PASD**
- pasd: Primary ASID (PASID) at trace entry.

**SASD**
- sasd: Secondary ASID (SASID) at trace entry.

**Time Format**
- timestamp----------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

**CP**
- The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace
System trace

entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**AINT trace entries**

An AINT trace entry represents an adapter interruption.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW------</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format-------</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>AINT</td>
<td>aism</td>
<td>i/o-old-pswaddr</td>
<td>isc</td>
<td>desc</td>
<td>desc</td>
<td>desc</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR**

pr: Identifier of the processor that produced the TTE.

**ASID**

home: Home address space identifier (ASID) associated with the TTE.

**WU-Addr-**

wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**

The TTE identifier, as follows:

AINT Adapter interruption.

**CD/D**

aism: The adapter interruption source mask.

**PSW------**

i/o-old-pswaddr / i/o-old-pswcntl: I/O old PSW.

**Unique-1/Unique-2/Unique-3**

**Unique-4/Unique-5/Unique-6**

- isc: Interruption subclass.
- desc: Description of the adapter types represented by the adapter interruption source mask (IQP, PCIE, Crypto).

**PSACLHS-**

psaclhs-: String for the current lock held, from the PSACLHS field of the PSA.

**PSACLHSE**

psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA.

**PSALOCAL**

psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA.

**PASD**

pasd: Primary ASID (PASID) at trace entry.

**SASD**

sasd: Secondary ASID (SASID) at trace entry.

**Time Format**

timestamp--------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

**CP**

The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace
**System trace**

entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**ALTR trace entries**

An ALTR trace entry represents alteration of the system trace options. Alter the options with a TRACE ST operator command.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALocal</th>
<th>PASD</th>
<th>SASD</th>
<th>Time</th>
<th>Format-</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>*ALTR</td>
<td>tobtropt</td>
<td>gpr0----</td>
<td>gpr1----</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR**
pr: Identifier of the processor that produced the TTE.

**ASID**
home: Home address space identifier (ASID) associated with the TTE.

**WU-Addr-**
wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**
The TTE identifier, as follows:

**ALTR** Alteration of the trace option. An asterisk (*) always appears before ALTR to indicate an unusual condition.

**CD/D**
Blank

**PSW----- Address-**
Blank

**Unique-1/Unique-2/Unique-3**

**Unique-4/Unique-5/Unique-6**

- tobtropt: Trace options in control register 12 format, from the TOBTROPT field of the system trace option block (TOB)
- gpr0----: General register 0
- gpr1----: General register 1
- pol-: The number of processor with tracing active or suspended, from the TOBTRPOL field of the TOB
- buf-: The number of trace buffers per processor, from the TOBTRBUF field of the TOB

**PSACLHS-**
Blank

**PSACLHSE**
Blank

**PSALocal**
Blank

**PASD**
pasd: Primary ASID (PASID) at trace entry.

**SASD**
sasd: Secondary ASID (SASID) at trace entry.
System trace

**Time Format**

- **timestamp---------**: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

- **CP**: The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**BR trace entries**

A BR trace entry represents processing of a Branch and Link (BALR), Branch and Save (BASR), Branch and Save and Set Mode (BASSM), or Branch and Stack (BAKR) instruction, when the $R_2$ field in the instruction is not zero. These branches are traced only when a TRACE operator command requests branch tracing by BR=ON.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSM------</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALocal</th>
<th>PASD</th>
<th>Sasd</th>
<th>Time</th>
<th>Format</th>
<th>-------</th>
<th>CP</th>
<th>Unique-4</th>
<th>Unique-5</th>
<th>Unique-6</th>
<th>PSACLHSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>last</td>
<td>wu-addr-</td>
<td>BR</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **PR**: Identifier of the processor that produced the TTE.
- **ASID**: 
  - **last**: Last home address space identifier (ASID) in the trace buffer.
- **WU-Addr-**: Address of the task control block (TCB) for the current task or the work element block (WEB).
- **Ident**: The TTE identifier, as follows:
  - **BR**: Branch instruction

- **CD/D**: Blank

- **PSM------**: Address- 
  - **Unique-1/Unique-2/Unique-3**
  - **Unique-4/Unique-5/Unique-6**
  - **PSACLHS-**
  - **PSACLHSE**
  - **PSALocal**
  - **PASD**
  - **SASD**

- **address-**: Successful branch address, repeated for consecutive branches on the BR entry. Addresses appear in the following formats:

<table>
<thead>
<tr>
<th>Addressing mode and location</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-bit address</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>31-bit address</td>
<td>xxxxxxxxxx</td>
</tr>
<tr>
<td>64-bit address with zeros in high order bits</td>
<td>00_xxxxxxxx</td>
</tr>
<tr>
<td>64-bit address with non-zero high order bits</td>
<td>xxxxxxxx_xxxxxxxx</td>
</tr>
</tbody>
</table>
BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries

These trace entries represent processing of a cross memory instruction:

- A BSG trace entry represents a Branch on Subspace group (BSG) control instruction
- A PC trace entry represents a Program Call (PC) control instruction
- A PR trace entry represents a Program Return (PR) control instruction
- A PT trace entry represents a Program Transfer (PT) control instruction
- A PTI trace entry represents a Program Transfer with Instance (PTI) control instruction
- An SSAR trace entry represents a Set Second Address Space Number (SSAR) control instruction
- An SSIR trace entry represents a Set Secondary Address Space Number with Instance (SSAIR) control instruction

---

**PR**

pr: Identifier of the processor that produced the TTE.

**ASID**

last: Last home address space identifier (ASID) associated with the TTE.

**WU-Addr-**

wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**

The TTE identifier, as follows:

- **PC** Program Call control instruction
- **PR** Program Return control instruction
- **PT** Program Transfer control instruction
- **PTI** Program Transfer with Instance (PTI) control instruction
- **SSAR** Set Secondary Address Space Number control instruction
- **SSIR** Set Secondary Address Space Number with Instance (SSAIR) control instruction
- **BSG** Branch on Subspace Group control instruction
System trace

CD/D
Blank

PSW----- Address-

- alet: ALET word during BSG execution
- newsasid: New SASID from the SSAR instruction
- return--: Caller’s return address
- psw-key--: Program status word (PSW) key
- pkey-flag: Program status word (PSW) key and flags. The flag value is either blank or a hexadecimal value of 1-3:
  - 0 - PSW bit 31 was replaced by a zero and PSW bit 31 was a zero before being replaced.
  - 1 - PSW bit 31 was replaced by a one and PSW bit 31 was a zero before being replaced.
  - 2 - PSW bit 31 was replaced by a zero and PSW bit 31 was a one before being replaced.
  - 3 - PSW bit 31 was replaced by a one and PSW bit 31 was a one before being replaced.
- pc-addr--: Return address from the PC instruction
- pr-addr--: New instruction address as updated by the PR instruction
- pt-addr--: New instruction address as updated by the PT instruction
- bsg-addr: New instruction address as updated by the BSG instruction

Addresses appear in the following formats:

<table>
<thead>
<tr>
<th>Addressing mode and location</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-bit address</td>
<td>xxxxxxx</td>
</tr>
<tr>
<td>31-bit address</td>
<td>xxxxxxxx</td>
</tr>
<tr>
<td>64-bit address with zeros in high order bits</td>
<td>00_xxxxxxxx</td>
</tr>
<tr>
<td>64-bit address with non-zero high order bits</td>
<td>xxxxxxxx_xxxxxxx</td>
</tr>
</tbody>
</table>

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- pc#-----: PC number from the PC instruction
- pr-faddr: Address of the location following the PR instruction
- pt-asid--: New ASID specified on the PT instruction

PSACLHS-
This field will contain descriptive text for some PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PSACLHSE
Blank

PSALOCAL
This field will contain descriptive text for some PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PASD
pasd: Primary ASID (PASID) at trace entry. This field will contain descriptive text for some PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.
SASD
sasd: Secondary ASID (SASID) at trace entry. This field will contain descriptive text for some PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

**Time Format**
Blank

**CP**
Blank

**CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries**
The following trace entries represent an interruption:

- These entries represent external interruptions:
  - A CALL trace entry is for an external call
  - A CLKC trace entry is for a clock comparator
  - An EMS trace entry is for an emergency signal
  - An EXT trace entry is for a general external interruption
  - An SS trace entry is for a service signal
  - A TIMR trace entry is for a timer interruption
  - A WTI trace entry is for a warning track interruption
  - A Key entry for an interrupt key interruption
  - A TimA entry for a timing alert interruption
- An I/O trace entry is for an I/O interruption
- An MCH trace entry is for a machine check
- An RST trace entry is for a restart

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr</th>
<th>Ident</th>
<th>CD/O</th>
<th>PSW-------</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>Unique-4</th>
<th>Unique-5</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format</th>
<th>CP</th>
<th>PSACLHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>EXT</td>
<td>CALL</td>
<td>ext-old-</td>
<td>psw------</td>
<td>psaeepsw</td>
<td>pccarph-</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>EXT</td>
<td>CLKC</td>
<td>ext-old-</td>
<td>pswaddr-</td>
<td>psaepsw</td>
<td>tqe-addr</td>
<td>tqe-asid</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>EXT</td>
<td>EMS</td>
<td>ext-old-</td>
<td>pswaddr-</td>
<td>psaepsw</td>
<td>pccaemsi</td>
<td>pccaemsp</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>EXT</td>
<td>code</td>
<td>ext-old-</td>
<td>pswaddr-</td>
<td>psaepsw</td>
<td>pccaemse</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>I/O</td>
<td>dev i/o-old</td>
<td>pswaddr-</td>
<td>flg-ctl-</td>
<td>ccw-addr</td>
<td>dvch-cnt</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>MCH</td>
<td>mch-old-</td>
<td>pswaddr-</td>
<td>machine-</td>
<td>chk-code</td>
<td>pssuper</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>RST</td>
<td>rst-old-</td>
<td>pswaddr-</td>
<td>gpr15---</td>
<td>gpr0----</td>
<td>gpr1----</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>EXT</td>
<td>SS</td>
<td>ext-old-</td>
<td>pswaddr-</td>
<td>psaepsw</td>
<td>pseparm</td>
<td>msf-bcmd</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>EXT</td>
<td>TIMR</td>
<td>ext-old-</td>
<td>psw------</td>
<td>psaeepsw</td>
<td>pccarph-</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>EXT</td>
<td>WTI</td>
<td>ext-old-</td>
<td>psw------</td>
<td>psaeepsw</td>
<td>pccarph-</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td>pr</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR** pr: Identifier of the processor that produced the TTE.
System trace

**ASID**
home: Home address space identifier (ASID) associated with the TTE.

**WU-Addr**
wu-addr: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**
The TTE identifier, as follows:

- **EXT** General external interruption. An asterisk before EXT indicates that the interrupt is a malfunction alert (MFA) or is the result of pressing the External Interrupt key.
- **I/O** I/O interruption. An asterisk before I/O indicates that one of the following bits in IRBFLAGS field of the interrupt request block (IRB) is ON. The IRBFLAGS field is in the Unique-1 column of the I/O entry.
  - IRBN for path not operational
  - IRBSALRT for alert status
- **MCH** Machine check interruption.
- **RST** Restart interruption.

**CD/D**
- **CALL** External call external interruption
- **CLCK** Clock comparator external interruption
- **EMS** Emergency signal external interruption
- **SS** Service signal external interruption
- **TIMR** Timer interruption
- **WTI** Warning track interruption

**code** External interruption code

**dev** Device number associated with the I/O or, for a co-processor device, the I/O co-processor identifier, for example, ADM

**PSW----- Address-----**
The z/Architecture 128-bit PSW address appears on two lines:
- **pswaddr**: Two words, containing the 64-bit address portion of the PSW
- **pswctrl**: Two words, containing the 64-bit “control” portion of the PSW

The PSW represents different data, depending on the type of interrupt that was traced:
- **ext-old** pswaddr / **ext-old** pswctrl: External old program status word (PSW)
- **i/o-old** pswaddr / **i/o-old** pswctrl: I/O old PSW
- **mch-old** pswaddr / **mch-old** pswctrl: Machine check old PSW
- **rst-old** pswaddr / **rst-old** pswctrl: Restart old PSW

**Unique-1/Unique-2/Unique-3**
**Unique-4/Unique-5/Unique-6**
- **ccw-addr**: Address of the channel command word (CCW) for the I/O
- **-cnt**: Residual count
- **dvch**: Device status and subchannel status
- **ext-stat**: Extended status word
**System trace**

- `flg-brsp`: Maintenance and service support facility (MSSF) hardware flags and MSSF response code
- `flg-ctl-`: IRBFLAGS field in the IRB and the subchannel control bytes
- `gpr15--- gpr0---- gpr1----`: General registers 15, 0, and 1
- `machine- chk-code`: Machine check interruption code from the FLCMCIC filed in the prefix save area (PSA)
- `msf-bcmd`: Service processor command word
- `mssfasid`: Service processor address space ID
- `mssfatchb`: Service processor TCB address
- `pccaeemse`: PCCAEMSE field from the physical configuration communication area (PCCA)
- `pccaeensi`: PCCAEMSI field from the PCCA
- `pccaeemsp`: PCCAEMSP field from the PCCA
- `pccarphp`: PCCARPB field from the PCCA
- `psaeepsw`: PSAEEPSW field in the PSA. For CALL and EMS, bytes 1 and 2 contain the issuing processor's address. For all entries, bytes 3 and 4 contain the external interruption code.
- `psaeparm`: PSAEPARM field in the PSA, containing the MSSF buffer address
- `psamode`: PSAMODE field in the PSA
- `psasuper`: PSASUPER field in the PSA
- `tqe-asid`: ASID of the associated timer queue element (TQE)
- `tqe-addr`: TQE address
- `ucb-addr`: Unit control block (UCB) address

**PSACLHS-**
- `psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

**PSACLHSE**
- `psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

**PSALOCAL**
- `psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

**PASD**
- `pasd`: Primary ASID (PASID) at trace entry.

**SASD**
- `sasd`: Secondary ASID (SASID) at trace entry.

**Time Format**
- `timestamp--------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

**CP**
- The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.
System trace

CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries

These trace entries represent an input/output operation:
- A CSCH trace entry represents a clear subchannel operation
- An HSCH trace entry represents a halt subchannel operation
- An MSCH trace entry represents a modify subchannel operation
- An RSCH trace entry represents a resume subchannel operation
- An SSCH trace entry represents a start subchannel operation
- An SIGA trace entry represents a signal adapter operation
- An XSCH trace entry represents a cancel subchannel operation

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format------</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>asid</td>
<td>wu-addr-</td>
<td>CSCH</td>
<td>dev</td>
<td>cc</td>
<td>di</td>
<td>iosbaddr</td>
<td>ucb-addr</td>
<td>iqq-addr</td>
<td>asc-iosb</td>
<td>timestamp</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>asid</td>
<td>wu-addr-</td>
<td>HSCH</td>
<td>dev</td>
<td>cc</td>
<td>di</td>
<td>iosbaddr</td>
<td>ucb-addr</td>
<td>iqq-addr</td>
<td>asc-iosb</td>
<td>timestamp</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>asid</td>
<td>wu-addr-</td>
<td>MSCH</td>
<td>dev</td>
<td>cc</td>
<td>di</td>
<td>iosbaddr</td>
<td>ucb-addr</td>
<td>f1f2pmom</td>
<td>mbi-t2lb</td>
<td>timestamp</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>asid</td>
<td>wu-addr-</td>
<td>RSCH</td>
<td>dev</td>
<td>cc</td>
<td>di</td>
<td>iosbaddr</td>
<td>ucb-addr</td>
<td></td>
<td></td>
<td></td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>asid</td>
<td>wu-addr-</td>
<td>SSCH</td>
<td>dev</td>
<td>cc</td>
<td>di</td>
<td>iosbaddr</td>
<td>ucb-addr</td>
<td>orb-wrd2</td>
<td>orb-wrd3</td>
<td>orb-wrd4</td>
<td>cap-addr</td>
<td>bdev</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>asid</td>
<td>wu-addr-</td>
<td>XSCH</td>
<td>dev</td>
<td>cc</td>
<td>di</td>
<td>iosbaddr</td>
<td>ucb-addr</td>
<td>iqq-addr</td>
<td>bdev</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>asid</td>
<td>wu-addr-</td>
<td>SIGA</td>
<td>dev</td>
<td>cc</td>
<td>fc</td>
<td>qib-addr</td>
<td>subsysid</td>
<td>q-mask-1</td>
<td>q-mask-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR**: Identifier of the processor that produced the TTE.

**ASID**: Address space identifier (ASID) related to the I/O.

**WU-Addr-**: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**: The TTE identifier, as follows:
- **CSCH**: Clear subchannel operation
- **HSCH**: Halt subchannel operation
- **MSCH**: Modify subchannel operation
- **RSCH**: Resume subchannel operation
- **SSCH**: Start subchannel operation
- **XSCH**: Cancel subchannel operation
- **SIGA**: Signal adapter operation

An asterisk before RSCH, SSCH, or SIGA indicates that the condition code associated with the I/O was not 0.

**CD/D**: dev: One of the following:
The device number associated with the I/O, which will include the subchannel set identifier when appropriate.

• ADMF, if the IOSADMF macro was transferring data

PSW----- Address-

• cc: Condition code in bits 2 and 3 associated with the I/O
• di: Driver identifier associated with the I/O
• fc: Function code associated with the I/O
• iosbaddr: I/O supervisor block (IOSB) address associated with the I/O
• qib-addr: Queue identification block (QIB) address associated with the I/O

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

• asc-iosb: IOSB address for the associated SSCH request for the I/O
• bdev: The base device number if the I/O is associated with an alias device.
• cap-addr: Captured unit control block (UCB) address associated with the SSCH I/O. This field is blank if a below 16 megabyte UCB or actual above 16 megabyte UCB address was used for the start subchannel (SSCH) operation. The address of the actual above 16 megabyte UCB is in the ucb-addr field.
• f1f2pmom: From the subchannel information block (SCHIB) associated with the I/O, as follows:
  f1 SCHFLG1 flag field
  f2 SCHFLG2 flag field
  pm SCHLPM field
  om SCHPOM field
• iq-addr: I/O queue (IOQ) address associated with the I/O
• mbi-t2lb:
  mbi- SCHMBI field from the SCHIB
  t2 IOSOPT2 field from the IOSB
  lb IOSFLB field from the IOSB
• orb-wrd2: Word 2 of the operation request block (ORB) associated with the I/O
• orb-wrd3: Word 3 of the operation request block (ORB) associated with the I/O
• orb-wrd4: Word 4 of the operation request block (ORB) associated with the I/O
• q-mask-1: Read or write queue mask associated with the I/O
• q-mask-2: Read queue mask associated with the I/O
• subsysid: Subsystem ID associated with the I/O
• ucb-addr: Unit control block (UCB) address associated with the I/O

PSACLHSE-
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSAEND output.

PSACLHSE
Blank
System trace

**PSALOCAL**
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

**PASD**
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

**SASD**
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

**Time Format**

| timestamp---------: Time-of-day (TOD) clock value when system trace created |
| the trace entry. The value is in the format that was specified for TIME. If TIME |
| was not specified, the default format is TIME(HEX). |

**CP**
The CP column contains 2 hex digits of the processor model-dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**DSP, SRB, SSRB, and WAIT trace entries**
These trace entries represent the dispatch of a unit of work:

- A DSP trace entry represents dispatch of a task
- An SRB trace entry represents the initial dispatch of a service request
- An SSRB trace entry represents dispatch of a suspended service request
- A WAIT trace entry represents dispatch of the wait task

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time</th>
<th>Format</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>DSP</td>
<td>dspt-new</td>
<td>pswaddr-</td>
<td>psamodew</td>
<td>gpr0----</td>
<td>gpr1----</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>SRB</td>
<td>srb-new</td>
<td>pswaddr-</td>
<td>safnasid</td>
<td>gpr0----</td>
<td>gpr1----</td>
<td>srbhlhi-</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>SSRB</td>
<td>ssrb-new</td>
<td>pswaddr-</td>
<td>safnasid</td>
<td>gpr1----</td>
<td>psaclhs-4</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>WAIT</td>
<td>wu-addr-</td>
<td>pswctrl-</td>
<td>purgetcb</td>
<td>flg-srb-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR**
pr: Identifier of the processor that produced the TTE.

**ASID**
home: Home address space identifier (ASID) associated with the TTE.

**WU-Addr-**
wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**
The TTE identifier, as follows:

- **DSP** Task dispatch
- **SRB** Initial service request dispatch
SSRB  Suspended service request dispatch
WAIT  Wait task dispatch
CD/D  Blank

PSW----- Address-
The z/Architecture 128-bit PSW address appears on two lines:
  * pswaddr: Two words, containing the 64-bit address portion of the PSW
  * pswctrl: Two words, containing the 64-bit “control” portion of the PSW
The PSW represents different data, depending on the type of work that was dispatched:
  * dsp-new- pswaddr / dsp-new- pswctrl: Program status word (PSW) to be dispatched
  * srb-new- pswaddr / srb-new- pswctrl: PSW to receive control on the SRB dispatch
  * ssrb-new pswaddr / ssrb-new pswctrl: PSW to receive control on the SSRB redispach

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
  * gpr0----: General register 0
  * gpr1----: General register 1
  * psamodew: PSAMODEW field in the PSA
  * safnasid: LCCASAFN field in the logical configuration communication area (LCCA) and the related ASID
  * flg-srb: SRBFLGS field from the SRB
  * purgetcb: TCB (located in address space of the scheduler of the SRB or SSRB) that gets control if the SRB or SSRB abends and percolates

PSACLHSS-
  * psaclhs-: String for the current lock held, from the PSACLHSS field of the PSA.
  * psaclhs4: PSACLHSS4 field of the PSA
  * srbh1hi-: SRBHLHI field in the SRB
This field will contain descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PSACLHSE
  Blank

PSALOCAL
psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA. This field will contain descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PASD
pasd: Primary ASID (PASID) at trace entry. This field will contain descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

SASD
sasd: Secondary ASID (SASID) at trace entry. This field will contain descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.
System trace

Text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

**Time Format**

*timestamp*---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

**CP**
The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**MODE and MOBR trace entries**

These trace entries represent a change of addressing mode:

- A MODE trace entry represents a change into or out of 64-bit addressing mode
- A MOBR trace entry represents a change into or out of 64-bit addressing mode along with a change of instruction address

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format---------</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>last wu-addr-</td>
<td>MODE</td>
<td>target</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>last wu-addr-</td>
<td>MOBR</td>
<td>target</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>address-</td>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR**

*pr*: Identifier of the processor that produced the TTE.

**ASID**

*last*: Last home address space identifier (ASID) in the trace buffer.

**WU-Addr-**

*wu-addr-*: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**
The TTE identifier, as follows:

- **MODE**
  - Addressing mode change instruction

- **MOBR**
  - Addressing mode change combined with a branch instruction

**CD/D**
Blank

**PSW-----**

*target*: Target addressing mode.

- **24 OR 31**
  - Target addressing mode is either 24-bit or 31-bit.

- **64**
  - Target addressing mode is either 64-bit.

**Address-**

- **Unique-1/Unique-2/Unique-3**
- **Unique-4/Unique-5/Unique-6**
- **PSACLHS-**
- **PSACLHSE**
- **PSALocal**
System trace

address--: Target address. Addresses appear in the following formats:

<table>
<thead>
<tr>
<th>Addressing mode and location</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-bit address</td>
<td>xxxxxxx</td>
</tr>
<tr>
<td>31-bit address</td>
<td>xxxxxxxx</td>
</tr>
<tr>
<td>64-bit address with zeros in high-order bits</td>
<td>00_xxxxxxxx</td>
</tr>
<tr>
<td>64-bit address with nonzero high-order bits</td>
<td>xxxxxxxx_xxxxxxxx</td>
</tr>
</tbody>
</table>

Time Format
Blank
CP    Blank

PCIL trace entries

A PCIL trace represents a PCI load instruction.

PR   pr: Identifier of the processor that produced the TTE.
ASID home: Home address space identifier (ASID) associated with the TTE.
WU-Addr- wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).
Ident The TTE identifier, as follows:
PCIL  PCI load instruction.
CD/D pfid: The PCIE function identifier for the PCIE device.
PSW---- Address-
    cc: Condition code from the PCI load instruction.
    reqaddr: Address of the program that requested the PCI load instruction to be issued.
Unique-1/Unique-2/Unique-3
    traceid: Program defined trace identifier that can be used to determine why the PCI load instruction is being issued.
    operand1_operand1: First operand on the PCI load instruction.
    operand2_operand2: Second operand on the PCI load instruction.
    operand3_operand3: Third operand on the PCI load instruction.
Unique-4/Unique-5/Unique-6
    psaclhs-: String for the current lock held, from the PSACLHS field of the PSA.
System trace

PSACLHSE
psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
pasd: Primary ASID (PASID) at trace entry.

SASD
sasd: Secondary ASID (SASID) at trace entry.

Time Format
timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

PCIS trace entries
A PCIS trace represents a PCI store instruction.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHSE-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format--------</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>PCIS</td>
<td>pfid</td>
<td>cc</td>
<td>reqaddr-</td>
<td>traceid</td>
<td>operand1</td>
<td>operand1</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td></td>
<td></td>
<td>timestamp--------</td>
<td>CP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PR pr: Identifier of the processor that produced the TTE.

ASID
home: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

PCIS PCI store instruction.

CD/D
pfid: The PCIE function identifier for the PCIE device.

PSW----- Address-
• cc: Condition code from the PCI store instruction.
• reqaddr: Address of the program that requested the PCI store instruction to be issued.

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
• traceid: Program defined trace identifier that can be used to determine why the PCI store instruction is being issued.
• operand1_operand1: First operand on the PCI store instruction.
System trace

- operand2_operand2: Second operand on the PCI store instruction.
- operand3_operand3: Third operand on the PCI store instruction.

PSACLHS-
  psaclhs-: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
  psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
  psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
  pasd: Primary ASID (PASID) at trace entry.

SASD
  sasd: Secondary ASID (SASID) at trace entry.

Time Format
  timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP
  The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

PDMX trace entries
  A PDMX trace represents a PCIE adapter interruption de-multiplexing event.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format-</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>PDMX</td>
<td>pfid</td>
<td>pdmxaddr</td>
<td>devtype</td>
<td>cbck0</td>
<td>cbparm1</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>timestamp-</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PR
  pr: Identifier of the processor that produced the TTE.

ASID
  home: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
  wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
  The TTE identifier, as follows:

  PDMX
  PCIE adapter interruption de-multiplexing event.

CD/D
  pfid: PCIE function identifier for the PCIE device.

PSW----- Address-
  pdmxaddr: Address of the program that is performing the de-multiplexing.

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
System trace

- devtype: PCIE device type.
- cbck$: Callback routine address.
- cbparm1: First word of callback routine parameters.
- cbparm2: Second word of callback routine parameters.

**PSACLHS**
- psaclhs$: String for the current lock held, from the PSACLHS field of the PSA.

**PSACLHSE**
- psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA.

**PSALOCAL**
- psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA.

**PASD**
- pasd: Primary ASID (PASID) at trace entry.

**SASD**
- sasd: Secondary ASID (SASID) at trace entry.

**Time Format**
- timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

**CP**
The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**PGM, SPER and SPR2 trace entries**
These trace entries represent a program event:
- A PGM trace entry is for a program interrupt
- An SPER trace entry is for a PER event requested in a SLIP trap
- An SPR2 trace entry is for a PER event requested in a SLIP trap, when the STDERR keyword is specified on the trap

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>Unique-4</th>
<th>Unique-5</th>
<th>Unique-6</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format-</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>PGM</td>
<td>code</td>
<td>pgm-old-</td>
<td>pswaddr-</td>
<td>ilc-code</td>
<td>tea-</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp-</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>SPER</td>
<td>code</td>
<td>pgm-old-</td>
<td>pswctrl-</td>
<td>per-addH</td>
<td>per-addL</td>
<td>trap------</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp-</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>SPR2</td>
<td>code</td>
<td>pgm-old-</td>
<td>pswaddr-</td>
<td>var1</td>
<td>var2</td>
<td>var3</td>
<td>var4</td>
<td>var5</td>
<td>spc-exc</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>pasd</td>
<td>SASD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR**
- pr: Identifier of the processor that produced the TTE.

**ASID**
- home: Home address space identifier (ASID) associated with the TTE.

**WU-Addr**
- wu-addr$: Address of the task control block (TCB) for the current task or the work element block (WEB).
System trace

Ident
The TTE identifier, as follows:

PGM  Program interruption. An asterisk (*) before PGM indicates an unusual condition. PGM trace entries for program interrupts that may be resolved are not flagged. If the program interrupt is not resolved, then a subsequent RCVY trace entry is created and flagged with an asterisk.

SPER  SLIP program event recording

CD/D
• code for PGM entry: Program interruption code
• code for SPER entry: PER number

PSW----- Address-
The z/Architecture 128-bit old PSW appears on two lines:
• psaddr: Two words, containing the 64-bit address portion of the PSW
• psctrl: Two words, containing the 64-bit “control” portion of the PSW

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
• ilc-code: Instruction length code and interruption code.
• per-addh: high-order bits of the SLIP/PER status address.
• per-addl: low-order bits of the SLIP/PER status address.
• tea------: Translation exception address. In the high-order bit, 0 indicates primary and 1 indicates secondary.
• trap-----: SLIP/PER trap identifier in the form ID=xxxx.
• var1, var2, var3, var4, var5: Each contains one word of variable data as specified by the STDATA keyword.
• spc-exc: The message SpaceExc if more than five words of variable data are requested in the STDATA keyword.

PSACLHSH-
psaclhs-: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
pasd: Primary ASID (PASID) at trace entry.

SASD
sasd: Secondary ASID (SASID) at trace entry.

Time Format
timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.
RCVY trace entries

A RCVY trace entry represents entry into a recovery routine following an error or interruption. Several types of recovery events require reentry in a new environment or address space. Table 33 summarizes when an RCVY trace event requires reentry. RCVY also writes records to the system trace table for ESTAE type recovery exits and when the ESTAE type recovery exit requests retry.

Table 33. RCVY trace events that require reentry

<table>
<thead>
<tr>
<th>Trace Entry for Recovery Event</th>
<th>Reentry</th>
<th>Trace Entry for Reentry</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCVY ABT</td>
<td>Required only if the task to be ended resides in an address space other than the current home address space</td>
<td></td>
</tr>
<tr>
<td>RCVY ITRM</td>
<td>Always required</td>
<td>RCVY ITRR, if the unit of work ending is locally locked or has an EUT FRR established</td>
</tr>
<tr>
<td>RCVY MEM</td>
<td>Always required</td>
<td>RCVY MEMR</td>
</tr>
<tr>
<td>RCVY ABTR</td>
<td>Always required</td>
<td>RCVY RCMR</td>
</tr>
<tr>
<td>RCVY RCML</td>
<td>Always required</td>
<td>RCVY RCMR</td>
</tr>
<tr>
<td>RCVY STRM</td>
<td>Always required</td>
<td>RCVY STRR, if the unit of work ending is in SRB mode, is locally locked, or has an EUT FRR established</td>
</tr>
</tbody>
</table>
**PR** pr: Identifier of the processor that produced the TTE.
System trace

**ASID**
- Home address space identifier (ASID) associated with the TTE.

**WU-Addr-**
- WU-addr- : Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**
- The TTE identifier, as follows:
  - **RCVY** Recovery event. An asterisk (*) always appears before RCVY to indicate an unusual condition.

**CD/D**
- Type of recovery event, as follows:
  - **ABRT**: Abort processing for an unrecoverable error during any recovery termination management (RTM) processing
  - **ABT** : Request for abnormal end of a task by a CALLRTM TYPE=ABTERM macro, with a system or user completion code
  - **ABTR** : Rescheduling of a CALLRTM TYPE=ABTERM request for end of a task, when the task is not in the home address space
  - **DAT** : RTM1 entered for a dynamic address translation (DAT) error
  - **ESTA** : RTM has set up this ESTAE type recovery exit to receive control. Note that when a RCVY ESTA record follows another RCVY ESTA record without an intervening RCVY ESTR record, this implies that the first recovery exit either abended or percolated
  - **ESTR** : Retry was requested by this ESTAE type recovery exit. The exit and scb fields in this record can be used to help match it to its corresponding RCVY ESTA record.
  - **FRR** : RTM1 processing to invoke a function recovery routine (FRR)
  - **ITRM** : The system requested RTM1 to end an interrupted task
  - **ITRR** : ITRM reentry, to process a request to end an interrupted task
  - **MCH** : RTM1 entered for a machine check interruption
  - **MEM** : Request for abnormal memory end by a CALLRTM TYPE=MEMTERM macro, with a completion code
  - **MEMR** : Processing for an abnormal memory end following a MEM event
  - **PERC** : Percolation from RTM1 to RTM2 to continue recovery processing
  - **PROG** : RTM1 was entered for a program check interruption
  - **RCML** : RTM1 was entered to perform special end processing for a task in a failing address space. The failing address space held the local lock of another address space.
  - **RCMR** : RCML reentry, to process an abnormal end by a resource manager
  - **RESM** : Resume from an FRR after a RESTART request following an RSRT entry
  - **RSRT** : RTM entered for a RESTART request from the operator
  - **RTRY** : Retry from an FRR
  - **SABN** : The system requested RTM1 to end abnormally the current unit of work
  - **SKFE** : RTM has bypassed giving control to a FESTAE recovery exit because its address is zero. This situation can happen because of a timing window in FESTAE processing, in which case it is not a concern
  - **SPRC** : Final percolation from service request block (SRB) recovery
SRBT: Request for abnormal termination of a pre-emptable SRB by a
CALLRTM TYPE=SRBTERM macro, with a system or user completion code
STRM: The system requested RTM1 to end abnormally a suspended task
STRR: STRM reentry, to process the abnormal end of a suspended task

PSW----- Address-
For the RESM and RTRY entries, this field contains the z/Architecture 128-bit
old PSW, which appears on two lines:
• pswaddr--: Two words, containing the 64-bit address portion of the PSW
• pswctrl1-: Two words, containing the 64-bit “control” portion of the PSW
• exit-----: Address of an ESTAE type recovery exit. This is always zero for
RCVY SKFE entries.
• retry---: Retry address requested by an ESTAE type recovery
exit. When retry is to CVTBSM0F, contains the contents of 64-bit GPR15 at
the time of the retry request instead of the address of CVTBSM0F.
The frr-new- psw----- field contains the new program status word (PSW) to
give control to the FRR. For all other types of recovery events, the return--
field contains the caller’s return address.

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

• alet----: Contains the ALET of the parameter area for ESTAE type recovery
exits with an ALET qualified parameter area pointer. Otherwise, it contains
zero.
• asid----: Target ASID for end processing.
  – In a SPRC entry, the ASID is for the task that will be abnormally ended
    by SRB-to-task percolation. If this field and the
tcb----- field are zero, then no SRB-to-task percolation is performed.
• exit-----: Contains the ESTAE type recovery exit address
• comp----: System or user completion code
• cpu-----: Target processor for a restart error indicated on a request for an
FRR resume, after an operator RESTART request
• fpw-----: FRR processing word, in the following format:
  rsxxxxxp xxxxxxxx ssssssss eeeeeeee
  r  Bit 0 = 1 means a resource manager entry to the FRR
  s  Bit 1 = 1 means the FRR was skipped
  p  Bit 7 = 0 means a not serialized SRB-to-task percolation
     Bit 7 = 1 means a serialized SRB-to-task percolation

ssssssss
The stack index, which is an index of the FRR stack. The index means
the following:
  – 0 Normal stack
  – 1 SVC I/O dispatcher super stack
  – 2 Machine check super stack
  – 3 PC FLIH super stack
  – 4 External FLIH super stack 1
  – 5 External FLIH super stack 2
  – 6 External FLIH super stack 3
The entry index, which is an index of the FRR entry on the stack. The index ranges from 0 through 16. If the current stack is a super stack, an index of 0 indicates a super FRR.

- parm----: Contains the 31-bit parameter address, or the lower 32 bits of the 64-bit parameter address, which is provided to the ESTAE type recovery exit when it gains control.
- parm64--: Contains the upper 32 bits of the 64-bit parameter address when the ESTAE type recovery exit was established in AMODE 64
- pppp---- pppp----: PSW of the interrupted unit of work.
  - The instruction in the PSW may not be the cause of the failure. For example, an interruption can occur because a time limit expired, so that the interrupted instruction is not at fault.
- psasuper: PSASUPER field in the prefix save area (PSA)
- rc-------: Return code from CALLRTM
- reas------: Reason code accompanying the completion code appearing in the entry. If not provided, NONE.
- srbdtoken: Uniquely identifies the preemptable SRB, provided via the IEAMSCHD macro and consisting of four unique words on the second line for the RCVY SRBT trace entry only.
- tasn----: Target ASID for RCML reentry
- tcb------: Target task control block (TCB) for end processing
  - In a SPRC entry, the TCB is for the task that will be abnormally ended by SRB-to-task percolation. If this field and the asid---- field are zero, then no SRB-to-task percolation is performed.
  - In a STRM or STRR entry, a TCB address of zero indicates that the request was for ending of a suspended SRB.
- trk------: RTM1 error tracking area
- scb------: Contains the address of the STAE control block that represents this ESTAE type recovery exit. Note that for ARR and IEAARR recovery routines, RTM creates a 'pseudo-SCB'. Thus the same SCB address can be seen for multiple ARR or IEAARR recovery exits
- sdwa----: Contains the address of the system diagnostic work area that is provided to the ESTAE type recovery exit. If no SDWA is available to the exit, the field contains 0000000C

PSACLHS-
  psaclhs-: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
  psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
  psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
  pasd: Primary ASID (PASID) at trace entry.
System trace

SASD
sasd: Secondary ASID (SASID) at trace entry.

Time Format
timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

SPIN trace entries
A SPIN trace entry represents the starting (at least one second in), the middle (when special processing is done), or the stopping of a system spin attempting to obtain a resource. The spinning module will identify the resource within the trace entry.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSM..</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>Unique-4</th>
<th>Unique-5</th>
<th>Unique-6</th>
<th>PSACLHSE</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format</th>
<th>---------</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>SRC/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur holder--</td>
<td>rstrscid</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>ACO/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur holder--</td>
<td>infncode</td>
<td>asch-add</td>
<td>nmp-pam</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>BBR/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur cpu-spin</td>
<td>infncode</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>INT/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur cpu-spin</td>
<td></td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>LXX/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur lockword</td>
<td>lock-add</td>
<td>plhhson</td>
<td>plclhsp</td>
<td>lock-ent</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>RJ/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur cpu-spin</td>
<td>req-code</td>
<td>rcv-addr</td>
<td>pccaaddr</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>SGP/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur cpu-spin</td>
<td>para-reg</td>
<td>sigcode</td>
<td>sigstat</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>SPM/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur cpu-spin</td>
<td>reg0-</td>
<td>reg1-</td>
<td></td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home wu-addr</td>
<td>SPIN</td>
<td>MTC/[s]/p</td>
<td></td>
<td>return--</td>
<td>spin-dur phycpu</td>
<td>asid-</td>
<td></td>
<td>psaclhs-</td>
<td>psalocal</td>
<td>psd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PR pr: Identifier of the processor that produced the TTE.

ASID
home: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
uu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

SPIN System spin.

CD/D The CD/D field indicates the spinning module, using the last two or three characters of its module name, followed by a forward slash (/) and S (start), M (middle), or P (stop). For more information about each spinning module, see “Spinning modules” on page 200.
System trace

PSW----- Address-
return--: Caller's return address

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
For more information about each spinning module, see “Spinning modules.”

PSACLHSE-
psaclhse: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
pasd: Primary ASID (PASID) at trace entry.

SASD
sasd: Secondary ASID (SASID) at trace entry.

Time Format
timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP
The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

Spinning modules
The following section summarize information about the spinning modules.

BLWRESRC:
CD/D

SRC/S
The start spin entry of the BLWRESRC module.

SRC/P
The stop spin entry of the BLWRESRC module.

Unique-1
spin-dur
Spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

Unique-2
holder--
CPU that is a current holder.

Unique-3
rstrscid
The restart resource ID of the caller (input parameter 1).

IEAVEAC0:
CD/D
**System trace**

**AC0/S**
The start spin entry of the IEAVEAC0 module.

**AC0/P**
The stop spin entry of the IEAVEAC0 module.

**Unique-1**

**spin-dur**
The spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

**Unique-2**

**holder--**
CPU that is a current holder.

**Unique-3**

**infncoce**
The input function code.

**Unique-4**

**ascb-add**
Current ASCB

**Unique-5**

**inp-parm**
input_parm

**IEAVEBBR:**

**CD/D**

**BBR/S**
The start spin entry of the IEAVEBBR module.

**BBR/P**
The stop spin entry of the IEAVEBBR module.

**Unique-1**

**spin-dur**
The spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

**Unique-2**

**cpu-spin**
The CPU address being spun for.

**Unique-3**

**infncoce**
The input function code.

**Unique-4**

**reg1----**
input reg 1: when applicable, the ASID in bits 16-31.

**IEAVEINT:**

**CD/D**

**INT/S**
The start spin entry of the IEAVEINT module.
System trace

INT/P
The stop spin entry of the IEAVEINT module.

Unique-1
spin-dur
The spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

Unique-2
cpu-spin
The CPU address being spun for.

IEAVELKX:
CD/D
LKX/S
The start spin entry of the IEAVELKX module.

LKX/P
The stop spin entry of the IEAVELKX module.

Unique-1
spin-dur
Spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

Unique-2
lockword
The lock word that contains the CPU address being spun for.

Unique-3
lock-add
Input reg 11 (lockword address).

Unique-4
pllhsom-
Lock held obtained mask (PLLHSOM, via input reg 12).

Unique-5
plclhsp-
Lock held string pointer (PLCLHSP, via input reg 12).

Unique-6
lock-ent
Input reg 13. The lock routine entry point address.

IEAVERI:
CD/D
RI/S
The start spin entry of the IEAVERI module.

RI/M
The middle spin entry of the IEAVERI module.

RI/P
The stop spin entry of the IEAVERI module.
**System trace**

- **spin-dur**
  The spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

- **Unique-2**
  - **cpu-spin**
    The CPU address being spun for.

- **Unique-3**
  - **req-code**
    Input register 0. Request code and, when appropriate, ASID.

- **Unique-4**
  - **rcv-addr**
    Input reg 12. Receiving routine’s entry point address.

- **Unique-5**
  - **pccaaddr**
    Input reg 1. PCCA address of the receiving CPU. If this identifies the same CPU as Unique 2, Unique 2 value can be used.

**IEAVESGP:**

- **CD/D**
  - **SGP/S**
    The start spin entry of the IEAVESGP module.
  - **SGP/P**
    The stop spin entry of the IEAVESGP module.

- **Unique-1**
  - **spin-dur**
    Spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

- **Unique-2**
  - **cpu-spin**
    The CPU address being spun for.

- **Unique-3**
  - **para-reg**
    Input reg 1. Parameter register for status and prefix order codes.

- **Unique-4**
  - **sigpcode**
    Input reg 2. The SIGP order code.

- **Unique-5**
  - **sigpstat**
    The status returned from the last SIGP

**IEAVESPN:**

- **CD/D**
  - **SPN/S**
    The start spin entry of the IEAVESPN module.
  - **SPN/P**
    The stop spin entry of the IEAVESPN module.
System trace

1. **Unique-1**
   - **spin-dur**
     The spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

2. **Unique-2**
   - **cpu-spin**
     The CPU address being spun for.

3. **Unique-3**
   - **reg0----**
     Input reg 0.

4. **Unique-4**
   - **reg1----**
     Input reg 1.

5. **Unique-5**
   - **reg3----**
     Input reg 3.

**IEAVTMTCE:**

- **CD/D**
  - **MTC/S**
    The start spin entry of the IEAVTMTCE module.
  - **MTC/P**
    The stop spin entry of the IEAVTMTCE module.

6. **Unique-1**
   - **spin-dur**
     The spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

7. **Unique-2**
   - **phycpu--**
     The physical CPU number of some CPU that is still running with the terminating ASCB for "S" and 0 for "P". If an ACR condition is encountered, value can also be 0 for "S".

8. **Unique-3**
   - **asid----**
     The ASID that is the target of CALLRTM TYPE=MEMTERM.

**SSRV trace entries**

An SSRV trace entry represents entry to a system service. The service can be entered by a PC instruction or a branch.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSM----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time</th>
<th>Format</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr</td>
<td>SSRV</td>
<td>ssid</td>
<td>return-</td>
<td>data----</td>
<td>data----</td>
<td>data----</td>
<td>desc----</td>
<td>desc----</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp</td>
<td>06</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR**
- **pr**: Identifier of the processor that produced the TTE.

**ASID**
- **home**: Home address space identifier (ASID) associated with the TTE.
System trace

wu-addr-

wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident

The TTE identifier, as follows:

SSRV  Request for a system service

CD/D

ssid: One of the following SSRV entry identifiers:

<table>
<thead>
<tr>
<th>ssid (hexadecimal)</th>
<th>Macro for SSRV Request</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>WAIT</td>
<td>Task management</td>
</tr>
<tr>
<td>0002</td>
<td>POST</td>
<td>Task management</td>
</tr>
<tr>
<td>0004</td>
<td>GETMAIN</td>
<td>Virtual storage management</td>
</tr>
<tr>
<td>0005</td>
<td>FREEMAIN</td>
<td>Virtual storage management</td>
</tr>
<tr>
<td>000A</td>
<td>GETMAIN, FREEMAIN</td>
<td>Virtual storage management</td>
</tr>
<tr>
<td>005F</td>
<td>SYSEVENT</td>
<td>System resource manager</td>
</tr>
<tr>
<td>0078</td>
<td>GETMAIN, FREEMAIN</td>
<td>Virtual storage management</td>
</tr>
<tr>
<td>007A</td>
<td>SPI, SPIINT</td>
<td>Service processor interface</td>
</tr>
<tr>
<td>0100</td>
<td>ETCON</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0101</td>
<td>ETCRE</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0102</td>
<td>ATSET</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0103</td>
<td>AXSET</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0104</td>
<td>AXEXT</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0105</td>
<td>AXFRE</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0106</td>
<td>AXRES</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0107</td>
<td>ETDES</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0108</td>
<td>ETDIS</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>0109</td>
<td>LXFRE</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>010A</td>
<td>LXRES</td>
<td>PC/AUTH</td>
</tr>
<tr>
<td>010E</td>
<td>SUSPEND</td>
<td>Supervisor control</td>
</tr>
<tr>
<td>010F</td>
<td>RESUME</td>
<td>Supervisor control</td>
</tr>
<tr>
<td>0110</td>
<td>SCHEDULE</td>
<td>Supervisor control</td>
</tr>
<tr>
<td>0111</td>
<td>SCHEDULE</td>
<td>Supervisor control</td>
</tr>
<tr>
<td>0112</td>
<td>SCHEDULE</td>
<td>Supervisor control</td>
</tr>
<tr>
<td>0113</td>
<td>DSGNL</td>
<td>Supervisor control</td>
</tr>
<tr>
<td>0114</td>
<td>RPSGNL</td>
<td>Supervisor control</td>
</tr>
<tr>
<td>0115</td>
<td>SCHEDULE</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>0116</td>
<td>SCHEDULE</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>0117</td>
<td>SCHEDULE</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>0118</td>
<td>SUSPEND</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>0119</td>
<td>RESUME</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>011A</td>
<td>RESUME</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>011B</td>
<td>RESUME</td>
<td>Supervisor Control</td>
</tr>
</tbody>
</table>
### System trace

<table>
<thead>
<tr>
<th>ssid (hexadecimal)</th>
<th>Macro for SSRV Request</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>011C</td>
<td>SCHEDULE</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>011D</td>
<td>IEAMSCHD</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>011E</td>
<td>Pause (IEAVPSE / IEAVPSE2 / IEA4PSE / IEA4PSE2)</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>011F</td>
<td>Release (IEAVRLS / IEAVRLS2 / IEA4RLS / IEA4RLS2)</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>0120</td>
<td>Timer DIE</td>
<td>Supervisor Control</td>
</tr>
<tr>
<td>0128</td>
<td>WAIT</td>
<td>Task management</td>
</tr>
<tr>
<td>0129</td>
<td>POST</td>
<td>Task management</td>
</tr>
<tr>
<td>012A</td>
<td>POST</td>
<td>Task management</td>
</tr>
<tr>
<td>012B</td>
<td>POST</td>
<td>Task management</td>
</tr>
<tr>
<td>012C</td>
<td>ASCBCHAP</td>
<td>Task management</td>
</tr>
<tr>
<td>012D</td>
<td>STATUS</td>
<td>Task management</td>
</tr>
<tr>
<td>012E</td>
<td>STATUS</td>
<td>Task management</td>
</tr>
<tr>
<td>0132</td>
<td>STORAGE OBTAIN</td>
<td>Virtual storage management</td>
</tr>
<tr>
<td>0133</td>
<td>STORAGE RELEASE</td>
<td>Virtual storage management</td>
</tr>
<tr>
<td>0146</td>
<td>SPI, SPIINT</td>
<td>Service processor interface</td>
</tr>
<tr>
<td>014B</td>
<td>IARV64</td>
<td>Real storage management</td>
</tr>
<tr>
<td>014C</td>
<td>IGENQ</td>
<td>Global resource serialization</td>
</tr>
<tr>
<td>014D</td>
<td>ENQ/RESERVE</td>
<td>Global resource serialization</td>
</tr>
<tr>
<td>014E</td>
<td>DEQ</td>
<td>Global resource serialization</td>
</tr>
<tr>
<td>014F</td>
<td>SYSCALL</td>
<td>UNIX System Services</td>
</tr>
<tr>
<td>0150</td>
<td>ICYDIE</td>
<td>DFSMS Media Manager</td>
</tr>
<tr>
<td>1050</td>
<td>CF CPU command or internal request</td>
<td>Reconfiguration</td>
</tr>
</tbody>
</table>

**PSW----- Address-**

return---:

- For PC/AUTH, supervisor control, and task management: Caller’s return address if the service was entered by a branch; 0 if the service was entered by a PC instruction.
- For virtual storage management: For SSRV 132 (Storage Obtain) and SSRV 133 (Storage Release), it is the ALET. For other VSM SSRVs (004, 005, 00A, 078), it is the caller’s return address.
- For z/OS UNIX System Services: the syscall code.
- For real storage management: Bytes as follows:

<table>
<thead>
<tr>
<th>Request type identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>GETSTOR</td>
</tr>
<tr>
<td>02</td>
<td>GETSHARED</td>
</tr>
<tr>
<td>03</td>
<td>DETACH</td>
</tr>
<tr>
<td>04</td>
<td>PAGEFIX</td>
</tr>
<tr>
<td>05</td>
<td>PAGEUNFIX</td>
</tr>
<tr>
<td>06</td>
<td>PAGEOUT</td>
</tr>
<tr>
<td>Request type identifier:</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>DISCARDDATA</td>
</tr>
<tr>
<td>08</td>
<td>PAGEIN</td>
</tr>
<tr>
<td>09</td>
<td>PROTECT</td>
</tr>
<tr>
<td>0A</td>
<td>SHAREMEMOBJ</td>
</tr>
<tr>
<td>0B</td>
<td>CHANGEACCESS</td>
</tr>
<tr>
<td>0C</td>
<td>UNPROTECT</td>
</tr>
<tr>
<td>0D</td>
<td>CHANGEGUARD</td>
</tr>
<tr>
<td>0F</td>
<td>GETCOMMON</td>
</tr>
<tr>
<td>11</td>
<td>PCIEFIX</td>
</tr>
<tr>
<td>12</td>
<td>PCIEUNFIX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>GETSTOR GETSHARED Request flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COND=YES request</td>
</tr>
<tr>
<td>1..</td>
<td>FPROT=NO request</td>
</tr>
<tr>
<td>1..</td>
<td>CONTROL=AUTH request (only applies to GETSTOR)</td>
</tr>
<tr>
<td>1..</td>
<td>SVCDUMPRGN=NO request (only applies to GETSTOR)</td>
</tr>
<tr>
<td>...</td>
<td>CHANGEACCESS = GLOBAL request (only applies to GETSTOR)</td>
</tr>
<tr>
<td>...</td>
<td>GUARDLOC=HIGH request (only applies to GETSTOR)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>DETACH Request flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COND=YES request</td>
</tr>
<tr>
<td>1..</td>
<td>MATCH=USERTOKEN request</td>
</tr>
<tr>
<td>1..</td>
<td>AFFINITY=SYSTEM request</td>
</tr>
<tr>
<td>1..</td>
<td>OWNER=NO request</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>SHAREMEMOBJ Request flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COND=YES request</td>
</tr>
<tr>
<td>1..</td>
<td>SVCDUMPRGN=NO request</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>CHANGEGUARD Request flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COND=YES request</td>
</tr>
<tr>
<td>1..</td>
<td>TOGUARD request</td>
</tr>
<tr>
<td>1..</td>
<td>FROMGUARD request</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>PAGEFIX Request flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LONG=NO request</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>DISCARDDATA Request flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLEAR=NO request</td>
</tr>
<tr>
<td>1..</td>
<td>KEEPREAL=NO request</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>CHANGEACCESS Request flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>READONLY request</td>
</tr>
<tr>
<td>1..</td>
<td>SHAREDWRITE request</td>
</tr>
<tr>
<td>1..</td>
<td>HIDDEN request</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>GETCOMMON Request flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COND=YES request</td>
</tr>
</tbody>
</table>
### System trace

<table>
<thead>
<tr>
<th>0</th>
<th>Request type identifier:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>PCIeFIX Request flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>Keys Used flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>Miscellaneous Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>•</td>
<td>Storage Key for GETSTOR, GETSHARED, and GETCOMMON requests</td>
</tr>
<tr>
<td>•</td>
<td>Number of ranges in range list for range list requests</td>
</tr>
<tr>
<td>•</td>
<td>0 for all other requests</td>
</tr>
</tbody>
</table>

- For CONFIGURE CPU: Bytes as follows:

<table>
<thead>
<tr>
<th>Bytes 0 and 1</th>
<th>Target CPU ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 2</td>
<td>Internal flags</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Bits 0-3 Reserved</td>
</tr>
<tr>
<td></td>
<td>Bits 4-7 Direction and source</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits 4-7</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Online or Offline at MSI time</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Online Operator request</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Offline Operator request</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Online WLM request</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Offline WLM request</td>
<td></td>
</tr>
</tbody>
</table>

**Unique-1/Unique-2/Unique-3**

**Unique-4/Unique-5/Unique-6**

Data----: Data. The unique trace data for each event is obtained from data areas. The areas for PC/AUTH, supervisor control, and task management are in the [z/OS MVS Data Areas in z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/).

- For an SSRV request to the PC/AUTH component: the PCTRC data area
- For an SSRV request to supervisor control: the SPTRC data area
- For an SSRV request to task management: the TMTRC data area
- For an SSRV request to virtual storage management, the data is:
  - Under Unique-1: Information input to the VSM STORAGE (OBTAIN and RELEASE) service: Bytes as follows:

<table>
<thead>
<tr>
<th>0</th>
<th>Flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### System trace

<table>
<thead>
<tr>
<th>0</th>
<th>Flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1..</td>
<td>....</td>
</tr>
<tr>
<td>.1..</td>
<td>....</td>
</tr>
<tr>
<td>.0..</td>
<td>....</td>
</tr>
<tr>
<td>...1</td>
<td>....</td>
</tr>
<tr>
<td>....</td>
<td>1..</td>
</tr>
<tr>
<td>....</td>
<td>0..</td>
</tr>
<tr>
<td>....</td>
<td>.1..</td>
</tr>
<tr>
<td>....</td>
<td>.00</td>
</tr>
<tr>
<td>....</td>
<td>.01</td>
</tr>
<tr>
<td>....</td>
<td>.10</td>
</tr>
<tr>
<td>....</td>
<td>.11</td>
</tr>
</tbody>
</table>

1 Storage key (bits 0 through 3). Ignore when CALLRKY=YES is flagged in byte 0.

2 Subpool number

3 Request flags:

| 1.. | .... | ALET operand specified |
| .1.. | .... | Storage can be backed anywhere |
| .00 | .... | Storage must have callers residency |
| .01 | .... | Storage must have a 24-bit address |
| .10 | .... | The request is for an explicit address |
| .11 | .... | Storage can have a 24- or 31-bit address |
| .... | 1.. | Maximum and minimum request |
| .... | .1.. | Storage must be on a page boundary |
| .... | .11 | Unconditional request |
| .... | .0 | OBTAIN request |
| .... | .1 | RELEASE request |

- Under Unique-2:
  - In an SSRV trace entry for a VSM STORAGE OBTAIN or GETMAIN, one of the following:
    - The length of the storage successfully obtained
    - The maximum storage requested, if the storage was not obtained
  - In an SSRV trace entry for a VSM STORAGE RELEASE or FREEMAIN:
    - the length of the storage to be released, or zero if a subpool release was requested.

- Under Unique-3:
  - In an SSRV trace entry for a VSM STORAGE OBTAIN or GETMAIN, one of the following:
    - The address of the storage successfully obtained, if you specified address; otherwise, zero.
    - The minimum storage requested, if the storage was not obtained
  - In an SSRV trace entry for a VSM STORAGE RELEASE or FREEMAIN:
**System trace**

- The address of the storage to be released.
  - Under Unique-4:
    - Left 2 bytes under Unique-4: ASID of the target address space
    - Next byte under Unique-4: Reserved
    - Right byte under Unique-4:
      - If the GETMAIN/FREEMAIN/STORAGE OBTAIN/STORAGE RELEASE is unconditional, an abend will be issued and the SSRV trace entry 3rd byte of Unique-4 will contain X'FF'. If the GETMAIN/FREEMAIN/STORAGE OBTAIN/STORAGE RELEASE is conditional, no abend will be issued and the SSRV trace entry 3rd byte of Unique-4 will contain the actual return code from the storage service.

- For an SSRV request to real storage management (SSID 14B), the IARV64 data is:
  - Under Unique-1
    - Return Code/Abend Code (4 bytes)
  - Under Unique-2
    - Reason Code (4 bytes)
  - Under Unique-3
    - ALET specified on the IARV64 request (4 bytes)
  - Additional Unique fields depending on the IARV64 service that follows:
    - GETSTOR/GETSHARED/GETCOMMON
      - Origin address of the memory object - 8 bytes
      - Size of the memory object - 8 bytes
      - User token - 8 bytes
    - DETACH
      - Memory object start address (for MATCH=SINGLE requests) zeroes (for MATCH=USERTOKEN requests) - 8 bytes
      - User token - 8 bytes
    - PAGEFIX, PAGEUNFIX, PAGEOUT, PAGEIN, DISCARDATA, CHANGEACCESS, PROTECT, UNPROTECT, PCIEFIX, PCIEUNFIX
      - Address of rangelist - 8 bytes
      - VSA from 1st range list entry - 8 bytes
      - Number of blocks from 1st range list entry - 8 bytes
    - CHANGEGUARD
      - Memory object start (if ConvertStart was not specified), or convert start address (if ConvertStart was specified) - 8 bytes
      - Number of segments to be converted - 8 bytes
    - SHAREMEMOBJ
      - Range list address - 8 bytes
      - VSA from 1st range list entry - 8 bytes
      - User token - 8 bytes

- In an SSRV trace entry for global resource serialization with SSID (14C), the ISGENQ data is:
  - Under Unique-1:
    - Return address (4 bytes)
  - Under Unique-2:
Two bytes of flags as follows:

<table>
<thead>
<tr>
<th>Flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>01.</td>
</tr>
<tr>
<td>10.</td>
</tr>
<tr>
<td>11.</td>
</tr>
<tr>
<td>..1</td>
</tr>
<tr>
<td>...0</td>
</tr>
<tr>
<td>...1</td>
</tr>
<tr>
<td>...1</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1...</td>
</tr>
<tr>
<td>.1...</td>
</tr>
<tr>
<td>..1...</td>
</tr>
<tr>
<td>...1</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>....</td>
</tr>
</tbody>
</table>

**Note:** If the last bit of byte one and the first bit of byte two are both off, the system default for SYNCHRES is used.

ISGENQ reason code (2 bytes): If a list request was provided, this field will provide the reason code for the particular list entry in error. If more than one entry is in error, it will provide the highest reason code.

- Under Unique-3:
  - Primary ASID (2 bytes)
  - The last 2 bytes may represent
    - X'FFFF' if an incomplete trace entry. An incomplete entry may be the result of a program check or an error was detected. The entry will be populated only with data we know we can trust. Therefore, some flags may only be partially filled in. To avoid confusion, having a X'FFFF' as a device number and having the reserve request bit off will inform the user the entry is incomplete. (2 bytes)
    - X'0000' if not a reserve request (2 bytes)
    - Device number if a reserve request (2 bytes)

- Under Unique-4
  - First 4 bytes of the QNAME (4 bytes). For a list request, this represents the first QNAME in the request.

- Under Unique-5
  - Last 4 bytes of the QNAME (4 bytes). For a list request, this represents the first QNAME in the request.
In an SSRV trace entry for global resource serialization with SSID (14D) the ENQ and SSID (14E) the DEQ, the information is:

- **Under Unique-1:**
  - Return address (4 bytes)

- **Under Unique-2:**
  - Refer to the PEL mapping for explanation of PELLAST and PELXFLG1. See z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).
  - 3 bytes of flags.
  - Byte 1 is:
    - 0... ..... Exclusive request
    - .0.. ..... STEP
    - .1.. ..... SYSTEM
    - .0.. ..... SYSTEM w/UCB
    - .0.. 1... SYSTEMS w/UCB
    - .1.. 0... SYSTEMS
    - .1.. 0... An exit changed the request
    - ... 1... RNLs changed scope
    - .... .000 RET=NONE
    - .... .001 RET=HAVE
    - .... .010 RET=CHNG
    - .... .011 RET=USE
    - .... .100 RET=ECB
    - .... .101 RESERVED
    - .... .110 RESERVED
    - .... .111 RET=TEST

- Byte 2 represents PELLAST
  - Bit 4 is ignored.
- Byte 3 represents PELXFLG1
  - Bit 8 is ignored.

- ENQ return code (SSID14D) (1 byte) or DEQ return code (SSID 14E) (1 byte)
  - If a list request was provided, this field will provide the Return Code for the particular list entry in error. If more than one entry is in error, it will provide the highest Return Code.
  - If ABEND, this field is in the form X’Fn’ where n signifies the first hex digit of the ABEND code. For example, a X’F7’ signifies a X’738’ ABEND and X’F4’ signifies a X’438’ ABEND.

- **Under Unique-3:**
  - Primary ASID (2 bytes)
  - The last 2 bytes may represent
    - X’FFFF’ if an incomplete trace entry. An incomplete entry may be the result of a program check or an error was detected. The entry will be populated only with data we know we can trust. Therefore, some
flags may only be partially filled in. To avoid confusion, having a 
X’FFFF’ as a device number and having the reserve request bit off 
will inform the user the entry is incomplete. (2 bytes)
  • X’0000’ if not a reserve request (2 bytes)
  • Device number if a reserve request (2 bytes)

- Under Unique-4
  - First 4 bytes of the QNAME (4 bytes). For a list request, this represents 
the first QNAME in the request.

- Under Unique-5
  - Last 4 bytes of the QNAME (4 bytes). For a list request, this represents 
the first QNAME in the request.

• For an SSRV request to UNIX system services, the data is:
  - Under Unique-1
    The address of the PPRT control block

  - Under Unique-2:
    For an 8 byte parameter of an AMODE 64 caller, the low four bytes of the 
first parameter, otherwise the first four bytes of the first parameter, if 
available. Zero, if parameter not available.

  - Under Unique-3
    For an 8 byte parameter of an AMODE 64 caller, the low four bytes of the 
second parameter, otherwise the first four bytes of the second parameter, 
if available. Zero, if parameter not available.

  - Under Unique-4
    For an 8 byte parameter of an AMODE 64 caller, the low four bytes of the 
third parameter, otherwise the first four bytes of the third parameter, if 
available. Zero, if parameter not available.

• For an SSRV request to DFSMS Media Manager, the data is:
  - Under Unique-1
    The address of the IOSB control block

  - Under Unique-2:
    High order word of MMRE address (Media Manager Request Element) or 
zero

  - Under Unique-3
    Low order word of MMRE address

  - Under Unique-4:
    High order word of MMIB address (Media Manager Information Block) or 
zero

  - Under Unique-5
    Low order word of MMIB address

• In an SSRV trace entry for CONFIGURE CPU with SSID (1050), the 
information is:
  - Under Unique-1
    Contents of an internal return code field.

  - Under Unique-2
    Shows which 8-byte block of CSD_CPU_ALIVE.

  - Under Unique-4 through Unique-5
    The 8-byte contents of CSD_CPU_ALIVE mask at the 8-byte block offset 
in Unique-2 as updated by the CF CPU command.
**System trace**

**PSACLHS**
- desc----: String for the current lock held, from the PSACLHS field of the PSA. This field will contain descriptive text for some SSRV trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

**PSACLHSE**
- Blank.

**PSALOCAL**
- desc----: Locally locked address space indicator, from the PSALOCAL field of the PSA. This field will contain descriptive text for some SSRV trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

**PASD**
- pasd: Primary ASID (PASID) at trace entry. This field will contain descriptive text for some SSRV trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

**SASD**
- sasd: Secondary ASID (SASID) at trace entry. This field will contain descriptive text for some SSRV trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

**Time Format**
- timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

**CP**
- The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**SUSP trace entries**
An SUSP trace entry represents a request for a suspend type lock when the requestor had to be suspended because the lock was not available.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSM-----</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format-------</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>SUSP</td>
<td></td>
<td>return--</td>
<td>rb-addr-</td>
<td>suspndid</td>
<td>rel-addr</td>
<td>ssrbaddr</td>
<td>psaclhs-</td>
<td>psalocal</td>
<td></td>
<td></td>
<td>timestamp---------</td>
<td>02</td>
</tr>
</tbody>
</table>

**PR**
- pr: Identifier of the processor that produced the TTE.

**ASID**
- home: Home address space identifier (ASID) associated with the TTE.

**WU-Addr-**
- wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**
- The TTE identifier, as follows:
  - **SUSP**  
    Lock suspension
  - **CD/D**
    Blank
System trace

PSW----- Address-
    return--: Caller's return address

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
   • rb-addr: Address of the suspended request block (RB)
   • rel-addr: Address associated with the type of lock suspension:
       – 0: For LOCL lock
       – ASCB address: For CML lock
       – Lockword address: For CEDQ, CLAT, CMS, and CSMF locks
   • ssrbaddr: Address of the suspended service request block (SSRB)
   • suspndid: Identifier of the lock suspension type: CEDQ, CLAT, CML, CMS, CSMF, or LOCL

PSACLHS-
    psaclhs-: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
    psaclhse: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCA
    psalocal: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
    Blank

SASD
    Blank

Time Format
    timestamp---------: Time-of-day (TOD) clock value when system trace created
    the trace entry. The value is in the format that was specified for TIME. If TIME
    was not specified, the default format is TIME(HEX).

CP
    The CP column contains 2 hex digits of the processor model dependent
    information, which is intended to identify the physical CP that made the trace
    entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not
    provided for SYSUDUMP, SYSABEND, or SNAP.

SVC, SVCE, and SVCR trace entries
    These trace entries represent a supervisor event:
    • An SVC trace entry is for processing of a Supervisor Call (SVC) instruction
    • An SVCE trace entry is for an error during processing of an SVC instruction
    • An SVCR trace entry is for return from SVC instruction processing
System trace

PR pr: Identifier of the processor that produced the TTE.

ASID home: Home address space identifier (ASID) associated with the TTE.

WU-Addr- wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident The TTE identifier, as follows. An asterisk before SVC, SVCE, or SVCR indicates that the SVC is for an abend (SVC D) and the abend is not for a normal end of task, that is, bit X'08' in the leftmost byte of register 1 (in the Unique-3 column) is not on.

SVC Supervisor call (SVC) interruption
SVCE SVC error
SVCR SVC return

CD/D code:
• For SVC and SVCE, and for SVCR when not X'FFxx': SVC number.
• For SVCR when X'FF00': completion of the system-initiated processing involved with ATTACH, LINK, SYNCH, or XCTL processing before the target routine getting control.
• For SVCR when X'FF01': initial system-initiated processing involved with XCTL processing prior to the target routine getting control.

PSW------ Address-
The z/Architecture 128-bit old PSW, which appears on two lines:
• pswaddr: Two words, containing the 64-bit address portion of the PSW
• pswctrl: Two words, containing the 64-bit “control” portion of the PSW
The contents of this field varies, depending on the type of supervisor event and the value of code:
• ret-new- pswaddr / ret-new- pswctrl:
  – For SVC and SVCE, and for SVCR when code is not X'FFxx': Program status word (PSW) to receive control when the SVC is dispatched again.
  – For SVCR when code is X'FF00': PSW of the target routine that will get control as a result of ATTACH, LINK, SYNCH, or XCTL processing.
  – For SVCR when X'FF01': PSW of a system routine that will get control as a result of initial system processing involved with XCTL.
• svc-old- pswaddr / svc-old- pswctrl: SVC old PSW

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
gpr15--- gpr0---- gpr1----: General registers 15, 0, and 1

env-data: Characteristics of failing environment. One of the following values:
• 00000004 - issuer of SVC was in SRB mode
• 00000008 - issuer of SVC was locked
• 0000000C - issuer of SVC was disabled
• 00000010 - issuer of SVC was in cross memory mode
• 00000014 - issuer of SVC was in EUT FRR mode
• 00000018 - issuer of SVC was in AR mode
PSACLHS-
psaclhs-: For SVCE, string for the current lock held, from the PSACLHS field of the PSA. This field will contain descriptive text for some SVC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PSACLHSE
psclhse: For SVCE, extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
psalocal: For SVCE, locally locked address space indicator, from the PSALOCAL field of the PSA. This field will contain descriptive text for some SVC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PASD
pasd: Primary ASID (PASID) at trace entry. This field will contain descriptive text for some SVC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

SASD
sasd: Secondary ASID (SASID) at trace entry. This field will contain descriptive text for some SVC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

Time Format
timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP
The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

TIME trace entries
A TIME trace entry represents a dynamic time-of-day (TOD) clock adjustment by the timer services component.
System trace

**CD/D**

Contains a value of 1, indicating that word1 and word2 contain the amount of time that the system advances the time-of-day (TOD) clock when the TOD clock and the External Time Reference (ETR) get out of synchronization.

**PSW----- Address-**

return: Return address of the program that issued the PTRACE macro

**Unique-1/Unique-2/Unique-3**

**Unique-4/Unique-5/Unique-6**

word1, word2: For a code value of 1, the amount of time that the system advances the TOD clock when the TOD clock and the ETR get out of synch.

**PSACLHS-**

Blank

**PSACLHSE**

Blank

**PSALOCAL**

Blank

**PASD**

pasd: Primary ASID (PASID) at trace entry.

**SASD**

sasd: Secondary ASID (SASID) at trace entry.

**Time Format**

timestamp---------: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

**CP**
The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**USRn trace entries**

A USRn trace entry represents processing of a PTRACE macro in an authorized program. The trace entry contains data from the macro.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time Format</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>USRn</td>
<td>return--</td>
<td>data----</td>
<td>data----</td>
<td>data----</td>
<td>data----</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>home</td>
<td>wu-addr-</td>
<td>USRn</td>
<td>return--</td>
<td>idc-</td>
<td>rbc</td>
<td>data----</td>
<td>data----</td>
<td>pasd</td>
<td>sasd</td>
<td>timestamp---------</td>
<td>CP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PR**

pr: Identifier of the processor that produced the TTE.

**ASID**

home: Home address space identifier (ASID) associated with the TTE.

**WU-Addr-**

wu-addr-: Address of the task control block (TCB) for the current task or the work element block (WEB).

**Ident**

The TTE identifier, as follows:
System trace

**USRn**  User event. n is a number from `X'0'` to `X'F'.

**CD/D**
Blank

**PSW------ Address--**
return: Return address of the program that issued the PTRACE macro

**Unique-1/Unique-2/Unique-3**

**Unique-4/Unique-5/Unique-6**
- data----: User-defined data from the PTRACE macro
- idc=: PTRACE identification count
- rbc: Relative byte count

**PSACLHS-**
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

**PSACLHSE**
Blank.

**PSALOCAL**
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

**PASD**
pasd: Primary ASID (PASID) at trace entry. This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

**SASD**
sasd: Secondary ASID (SASID) at trace entry. This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

**Time Format**
timestamp---------:
Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

**CP**
The CP column contains 2 hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

**Multiple trace entries for a user event**
A single user event appears in more than one trace entry if the PTRACE macro requests recording of more than 5 fullwords of trace data. For example, the following PTRACE macro requests recording of 11 fullwords of trace data:

\[
\text{PTRACE TYPE=USER3,REGS=(2,12),SAVEAREA=STANDARD}
\]

For this macro, system trace places three entries in the trace table. The entries contain the following:
- The first entry contains the 5 fullwords of trace data in registers 2 through 6.
- The second entry contains the 5 fullwords of trace data in registers 7 through 11.
- The third entry contains the fullword of trace data in register 12.
If the program issuing the PTRACE macro is interrupted, the three trace entries may not be consecutive in the trace table. The multiple entries contain continuation information as the data for Unique-1. The format of the continuation information is as follows:

\[
\text{n}nnn \text{ h}hh
\]

**nnnn**

The PTRACE identification count. This is a hexadecimal number assigned by PTRACE to all entries for one macro processing.

**hhh**

The byte offset, in hexadecimal, of the next byte of trace data, which is under Unique-2. For the first entry, the offset is X'000'. For the second entry, the offset is X'014'. For the third entry, the offset is X'028'.

For example, Figure 72 shows three trace entries from the preceding PTRACE macro.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-</th>
<th>Address-</th>
<th>Unique-1</th>
<th>Unique-2</th>
<th>Unique-3</th>
<th>Unique-4</th>
<th>Unique-5</th>
<th>Unique-6</th>
<th>PSACLHS-</th>
<th>PSALOCAL</th>
<th>PASD</th>
<th>SASD</th>
<th>Time</th>
<th>Hex----------</th>
<th>CP</th>
<th>PSACLHSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0000</td>
<td>00AFF0</td>
<td>USR3</td>
<td>81A007B6</td>
<td>0030</td>
<td>000</td>
<td>00000001</td>
<td>00000002</td>
<td>00000003</td>
<td>00000004</td>
<td>00000005</td>
<td>00000006</td>
<td>00000007</td>
<td>00000008</td>
<td>00000009</td>
<td>0000000A</td>
<td>0000000B</td>
<td>0000000C</td>
<td>0000000D</td>
<td>0000000E</td>
</tr>
<tr>
<td>01</td>
<td>0000</td>
<td>00AFF0</td>
<td>USR3</td>
<td>81A007B6</td>
<td>0030</td>
<td>014</td>
<td>00000006</td>
<td>00000007</td>
<td>00000008</td>
<td>00000009</td>
<td>0000000A</td>
<td>0000000B</td>
<td>0000000C</td>
<td>0000000D</td>
<td>0000000E</td>
<td>0000000F</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0000</td>
<td>00AFF0</td>
<td>USR3</td>
<td>81A007B6</td>
<td>0030</td>
<td>028</td>
<td>00000008</td>
<td>00000009</td>
<td>0000000A</td>
<td>0000000B</td>
<td>0000000C</td>
<td>0000000D</td>
<td>0000000E</td>
<td>0000000F</td>
<td>CP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 72. Example: Three trace entries from the PTRACE macro*
Chapter 9. Master trace

Master trace maintains a table of all recently issued system messages. This creates a log of external system activity; the other traces log internal system activity. Master trace is activated automatically at system initialization, but you can turn it on or off using the TRACE command.

Master trace can help you diagnose a problem by providing a log of the most recently issued system messages. For example, master trace output in a dump contains system messages that may be more pertinent to your problem than the usual component messages issued with a dump.

The following sections contain more information about how to request, customize, and use the master trace.

Master trace and the hardcopy log

Master trace lists the same messages that the system saves automatically and permanently in the hardcopy log, but the entries are maintained in a wraparound table, which means that master trace overwrites old entries when the table is full. You can use master trace data in a dump as a substitute for the hardcopy log when the dump contains the required messages. If the master trace table wraps and overwrites the messages related to your problem before you can request a dump, the dump will not contain useful messages.

Consider the following conditions:

- The master trace table wraps at 9 p.m.
- The system issues messages related to a problem between 9:10 and 9:20 p.m.
- The system issues an SVC dump at 9:30 p.m.

In this example, the messages pertinent to the problem will be in the master trace data in the dump, since the problem occurred between the time the trace table wrapped and the time the dump was issued.

To print the system-managed data set containing the hardcopy log, use the JESDS parameter of the OUTPUT JCL statement.

Customizing master trace

At initialization, the master scheduler sets up a master trace table of 24 kilobytes. A 24-kilobyte table holds about 336 messages, assuming an average length of 40 characters. You can change the size of the master trace table or specify that no trace table be used by changing the parameters in the SCHEDxx parmlib member.

You can also change the size of the table using the TRACE command. For example, to change the trace table size to 36 kilobytes, enter:

```
TRACE MT,36K
```

See IBM MVS Initialization and Tuning Reference for more information about the SCHEDxx member.
Master trace

Requesting master trace

Start, change, or stop master tracing by entering a TRACE operator command from a console with master authority. For example, to start the master tracing:

TRACE MT

To stop master tracing:

TRACE MT,OFF

You can also use the TRACE command to obtain the current status of the master trace. The system displays the status in message IEE839I. For example, to ask for the status of the trace, enter:

TRACE STATUS

In the output shown in [Figure 73], master tracing is active with a trace table of 140 kilobytes, as indicated by MT=(ON,140K).

Figure 73. Example: TRACE STATUS output

If you want to check the current status of system, master, and component tracing, use the DISPLAY TRACE command. The system displays the status in message IEE843I. For example, to ask for the status of the three traces, enter:

DISPLAY TRACE

In [Figure 74], master tracing is active with a master trace table of 140 kilobytes, as indicated by MT=(ON,140K).

Figure 74. Example: DISPLAY TRACE output

See z/OS MVS System Commands for details about the TRACE and DISPLAY operator commands. See z/OS MVS System Messages, Vol 7 (IEB-IEE) for information about IEE839I and IEE843I messages.

Receiving master trace

Master trace writes trace data in the master trace table, which resides in the master scheduler address space (ASID 1). You can obtain master trace data in a stand-alone, SVC, or unformatted dump, if the dump options list includes TRT to request trace data. Format the master trace data by specifying the IPCS VERBEXIT
MTRACE subcommand or using the IPCS Trace Processing selection panel. Table 34 shows the dumps that contain master trace data.

Table 34. Summary of dumps that contain master trace data

<table>
<thead>
<tr>
<th>Dump</th>
<th>Master trace data in the dump?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stand-alone dump</td>
<td>Default</td>
</tr>
<tr>
<td>SVC dump for SDUMP or SDUMPX macro</td>
<td>Default</td>
</tr>
<tr>
<td>SVC dump for DUMP operator command</td>
<td>Default</td>
</tr>
<tr>
<td>SVC dump for SLIP operator command with ACTION=SVCD, ACTION=STDUMP, ACTION=SYNCSVCD, or ACTION=TRDUMP</td>
<td>Default</td>
</tr>
<tr>
<td>Any unformatted dump customized to exclude trace data</td>
<td>Yes, Request SDATA=TRT</td>
</tr>
<tr>
<td>ABEND dump to SYSABEND</td>
<td>Not available</td>
</tr>
<tr>
<td>ABEND dump to SYSMDUMP</td>
<td>Not available</td>
</tr>
<tr>
<td>ABEND dump to SYSUDUMP</td>
<td>Not available</td>
</tr>
<tr>
<td>SNAP dump</td>
<td>Not available</td>
</tr>
</tbody>
</table>

See z/OS MVS IPCS Commands for information about the VERBEXIT MTRACE subcommand. See z/OS MVS IPCS User’s Guide for information about the panel.

Reading master trace data

The following topics in this section show the format of master trace entries:
- “Master trace output formatted in a dump”
- “Master trace table in storage” on page 224

Master trace output formatted in a dump

The entries in the master trace table are listed in first-in, first-out (FIFO) order, which resembles a hardcopy log. The messages might not be in chronological order because presumably the messages were not put in the master trace table in the order the messages were issued.

Figure 75 on page 224 shows an example of master trace data in a dump that is formatted by IPCS. The subcommand that is issued on the IPCS Subcommand Entry panel:

VERBEXIT MTRACE
Master trace

*** MASTER TRACE TABLE ***

<table>
<thead>
<tr>
<th>TAG</th>
<th>IMM DATA</th>
<th>MESSAGE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.76 00000000 $HASP468 JES2 INIT DECK PROCESSED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 NC000000 SCOTT01 03147 21:24:22.77 INTERNAL 00000290 REPLY 0002, N1 AUTH= (NET= YES), NAME= SCOTT01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 W CO4000 SCOTT01 03147 21:24:22.76 00000000 +0002 $HASP469 REPLY PARAMETER STATEMENT, CANCEL, OR END</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000009 NR04000 SCOTT01 03147 21:24:22.77 INTERNAL 00000090 IEE6001 REPLY TO 0002 IS; N1 AUTH= (NET= YES), NAME= SCOTT01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.77 00000290 $HASP466 CONSOLE STMT 126 N1 AUTH= (NET= YES), NAME= SCOTT01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.77 00000090 $HASP826 NODE(1) NAME= SCOTT01, AUTH= (DEVICE= YES, JOB= YES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.77 00000090 $HASP826 NAME= SCOTT01, AUTH= (NET= YES, SYSTEM= YES), TRANSMIT= BOTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.78 00000090 $HASP826 RECEIVE= BOTH, HOLD= NONE, PENCYRYPT= NO, ENDNODE= NO, REST= SENTREST= ACCEPT, TRACE= NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.78 00000090 $HASP826 COMPACT= 0, LINE= 0, LOGMODE=, LOGON= 0, SEND= (NOTSET), END</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.78 00000090 $HASP826 PASSWORD= (VERIFY= (NOTSET), SEND= (NOTSET)), TRACE= NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.79 00000090 $HASP826 PATHMGR= YES, PRIVATE= NO, SUBNET=, TRANSENO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 NC000000 SCOTT01 03147 21:24:22.78 INTERNAL 00000000 $HASP466 CONSOLE STMT 126 END</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N CO4000 SCOTT01 03147 21:24:22.79 00000290 IEF196I IEF285I CONSOLE.OSV142.PARMLIB KEPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N 0000000 SCOTT01 03147 21:24:22.79 00000290 IEF196I IEF285I SYS1.PARMLIB KEPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 00000013 N 0000000 SCOTT01 03147 21:24:22.79 00000290 IEF196I IEF285I VOL SER NOS= D72666.KEPT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 75. Example of master trace data in a dump formatted by IPCS

The meaning of the highlighted text in the preceding example is as follows:

**TAG**

A halfword that contains the identity of the caller. TAG can be:

- **Tag**
  - 000: Reserved
  - 001: WTO SVC
  - 002: Master scheduler
  - 003: Trace command

Current identifiers are defined in the macro, IEZMTPRM, which maps the parameter list.

**IMM DATA**

A fullword of immediate data, consisting of the 32 bits defined by the caller. The significance of the immediate data is defined by the caller.

**MESSAGE DATA**

The message. If a problem occurs during processing, the line that follows the message indicates the problem.

Master trace table in storage

This topic describes master trace data as it is recorded in the master trace table in the master scheduler address space. You can use this information to write your own formatting or analysis routines for master trace information. Master trace places entries in FIFO order. Thus, a current entry is in front of the older entries. When the table is full, master trace wraps, and resumes recording entries at the end of the table.
Master trace

Note that the messages may not be in chronological order because the messages may not be put in the master trace table in the order in which they are issued.

You can locate the master trace table from the communication vector table (CVT), as shown in Table 35.

Table 35. How to locate master trace table from CVT

<table>
<thead>
<tr>
<th>At the location:</th>
<th>In field:</th>
<th>Find the following address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVT+X’94’</td>
<td>CVTMSER</td>
<td>IEEBASEA (master scheduler resident data area)</td>
</tr>
<tr>
<td>IEEBASEA+ X’8C’</td>
<td>BAMTTBL</td>
<td>Start of the master trace table</td>
</tr>
</tbody>
</table>

The unformatted master trace table in the master scheduler address space contains a header and, for each message logged in the table, an entry. The following two topics show the fields in the header and an entry. The master trace table header and entries are mapped by the MTT mappings in the IEEZB806 macro, which can be found in z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

Header in the master trace table

<table>
<thead>
<tr>
<th>TABLE ID</th>
<th>CURRENT</th>
<th>START</th>
<th>END</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBPOOL</td>
<td>LENGTH</td>
<td>WRAP TIME</td>
<td></td>
</tr>
<tr>
<td>WRAP POINT</td>
<td>RESERVED1</td>
<td>DATA LENGTH</td>
<td>RESERVED2...</td>
</tr>
</tbody>
</table>

**TABLE ID**
A fullword field containing MTT. MTT is an eye-catcher to mark the beginning of the master trace table.

**CURRENT**
A fullword field containing the address of the current (most recently stored) entry.

**START**
A fullword field containing the address of the first byte of the trace area.

**END**
A fullword field containing the address of the first byte beyond the end of the trace area.

**SUBPOOL**
A one-byte field containing the number of the subpool in which this table resides.

**LENGTH**
A three-byte field containing the length, in bytes, of the table header and the area containing the entries. This length is the default table size or the size specified on the TRACE command.

**WRAP TIME**
A double word field containing a time, either when the table was initialized or when the last table wrap occurred. The time is in the XXHH:MM:SS.T form:

- **XX** Possible values can be IT or WT.
- **IT** Indicates the time that the table was initialized.
- **WT** Indicates the time the table last wrapped.
- **HH** hours
Master trace

**MM** minutes

**SS** seconds

**T** tenths of a second

**WRAP POINT**
A fullword field containing the address of the first byte of the last entry stored before the most recent table wrap.

**Note:** This address is initialized to zero and remains zero until the first table wrap.

**RESERVED1**
A fullword reserved field.

**DATA LENGTH**
A fullword field containing the length, in bytes, of the data area part of the table.

**RESERVED2**
A 21-word field.

**Entry in the master trace table**

<table>
<thead>
<tr>
<th>FLAGS</th>
<th>TAG</th>
<th>IMM DATA</th>
<th>LEN</th>
<th>CALLER-PASSED DATA</th>
</tr>
</thead>
</table>

**Entry header**
10-byte header for the entry.

**FLAGS**
A halfword that contains the flags that are set by the caller in the parameter list that is passed to master trace.

**TAG**
A halfword that contains the identity of the caller. **TAG** can be:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Caller</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Reserved</td>
</tr>
<tr>
<td>0001</td>
<td>WTO SVC</td>
</tr>
<tr>
<td>0002</td>
<td>Master scheduler</td>
</tr>
<tr>
<td>0003</td>
<td>Trace command</td>
</tr>
</tbody>
</table>

Current identifiers are defined in the macro, IEZMTPRM, which maps the parameter list.

**IMM DATA**
A fullword that contains 32 bits defined by the caller. Master trace stores these bits in the table without checking them for validity. The significance of IMMEDIATE DATA is defined by the caller; likely values are a counter, a control block address, or flags describing the passed data.

**LEN**
A halfword that contains the length of the caller-passed data.

**CALLER-PASSED DATA**
A variable-length field that contains the data that is provided by the caller.
The master trace table entries vary in length. If the caller specifies the length of the caller-passed data as zero, the entry in the master trace table consists of only the 10-byte header.
Chapter 10. The Generalized Trace Facility (GTF)

The generalized trace facility (GTF) is a service aid you can use to record and diagnose system and program problems. GTF is part of the MVS system product, and you must explicitly activate it by entering a `START GTF` command.

Use GTF to record a variety of system events and program events on all of the processors in your installation. If you use the IBM-supplied defaults, GTF lists many of the events that system trace lists, showing minimal data about them. However, because GTF uses more resources and processor time than system trace, IBM recommends that you use GTF when you experience a problem, selecting one or two events that you think might point to the source of your problem. This will give you detailed information that can help you diagnose the problem. You can trace combinations of events, specific incidences of one type of event, or user-defined program events that the GTRACE macro generates. For example, you can trace:

- Channel programs and associated data for start and resume subchannel operations, in combination with I/O interruptions
- I/O interruptions on one particular device
- System recovery routine operations

The events that GTF traces are specified as options in a parmlib member. You can use the IBM supplied parmlib member or provide your own. Details of GTF operation, which include storage that is needed, where output goes, and recovery for GTF are defined in a cataloged procedure in SYS1.PROCLIB.

GTF can trace system and program events both above and below 16 megabytes. For each event it traces, GTF produces trace records as its output. You can have GTF direct this output to one of the following places:

- A trace table in virtual storage.
- A data set on a tape or direct access storage device (DASD).

Choose a trace table for your GTF output when maintaining good system performance is very important to your installation. The trace table cannot contain as much GTF trace data as a data set, but will not impact performance as much as a data set because there is no I/O overhead.

Choose a data set or sets when you want to collect more data than will fit in a trace table. Writing trace data to a data set does involve I/O overhead, so choosing this option will impact system performance more than a trace table.

GTF can use only one table in virtual storage, but can use up to 16 data sets. If you specify more than one data set, all of them must reside on devices of the same class, tape, or DASD.

Other components, such as OPEN/CLOSE/EOV and VSAM have special GTF support. See z/OS DFSMSdfp Diagnosis for complete details.
GTF and IPCS

You can use IPCS to merge, format, display, and print GTF output. See z/OS MVS IPCS User’s Guide and z/OS MVS IPCS Commands for information about the COPYTRC, GTFTRACE, and MERGE subcommands, and the trace processing option of the IPCS dialog.

GTF and the GTRACE macro

You can use GTF in combination with the GTRACE macro, provided you activate GTF with TRACE=USR. Then, your programs can issue GTRACE macros to generate trace records, which GTF can store in the trace table. See z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG for information about coding the GTRACE macro.

GTF and system trace

You can use GTF in combination with system trace. System trace records predetermined system events, and provides minimal details about each event. Supplement system trace information by selecting specific GTF options to provide more detailed information about system and user events. For further information about system trace, see Chapter 8, “System trace,” on page 169.

Using IBM defaults for GTF

IBM supplies both a SYS1.PARMLIB (also called parmlib) member that contains predefined GTF trace options and a cataloged procedure for GTF, should you want to use IBM’s defaults for GTF operation. You can override some of the default options by specifying certain parameters on the START command that activates GTF.

The IBM-Supplied parmlib member of GTF trace options

IBM supplies the GTFPARM parmlib member, which contains the GTF trace options shown in Figure 76.

<table>
<thead>
<tr>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE=SYSM,USR,TRC,DSP,PCI,SRM</td>
</tr>
</tbody>
</table>

Figure 76. IBM-Supplied GTFPARM parmlib member

Briefly, the following options request GTF traces. For more details about these trace options, see “GTF trace options” on page 247.

- **SYSM**: Selected system events
- **USR**: User data that the GTRACE macro passes to GTF
- **TRC**: Trace events associated with GTF
- **DSP**: Dispatchable units of work
- **PCI**: Program-controlled I/O interruptions
- **SRM**: Trace data associated with the system resource manager (SRM)

The IBM-Supplied cataloged procedure

IBM supplies the GTF cataloged procedure, which resides in SYS1.PROCLIB. The cataloged procedure defines GTF operation, output location, recovery facilities, trace output data sets, and the parmlib member that contains GTF options and defaults. Figure 77 on page 231 illustrates the content of the IBM supplied
The statements in this cataloged procedure are:

PROC
  Defines the GTF cataloged procedure.

EXEC

  PGM=AHLGTF
  Calls for the system to run program AHLGTF.

  PARM='MODE=EXT,DEBUG=NO,TIME=YES',
  The parameters selected specify that GTF direct trace data to a data set on
tape or DASD, attempt recovery if it encounters an error, and give every
trace record a time stamp. See the explanation for the EXEC statement
under ["Setting up a cataloged procedure" on page 232] for detailed
information.

  TIME=1440
  The amount of time, in seconds, that GTF will remain active.

  REGION=4M
  Specifies the maximum size of the storage that GTF can use.

IEFRDER DD
Defines the trace output data set:

  The common data set name is SYS1.TRACE.
  The data set can reside on a DASD if:
    The primary space amount is large enough for the data set to contain at
    least 20 physical blocks. After completely filling the primary space, GTF
    will overlay previously written records with new trace records, starting at
    the beginning of the output data set.
    The set is in basic format, large format sequential, or does not have a
    DSORG value.
    The set is a VSAM linear data set with a control interval size (CISIZE) of
    32 KB.
  The data set can reside on a tape if the tape has IBM standard labels or no
  labels. If GTF fills the volume, the system will request another volume to
  continue writing. GTF does not overlay the trace records of this GTF written
  instance.

Restrictions to IEFRDER:
  A DSNTYPE=LARGE data set can only be used if the trace is both written
  and processed on an V1R7 system or a later release.

SYSLIB DD
Defines the IBM-supplied GTFPARM parmlib member that contains GTF trace
options and their default values. Multiple instances of GTF can be active at the
same time. Each instance of GTF requires a unique trace dataset. The default
trace dataset in the cataloged procedure can be overridden by specifying a different data set on the START command, or by setting up a cataloged procedure for each instance of GTF to be activated.

Customizing GTF

You can customize GTF to the needs of your installation by either overriding IBM’s defaults through the START GTF command, or providing your own parmlib member and cataloged procedure for GTF. Customize GTF in one of the following ways:

- Predefine the GTF trace options in a parmlib or data set member. See “Defining GTF trace options.”
- Set up a cataloged procedure. See “Setting up a cataloged procedure.”
- Override the defaults in the IBM supplied GTF cataloged procedure using the START command. See “Using the START command to invoke GTF” on page 239.
- Determine how much storage GTF needs for the trace options you choose. See “Determining GTF’s storage requirements” on page 237.
- Specify trace options directly through the console after entering the START command. See “Specifying or changing GTF trace options through system prompting” on page 240.

Defining GTF trace options

If you supply your own parmlib member or data set containing GTF trace options, you can select any of the options listed in “GTF trace options” on page 247. Each instance of GTF can be activated with the same or different set of options.

The member containing predefined trace options does not have to reside in the parmlib member. GTF will accept any data set specified in the SYSLIB DD statement of the cataloged procedure, or in the START command, as long as that data set’s attributes are compatible with those of SYS1.PARMLIB.

Setting up a cataloged procedure

Set up your own GTF cataloged procedure when you want to control details of GTF operation such as:

- Amount of storage needed for tracing
- Recovery for GTF
- Number and type of trace output data sets.

If you choose to supply your own cataloged procedure, include the following statements:

```
PROC
  Defines your cataloged procedure.
EXEC
  PGM=AHLGTF
    Calls for the system to run program AHLGTF.
  PARM=’parm, parm...’
    Options specified on the PARM parameter specify where GTF writes trace data and the amount of storage needed for GTF to collect and save trace data in various dump types. parm can be any of the following:
```

z/OS V2R2 MVS Diagnosis: Tools and Service Aids
MODE={INT|EXT|DEFER}

Defines where GTF writes the trace data. MODE=INT directs data to a trace table in virtual storage, and MODE=EXT directs data to a data set on tape or DASD.

If MODE=INT, each instance of GTF will direct the trace data to a separate trace table in virtual storage, and will ignore any DD statements that define GTF output data sets. Choose this option when it is very important to you to maintain good system performance while GTF runs. The trace table cannot contain as much GTF trace data as a data set, but will not impact performance as much as a data set because there is no I/O overhead.

If MODE=EXT or MODE=DEFER, each instance of GTF directs the output to a separate trace data set defined by GTFOUtxx or IEFRDER DD statements. MODE=EXT is the default value. Choose MODE=EXT or MODE=DEFER when you want to collect more data than will fit in a trace table. Writing trace data to a data set does involve I/O overhead, so choosing one of these options will impact system performance more than MODE=INT.

MODE=DEFER will place the trace data in the GTF address space until you enter the STOP GTF command. Every instance of GTF runs in its own address space. During GTF end processing, each instance of GTF will transfer the data from its address space to the output data set.

The amount of data transferred for MODE=EXT or MODE=DEFER is one of the following:

- The default amount
- The amount specified on the SADMP|SA keyword

When the trace output data set is full, GTF continues as follows:

- **Direct access**: GTF resumes recording at the beginning of the data set, when the primary allocation is full. Thus, GTF writes over earlier trace data.
- **Tape**: GTF writes an end-of-file record. The tape is rewound and unloaded, then a new volume is mounted on the tape unit. If GTF has only one tape data set and only one unit for the data set, GTF does not write trace records while the tape is unavailable, thus losing trace data.

GTF can write to multiple tape units in two ways:

- Multiple GTFOUtxx DD statements can specify tape data sets. When GTF fills one data set, it changes to the next data set.
- The IEFRDER DD statement can specify two tape units; in this case, GTF resumes writing the current trace data on the other unit, while rewinding and unloading the full volume.
Generalized Trace Facility

Specifies the amount of storage needed to save GTF trace data for stand-alone dumps. Specify the amount of storage in terms of either kilobytes (K) or megabytes (M). The minimum amount is 40K, and the maximum is 2048M minus 400K, or 2096752K. GTF rounds up the amount to the block size boundaries for DASD data sets, or 32K boundaries for tape data sets or internal mode. The default value for this parameter is 40K (rounded up to the correct boundary).

SDUMP|SD={nnnnnnK|nnnnM|40K}
Specifies the amount of storage needed to save GTF trace data for SVC dumps. Specify the amount of storage in terms of either kilobytes (K) or megabytes (M). The minimum amount is zero, and the maximum cannot exceed the maximum amount of storage defined by the SADMP parameter. GTF rounds up the amount to the block size boundaries for DASD data sets, or 32K boundaries for tape data sets or internal mode. The default value for this parameter is 40K (rounded up to the correct boundary).

NOPROMPT|NP
If specified, indicates that you will not be prompted to specify trace options. Message AHL125A and AHL100A will not be issued. Use this parameter when you have a parmlib member set up with the desired GTF options and you want to avoid multiple replies in a sysplex environment.

ABDUMP|AB={nnnnnnK|nnnnM|0K}
Specifies the amount of GTF trace data to be formatted in an ABEND or SNAP dump. Specify the amount of trace data in terms of either kilobytes (K) or megabytes (M). The minimum amount is zero, and the maximum cannot exceed the maximum amount of storage defined by the SADMP parameter. GTF rounds up the amount to the block size boundaries for DASD data sets, or 64K boundaries for tape data sets or internal mode. The default value for this parameter is 0K, which means that no GTF data will appear in ABEND or SNAP dumps.

For ABEND or SNAP dumps. GTF formats only those records that are directly associated with the failing address space. GTF does not format the channel program trace data associated with the failing address space.

BLOK={nnnn | nnnnnK | nnnnnM | 40K}
Specifies the amount of virtual storage (E)CSA that GTF will use to collect trace data. Specify this storage amount in 4096-byte pages (nnnn), or in bytes (nnnnK or nnnnnM). The maximum amount is 999999; the default is 40K. If the amount is less than 40K, GTF will use the default.

SIZE={nnnnnK|nnnnnM|1024K}
Specifies the size of the buffers. Specify this amount in bytes (nnnnnK or nnnnnM). The range for the size keyword is 1M to 2046M. The maximum amount is 2046M; the default is 1024K. If the amount is less than 1024K, GTF will use the default.

TIME=YES
Specifies that every GTF trace record have a time stamp, as well as the block time stamp associated with every block of data. The time stamp is the eight-byte time of day (TOD) clock value at the local time in which GTF puts the record into the trace table. GTF does not accept TIME=NO; all output records will have time stamps. Local time is
calculated using a time zone offset that GTF establishes at the time that the trace starts. If the system time zone offset is changed during tracing, e.g. in response to daylight saving time going into effect, local times formatted by GTF will not correspond with system times afterward.

When you use IPCS to format and print the trace records, a time stamp record follows each trace record. You can use these time stamp records to calculate the elapsed time between trace entries. The time stamp record is described in [“Time stamp records” on page 269].

**DEBUG={YES|NO}**

Specifies whether GTF should attempt recovery after it encounters an error. If DEBUG=YES, GTF will not attempt any recovery. Instead, GTF will issue an error message and end after encountering any error, so that the contents of the trace table immediately prior to the error remain intact.

If DEBUG=NO, which is the default, GTF does the following:

- For errors in GTF processing, GTF continues processing after doing one or more of the following:
  - Flagging the trace record or trace record field associated with the error
  - Issuing a message to the console to inform that an error occurred
  - Suppressing the error or function in which the error occurred.
- For errors that do not occur in GTF itself, GTF ends abnormally. If GTF stops processing, that will not cause any other task to also stop.

**TIME=1440**

Specifies unlimited processor time for GTF.

**REGION=nnnnK|M**

Specifies the maximum size of the storage that GTF can use. If the REGION parameter is omitted, then the GTF program defaults to using the REGION value that was specified during the system installation. For information about determining the value for REGION, see [“Determining GTF’s storage requirements” on page 237]. For general information about the REGION parameter, see [z/OS MVS JCL Reference].

**IEFRDER DD or GTFOUTxx DD**

Defines the trace output data set or data sets. This statement is required only if you do one of the following:

- Specify MODE=EXT or MODE=DEFER
- Use the default MODE=EXT

IEFRDER DD can be used, but does not have to be used, for one trace output data set. Additional data sets must be defined on GTFOUTxx DD statements, where xx is one or two characters that are valid in DDNAMES. The trace output data set or data sets must be unique and cannot be shared across active instances of GTF. See [“Guidelines for defining GTF trace output data sets in a cataloged procedure” on page 236] for guidance on how to define output data sets for GTF.

**SYSLIB DD**

Optionally include a SYSLIB DD statement to define the IBM-supplied member, or the installation-supplied member, that contains GTF trace options. If the member exists, GTF will use the options in that member. If the member does not exist, GTF will issue an error message and stop.
If you code a procedure that does not contain a SYSLIB DD statement, GTF issues message AHL100A to prompt for options after the START GTF command. In response, you can supply the desired trace options through the console. See “Specifying or changing GTF trace options through system prompting” on page 240 for examples of specifying options through the console.

Guidelines for defining GTF trace output data sets in a cataloged procedure

The trace output data sets must be specific to each instance of GTF and can be defined in the cataloged procedure. Each instance of GTF to be started must have a separate cataloged procedure, or if the same cataloged procedure is used, then a different trace data set must be supplied with the START command.

Use the following guidelines for specifying trace output data sets on the IEFRDER DD or GTFOUTxx DD statements:

- You can define up to 16 output data sets for GTF to use. If you define more than 16 data sets, GTF will accept the first 16 and ignore the rest.
- If GTF cannot open all of the data sets, it issues a message that identifies those that are unopened, and continues processing with those that are open.
- Do not specify the RLSE option while using the SPACE parameter because the output data sets are opened and closed more than once while GTF runs.

**Note:** If the GTF trace output resides on an SMS volume, you should ensure that the SMS management class does not allow partial release.

- Do not request secondary extents for trace data sets. GTF will only use the first extent.

To obtain the maximum degree of control over the number of trace entries for which space is allocated, specify space allocation in blocks, using a block length that matches the BLKSIZE of your trace data set. Do not specify any secondary space. Use the CONTIG option to request contiguous blocks. For example, if your BLKSIZE is 8192, code the SPACE keyword as follows:

```
SPACE=(8192,(500,0),,CONTIG)
```

- All data sets must be in the same device class: either DASD or tape, but not both. If you mix device classes, GTF will ignore the tapes and use only DASD. However, the data sets can have different device types; for example, you can mix 3380 and 3390 device types.

- If the data set is on a DASD:
  - The data set can be sequential, basic format, large format, or extended format. Basic format is limited to 65535 tracks on the volume. Large format is limited to 16,777,215 tracks per volume. The extended format limit is much higher. Extended format data sets can be compressed format, striped, or both. You cannot use the WRAP option in compressed format.
  - The data set must be linear if it is VSAM and must have a 32 KB control interval size. The data set might be extended format, striped, and might have the extended addressing option to allow it to be larger than 4 GB.

When WRAP processing is requested, the primary space request is consulted and only control intervals that are contained within that space are used. Unlike non-VSAM data sets used for WRAP processing, the primary space does not have to be satisfied using a single extent. A data set must be empty or defined with the REUSE attribute. If neither of the two conditions exists, GTF rejects the use of the data set.
When WRAP processing is not requested, control intervals are filled until GTF is stopped or the data set is full.

**Note:** GTF and CTRACE accept a single VSAM linear data set as output. VSAM's support for striping can increase data rate without the complexity that is associated with the use of distinct data sets.

- GTF and CTRACE support placement of NOWRAP traces in cylinder-managed space. WRAP traces placed in VSAM linear data sets can reside in cylinder-managed space too. WRAP traces in non-VSAM data sets cannot be placed in large format data sets, extended format data sets, or cylinder-managed space.
- To ensure the most efficient GTF processing, do not specify any particular block size for the output data set or data sets in either:
  - The cataloged procedure for GTF
  - The JCL, TSO/E commands, or interactive system productivity facility/program development facility (ISPF/PDF) panels that you might use to preallocate the data sets.
  
  The system computes an optimal block size when it opens each data set.

**Note:** If you want GTF to use an unlabeled tape as the output data set, you must specify the logical record length and block size when you allocate that data set.

- If you define more than one data set, you should ensure that the number of paths to the data sets equals the number of data sets.
- You can specify the number of channel programs for each output data set using the NCP parameter on each DD statement. The NCP value determines the rate at which GTF transfers data to the output data sets. For example, if you want to transfer data to your data sets at a rate of 25 buffers per second and you have 5 data sets, you will need to specify an NCP value of 5. GTF then transfers data to the 5 data sets at a rate of 5 buffers per second per data set for a total rate of 25 buffers per second.

  The maximum value for NCP is 255. If you do not specify a value for NCP, or if you specify a value less than four, GTF will use the following default values:
  - For tape: four
  - For DASD: the number of output blocks per track, which is multiplied by four.

- If, when you enter the START command, you override any of the DD statements for multiple output data sets, you must use symbolic parameters in those DD statements. See “Using the START command to invoke GTF” on page 239 for more information.

### Determining GTF's storage requirements

The storage that GTF requires depends on the trace options that you choose. Modern systems consume large amounts of virtual storage, and GTF is no exception. Use the REGION parameter to specify at least 20 MB of available storage for GTF. Then, if you need to impose constraints on the private storage of the GTF address space, use the REGION parameter to restrict GTF's usage. If the REGION parameter is omitted, then the GTF program defaults to using the REGION value that was specified during the system installation. For general information about the REGION parameter, see [z/OS MVS JCL Reference](#).
After determining the GTF trace options, use Figure 78 to determine several storage requirements for either your cataloged procedure or the START GTF command. There are several types of storage to calculate:

- Extended pageable link pack area (EPLPA)
- System queue area (SQA)
- Extended common service area (ECSA)
- Region storage

Use the formulas in Figure 78 to calculate the amount of storage needed for each storage type. Then add them all together to arrive at the final figure to specify on the REGION parameter. For information about the options mentioned in the figure, see “GTF trace options” on page 247.

<table>
<thead>
<tr>
<th>Extended Pageable Link Pack Area</th>
<th>System Queue Area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fix</strong> = Opt + Prmp + 8K</td>
<td><strong>SQA</strong> = 16500 + REG + SAVE + CBLOC</td>
</tr>
<tr>
<td><strong>Fix</strong>: Fixed storage in pageable EPLPA while GTF is active.</td>
<td><strong>SQA</strong>: System Queue Area storage requirement.</td>
</tr>
<tr>
<td><strong>Opt</strong>: Sum of storage required for each GTF option specified. See the table below to calculate OPT.</td>
<td><strong>REG</strong>: 232 bytes per processor are required for register save areas, regardless of whether or not GTF is active.</td>
</tr>
<tr>
<td><strong>Prmp</strong>: Optional additional 1.5K if any prompting options specified.</td>
<td><strong>SAVE</strong>: 1352 bytes per processor are required for save/work areas when GTF is active.</td>
</tr>
<tr>
<td><strong>8K</strong>: 8K required for services.</td>
<td><strong>CBLOC</strong>: 1700-2200 bytes are needed for control blocks when GTF is active.</td>
</tr>
</tbody>
</table>

### Extended Pageable Link Pack Area

<table>
<thead>
<tr>
<th>Option</th>
<th>Size Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSM</td>
<td>4K</td>
</tr>
<tr>
<td>SYS with DSP and/or SRM and/or RNIO</td>
<td>7K</td>
</tr>
<tr>
<td>SYS, SYS BP</td>
<td>18K</td>
</tr>
<tr>
<td>PI, DSP, PIP</td>
<td>2.5K</td>
</tr>
<tr>
<td>EXT</td>
<td>2K</td>
</tr>
<tr>
<td>IO, IOP, SIO, SIDP, SSCH, SSCHP</td>
<td>6K</td>
</tr>
<tr>
<td>SVC, SVCP</td>
<td>10K</td>
</tr>
<tr>
<td>SRM, RR, RNIO</td>
<td>3K</td>
</tr>
<tr>
<td>SLIP</td>
<td>8K</td>
</tr>
<tr>
<td>USR, USRP</td>
<td>1.5K</td>
</tr>
<tr>
<td>PCI, TRC</td>
<td>No Requirement</td>
</tr>
<tr>
<td>CCW, CCWP</td>
<td>9.3K</td>
</tr>
</tbody>
</table>

### System Queue Area

<table>
<thead>
<tr>
<th>Option Size Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ESQA</strong> = N</td>
</tr>
<tr>
<td><strong>N</strong>: Approximately 4500 times the number of blocks specified on the BLOCK = keyword parameter of the GTF START command.</td>
</tr>
<tr>
<td>The default is 4506 bytes.</td>
</tr>
</tbody>
</table>

### Notes:

1. When you specify more than one event from a line, the size requirement is the same as if you specified only one option. For example, DSP and PI require 2.5K.
2. For the maximum storage requirement round up the storage requirement for each option you specified to the nearest 4K boundary.
3. For the minimum storage requirement, round up the Fix value to the nearest 4K boundary.

### Example:

1) Options = PIP, DSP, SLIP
   - Fix = 10.5 + 1.5 + 8 = 20K minimum or 12 + 1.5 + 8 = 21.5 + 24K maximum
2) Options = SYSM, SRM USR, TRC
   - Fix = 8.5 + 0 + 8K = 16.5 = 20K minimum or 12 + 0 + 8K = 20K maximum

Figure 78. GTF storage requirements
Starting GTF

Multiple instances of GTF can be active in a system at the same time. When you activate GTF, each instance operates as a system task, in its own address space. The only way to activate GTF is to enter a START GTF command from a console with master authority. Using this command, select either IBM’s procedure or your cataloged procedure for GTF. The cataloged procedure defines GTF operation; you can accept the defaults that the procedure establishes, or change the defaults by specifying certain parameters on the START GTF command.

Because GTF sends messages to a console with master authority, enter the command only on a console that is eligible to be a console with master authority. Otherwise, you cannot view the messages from GTF that verify trace options and other operating information.

Each instance of GTF can be assigned a unique identifier that is specified on the START GTF command after the GTF keyword. This will allow you to recognize and control specific instances of GTF. If a unique identifier is not specified, the operating system assigns the default, which is the device number of the device where the trace data set resides. See the example in the topic “Starting GTF with trace output to an existing data set on tape” on page 242 for an instance of GTF with the default identifier.

Using the START command to invoke GTF

The START command, without any parameters other than the IBM-supplied procedure name and an identifier, uses the defaults of the cataloged procedure. If that source JCL contains a DD statement for the data set member of predefined trace options, GTF will issue a message that lists those options, and will allow you to override them. Otherwise, GTF will prompt you to specify trace options directly through the console. See “Specifying or changing GTF trace options through system prompting” on page 240 for more information.

To invoke GTF, enter the following START command. For information about the command and parameters you can use to change the GTF cataloged procedure, see z/OS MVS System Commands.

```
{START|S}{GTF|membername}[,identifier]
```

Guidelines for overriding JCL statements in the GTF cataloged procedure

You can override the parameters of only one output data set using the keyword=option parameter on the START command. If you have defined more than one output data set, and you used IEFRDER as the DDNAME for one of the DD statements, the keywords specified on the START command will override the attributes of the data set that IEFRDER defines. If you want to alter the attributes of another data set, or more than one data set, you must:

- Use symbolic parameters in the JCL DD statements for those attributes you want to change. You cannot use DD statement keywords as symbolic parameters; for example, you cannot code UNIT=&UNIT;
- Assign values to the symbolic parameters in the EXEC or PROC statements in the cataloged procedure.
- Specify keywords in the START command to override the symbolic parameter values specified on the EXEC or PROC statements.
Examples of overriding the JCL statements in the GTF cataloged procedure

The following shows examples of setting up a cataloged procedure when you want to override JCL statements in the procedure using the keyword=option parameter on the START command. Note that the DD statement parameters in both of the following procedures are for example only; the needs of your installation might require you to provide DD parameters in addition to, or other than, DSNAME, UNIT, and DISP.

If you want to alter just one data set using the START command, your cataloged procedure could look like [Figure 79]

```plaintext
//GTABC PROC MEMBER=GTFPARM
//IEFPROC EXEC PGM=AHKGTF,REGION=2880K,TIME=1440,
  // PARM=('MODE=EXT,DEBUG=NO')
//IEFRDER DD DSNAME=SYS1.GTFTRC,UNIT=SYSDA,
  // SPACE=(4096,20),DISP=(NEW,KEEP)
//SYSLIB DD DSN=SYS1.PARMLIB(&MEMBER),DISP=SHR
//GTFOUT1 DD DSNAME=SYS1.TRACE1,UNIT=SYSDA,DISP=(NEW,KEEP)
//GTFOUT2 DD DSNAME=SYS1.TRACE2,UNIT=SYSDA,DISP=(NEW,KEEP)
//GTFOUT3 DD DSNAME=SYS1.TRACE3,UNIT=SYSDA,DISP=(NEW,KEEP)
```

[Figure 79. Example: altering one data set]

Enter `START GTABC,,,UNIT=TAPE`, to alter only the data set that IEFRDER defines.

If you want to alter the attributes of more than one data set with the START command, use the JCL statements in [Figure 79] in your cataloged procedure.

```plaintext
//GTABC PROC MEMBER=GTFPARM
//IEFPROC EXEC PGM=AHKGTF,REGION=2880K,TIME=1440,
  // PARM=('MODE=EXT,DEBUG=NO')
//SYSLIB DD DSN=SYS1.PARMLIB(&MEMBER),DISP=SHR
//GTFOUT1 DD DSNAME=&NAME1,UNIT=&DEVICE,DISP=&DSPS;
//GTFOUT2 DD DSNAME=&NAME2,UNIT=&DEVICE,DISP=&DSPS;
//GTFOUT3 DD DSNAME=&NAME3,UNIT=&DEVICE,DISP=&DSPS;
```

[Figure 80. Example: Altering More Than One Data Set]

Enter `START GTABC,,,DEVICE=TAPE`, to override the default value of the UNIT parameter for each output data set in your cataloged procedure.

See [z/OS MVS JCL Reference] for more information about using symbolic parameters in JCL statements.

Specifying or changing GTF trace options through system prompting

After you enter the START command, GTF issues message AHL100A or AHL125A which allows you to specify or change trace options. If the cataloged procedure or START command did not contain a member of predefined options, GTF issues message AHL100A, which allows you to enter trace options. If the procedure or command did include a member of predefined options, GTF identifies those
options by issuing the console messages AHL121I and AHL103I. Then you can accept the options, or reject and specify new options.

GTF allows overlapping of trace options when multiple instances are active. This sequence of messages appears as:

AHL121I trace option input indicated from member memname of PDS dsname
AHL103I trace options selected -
keywd=(value),...,keywd=(value)
keywd,keywd,...,keywd
AHL125A Respecify trace options or reply U

Note: If you specify NOPROMPT or NP on the START GTF command, the system will not issue message AHL125A to request the respecification of trace options or the continuation of initialization.

If you choose to reject any options in the member, you are rejecting all of the options specified in that member. Respecifying trace options does not modify the options in the data set member.

The format of the response is:
TRACE=trace option[,trace option]...

The trace options determine the amount of storage GTF requires. See “Determining GTF’s storage requirements” on page 237.

GTF will accept the trace options listed under “GTF trace options” on page 247.

Examples of starting GTF

In this topic you will find the following examples:
- “Starting GTF with a cataloged procedure and parmlib member”
- “Starting GTF with internal tracking” on page 242
- “Starting GTF with trace output to an existing data set on tape” on page 242
- “Starting GTF with trace options stored in SYS1.PARMLIB” on page 243
- “Starting GTF without trace options in a member” on page 244
- “Starting GTF to trace VTAM remote network activity” on page 244

Starting GTF with a cataloged procedure and parmlib member

Figure 81 on page 242 shows GTF started with a cataloged procedure that indicates the GTFPARM parmlib member. The trace options are specified in the parmlib member record. In this example, message AHL103I displays the options specified in the GTFPARM member: TRACE=SYSM, DSP, PCI, SRM, TRC, USR. This example shows the messages and the reply generated by the initial START command, and the GTFPARM specifications that are in effect. This instance of GTF can be recognized by the EXAMPLE 1 identifier.
Starting GTF with internal tracking

Figure 82 shows GTF, with identifier EXAMPLE2, started with MODE=INT. The trace data is maintained in virtual storage and is not recorded on an external device. In this example, you can override the trace options given in the supplied parmlib member:

```
START GTF.EXAMPLE2,,,(MODE=INT),DSN=NULLFILE
AHL121I TRACE OPTION INPUT INDICATED FROM MEMBER memname OF PDS dsname
AHL103I TRACE OPTIONS SELECTED -- SYSM,USR,TRC,DSP,PCI,SRM
00 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 00,U
AHL031I GTF INITIALIZATION COMPLETE
```

Figure 82. Example: Starting GTF with internal tracking

Starting GTF with trace output to an existing data set on tape

Figure 83 on page 243 shows how the START command is used to direct GTF trace output to an existing data set on tape rather than to an existing data set on a DASD. The device type and volume serial number are supplied. The disposition and name of the trace data set are changed from DISP=(NEW,KEEP) and DSNAME=SYS1.TRACE to DISP=(OLD,KEEP) and DSNAME=TPOUTPUT. The specified tape has a volume serial of TRCTAP and resides on a 3400 tape drive. Note that the GTFPARM parmlib member is used to specify the trace options.

Here the GTF keyword is not followed by a unique identifier and defaults to volume serial number.
Starting GTF with trace options stored in SYS1.PARMLIB

Figure 84 shows how to store trace options in a member of SYS1.PARMLIB. This can save you time when starting GTF. First store one or more combinations of trace options as members in SYS1.PARMLIB, and include a SYSLIB DD statement in the cataloged procedure. When you start GTF, GTF will then retrieve the trace options from SYS1.PARMLIB, instead of prompting you to supply them through the console. GTF displays the trace options for you, and then issues message AHL125A, to which you can reply U to accept the parmlib options.

This example shows the job control statements and utility JCL statements needed to add trace options to SYS1.PARMLIB using IEBUPDTE.

```plaintext
//GTFPARM JOB MSGLEVEL=(1,) EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=A //SYSPUT DD SYSPUT=A
//SYSPUT DD SYSOUT=SYS1.PARMLIB,DISP=SHR
//SYSIN DD DD NAME=GTFPARM,LIST=ALL,SOURCE=0
./ ADD NAME=GTFA,LIST=ALL,SOURCE=0
TRACE=SYSP,USR SVC=(1,2,3,4,10),IO=(D34,D0C),SSCH=ED8,PI=15
./ ADD NAME=GTFB,LIST=ALL,SOURCE=0
TRACE=IO,SSCH,TRC
./ ADD NAME=GTFC,LIST=ALL,SOURCE=0
TRACE=SYS,PCI
/*
```

Figure 84. Example: Starting GTF with trace options stored in SYS1.PARMLIB

A sample SYSLIB DD statement to be included in a GTF cataloged procedure might look like this:

```plaintext
//SYSLIB DD DSN=SYS1.PARMLIB(GTFA),DISP=SHR
```

The new member name can also be specified on the START command while using the IBM-supplied GTF procedure, as in the following example:

```
S GTF,,,(MODE=EXT,TIME=YES),MEMBER=GTFB
```

For more information see the following references:

- See z/OS DFSMSdfp Utilities for descriptions of the statements.
- See z/OS MVS JCL Reference for descriptions of the statements.
- See z/OS MVS Initialization and Tuning Reference for further information about SYS1.PARMLIB.
Starting GTF without trace options in a member

Figure 85 shows an installation-written procedure where there is no predefined member with trace options specified. The procedure contains no SYSLIB DD statement. When GTF is started with a procedure containing no SYSLIB DD statement, message AHL100A is issued to prompt for GTF trace options.

In this example, an installation-written cataloged procedure, USRPROC, is invoked to start GTF in external mode to a direct access data set, ABCTRC, on device 250. The trace options selected result in trace data being gathered for:

- All SVC and IO interruptions
- All SSCH operations
- All matching SLIP traps with a tracing action specified or SLIP traps in DEBUG mode
- All dispatcher events
- All issuers of the GTRACE macro will have their user data recorded in the trace buffers.

The trace data is written into the data set ABCTRC. (Note that when the end of the primary extent is reached, writing continues at the beginning.)

```
START USRPROC,250,333005,(MODE=EXT),DSN=ABCTRC
00 AHL100A SPECIFY TRACE OPTIONS
REPLY 00,TRACE=SVC,SSCH,IO,DSP,SLIP,USR
AHL103I TRACE OPTIONS SELECTED--USR,DSP,SVC,IO,SLIP,SSCH
01 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 01,U
AHL031I GTF INITIALIZATION COMPLETE
```

Figure 85. Example: Starting GTF without trace options in a member

Starting GTF to trace VTAM remote network activity

GTF can trace VTAM activity only if VTAM is started with the GTF option. See z/OS Communications Server: SNA Operation for details. In Figure 86 on page 245 GTF options are not stored in parmlib; they are entered at the console. Three GTF options are required to record all VTAM traces:

- RNIO must be specified so that the VTAM I/O trace can function for an NCP or a remote device attached to the NCP.
- IO or IOP must be specified so that the VTAM I/O trace can function for a local device.
- USR or USRP must be specified so that the VTAM buffer and the NCP line traces can function.

You must enter the START GTF command before a trace can be activated from VTAM.
Stopping GTF

You can enter the STOP command at any time during GTF processing. The amount of time you let GTF runs depends on your installation and the problem you are trying to capture, but a common time is between 15 and 30 minutes.

If you are running GTF to gather information related to a problem for which a SLIP trap has been defined, you can instruct SLIP to stop all instances of GTF when the trap conditions are satisfied. For additional information, see the SLIP Command documentation in z/OS MVS System Commands.

To stop GTF processing, enter the STOP command. The STOP command must include either the GTF identifier specified in the START command, or the device number of the GTF trace data set if you specified MODE=EXT or MODE=DEFER to direct output to a data set. If you have not specified the GTF identifier in the START command, then those instances of GTF will have the same identifier: the volume serial number.

If you are not sure of the identifier or the device number of the trace data set, enter the following command:

DISPLAY A,LIST

Figure 87 shows the output produced by the DISPLAY A,LIST command. In this example, the identifier for GTF is EVENT1.

You must enter the STOP command at a console with master authority. The general format of the STOP command is as follows:
For example, to stop GTF for the identifier EVENT1 (as shown in Figure 87 on page 245), enter the command:

\[ \text{STOP EVENT1} \]

When the STOP command takes effect, the system issues message AHL006I. If the system does not issue message AHL006I, then GTF tracing continues, remaining active until a STOP command takes effect or the next initial program load (IPL). When this happens, you will not be able to restart GTF tracing. In this case, you can use the FORCE ARM command to stop GTF.

If there were several functions started with the same identifier on the START command, using the same identifier on the STOP command will stop all those functions.

If the volume serial number is used on the STOP command, all instances of GTF with trace data directed to a data set on that volume serial are stopped. This is independent of the identifier assigned to each instance of GTF.

For example, if three instances of GTF are active with the identifiers EX1, EX2, and EX3 directing trace data to different data sets to the same volume with volume serial number 1020, then the following command will stop all the 3 instances of GTF.

\[ \text{STOP 1020} \]

See [z/OS MVS System Commands](#) for more information about the STOP and FORCE ARM commands.

You can also use an identifier to stop GTF. In this example, the following command starts GTF tracing with the identifier EXAMPLE and with trace data maintained in the GTF address space. The DSN keyword is entered to prevent allocation of an external trace data set as specified in the cataloged procedure.

\[ \text{START GTF.EXAMPLE,,,(MODE=INT),DSN=NULLFILE} \]

To stop GTF tracing, you would issue the following command:

\[ \text{STOP EXAMPLE} \]

In some instances, you may need to display the active jobs before stopping GTF. The example shown in Figure 88 starts GTF tracing with trace data recorded on an external device, data set GTF.TEST01. Another instance of GTF with an identifier EX1 is started with trace data directed to another data set on the same volume. Note that you do not have to specify MODE=EXT, because it is the default.

\[ \text{STOP|P} \text{ identifier} \]

Figure 88. Example: Starting instances of GTF

Because it is not apparent which is the GTF recording device, you have to display active jobs with the DISPLAY A,LIST command before stopping GTF. In Figure 89 on page 247, the device number for GTF is 0227.
If you only want to stop only the second instance of GTF, issue the following command:
STOP EX1

If you want to stop both instances, issue the following command:
STOP 227

GTF trace options

This topic describes the GTF options you can specify through either system prompting in response to the START GTF command or in a predefined parmlib or data set member. However, GTF will not use certain combinations of options; see Table 36 on page 252 for a list of those combinations.

Some GTF trace options also require keywords. If you specify options requiring keywords in the member or data set containing the predefined options, it must also contain the associated keywords. These are explained in “Prompting keywords” on page 252.

ASIDP
Requests that GTF tracing be limited to a subset of address spaces. ASIDP requests GTF prompting for one to five address space identifiers (ASID) in which you want GTF tracing to occur. ASIDP works only with a GTF option that generates tracing, such as SVC or IO. For information about responding to GTF prompts, see “Prompting keywords” on page 252.

CCW
Requests tracing of channel programs and associated data for I/O events. CCW is valid only if the other trace options include SSCH, SSCHP, IO, or IOP. See Table 36 on page 252.

CCWP
Requests tracing of channel programs and associated data for I/O events, and requests GTF prompting for the following information:
- Tracing channel command words (CCW) or device command words (DCW) for start subchannel (SSCH) operations or I/O interruptions or both
- Maximum number of CCWs or device command words (DCW) for each event
- Maximum number of bytes of data for each CCW or DCW
- Optional input/output supervisor block (IOSB), input/output block extension (IOBE, zHPF channel programs only), and error recovery procedure work area (EWA) tracing
- Size of the program controlled interrupt (PCI) table
For information about responding to GTF prompts, see “Prompting keywords” on page 252.
**Generalized Trace Facility**

CCWP is valid only if the other trace options include SSCH, SSCHP, IO, or IOP. See Table 36 on page 252.

CSCH
Requests recording for all clear subchannel operations. See Table 36 on page 252 for more information on combining this option with other GTF options.

DSP
Requests recording for all dispatchable units of work: service request block (SRB), local supervisor routine (LSR), task control block (TCB) and Supervisor Call (SVC) prolog dispatch events. If you specify both the SYSM and DSP trace options, GTF records minimal trace data for DSP. Otherwise, GTF records comprehensive trace data for DSP.

EXT
Requests comprehensive recording for all external interruptions. See Table 36 on page 252 for more information on combining this option with other GTF options.

HSCH
Requests recording for all halt subchannel operations. See Table 36 on page 252 for more information on combining this option with other GTF options.

IO
Requests recording of all non-program-controlled I/O interruptions. Unless you also specify the PCI trace option, GTF does not record program-controlled interruptions. See Table 36 on page 252 for more information on combining this option with other GTF options.

IOX
Requests recording of all non-program-controlled I/O interruptions providing a summary of a complete channel program for the I/O interruption in an I/O summary trace record. Unless you also specify the PCI trace option, GTF does not record program-controlled interruptions.

IOP
Requests GTF prompting for specific device numbers for which you want GTF to record non-program-controlled I/O interruptions. Unless you specify the PCI trace option, GTF does not record program-controlled interruptions. For information about responding to GTF prompts, see “Prompting keywords” on page 252.

IOXP
Requests GTF prompting for specific device numbers for which you want GTF to record non-program-controlled I/O interruptions providing a summary of a complete channel program for the I/O interruption in an I/O summary trace record. Unless you specify the PCI trace option, GTF does not record program-controlled interruptions. For more information on responding to GTF prompts, see “Prompting keywords” on page 252.

If an installation chooses to specify either IO or IOP in addition to IOX or IOXP, they will receive IOX records for DASD and tape devices and IO records for all other devices.

JOBNAMEP
Requests that GTF tracing be limited to a subset of jobs. JOBNAMEP requests GTF prompting for one through five job names for which you want GTF tracing to occur.
These job names can be generic, as well as specific, job names. If you want to specify generic job names, use * or % in the job name.

The asterisk is a placeholder for one or more valid job name characters, or indicates no characters. For example, if you enter JOBNAMEP=*MS*, GTF will process trace data for address spaces with job names IABMS01, IAMS, IMS, IMSA, IMS00012, and so on. However, if you enter JOBNAMEP=*MASTER*, that job name represents only the master address space.

The percent symbol is a placeholder for a single valid job name character. For example, if you enter JOBNAMEP=I%MS%%, GTF will process trace data for address spaces with job names IAMS01 and IXMSBC, but not job names IMS001 or I999MS. The combination %* is a placeholder for at least one character.

JOBNAMEP works only with a GTF option that generates tracing, such as SVC or IO. For information on responding to GTF prompts, see “Prompting keywords” on page 252.

**MSCH**
Requests recording for all modify subchannel operations. See Table 36 on page 252 for more information on combining this option with other GTF options.

**PCI**
Requests recording of intermediate status interruptions in the same format as other I/O trace records that GTF creates. Specifically, PCI causes GTF to record program-controlled I/O interruptions, initial status request interruptions, resume subchannel operation instruction, and suspend channel program interruptions. When you select specific devices as a result of prompting for I/O events (IOP trace option), GTF records intermediate status interruptions for only those devices. PCI is valid only when the other trace options include IO, IOP, SYS, SYSM, or SYSP.

**PCIE**
Requests tracing of PCI load and store instructions, adapter interrupts, and PCIE de-multiplexing requests.

**PFIDP**
Requests that GTF tracing of PCIE-related events be limited to a subset of the PCIE function identifiers (PFIDs). PFIDP requests GTF prompting for 1 to 256 PFIDs or PFID ranges in which you want GTF tracing to occur. PFIDP is only valid when the PCIE trace option is specified. For information on responding to GTF prompts, see “Prompting keywords” on page 252.

**PI**
Requests comprehensive recording for all program interruptions (0-255). See Table 36 on page 252 for more information on combining this option with other GTF options.

**PIP**
Requests GTF prompting for those interruption codes for which you want GTF to record program interruptions. For information about responding to GTF prompts, see “Prompting keywords” on page 252. See Table 36 on page 252 for more information on combining this option with other GTF options.

**RNIO**
Requests recording of all Virtual Telecommunications Access Method (VTAM) network activity. If you specify both the SYSM and RNIO trace options, GTF will record minimal trace data for RNIO. Otherwise, GTF records comprehensive trace data for RNIO.
Generalized Trace Facility

**RR**
Requests comprehensive recording of data associated with all invocations of recovery routines (such as STAE and ESTAE routines). GTF creates a trace record describing the activity of the recovery routine when control passes from the recovery routine back to the recovery termination manager (RTM). See Table 36 on page 252 for more information on combining this option with other GTF options.

**SIO|SIOP**
If you specify the SIO or SIOP trace option, GTF processes that request as a request for SSCH or SSCHP. GTF issues message AHL138I to indicate this substitution. Subsequent messages refer to the original SIO or SIOP trace option.

**Note:** The SIO keyword is provided only for compatibility; it is recommended that you use the SSCH keyword instead. The SIOP option is provided only for compatibility; it is recommended that you use the SSCHP option instead.

**SLIP**
Requests that a trace entry be made each time:
- A match occurs for a SLIP trap with ACTION=TRACE
- A SLIP trap with the SLIP DEBUG option is checked

The amount of data and the type of SLIP trace record to be built is specified on the SLIP command.

**SRM**
Requests recording of trace data each time the system resource manager (SRM) is invoked. If you specify both the SYSM and SRM trace options, GTF records minimal trace data for SRM. Otherwise, GTF records comprehensive trace data for SRM.

**SSCH**
Requests recording for start subchannel and resume subchannel operations. See Table 36 on page 252 for more information on combining this option with other GTF options.

**SSCHP**
Requests GTF prompting for the specific device numbers for which you want GTF to record start subchannel and resume subchannel operations. For information about responding to GTF prompts, see “Prompting keywords” on page 252. See Table 36 on page 252 for more information on combining this option with other GTF options.

**SVC**
Requests comprehensive recording for all SVC interruptions. See Table 36 on page 252 for more information on combining this option with other GTF options.

**SVCP**
Requests GTF prompting for those SVC numbers for which you want data recorded. For information about responding to GTF prompts, see “Prompting keywords” on page 252. See Table 36 on page 252 for more information on combining this option with other GTF options.

**SYS**
Requests recording of comprehensive trace data for all of the following:
- Clear subchannel operations
- External interruptions
Generalized Trace Facility

- Halt subchannel operations
- I/O interruptions
- Modify subchannel operations
- Program interruptions
- Recovery routines
- Start subchannel and resume channel operations
- SVC interruptions.

Because specifying SYS automatically causes GTF to trace all of these events, GTF will ignore the following trace options if you specify them in any form: CSCH, HSCH, MSCH, SSCH, EXT, IO, PI, RR, SVC. See Table 36 on page 252 for more information on combining this option with other GTF options.

SYSM
Requests recording of minimal trace data for the same events as SYS.

Because specifying SYSM automatically causes GTF to trace all of these events, GTF will ignore the following trace options if you specify them in any form: CSCH, HSCH, MSCH, SSCH, EXT, IO, PI, RR, SVC.

If you specify DSP, RNIO, or SRM in addition to SYSM, GTF produces minimal, rather than comprehensive, trace data for those events.

SYSP
Requests recording for the same events as the SYS option, but causes GTF to prompt for the selection of specific SVC, IO, SSCH, and PI events that you want recorded. For information about responding to GTF prompts, see “Prompting keywords” on page 252.

Because specifying SYSP automatically causes GTF to trace all of these events, GTF will ignore the following trace options if you specify them in any form: CSCH, HSCH, MSCH, SSCH, EXT, IO, PI, RR, SVC. See Table 36 on page 252 for more information on combining this option with other GTF options.

TRC
Requests recording of those trace events that are associated with GTF itself. Unless you request TRC, GTF will not trace these events. TRC works only with a GTF option that generates tracing, such as SVC or IO.

USR
Requests recording of all data that the GTRACE macro passes to GTF. You must specify USR or USRP to trace data from the GTRACE macro. Use USRP for specific events. If USR is used instead of USRP, the trace data set might be full of unwanted records. When you code the GTRACE macro but do not specify USR or USRP, GTF ignores the GTRACE macro. See Table 36 on page 252 for more information on combining this option with other GTF options.

Reference
See z/OS MVS Programming: Assembler Services Reference ABE-HSP for information about coding the GTRACE macro.

USRP
Requests GTF prompting for specific event identifiers (EID) of the data that the GTRACE macro passes to GTF. The EIDs represent user, program product, or IBM subsystem and component events. See Table 39 on page 258 for a list of EID values.
**Generalized Trace Facility**

See Table 36 for more information on combining this option with other GTF options. For information about responding to GTF prompts, see “Prompting keywords.”

**XSCH**

Requests recording all cancel subchannel operations.

See Table 36 for more information on combining this option with other GTF options. For information about responding to GTF prompts, see Table 37.

**Combining GTF options**

Table 36 shows those TRACE options that GTF will not use in combination. If two or more options from the same row are specified, GTF uses the option that has the lower column number and ignores the other options. For example, if you specify both SYSP and PI (see row D), GTF uses SYSP (column 2) and ignores PI (column 5).

**Table 36. Combining GTF options**

<table>
<thead>
<tr>
<th>Row</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>SYSM</td>
</tr>
<tr>
<td>B</td>
<td>SYSM</td>
</tr>
<tr>
<td>C</td>
<td>SYSM</td>
</tr>
<tr>
<td>D</td>
<td>SYSM</td>
</tr>
<tr>
<td>E</td>
<td>SYSM</td>
</tr>
<tr>
<td>F</td>
<td>SYSM</td>
</tr>
<tr>
<td>G</td>
<td>SYSM</td>
</tr>
<tr>
<td>H</td>
<td>SYSM</td>
</tr>
<tr>
<td>I</td>
<td>SYSM</td>
</tr>
<tr>
<td>J</td>
<td>SYSM</td>
</tr>
<tr>
<td>K</td>
<td>CCWP</td>
</tr>
<tr>
<td>L</td>
<td>USRP</td>
</tr>
</tbody>
</table>

If an installation chooses to specify either IO or IOP in addition to IOX or IOXP, they will receive IOX records for DASD and tape devices and IO records for all other devices.

**Prompting keywords**

When you specify any of the trace options listed in Table 37, GTF prompts for specific values by issuing message AHL101A:

AHL101A SPECIFY TRACE EVENT KEYWORDS - keyword=,...,keyword=

The keywords issued in the message correspond to the trace options specified. Enter only the trace event keywords appearing in the message text. The trace options and their corresponding keywords are:

**Table 37. GTF trace options and corresponding prompting keywords**

<table>
<thead>
<tr>
<th>Trace Option</th>
<th>Prompting Keyword</th>
<th>Number of Prompting Values Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIDP</td>
<td>ASID=</td>
<td>5</td>
</tr>
<tr>
<td>CCWP</td>
<td>CCW=</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Table 37. GTF trace options and corresponding prompting keywords (continued)

<table>
<thead>
<tr>
<th>Trace Option</th>
<th>Prompting Keyword</th>
<th>Number of Prompting Values Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOP, IOXP, SYSP</td>
<td>IO=SSCH=</td>
<td>Unlimited</td>
</tr>
<tr>
<td>IOP, IOXP, SSCHP, SYSP</td>
<td>IO=SSCH=</td>
<td>Unlimited</td>
</tr>
<tr>
<td>JOBNAMEP</td>
<td>JOBNAME=</td>
<td>5</td>
</tr>
<tr>
<td>PFIDP</td>
<td>PFID=</td>
<td>256</td>
</tr>
<tr>
<td>PIP, SYSP</td>
<td>PI=</td>
<td>50</td>
</tr>
<tr>
<td>SSCHP, SIO, SYSP</td>
<td>SIO=</td>
<td>Unlimited</td>
</tr>
<tr>
<td>SSCHP, SIO, SYSP</td>
<td>SSCH=</td>
<td>Unlimited</td>
</tr>
<tr>
<td>SVCP, SYSP</td>
<td>SVC=</td>
<td>50</td>
</tr>
<tr>
<td>USRP</td>
<td>USR=</td>
<td>50</td>
</tr>
</tbody>
</table>

Note:
1. The SIO keyword is provided only for compatibility; it is recommended that you use the SSCH keyword instead. The SIOF option is provided only for compatibility; it is recommended that you use the SSCHP option instead.
2. Tracing a PAV base device number will cause all PAV aliases associated with that base device number to also be traced. If I/O tracing is needed to locate issues related to PAV alias device numbers when they are not associated with a PAV base device number, specify the device numbers of the PAV alias devices explicitly.

Guidelines for specifying values for prompting keywords: Use the following guidelines when replying to message AHL101A for prompting keywords:

- If you do not specify a reply for each of the keywords displayed in message AHL101A, GTF records all the events for that trace option, which increases the amount of storage that GTF requires. IBM recommends that you specify values for each keyword displayed, selecting the values that will help you debug your problem.

- You can only enter values for keywords displayed in message AHL101A.

- GTF limits the number of specific values that you can supply through prompting, see Table 37 on page 252 for the maximum number of values allowed for each keyword. If you specify more than the maximum values, GTF issues a message to which you reply by respecifying values for all appropriate keywords.

- Keep in mind that prompting increases the amount of storage that GTF requires, because storage requirements depend on the trace options you specify. See "Determining GTF's storage requirements" on page 237 for further information.

- Within a given reply, each keyword that you specify must be complete. If you need more values for a keyword than will fit into one reply, repeat the keyword in the next reply, and code the additional values for that keyword. The following are examples of correct replies:
  - REPLY 01 10=(191-193), SVC=(1,2,3,4,5)
  - REPLY 01 SVC=(6,7,8,9,10)

  The maximum number of values that GTF allows for a keyword does not change, regardless of whether you enter one or more replies to specify all the values for the keyword.

- After supplying all keywords and values, you must enter the END keyword, which signifies that the event definition is complete. If the system does not find the END keyword in a reply, the system issues message AHL102A to prompt for
additional event keywords and values. When the system finds the END keyword, the system issues message AHL103I to list all of the trace options that are in effect.

For sample prompting sequences, see "Examples of sample prompting sequences" on page 258.

Use the following keywords when GTF prompts for values by issuing message AHL101A:

**ASID=(asid1[,asidn]...[,asid5])**

Specifies one through five identifiers for address spaces in which you want GTF tracing to occur. The values ‘asid1’ through ‘asid5’ are hexadecimal numbers from X’0001’ to the maximum number of entries in the address space vector table (ASVT). If you specify ASIDP, but do not specify ASID= before replying END, GTF traces all address space identifiers.

If the number of values for ASIDP requires more than one line, and a particular ASID value is incorrect, GTF allows you to respecify the correct value without having to reenter all ASIDs.

If you specify both ASIDP and JOBNAMEP, GTF will trace address spaces that ASIDP did not identify, if some of the jobs that JOBNAMEP identifies run in other address spaces.

**CCW=(\(S|I|SI\)[,CCWN=nnnnn][,DATA=nnnnn][,IOSB][,PCITAB=n])**

Specifies different options for tracing channel programs. If you specify CCW= more than once, GTF uses the last specification of CCW=.

If you specify CCWP, but do not specify a value for keyword CCW=, GTF’s default CCW tracing depends on what other trace options were specified. The following table shows the defaults for CCW tracing depending on other trace options specified:

<table>
<thead>
<tr>
<th>Other Trace Options Selected</th>
<th>CCW Subparameter Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSCH or SSCHP</td>
<td>S</td>
</tr>
<tr>
<td>IO or IOP or IOX or IOXP</td>
<td>I</td>
</tr>
<tr>
<td>SSCH or SSCHP or IO or IOP or IOX or IOXP</td>
<td>SI</td>
</tr>
<tr>
<td>PCI</td>
<td>PCITAB=1</td>
</tr>
<tr>
<td>SSCH or SSCHP or IO or IOP or IOX or IOXP</td>
<td>CCWN=50</td>
</tr>
<tr>
<td>SSCH or SSCHP or IO or IOP or IOX or IOXP</td>
<td>DATA=20</td>
</tr>
</tbody>
</table>

Examples:

TRACE=IO,CCWP \> CCW defaults to: CCW=(I,CCWN=50,DATA=20)
TRACE=IOP,SSCH,PCI,CCWP \> CCW defaults to: CCW=(SI,CCWN=50,DATA=20,PCITAB=1)

If you specify an option more than once in one line, GTF uses the last specification of that option. An exception is that GTF uses the first specification of S, I, or SI. If a line contains an error, GTF prompts you to respecify the value.

**S|I|SI**

Specifies the type of I/O event for which you want channel programs traced. If you specify more than one option, GTF uses the first option.
Generalized Trace Facility

S Specifies GTF tracing of channel programs for start subchannel and resume subchannel operations. CCW=S works only with the SSCH or SSCHP trace options.

I Specifies GTF tracing of channel programs for I/O interruptions, including program-controlled interruptions if you specify PCI as a trace option. CCW=I works only with the IO or IOP trace options.

SI Specifies GTF tracing of channel programs for start subchannel and resume subchannel operations and I/O interruptions. CCW=SI works only with either SSCH or SSCHP and either IO or IOP as trace options.

**CCWN=nnnnn**
Specifies the maximum number of CCWs or DCWs traced for each event. The value *nnnnn* is any decimal number from 1 to 32767. The default is 50.

**DATA=nnnnn**
Specifies the maximum number of bytes of data traced for each CCW or DCWs. The value *nnnnn* is any decimal number from 0 to 32767. The default is 20.

GTF traces *nnnnn* bytes of data for each CCW on the data chain. GTF traces *nnnnn* bytes of data for each word in an indirect data addressing word (IDAW) list, a modified indirect addressing word (MIDAW) list, or a transport indirect addressing word (TIDAW) list.

For start subchannel or resume subchannel operations, GTF does not trace data for read, read backwards, or sense commands in the channel programs. If no data is being transferred, regardless of the type of I/O operation, GTF does not trace data for read, read backwards, or sense commands.

When the data count in the CCW or DCW is equal to or less than *nnnnn*, GTF traces all data in the data buffer. When the data count in the CCW or DCW is greater than *nnnnn*, GTF traces data only from the beginning and end of the data buffer. If you examine the traced data, you can tell whether the channel completely filled the buffer on a read operation.

GTF uses a different CCW or DCW tracing method for a data transfer that is in progress when an I/O interruption occurs. Instead of using the data count in the CCW or DCW, GTF tracing depends on the transmitted data count. The transmitted data count is the difference between the data count in the CCW or DCW and the residual count in the subchannel status word (SCSW), for non-zHPF channel programs and the transport status block (TSB), for zHPF channel programs.

- When the residual count is greater than the data count in the CCW or DCW, then GTF traces all of the data in the CCW or DCW.
- When the transmitted data count is less than or equal to *nnnnn*, GTF traces all of the transmitted data.
- When the transmitted data count is greater than *nnnnn*, GTF traces data only from the beginning and end of the transmitted data.

**IOSB**
Specifies tracing of the input/output supervisor block (IOSB), the input/output block extension (IOBE), for zHPF channel programs, and, if available, the error recovery procedure work area (EWA), for all CCW events. If you do not specify IOSB, then GTF performs IOSB and EWA tracing only if GTF encounters an exceptional condition when tracing a channel program.
Generalized Trace Facility

**PCITAB=n**

Specifies a decimal number of 100-entry increments for GTF to allocate in an internal program-controlled interruption (PCI) table. The value of n is an integer from 1 to 9. The default is 1 (100 entries).

The PCI table keeps track of the channel programs that use PCI. One entry in the PCI table contains information about a program-controlled interruption in one channel program. An entry in the PCI table includes a CCW address and an IOSB address.

**IO=(DEVCLASS=xxxx,DEVCLASS=xxxx,devnum1 [,devnum...,devnum])**

Specifies devices for which you want I/O interruptions traced. Devices are specified by entering a device number or a device class.

The device number must be specified in hexadecimal and is not the same as the subchannel number. If you specify any combination of IO= and SSCH=, and IO=SSCH=, the combined number of device numbers for all prompting keywords is unlimited. Specify device numbers individually, or as a range of device numbers with a dash (-) or colon (:) separating the lowest and highest number in the range. For example, to trace I/O interruptions for device numbers 193 through 198, you specify IO=(193-198).

The device class must be specified with the DEVCLASS= keyword parameter, which provides the ability to trace all devices in the specified device class. The allowable keyword parameters are:

- **TAPE** (magnetic tape devices)
- **COMM** (communications)
- **DASD** (direct access storage device)
- **DISP** (display)
- **UREC** (unit/record)
- **CTC** (channel to channel)

If you specify IOP, IOXP, or SYSP and does not specify IO= in the response to the prompting messages, GTF processing proceeds as if you specified IO, IOX or SYS event keywords respectively.

For the following examples, the I/O device numbers and associated device types listed below are used:

**I/O Device Number**

<table>
<thead>
<tr>
<th>Device Type</th>
<th>I/O Device Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>3390 DASD</td>
<td>230</td>
</tr>
<tr>
<td>3490 Tape Drive</td>
<td>450</td>
</tr>
<tr>
<td>3480 Tape Drive</td>
<td>575</td>
</tr>
<tr>
<td>3380 DASD</td>
<td>663</td>
</tr>
<tr>
<td>3274 Communications Controller</td>
<td>020</td>
</tr>
</tbody>
</table>

In this example, the resulting trace includes information for all DASD devices and one 3490 tape drive at address 450.

**IO=(DEVCLASS=DASD,450)**

In the following example, the resulting trace includes information for all DASD and TAPE devices and the communications controller at address 020.

**IO=(DEVCLASS=DASD,DEVCLASS=TAPE,020)**

In this example, the resulting trace includes information for the devices at addresses 450, 575, and 663.
IO=(450-663)

IO=SSCH=(DEVCLASS=xxxx,DEVCLASS=xxxx,devnum1[,devnumm....[,devnum])

Specifies devices for which you want both I/O interruptions and start subchannel operations traced. See the IO= prompting keyword for a description of how to specify devices to be traced.

JOBNAME=(jobname1[,jobnamen]...[,jobname5])

Specifies one through five job names for which you want GTF tracing to occur. The values job1 through job5 must be valid job names. These job names can be generic, as well as specific, job names. If you want to specify generic job names, you must use * or % in the job name. The asterisk is a placeholder for one or more valid job name characters, or indicates no characters. For example, if you enter JOBNAMEP=IMS*, GTF will process trace data for address spaces with job names IABMS01, IAMS, IMS, IMSA, IMS00012, and so on. However, if you code JOBNAMEP=*MASTER*, that job name represents only the master address space. The percent symbol is a placeholder for a single valid job name character. For example, if you code JOBNAMEP=I%MS%%, GTF will process trace data for address spaces with job names IAMS01 and IXMSBC, but not job names IMS001 or I999MS. The combination %* is a placeholder for at least one character.

If you specify JOBNAME, but do not specify JOBNAME before replying END, GTF traces all job names.

If the number of values for JOBNAME requires more than one line, and a particular job name value is incorrect, GTF allows you to re-specify the correct value without having to reenter all job names.

If you specify both ASIDP and JOBNAMEP, GTF will trace jobs that JOBNAMEP did not identify, if some of the address spaces that ASIDP identifies contain jobs that JOBNAMEP did not identify.

PFID=(pfid1[,pfidn]...[,pfid256])

Specifies 1 to 256 PFIDs or PFID ranges for which you want GTF tracing to occur. The values are hexadecimal numbers from 0 to FFFFFFFF. Leading zeroes are allowed. For example: 1, 01, and 00000001 are all accepted. If a PFID range is specified, the first number in the range must be lower than the second number in the range.

Duplicate PFID values are ignored. Overlapping PFID ranges are accepted without issuing an error message.

If you specify PFIDP, but do not specify PFID= before replying END, GTF traces all PFIDs. If the number of values for PFIDP requires more than one line, and a particular PFID value is incorrect, GTF allows you to re-specify the correct value without having to reenter all PFIDs.

PI=(code0[,codeen]...[,code50])

Specifies 1 through 50 program interruption codes, in decimal notation, that you want traced. If you specify PIP or SYSP, and do not specify PI= in response to this prompting message, GTF traces all program interruptions.

SSCH=(DEVCLASS=xxxx,DEVCLASS=xxxx,devnum1[,devnumm....,devnum])

Specifies devices for which you want start subchannel operations traced. See the IO= "Prompting keywords" on page 252 for a description of how to specify devices to be traced.
**Generalized Trace Facility**

\[ \text{SVC} = \{svcnm1[,svcnm2],...,svcnm50] \}\]

Specifies 1 through 50 SVC numbers, in decimal notation, that you want traced. If you specify SVC or SYSP, and do not specify SVC= in response to the prompting message, GTF traces all SVC numbers. Both SVC entry and exit are be recorded.

\[ \text{USR} = \{event1[,eventn],...,event50] \}\]

Specifies 1 through 50 user event identifiers (EIDs) for which you want user data traced. The values for USR are three-digit hexadecimal numbers, as follows:

<table>
<thead>
<tr>
<th>Identifier (Hex)</th>
<th>Type of Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>000-3FF</td>
<td>User</td>
</tr>
<tr>
<td>400-5FF</td>
<td>Reserved for program products</td>
</tr>
<tr>
<td>600-FFF</td>
<td>Reserved for IBM subsystems and components</td>
</tr>
</tbody>
</table>

If you specify USRP and do not specify USR= in response to the prompting message, all instances of GTRACE issued with TEST=YES will return with an indication that tracing is not active.

**Examples of sample prompting sequences**

This example shows how to store prompting keywords in a SYS1.PARMLIB member.

If you start GTF with options requiring prompting keywords stored in SYS1.PARMLIB, these prompting keywords must also appear in the parmlib member. If prompting keywords are used in the parmlib member without the replies included, GTF will not obtain the replies from the console. GTF will use the options without the prompting (for example, SVCP becomes SVC). A SYSLIB DD statement in your cataloged procedure causes GTF to read the prompting keywords from the specified parmlib member. The second and subsequent logical records in the member should contain only the prompting keywords.

GTF uses either the END keyword, or end-of-file on the member as the indicator that there is no more prompting input from parmlib. If the number of events for one keyword require more than one record, respell the keyword in a subsequent prompting record with the additional events, as follows:

- **Record #1** \( \text{TRACE}=\text{IOP,SVCP,SSCH} \)
- **Record #2** \( \text{IO}=(D34,D0C),\text{SVC}=(1,2,3) \)
- **Record #3** \( \text{SVC}=(4,5,6,7,8,9,10),\text{END} \)

Do not respell the keyword through the system console at this point, because GTF will then override all of the options and keywords in the parmlib member.

When GTF finishes reading the options and prompting keywords in the parmlib member, it displays the options through message AHL103I:

- **AHL103I** TRACE OPTIONS SELECTED--IOP,SVCP,SSCH
- **AHL103I** IO=(D34,D0C),SVC=(1,2,3,4,5,6,7,8,9,10)

This message may be a multiple-line message, depending on the number of options you select. If the set of devices specified for IO= and SSCH= are identical, message AHL103I will show them as if specified by use of IO=SSCH.
Generalized Trace Facility

After GTF displays all of the options specified, you then have the opportunity to accept the parmlib options, or completely change the options by respecifying them through the console by replying to the following message:

AHL125A RESPECIFY TRACE OPTIONS OR REPLY U.

In Figure 90, you started GTF in external mode to the data set defined in the cataloged procedure. You selected two trace options in reply 00:

- SYSP requests that GTF trace specific system event types.
- USRP requests that GTF trace specific user entries that the GTRACE macro generates.

Message AHL101A instructed you to specify values for the SVC, IO, SSCH, PI, and USR keywords. In reply 01 to message AHL101A, you selected:

- Five SVCs
- Two devices for non-program-controlled I/O interruptions
- One device for SSCH operations
- Three user event identifiers.

GTF does not record any other SVC, IO, and SSCH events. Because you did not specify any program interruption codes for PI=, GTF would trace all program interruptions.

In Figure 90 you started GTF in external mode to the data set defined in the cataloged procedure. You selected two trace options in reply 00:

- SYSP requests that GTF trace specific system event types.
- USRP requests that GTF trace specific user entries that the GTRACE macro generates.

Message AHL101A instructed you to specify values for the SVC, IO, SSCH, PI, and USR keywords. In reply 01 to message AHL101A, you selected:

- Five SVCs
- Two devices for non-program-controlled I/O interruptions
- One device for SSCH operations
- Three user event identifiers.

GTF does not record any other SVC, IO, and SSCH events. Because you did not specify any program interruption codes for PI=, GTF would trace all program interruptions.

START MYPROC.EXAMPLE7..,(MODE=EXT)

00 AHL100A SPECIFY TRACE OPTIONS
REPLY 00,TRACE=SYSP,USR

01 AHL101A SPECIFY TRACE EVENT KEYWORDS--IO=,SSCH=,SVC=,PI=,USR=

01 AHL101A SPECIFY TRACE EVENT KEYWORDS--IO=

REPLY 01,SVC=(1,2,3,4,10),IO=(191,192),USR=(10,07A,AB)

02 AHL102A CONTINUE TRACE DEFINITION OR REPLY END
REPLY 02,SSCH=282,END

AHL103I TRACE OPTIONS SELECTED--SYSP,P1=,(191,192),SSCH=(282)
AHL103I SVC=(1,2,3,4,10),USR=(010,07A,0AB)

03 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 03,U

Figure 90. Example: Specifying prompting trace options SYSP and USRP

In Figure 91 on page 260 you started GTF in external mode, using the trace options defined in the data set specified in the cataloged procedure. You are prompted for the information as follows:

- Message AHL100A prompts for trace options.
- In reply 00, you selected six trace options: SSCHP, IOP, PCI, CCWP, SVC, and JOBNAMEP.
- Message AHL101A prompts to specify values for the IO, SSCH, CCW and JOBNAME prompting keywords.
- In reply 01, you select one device for tracing both IO and SSCH events and limit GTF tracing to one job.
In reply 02, you specify five options for CCW tracing.

The final result of these specifications is that GTF traces CCWs for both start subchannel operations and I/O interruptions at device 580 for the job BACKWARD, and all SVCs in BACKWARD’s address space. GTF would allocate 200 entries in the PCI table, and trace up to 100 CCWs or DCWs, up to 40 bytes of data for each CCW or DCW, and the IOSB, IOBE, and EWA.

START USRPROC,,(MOD=EXT)
00 AHL100A SPECIFY TRACE OPTIONS
REPLY 00, TRACE=SSCHP,IOP,PCI,CCWP,SVC,JOBNAMEP
01 AHL101A SPECIFY TRACE EVENT KEYWORDS
--IO=,SSCH=,CCW=,JOBNAME=,IOP=SSCH=
REPLY 01,JOBNAME=(BACKWARD),IOP=SSCH=580
02 AHL102A CONTINUE TRACE DEFINITION OR REPLY END
REPLY 02,CCW=(CCWN=100,DATA=40,PCITAB=2,IOSB,SI),END
AHL103I TRACE OPTIONS SELECTED--PCI,SVC,IO=SSCH=(580)
AHL103I CCW=(SI,IOSB,CCWN=100,DATA=40,PCITAB=2)
AHL103I JOBNAME=(BACKWARD)
03 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 03,U

Figure 91. Example: Specifying prompting trace options

Receiving GTF traces

GTF writes trace data in GTF trace tables in the GTF address space in storage. GTF trace data in storage are printed or viewed as part of a dump, if the dump options list includes TRT to request trace data. The following table shows the dumps that have TRT in their default options. For unformatted dumps that are printed or viewed through IPCS, format the trace data by specifying the IPCS GTFTRACE subcommand or using the IPCS Trace Processing selection panel.

To format and print GTF trace data in a GTfoutxx or IEFRDER data set, specify the IPCS GTFTRACE subcommand or use the IPCS Trace Processing selection panel.

If the GTF data was created for VTAM diagnosis, you can use the ACF/TAP program to format the VTAM data.

<table>
<thead>
<tr>
<th>Dump</th>
<th>How to obtain trace data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABEND dump to SYSABEND</td>
<td>Default</td>
</tr>
<tr>
<td>ABEND dump to SYSDUMP</td>
<td>Not available</td>
</tr>
<tr>
<td>ABEND dump to SYSUDUMP</td>
<td>Request SDATA=TRT</td>
</tr>
<tr>
<td>SNAP dump</td>
<td>Request SDATA=TRT</td>
</tr>
<tr>
<td>Stand-alone dump</td>
<td>Default</td>
</tr>
<tr>
<td>SVC dump for SDUMP or SDUMPX macro</td>
<td>Default</td>
</tr>
</tbody>
</table>
## Generalized Trace Facility

<table>
<thead>
<tr>
<th>Dump</th>
<th>How to obtain trace data</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC dump for DUMP operator command</td>
<td>Default</td>
</tr>
<tr>
<td>SVC dump for SLIP operator command with ACTION=SVCD, ACTION=STDUMP, ACTION=SYNCSVCD, or ACTION=TRDUMP</td>
<td>Default</td>
</tr>
<tr>
<td>Any dump customized to exclude trace data</td>
<td>Request SDATA=TRT</td>
</tr>
</tbody>
</table>

For more information, see the following references:
- See [z/OS MVS IPCS Commands](#) for the GTFTRACE subcommand.
- See [z/OS MVS IPCS User’s Guide](#) for the panel interface.

### Combining, extracting, and merging GTF trace output

GTF trace data can be combined with other data or extracted from dumps and data sets using two IPCS subcommands: COPYTRC and MERGE.

Use consolidated or merged trace output to show the chronology of events around the time of an error. Specify start and stop times for the merge to see events beginning a little before the error occurred and ending a little after. On the CTRACE and GTFTRACE subcommands, specify the jobs and address space identifiers (ASID) involved, so that the merged output contains only pertinent trace records.

Merging is most useful when several components are running traces; the system can also be running a GTF trace. Each component puts its trace records into its own buffers independently. GTF is independent from all of the component traces. You can merge these separate records into one chronological sequence to make diagnosis easier.

See [z/OS MVS IPCS Commands](#) for more information about COPYTRC and MERGE.

### Combining and extracting GTF output

Use the IPCS COPYTRC subcommand to do one or more of the following:
- Consolidate GTF trace data into one data set from:
  - Multiple GTF data sets
  - Multiple GTF data sets, dumps, or both
  - More than one system
- Extract GTF trace data from SVC dumps and stand-alone dumps
- Extract from merged data the GTF trace data for a specified list of systems

If you have GTF set up to write data for one system to multiple data sets, you can use the IPCS COPYTRC subcommand to consolidate the data into one data set. You should do this before you consolidate GTF data from multiple systems with the MERGE or COPYTRC subcommands.

[Figure 92 on page 262](#) shows an example of a GTF cataloged procedure with 3 data sets defined for GTF data from system SYS01.
From IPCS, issue the following command to consolidate the data from the data sets defined in the cataloged procedure into one data set, GTF.SYS01:

```
COPYTRC TYPE(GTF)
  INDATASET(SYS01.DSN1,SYS01.DSN2,SYS01.DSN3)
  OUTDATASET(GTF.SYS01)
```

In Figure 93, the COPYTRC subcommand is used to consolidate data from 3 systems, in data sets GTF.SYS01, GTF.SYS02, and GTF.SYS03, into one output data set, GTF.ALLSYS.

```
COPYTRC TYPE(GTF)
  INDATASET(GTF.SYS01,GTF.SYS02,GTF.SYS03)
  OUTDATASET(GTF.SYS01)
```

In Figure 93, the COPYTRC subcommand is used to consolidate data from 3 systems, in data sets GTF.SYS01, GTF.SYS02, and GTF.SYS03, into one output data set, GTF.ALLSYS.

```
COPYTRC TYPE(GTF)
  INDATASET(GTF.SYS01,GTF.SYS02,GTF.SYS03)
  OUTDATASET(GTF.SYS01)
```

Merging trace output

Use the IPCS MERGE subcommand to merge multiple traces into one chronological sequence. The traces can be all of the following:

- Component traces from the same dump on direct access storage (DASD)
- Component traces from different dumps on DASD
- GTF trace records from a dump or data set and on tape or DASD

In Figure 94, the MERGE subcommand is used to consolidate and format data from 3 systems, in data sets GTF.SYS04, GTF.SYS05, and GTF.SYS06, into one chronological sequence in output data set, GTF.SYSALL.

```
MERGE
GTFTRACE DSNAME(GTF.SYS04)
GTFTRACE DSNAME(GTF.SYS05)
GTFTRACE DSNAME(GTF.SYS06)
MERGEEEND
```

Figure 94. Example: Merging GTF output from multiple systems
Reading GTF output

This topic shows the format of the trace records that GTF creates. When you select your tracing options carefully, GTF provides detailed information about the system and user events where your problem lies, making it easier to diagnose.

This section contains the following topics:

- “Formatted GTF trace output” on page 264 which has information about trace records formatted by the IPCS GTFTRACE subcommand.
- “Unformatted GTF trace output” on page 320 which has information about unformatted trace records.

“Trace options” lists the GTF trace options and the trace records they generate in GTF trace output. Use this list to correlate the options you selected with their associated trace records. Some trace options in the table do not have trace records associated with them:

- ASIDP - Specifies that GTF trace only events from the select address spaces.
- JOBNAMEP - Specifies that GTF trace only events in selected jobs.
- END - Specifies the end of prompting keyword values specified.
- TRC - Specifies that GTF tracing includes the GTF address space.

### Trace options

<table>
<thead>
<tr>
<th>Trace Options</th>
<th>Trace Record Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIDP</td>
<td>N/A</td>
</tr>
<tr>
<td>CCW</td>
<td>CCW, TCW</td>
</tr>
<tr>
<td>CCWP</td>
<td>CCW, TCW</td>
</tr>
<tr>
<td>CSCH</td>
<td>CSCH</td>
</tr>
<tr>
<td>DSP</td>
<td>DSP, LSR, SDSP, SRB</td>
</tr>
<tr>
<td>END</td>
<td>N/A</td>
</tr>
<tr>
<td>EXT</td>
<td>EXT</td>
</tr>
<tr>
<td>HSCH</td>
<td>HSCH</td>
</tr>
<tr>
<td>IO</td>
<td>EOS, INTG, IO, IOCS</td>
</tr>
<tr>
<td>IOX</td>
<td>IOX</td>
</tr>
<tr>
<td>IOP</td>
<td>EOS, INTG, IO, IOCS</td>
</tr>
<tr>
<td>IOXP</td>
<td>IOXP</td>
</tr>
<tr>
<td>JOBNAMEP</td>
<td>N/A</td>
</tr>
<tr>
<td>MSCH</td>
<td>MSCH</td>
</tr>
<tr>
<td>PCI</td>
<td>PCI</td>
</tr>
<tr>
<td>PCIe</td>
<td>PCILG, PCISTG, ADINT, PCIDMX</td>
</tr>
<tr>
<td>PFIDP</td>
<td>N/A</td>
</tr>
</tbody>
</table>
**Formatted GTF trace output**

This topic describes GTF trace output records formatted by the IPCS GTFTRACE subcommand. In each formatted record, the length of each field is indicated by the number of characters. The characters indicate the type of data in the field, as follows:

- c Character
- d Decimal
- h Hexadecimal
- x Variable information
- y Variable information

The CCW trace record format uses additional letters to distinguish parts of fields.

A trace record can contain indicators to denote unusual conditions that occurred while GTF was tracing the event for the record. The indicators are:

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Not applicable. The field does not apply in this record. In a 2-byte field, not applicable appears as N/.</td>
</tr>
</tbody>
</table>
Generalized Trace Facility

| U/A    | Unavailable. GTF could not gather the information. In a 2-byte field, unavailable appears as U/. |
| PPPPPP | Unavailable because of a page fault encountered while GTF was gathering the data (SVC only). |
| SSSSSSS | Unavailable because of security considerations (SVC only). |
| ******* | Unavailable because of an error that occurred while GTF was gathering the data or due to the data being paged out. |
| X'EEEE' | Unavailable because of a severe error that occurred while GTF was gathering the data. This value appears in the first 2 data bytes of the trace record. The contents of the trace record are unpredictable. |

Trace record identifiers

Each trace record has an identifier to indicate the type of record. Table 40 lists the identifiers alphabetically and gives the page that shows the format for the record.

Table 40. Summary of trace record identifiers

<table>
<thead>
<tr>
<th>Trace Record Identifier</th>
<th>GTF Trace Record Parameter in SYS1.PARMLIB Member or Operator Reply</th>
<th>For format, see</th>
</tr>
</thead>
<tbody>
<tr>
<td>****</td>
<td>Time stamp</td>
<td>“Time stamp records” on page 269</td>
</tr>
<tr>
<td>****</td>
<td>Lost event</td>
<td>“Lost event records” on page 270</td>
</tr>
<tr>
<td>ADINT</td>
<td>Adapter interruption PCIE</td>
<td>“ADINT trace records” on page 270</td>
</tr>
<tr>
<td>CCW</td>
<td>Non-zHPF channel program CCW</td>
<td>“CCW trace records” on page 271</td>
</tr>
<tr>
<td>CSCH</td>
<td>Clear subchannel operation CSCH, SYS, SYSM, SYSP</td>
<td>“CSCH and HSCH trace records” on page 273</td>
</tr>
<tr>
<td>DSP</td>
<td>Task dispatch DSP</td>
<td>“DSP and SDSP trace records” on page 274</td>
</tr>
<tr>
<td>EOS</td>
<td>End-of-sense interruption IO, IOP, SYS, SYSM, SYSP</td>
<td>“EOS, INTG, IO, IOCS, and PCI trace records” on page 276</td>
</tr>
<tr>
<td>EXT</td>
<td>General external interruption EXT, SYS, SYSM, SYSP</td>
<td>“EXT trace records” on page 280</td>
</tr>
<tr>
<td>FRR</td>
<td>Functional recovery routine return RR, SYS, SYSM, SYSP</td>
<td>“FRR trace records” on page 282</td>
</tr>
<tr>
<td>HEXFORMAT</td>
<td>Unformatted trace event</td>
<td>“HEXFORMAT, SUBSYS, and SYSTEM trace records” on page 283</td>
</tr>
<tr>
<td>HSCH</td>
<td>Halt subchannel operation HSCH, SYS, SYSM, SYSP</td>
<td>“CSCH and HSCH trace records” on page 273</td>
</tr>
<tr>
<td>INTG</td>
<td>Interrogate input/output interruption IO, IOP, SYS, SYSM, SYSP</td>
<td>“EOS, INTG, IO, IOCS, and PCI trace records” on page 276</td>
</tr>
<tr>
<td>IO</td>
<td>Input/output interruption IO, IOP, SYS, SYSM, SYSP</td>
<td>“EOS, INTG, IO, IOCS, and PCI trace records” on page 276</td>
</tr>
<tr>
<td>IOCS</td>
<td>Input/output interruption with concurrent sense IO, IOP, SYS, SYSM, SYSP</td>
<td>“EOS, INTG, IO, IOCS, and PCI trace records” on page 276</td>
</tr>
<tr>
<td>IOX</td>
<td>Input/output interruption summary record format IOX, IOXP, SYS, SYSM, SYSP</td>
<td>“IOX trace records” on page 284</td>
</tr>
<tr>
<td>LSR</td>
<td>Local supervisor routine dispatch DSP</td>
<td>“LSR trace records” on page 287</td>
</tr>
</tbody>
</table>

Chapter 10. The Generalized Trace Facility (GTF) 265
## Generalized Trace Facility

### Table 40. Summary of trace record identifiers (continued)

<table>
<thead>
<tr>
<th>Trace Record Identifier</th>
<th>GTF Trace Record</th>
<th>Parameter in SYS1.PARMLIB Member or Operator Reply</th>
<th>For format, see:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSCH</td>
<td>Modify subchannel operation</td>
<td>MSCH, SYS, SYSM, SYSP</td>
<td>&quot;MSCH trace records&quot; on page 288</td>
</tr>
<tr>
<td>PCI</td>
<td>Program-controlled input/output interruption</td>
<td>PCI</td>
<td>&quot;EOS, INTG, IO, IOCS, and PCI trace records&quot; on page 276</td>
</tr>
<tr>
<td>PCIDMX</td>
<td>PCIE de-multiplexing event</td>
<td>PCIE</td>
<td>&quot;PCIDMX trace records&quot; on page 289</td>
</tr>
<tr>
<td>PCILG</td>
<td>PCI load instruction</td>
<td>PCIE</td>
<td>&quot;PCILG trace records&quot; on page 290</td>
</tr>
<tr>
<td>PCISTG</td>
<td>PCI store instruction</td>
<td>PCIE</td>
<td>&quot;PCISTG trace records&quot; on page 291</td>
</tr>
<tr>
<td>PGM</td>
<td>Program interruption</td>
<td>PI, PIP, SYS, SYSM, SYSP</td>
<td>&quot;PGM and PI trace records&quot; on page 292</td>
</tr>
<tr>
<td>PI</td>
<td>Program interruption</td>
<td>PI, PIP, SYS, SYSM, SYSP</td>
<td>&quot;PGM and PI trace records&quot; on page 292</td>
</tr>
<tr>
<td>RNIO</td>
<td>VTAM remote network input/output event</td>
<td>RNIO</td>
<td>&quot;RNIO trace records&quot; on page 293</td>
</tr>
<tr>
<td>RSCH</td>
<td>Resume subchannel operation</td>
<td>SSCH, SSCHP</td>
<td>&quot;RSCH trace records&quot; on page 294</td>
</tr>
<tr>
<td>SDSP</td>
<td>Task re-dispatch</td>
<td>DSP</td>
<td>&quot;DSP and SDSP trace records&quot; on page 274</td>
</tr>
<tr>
<td>SLIP</td>
<td>SLIP program event interruption</td>
<td>SLIP</td>
<td>&quot;SLIP trace records&quot; on page 295</td>
</tr>
<tr>
<td>SRB</td>
<td>Service request block routine dispatch or re-dispatch</td>
<td>DSP</td>
<td>&quot;SRB trace records&quot; on page 301</td>
</tr>
<tr>
<td>SRM</td>
<td>System resources manager return</td>
<td>SRM</td>
<td>&quot;SRM trace records&quot; on page 302</td>
</tr>
<tr>
<td>SSCH</td>
<td>Start subchannel operation</td>
<td>SSCH, SSCHP, SYS, SYSM, SYSP</td>
<td>&quot;SSCH trace records&quot; on page 303</td>
</tr>
<tr>
<td>STAE</td>
<td>STAE or ESTAE recovery routine return</td>
<td>RR, SYS, SYSM, SYSP</td>
<td>&quot;STAE trace records&quot; on page 304</td>
</tr>
<tr>
<td>SUBSYS</td>
<td>Unformatted trace event</td>
<td></td>
<td>&quot;HEXFORMAT, SUBSYS, and SYSTEM trace records&quot; on page 283</td>
</tr>
<tr>
<td>SVC</td>
<td>Supervisor call interruption</td>
<td>SVC, SVCP, SYS, SYSM, SYSP</td>
<td>&quot;SVC and SVCR trace records&quot; on page 306</td>
</tr>
<tr>
<td>SVCR</td>
<td>Supervisor call exit</td>
<td>SVC, SVCP, SYS, SYSM, SYSP</td>
<td>&quot;SVC and SVCR trace records&quot; on page 306</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Unformatted trace event</td>
<td></td>
<td>&quot;HEXFORMAT, SUBSYS, and SYSTEM trace records&quot; on page 283</td>
</tr>
<tr>
<td>TCW</td>
<td>zHPF channel program</td>
<td>CCW</td>
<td>&quot;TCW trace records&quot; on page 308</td>
</tr>
<tr>
<td>USR</td>
<td>User event</td>
<td>USR, USRP</td>
<td>&quot;USR trace records&quot; on page 312</td>
</tr>
<tr>
<td>XSCH</td>
<td>Cancel subchannel operation</td>
<td>XSCH, SYS, SYSM, SYSP</td>
<td>&quot;XSCH trace record&quot; on page 317</td>
</tr>
</tbody>
</table>
Example of formatted GTF trace output

This section contains screen images that show GTF records. IPCS produced the screens from an example dump. These records are in comprehensive format and are time stamped.

The GTFTRACE subcommand was issued on the IPCS Subcommand Entry panel shown in Figure 95.

Figure 95. Example: IPCS subcommand entry panel for GTFTRACE

Figure 96 on page 268 shows records for the start subchannel operation (SSCH) event.
The screen images in Figure 97 and Figure 97 show records for two input/output (IO) interruption events. The last two rows in the I/O statistics section will only appear for zHPF I/O events.

**Figure 96. Example: GTF record for SSCH events**

The screen images in **Figure 97** and **Figure 97** show records for two input/output (IO) interruption events. The last two rows in the I/O statistics section will only appear for zHPF I/O events.

**Figure 97. Example: GTF records for IO interruption events**
Formatted trace records for events

The following sections describe different types of formatted trace records.

Time stamp records

Time stamp records mark the time an event occurred.

Record Format After Each Trace Record

```
GMT-mm/dd/yy hh:mm:ss:dddddd LOC-mm/dd/yy hh:mm:ss:dddddd
```

 GMT-mm/dd/yy hh:mm:ss
Month/day/year and Greenwich mean time given in hour:minute:second format.

LOC-mm/dd/yy hh:mm:ss:dddddd
Month/day/year and local time given in hour:minute:second.microsecond format. Local time is calculated using a time zone offset established when tracing starts.

Source index records

Source index records are added when GTF trace records are consolidate using the IPCS COPYTRC subcommand. The records identify the system that produced the GTF trace record.

Record Format After Each Trace Record, if the GTF trace records are consolidated with the IPCS COPYTRC subcommand:
The source index record indicates that the GTF trace record was produced by the system with identifier 01. Identifiers include the system name and the trace options in effect for that system. The identifiers are listed at the top of the IPCS report.

Lost event records
A lost event record indicates that GTF lost the trace records for one or more events because of an error or overflow of the trace buffer.

Record Format When GTF Trace Buffer is Lost due to Error

```
**** ONE TRACE BUFFER LOST TIME hh.mm.ss.dddddd
```

**hh.mm.ss.dddddd**
The time of day (hour.minute.second.millisecond) when GTF placed the first trace record in the buffer.

The size of the GTF trace buffer is:
- Equal to the blocksize used by GTF when writing the trace records to a data set on a direct access storage device (DASD). The system displays the blocksize in message AHL906I. If the records are to be written to a data set, the system issues message AHL906I after starting GTF.
- 32,760 bytes, if GTF is writing the trace records to a data set on tape.
- 32,768 bytes, if GTF is writing the trace records only into internal trace buffers.

Record Format for Number of Trace Events Lost due to Errors or Trace Buffer Overflow

```
***** LOST EVENTS NUM ddddddddddddd LOCAL TIME mm/dd/yyyy hh.mm.ss.nnnnnn ***
```

**dddddddddd**
The number of lost events

**mm/dd/yyyy**
The date (in month/day/year format) when GTF placed the first trace record in the current trace buffer.

**hh.mm.ss.dddddd**
The time of day (hour.minute.second.millisecond) when GTF placed the first trace record in the current trace buffer.

ADINT trace records
ADINT records represent adapter interruptions.
**ASCB hhhhhhhh**  
Address of the ASCB for the address space that was active when the adapter interruption occurred on this CPU.

**CPUID hhhh**  
Address of the processor on which the adapter interruption occurred.

**JOBN cccccc**  
Name of the job associated with the address space that was active when the adapter interruption occurred on this CPU.

**PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh**  
Program status word (PSW) stored when the interruption occurred.

**ISC hh**  
Interuption subclass.

**AISM hh**  
Adapter source identification mask.

### CCW trace records
A CCW record represents the processing of a non-zHPF channel program. CCW trace records appear following EOS, IO, IOCS PCI, RSCH, or SSCH trace records; they do not appear alone. Any of the formats can appear in any combination in one CCW trace record.

```
<table>
<thead>
<tr>
<th>CCW CHAIN</th>
<th>FORMAT d</th>
<th>ccc</th>
<th>DEV... hhhh</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCB..... hhhhhhhh CPU..... hhhh JOBN..... cccccc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fhhhhhhh ---CCW-- ---CCW-- dddddddd dddddddd cccccccc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fhhhhhhh ---CCW-- ---CCW-- dddddddd dddddddd cccccccc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fhhhhhhh ---CCW-- ---CCW-- dddddddd dddddddd cccccccc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dddddddd dddddddd cccccccc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dddddddd dddddddd cccccccc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dddddddd dddddddd cccccccc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dddddddd dddddddd cccccccc</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--Back half of split data--

| dddddddd dddddddd cccccccc |
| dddddddd dddddddd cccccccc |
| Fhhhhhhh ---CCW-- ---CCW-- dddddddd dddddddd cccccccc |
| IDAW hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
| IOSB hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
| EwAx hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh, hhhhhhh cccccccc |
| hhhhhhh hhhhhhh hhhhhhh cccccccc |
```

**FORMAT d ccc**  
Format (d) and type of trace event (ccc): EOS, IO, IOCS, PCI, RSCH, or SSCH. Format is either zero or 1.

**DEV shhhhh**

**DEV snnnn**  
Device number from the UCBCHAN field of the UCB. This number is qualified with the subchannel identifier (UCBSID).

**ASCB hhhhhhhh**  
Same as the ASCB field in the IO, IOCS, SSCH, RSCH, PCI, or EOS base record.
Generalized Trace Facility

**CPU hhhh**
Same as the CPU ID field in the IO, IOCS, SSCH, RSCH, PCI, or EOS base record.

**JOBN cccccccc**
Same as the job named (JOBN) field in the IO, IOCS, SSCH, RSCH, PCI, or EOS base record.

**Fhhhhhhh**
Fullword address of the CCW. If the high order bit of the address is on, this is the real address of the CCW, otherwise this is the virtual address of the address of the CCW.

---CCW---
Is the CCW command. The command is either a format 0 or format 1 CCW.
- Format 0 CCW is in the format oaaaaaa ffuubbbb
- Format 1 CCW is in the format oooffbbbb aaaaaaaa
  - oo op code.
  - aaaaaa Real address of data associated with the CCW. If indirect address words (IDAWS) are present, this is the address of the IDAW list.
  - aaaaaaaaa Fullword real address of data associated with the CCW. If IDAWs are present, this is the address of the IDAW list.
  - ff CCW flags; if this flag is .... ..1.. , then this indicates that an IDAW list is present. If this flag is .... ...1., then a suspend of the channel program was requested. If this flag is .... ...1 , then a modified indirect addressing word list is present.
  - uu Not used by hardware; could contain a nonzero character.
  - bbbb Byte count.

<table>
<thead>
<tr>
<th>dddddddd ddddddd</th>
<th>cccccccc</th>
</tr>
</thead>
</table>
| Information transferred by the CCW. If there is not a series of dashes in this field, then all transferred data is displayed in four byte sections.

--Back half of split data--
Indicates there were more bytes of information transferred than were specified on the START command. The default value is 20 bytes, but you can specify the number of bytes to be shown. The specified value is halved; for an odd number, the larger section is shown first. The first section of data displayed comes from the beginning of the buffer from which the data was transferred. The last section comes from the end of the buffer.

**IDAW hhhhhhh or hhhhhhh_hhhhhhh hhhh**
Contents of the IDAW, a fullword real address for 31-bit IDAWs or a doubleword real address for 64-bit IDAWs, followed by a halfword specifying the length of the data at that address. The data at the address follows the halfword length. The hhhhhhh_hhhhhhh version of this parameter specifies 64-bit.

**MIDAW hhhhhhh hhhhhhh hhh hhhhhhh_hhhhhhh**
The modified indirect addressing word (MIDAW), which is 16 bytes, formatted in the GTF trace with the first 8 bytes on the first line containing the flags and data length and the second line containing the 64 bit data address. The length of the data is replicated after the first 8 bytes of MIDAW data to make it easier
to read and maintain consistency with the IDAW format. The data at the address follows the halfword length. The data for a MIDAW is not formatted if the skip indicator is on.

**IOSB hhhhhhhh**
Fullword virtual address of the IOSB followed by the contents of the IOSB. The fullword at offset X’34’ of the IOSB points to an error recover procedure work area (EWA), or is zero. The EWA is traced and documented directly below the IOSB and is formatted in the same manner as the IOSB.

**EWAx hhhhhhhh**
Fullword virtual address of the error recovery procedure work area, followed by the contents of EWA.

### CSCH and HSCH trace records

CSCH and HSCH records represent a clear subchannel operation and a halt subchannel operation.

#### Record Formats

<table>
<thead>
<tr>
<th>CSCH</th>
<th>shhhh</th>
<th>ASCB</th>
<th>hhhhhhhh</th>
<th>CPUID</th>
<th>hhhh</th>
<th>JOBN</th>
<th>cccccccc</th>
<th>DEV</th>
<th>hhhh</th>
<th>SFLS</th>
<th>hhhh</th>
<th>SID</th>
<th>hhhhhhhh</th>
<th>DVRID</th>
<th>hh</th>
<th>ARDID</th>
<th>hh</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSCH</td>
<td>shhhh</td>
<td>ASCB</td>
<td>hhhhhhhh</td>
<td>CPUID</td>
<td>hhhh</td>
<td>JOBN</td>
<td>cccccccc</td>
<td>DEV</td>
<td>hhhh</td>
<td>SFLS</td>
<td>hhhh</td>
<td>SID</td>
<td>hhhhhhhh</td>
<td>DVRID</td>
<td>hh</td>
<td>ARDID</td>
<td>hh</td>
</tr>
</tbody>
</table>

**CSCH shhhh**

**HSCH shhhh**
Device number from the UCBCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

**ASCB hhhhhhhh**
Address of the ASCB for the address space that started the I/O operation.

**CPUID hhhh**
Address of the processor on which the I/O operation started

**JOBN cccccccc**
One of the following:

- cccccccc
  - Name of the job associated with the task that requested the I/O operation
- N/A
  - No job is associated with the requested I/O

**DEV hhhh**
Device number from the UCBCHAN field of the UCB.

**SFLS hhhh**
Start flags from the UCBSFLS field of the UCB.

**SID hhhhhhhh**
Subchannel ID from the UCBSID field of the UCB.
Generalized Trace Facility

CC hh
CSCH or HSCH condition code in bits 2-3.

DVRID hh
Driver ID value from the IOSDVRID field of the IOSB.

ARDID hh
One of the following:
  hh  Associated request driver ID from the IOSDVRID field of the IOSB
  U/  Unavailable because the IOQ was unavailable

IOSLVL hh
Function level to provide serialization of I/O requests. This value comes from
the IOSLEVEL field of the IOSB.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

UCBWGT hh
Flags from the UCBWGT field of the UCB.

BASE shhhh
Device number from the UCBCHAN field of the UCB, which includes the
subchannel set identifier when appropriate.

DSP and SDSP trace records

A DSP record represents dispatching of a task. An SDSP record represents
re-dispatching of a task after an SVC interruption. SDSP interruptions also build
SVC exit records with label SDSP. When both DSP and SVC options are in effect,
the SVCR format of trace record is produced by IPCS.

If the trace data contains an SVC exit record, the label that appears in the
formatted output will depend on the options selected during IPCS.
1. If the SVC option is selected in the IPCS dialog, the SVC exit record and the
   SVC number will appear with the label SVCR.
2. If only the DSP option is chosen in the IPCS dialog, the formatted output
   record will remain unchanged; DSP and SDSP labels will appear in the
   formatted output and no SVC number is present.
3. If both DSP and SVC options are active in IPCS, the SVCR along with SVC
   number will appear.

It can be concluded, if SVC is one of the options selected during IPCS formatting,
all SVC exit records will appear with label SVCR along with SVC number.

Minimal Trace Record Formats

<table>
<thead>
<tr>
<th>DSP</th>
<th>ASCB.... hhhhhhhh</th>
<th>CPU..... hhh</th>
<th>PSW..... hhhhhhhhh hhhhhhh</th>
<th>R15..... hhhhhhhh R0...... hhhhhhh</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDSP</th>
<th>ASCB.... hhhhhhhh</th>
<th>CPU..... hhh</th>
<th>PSW..... hhhhhhhhh hhhhhhh</th>
<th>R15..... hhhhhhhh R0...... hhhhhhh</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Comprehensive Trace Record Formats

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP</td>
<td>ASCB... hhhhhhh CPU..... hhhh JOBN.... cccccccc</td>
</tr>
<tr>
<td></td>
<td>DSP-PSW. hhhhhhh hhhhhhh hhhhhhh hhhhhhh</td>
</tr>
<tr>
<td></td>
<td>TCB..... hhhhhhh MODN.... yyyyyyy R15..... hhhhhhh</td>
</tr>
<tr>
<td></td>
<td>R0..... hhhhhhh R1...... hhhhhhh</td>
</tr>
<tr>
<td>SDSP</td>
<td>ASCB... hhhhhhh CPU..... hhhh JOBN.... cccccccc</td>
</tr>
<tr>
<td></td>
<td>DSP-PSW. hhhhhhh hhhhhhh hhhhhhh hhhhhhh</td>
</tr>
<tr>
<td></td>
<td>TCB..... hhhhhhh MODN.... yyyyyyy R15..... hhhhhhh</td>
</tr>
<tr>
<td></td>
<td>R0..... hhhhhhh R1...... hhhhhhh</td>
</tr>
</tbody>
</table>

**ASCB** hhhhhhh
Address of address space control block.

**CPU** hhhh
Address of processor on which the task is dispatched.

**PSW** hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh
Program status word under which the task is dispatched.

**JOBN** cccccccc
One of the following:

- cccccccc
  Name of the job associated with the task being dispatched
- N/A
  The record is for a system or started task

**PPPPP**
A page fault occurred

********
An internal error occurred

**TCB** hhhhhhh
Address of the task control block.

**R15** hhhhhhh
**R0** hhhhhhh
**R1** hhhhhhh
Data that will appear in general registers 15, 0, and 1 when the task is dispatched.

**MODN** cccccccc
ccccccc is one of the following:

- *mod_name*
  The name of a module that will receive control when the task is dispatched.

**WAITTCB**
Indicates that the system wait task is about to be dispatched.

**SVC–T2**
Indicates that a type 2 SVC routine that resides in the nucleus is about to be dispatched.
Generalized Trace Facility

**SVC–RES**
Indicates that a type 3 SVC routine or the first load module of a type 4 SVC routine is about to be dispatched. The routine is located in the pageable link pack area (PLPA).

**SVC–cccc**
Indicates that the second or subsequent load module of a type 4 SVC routine is about to be dispatched. The module is located in the fixed or pageable link pack area (LPA). The last four characters of the module name are cccc.

****IRB****
Indicates that an asynchronous routine with an associated interruption request block (IRB) is about to be dispatched. No module name is available.

**cccccccc**
Indicates that error fetch is in the process of loading an error recovery module. The last seven characters of the module name are ccccccc.

**PPPPPPPPP**
A page fault occurred

*********
An internal error occurred

**EOS, INTG, IO, IOCS, and PCI trace records**
EOS records represent an end of sense interruption, INTG records represent an input/output (I/O) interruption that is used to signal the completion of a zHPF interrogate request, IO records represent an I/O interruption, IOCS records represent an I/O interruption that also contains concurrent sense information, for devices that support the concurrent sense facility, and PCI records a program-controlled interruption.

```
| EOS.... shhhh ASCB.... hhhhhh CPUID... hhhh JOBN.... cccccc PSW..... hhhhhh hhhhhh hhhhhh hhhhhh |
|--------|-----------------|-----------------|-----------------|-----------------|
| IRB..... hhhhhh SENSE... hhhh FLA..... hhh |
| TCB..... hhhhhh OPT..... hh DVRID... hh IOSLVL.. hh |
| UCBVL.. hh UCBWGT.. hh BASE.... shhhh |

I/O Statistics:
Connect. hhhhhh Pending. hhhhhh Discon.. hhhhhh
CUQ..... hhhhhh DAO..... hhhhhh Devbys.. hhhhhh
ICMR.... hhhhhh StartCt. hhhhhh SampCt.. hhhhhh
ZTotdev. hhhhhh ZDefer.. hhhhhh ZCUQ.... hhhhhh
ZDevbys. hhhhhh ZDAO.... hhhhhh
IntrDly. hhhhhh
```

```
| INTG.... shhhh ASCB.... hhhhhh CPUID... hhhh JOBN.... cccccc PSW..... hhhhhh hhhhhh hhhhhh hhhhhh |
|--------|-----------------|-----------------|-----------------|-----------------|
| IRB..... hhhhhh SENSE... hhhh FLA..... hh |
| TCB..... hhhhhh OPT..... hh DVRID... hh IOSLVL.. hh |
| UCBVL.. hh UCBWGT.. hh BASE.... shhhh |
```
The device number from the UCBCHAN field of the unit control block (UCB),
which includes the subchannel set identifier when appropriate.

**ASCB {hhhhhhhh|U/A}**

One of the following:

- **hhhhhhhh**
  Address of the address space control block (ASCB) for the address
  space that started the I/O operation.

- **U/A**
  Unavailable because the I/O supervisor block (IOSB) control block is
  unavailable.
Generalized Trace Facility

**CPU hhhh**
Address of the processor on which the interruption occurred.

**JOBN {cccccccc|N/A|U/A}**
One of the following:

- **cccccccc**
  Name of the job associated with the task that requested the I/O operation.
- **N/A**
  Not applicable.
- **U/A**
  Unavailable because the IOSB control block is unavailable.

**PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh**
Program status word (PSW) stored when the interruption occurred.

**IRB (see explanation)**
For the EOS, IO, and PCI trace records, this field contains the first five words, in hexadecimal, of the interruption response block (IRB) operand of the Test Subchannel (TSCb) instruction.

For the IOCS trace record, this field contains the first 16 words, in hexadecimal, of the interruption response block operand of the TSCH instruction. (Note that this IRB is not the interruption request block indicated as **IRB*** in a DSP trace record.)

**TCB {hhhhhhhh|N/A|U/A}**
One of the following:

- **hhhhhhhh**
  Address of the TCB for the task that requested the I/O operation.
- **N/A**
  Not applicable.
- **U/A**
  Unavailable because the IOSB control block is unavailable.

**SENSE {hhhh|N/A|U/A}**
One of the following:

- **hhhh**
  First 2 sense bytes from the IOSSNS field of the IOSB.
- **N/A**
  Not applicable.
- **U/A**
  Unavailable because the IOSB control block is unavailable.

**FLA {hh|U/A}**
One of the following:

- **hh**
  Flag byte from the IOSFLA field of the IOSB.
- **U/A**
  Unavailable because the IOSB control block is unavailable.

**OPT {hh|U/A}**
One of the following:

- **hh**
  IOSB options byte from the IOSOPT field of the IOSB.
- **U/A**
  Unavailable because the IOSB control block is unavailable.

**DVRID {hh|U/A}**
One of the following:

- **hh**
  Driver identifier from the IOSDVRID field of the IOSB.
- **U/A**
  Unavailable because the IOSB control block is unavailable.
Generalized Trace Facility

IOSLVL {hh|U/A}
   One of the following:
   hh   Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.
   U/A  Unavailable because the IOSB control block is unavailable.

UCBLVL hh
   UCB level value from the UCLEVEL field of the UCB.

UCBWGT hh
   Flags from the UCBWGT field of the UCB.

BASE shhhh
   Device number from the UCBCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

Connect hhhhhhhh
   The device connect time for this I/O request in units of 0.5 microseconds.

Pending hhhhhhhh
   The function pending time for this I/O request in units of 0.5 microseconds.

Discon hhhhhhhh
   The device disconnect time for this I/O request in units of 0.5 microseconds.

CUQ hhhhhhhh
   The control unit queueing time for this I/O request in units of 0.5 microseconds.

DAO hhhhhhhh
   The device active only time for this I/O request in units of 0.5 microseconds.

Devbsy hhhhhhhh
   The device busy time for this I/O request in units of 0.5 microseconds.

ICMR hhhhhhhh
   The initial command response time for this I/O request in units of 0.5 microseconds.

StartCt hhhhhhhh
   The number of SSCH/RSCH instructions that were issued for the device while GTF trace was active.

SampCt hhhhhhhh
   The number of SSCH/RSCH instructions for which data was collected for the device.

ZTotdev hhhhhhhh
   The total device time for this I/O request in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

ZDefer hhhhhhhh
   The control unit defer time for this I/O request in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

ZCUQ hhhhhhhh
   The control unit queue time for this I/O request in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.
Generalized Trace Facility

ZDevBsy hhhhhhh
The device busy time for this I/O request in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

ZDAO hhhhhhhh
The device active only time for this I/O request in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

IntrDly hhhhhhhh
The total interrupt delay time for all I/O requests in units of 128 microseconds.

EXT trace records
An EXT record represents a general external interruption.

Minimal Trace Record Format

EXT_CODE.... hhhh ASCB.... hhhhhhhh CPU..... hhhh
PSW...... hhhhhhhh hhghhhhhh hhhhhhhh hhhhhhhh
TCB...... hhhhhhhh ccc-TCB. hhhhhhhh

Comprehensive Trace Record Format

EXT..... hhhh ASCB.... hhhhhhhh CPU..... hhhh JOBN.... cccccccc
OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
TCB..... hhhhhhhh TQE FIELDS: FLAGS... hhhh
EXTADDR. hhhhhhhh TCB..... hhhhhhhh

EXT..... hhhh ASCB.... hhhhhhhh CPU..... hhhh JOBN.... cccccccc
OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
TCB..... hhhhhhhh TQE FIELDS: FLAGS... hhhh
EXTADDR. hhhhhhhh TCB..... hhhhhhhh

EXT..... hhhh ASCB.... hhhhhhhh CPU..... hhhh JOBN.... cccccccc
OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
TCB..... hhhhhhhh TQE FIELDS: FLAGS... hhhh
EXTADDR. hhhhhhhh TCB..... hhhhhhhh

EXT CODE hhhh
EXT hhhh
External interruption code.

ASCB hhhhhhhh
Address of ASCB for the address space that was current when the interruption occurred.

CPU hhhh
Address of the processor on which the interruption occurred.

JOBN cccccccc
One of the following:
cccccccc
Name of the job associated with the interrupted task
N/A The record is for a system or started task
PPPPPPPPP
A page fault occurred
****** An internal error occurred
PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
OLD-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word stored when the interruption occurred.

TCB hhhhhhhh
One of the following:

hhhhhhhh
Address of the TCB for the interrupted task

N/A Not applicable, as in the case of an interrupted SRB routine

INT-TCB hhhhhhhh
TQE-TCB hhhhhhhh
Address of the TCB. This interruption is indicated by interruption codes 12hh.

TQE FIELDS
Indicates a clock comparator or CPU timer interruption. These interruptions are indicated by interruption codes 'X'1004' or 'X'1005'. The following fields contain information from the timer queue element (TQE):

FLAGS hhhh
The flags from the TQEFLGS field.

EXTADDR hhhhhhhh
The first four hexadecimal digits are the contents of the TQEFLGS field; the last four hexadecimal digits are the contents of the TQEEXIT field.

ASCB hhhhhhhh
One of the following:

hhhhhhhh
Contents of the TQEASCB field.

PPPPPPPPP
A page fault occurred

********
An internal error occurred

The TQEASCB field is present only for a clock comparator interruption. TQEASCB contains the address of the ASCB for the address space in which the timer exit routine will be run.

TCB hhhhhhhh
One of the following:

hhhhhhhh
Contents of the TQETCB field.

N/A The record is for a system or started task

PPPPPPPPP
A page fault occurred

********
An internal error occurred

TQETCB contains the address of the TCB for the task under which the timer exit routine will run.

PARM hhhhhhhh
Signal passed on a signal processor interruption, which is indicated by interruption codes 12hh.
**Generalized Trace Facility**

**SIG-CPU hhhh**
Address of the processor on which a signal processor interruption occurred.

**FRR trace records**

An FRR record represents the return to the recovery termination manager (RTM) from a functional recovery routine (FRR). All fields, except the processor address, are gathered from the system diagnostic word area (SDWA) that was passed to the FRR.

**Minimal Trace Record Format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRR</td>
<td></td>
</tr>
<tr>
<td>ASCB....</td>
<td>Address of the ASCB for the address space in which the error occurred.</td>
</tr>
<tr>
<td>PPPPPPPP</td>
<td>A page fault occurred</td>
</tr>
<tr>
<td>PPPPPPPP</td>
<td>An internal error occurred</td>
</tr>
</tbody>
</table>

**Comprehensive Trace Record Format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRR</td>
<td></td>
</tr>
<tr>
<td>ASCB....</td>
<td>Address of the ASCB for the address space in which the error occurred.</td>
</tr>
<tr>
<td>JOBN....</td>
<td>Name of the job associated with the error</td>
</tr>
<tr>
<td>MMMMM</td>
<td>The record is for a system or started task</td>
</tr>
<tr>
<td>PPPPPPPP</td>
<td>A page fault occurred</td>
</tr>
<tr>
<td>PPPPPPPP</td>
<td>An internal error occurred</td>
</tr>
</tbody>
</table>

**ASCB hhhhhhh**
One of the following:

| hhhhhhh | Address of the ASCB for the address space in which the error occurred.      |

**CPU hhhh**
Address of the processor associated with the error.

**JOBN cccccccc**
One of the following:

| cccccccc | Name of the job associated with the error                                   |
| N/A      | The record is for a system or started task                                  |

**NAME cccccccc**
Name of the FRR routine.

**PSW hhhhhhh hhhhhhh hhhhhhh hhhhhhh**
One of the following:
Generalized Trace Facility

hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
   Program status word at the time of the error

PPPPPPPP
   A page fault occurred

********
   An internal error occurred

CC hhhhhhhh
ABCC hhhhhhhh
   One of the following:

hhhhhhhh
   The first three digits are the system completion code and the last three
digits are the user completion code

U/A Unavailable because the system diagnostic work area (SDWA) was
     unavailable

********
   An internal error occurred

FLG1 hhhhhhhh
FLG hhhhhhh
ERRT hhhhhhhh
   Error-type flags from the SDWAFLGS field of SDWA.

FLG2 hhhhhhh
   Additional flags from the SDWAMCHD and SDWAACF2 fields of SDWA. The
   flags are contained in the two low-order bytes of this printed field; the high
   order byte is meaningless.

RC hh
   Return code

RETRY hhhhhhhhh
RTRY hhhhhhhhh
   One of the following:

hhhhhhhh
   Retry address supplied by the FRR

N/A Not applicable, indicating an FRR return code other than 4

PPPPPPPPPPPPPP
   A page fault occurred

********
   An internal error occurred

RTCA hhhhhhhh
   Indicates if the recovery routine was a STAE or ESTAE.

HEXFORMAT, SUBSYS, and SYSTEM trace records
HEXFORMAT, SUBSYS, and SYSTEM records represent events for which GTF
could not format the records.
Generalized Trace Facility

HEXFORMAT AID hh FID hh EID hh hhhhhhh hhhhhhh ...  
SUBSYS AID hh FID hh EID hh hhhhhhh hhhhhhh ...  
SYSTEM AID hh FID hh EID hh hhhhhhh hhhhhhh ...

HEXFORMAT
Indicates an event signalled by a GTRACE macro. The macro specified no formatting routine (FID=00).

SUBSYS
Indicates an event signalled by a GTRACE macro. The macro specified a formatting routine (FID=hh) that could not be found.

SYSTEM
Indicates a system event. The trace record could not be formatted for one of the following reasons:

- If EEEE hex appears in bytes 0-1 or 8-9 of the recorded data, an unrecoverable error occurred in a GTF data-gathering routine. Message AHL118I is written on the console, identifying the module that caused the error and the action taken. (The message indicates that GTF will no longer trace this type of event. No more records for this type of event will appear in the trace output.)

- If EEEE hex does not appear in bytes 0-1 or 8-9 of the recorded data, the record could not be formatted because the GTF formatting routine could not be found.

AID hh
Application identifier, which should always be AID FF.

FID hh
Format identifier of the routine (AMDUSRhh or AMDSYShh) that was to format this record.

EID hh
Event identifier, which uniquely identifies the event that produced the record.

hhhhhhhh hhhhhhh ...
Recorded data (256 bytes maximum).

IOX trace records
IOX records represent an input/output (I/O) interruption for a completed channel program and a summary of a complete channel program for the I/O operation.
IOX....shhhh ASCB.... hhhhhhhhh CPU..... hhhh JOBN.... cccccccc
DEVN.... hhhh SID..... hhhh DRID.... hh
TVSN.... hh ECNO.... hhhh DVCLS.... hh
DSTAT.... hh AERCC.... hh FLAGO.... hh
VOLSER.. cccccc UCBTP.. hhhhhhh
DSNAME.. cccc.cccccc.cccccc.cccccc
NSSCH... hhhhhhhhh DSSCH... hhhhhhhhh SDCON... hhhhhhhhh
SRPEN... hhhhhhhhh SDISC... hhhhhhhhh SCQU... hhhhhhhhh
IOTST... hhhhhhhhh hhhhhhhhh AONLY... hhhhhhhhh
DVBSY... hhhhhhhhh ICMR.... hhhhhhhhh

CCW SECTION SQNO.... hh FGS1.... hh FGS2.... hh
RCNT.... hh BLKR.... hhhh BLKW.... hhhh
BTRD.... hhhhhhhhh BTWR.... hhhhhhhhh DCHN.... hhhh
CCHN.... hhhh DEGA.... hh DEGE.... hh
DREE.... hhhh SEEKLOC CCHH... hh
LRDP.... hh LRECT.. hh LREXOP.. hh
LREXPM.. hhhh

IOX shhhh
IOX identifies the beginning of an IOX record where hhhh is the device number and s is the subchannel set identifier.

ASCB {hhhhhhhh|U/A}
One of the following:

hhhhhhhh
Address of the address space control block (ACSB) for the address space that started the I/O operation.

U/A Unavailable because the I/O supervisor block (IOSB) control block is unavailable.

CPU hhhh
Address of the processor on which the interruption occurred.

JOBN {cccccccc|N/A|U/A}
One of the following:

ccccc
Name of the job associated with the task that requested the I/O operation.

N/A Not applicable.

U/A Unavailable because the IOSB control block is unavailable.

DEVN shhhh
Device number with the subchannel set identifier when appropriate.

SID hhhh
System ID

DRID hh
Driver ID from IOSB

TVSN hh
Trace version

ECNO hhhh
Record count

DVCLS hh
Device class
### Generalized Trace Facility

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DSTAT hh</strong></td>
<td>Device status</td>
</tr>
<tr>
<td><strong>AERRC hh</strong></td>
<td>Error codes found during CCW analysis. See &quot;CCW error codes&quot; on page 331 for a description.</td>
</tr>
<tr>
<td><strong>FLAG0 hh</strong></td>
<td>Flag byte</td>
</tr>
<tr>
<td><strong>VOLSER ccccccc</strong></td>
<td>Volume Serial</td>
</tr>
<tr>
<td><strong>UCBTYP hhhhhhhh</strong></td>
<td>UCB type</td>
</tr>
<tr>
<td><strong>DSNAME ccccc.ccccc.ccccc.ccccc</strong></td>
<td>44-byte data set name</td>
</tr>
<tr>
<td><strong>NSSCH hhhhhhhh</strong></td>
<td>Number of SSCH instructions.</td>
</tr>
<tr>
<td><strong>DSSCH hhhhhhhh</strong></td>
<td>Number of SSCH instructions for which data was collected.</td>
</tr>
<tr>
<td><strong>SDCON hhhhhhhh</strong></td>
<td>Summation of device connect times.</td>
</tr>
<tr>
<td><strong>SRPEN hhhhhhhh</strong></td>
<td>Summation of function pending times.</td>
</tr>
<tr>
<td><strong>SDISC hhhhhhhh</strong></td>
<td>Summation of device disconnect times.</td>
</tr>
<tr>
<td><strong>SCUQU hhhhhhhh</strong></td>
<td>Summation of control unit queuing times</td>
</tr>
<tr>
<td><strong>IODTS hhhhhhhh</strong></td>
<td>Time stamp from IOD</td>
</tr>
<tr>
<td><strong>AONLY hhhhhhhh</strong></td>
<td>Summation of device active only times</td>
</tr>
<tr>
<td><strong>DVBSY hhhhhhhh</strong></td>
<td>Summation of device busy times</td>
</tr>
<tr>
<td><strong>ICMR hhhhhhhh</strong></td>
<td>Summation of initial command response times</td>
</tr>
<tr>
<td><strong>SQNO hh</strong></td>
<td>Orientation Sequence Number</td>
</tr>
<tr>
<td><strong>FGS1 hh</strong></td>
<td>Flag byte 1</td>
</tr>
<tr>
<td><strong>FGS2 hh</strong></td>
<td>Flag byte 2</td>
</tr>
<tr>
<td><strong>RCNT hh</strong></td>
<td>Count of erase</td>
</tr>
<tr>
<td><strong>BLKR hhhh</strong></td>
<td>Number of blocks read</td>
</tr>
<tr>
<td><strong>BLKW hhhh</strong></td>
<td>Number of block written</td>
</tr>
</tbody>
</table>
BTRD hhhhhhhh
   Number of bytes read

BTWR hhhhhhhh
   Number of bytes written

DCHN hhh
   Number of data chain CCWs

CCHN hhh
   Number of COM chain CCWs

DEGA hh
   Definition of exterior global attribute

DEGE hh
   Definition of exterior global attribute extended

DEEE hhhhhh
   Definition of exterior end of extend CCH

SEEKLOCC hh
   The command code that performed the seek or locate record operation.

CCHHR hhhhhhhh
   CCHHR seek or search address

LROP hh
   The locate record operation code.

LRSECT hh
   The locate record sector number.

LREXOP hh
   The locate record extended operation code.

LREXPM hhhh
   The locate record extended parameters.

**LSR trace records**

An LSR record represents dispatching of a local supervisor routine in an address space.

**Minimal Trace Record Format**

```
LSR      ASCB.... hhhhhhhh CPU..... hhh
        PSW..... hhhhhhhh hhhhhhh hhhhhhh hhhhhhh
        TCB..... hhhhhhhh R15..... hhhhhhhh R0...... hhhhhhhh
        R1...... hhhhhhhh
```

**Comprehensive Trace Record Format**

```
LSR      ASCB.... hhhhhhhh CPU..... hhh
        PSW..... hhhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh
        LSR-PSW. hhhhhhhh hhhhhhh hhhhhhh hhhhhhh
        TCB..... hhhhhhhh
```

**ASCB hhhhhhhh**

Address of the address space control block.

**CPU hhhh**

Address of the processor on which the routine will be dispatched.
Generalized Trace Facility

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
LSR-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word under which the routine receives control.

JOBN cccccccc
One of the following:
cccccccc
Name of the job associated with the routine being dispatched
N/A Not applicable

PPPPPPPPP
A page fault occurred
******
An internal error occurred

TCB hhhhhhhh
One of the following:
hhhhhhhh
Address of the task control block associated with this routine (if the
routine is run as part of a task)
N/A Not applicable

R15 hhhhhhhh
R0 hhhhhhhh
R1 hhhhhhhh
One of the following:
hhhhhhhh
Data that will appear in general registers 15, 0, and 1 when the local
supervisor routine is dispatched

PPPPPPPPP
A page fault occurred
******
An internal error occurred

MSCH trace records
An MSCH record represents a modify subchannel operation.

<table>
<thead>
<tr>
<th>MSCH.... shhhh</th>
<th>ASCB.... hhhhhhhh</th>
<th>CPUID... hhh</th>
<th>JOBN.... cccccccc</th>
<th>SID..... hhhhhhhh</th>
<th>CC...... hh</th>
<th>OPT..... hh</th>
<th>OPT2.... hh</th>
<th>IOSLV.. hh</th>
<th>SCH1B1.. hhhhhhhh</th>
</tr>
</thead>
<tbody>
<tr>
<td>hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MSCH shhhh
Device number from the UCBCHAN field of the UCB with subchannel set
identifier when appropriate.

ASCB hhhhhhhh
Address of the ASCB for the address space that started the modify subchannel
operation.
CPU hhhh
Address of the processor on which the modify subchannel started.

JOBN cccccccc
One of the following:
cccccccc
Name of the job associated with the task that requested the modify
subchannel operation
N/A Not applicable

SID hhhhhhh
Subchannel ID from the UCBSID field of the UCB.

CC hh
MSCH condition code in bits 2 and 3.

OPT hh
IOSB option bytes from the IOSOPT field of the IOSB.

OPT2 hh
IOSB option bytes from the IOSOPT field of the IOSB.

IOSLVL hh
Function level to provide serialization of I/O requests. This value comes from
the IOSLEVEL field of the IOSB.

SCHIB1 hhhhhhh ... hhhhhhh
First 7 words of the subchannel information block. Input from the caller of
modify subchannel instruction. SCHIB address from the IOSSCHIB field of the
IOSB.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

SCHIB2 hhhhhhh ... hhhhhhh
First 7 words of the subchannel information block resulting from the modify
subchannel instruction.

UCBWGT hh
Flags from the UCBWGT field of the UCB.

BASE shhhh
Device number from the UCBCHAN field of the UCB, which includes the
subchannel set identifier when appropriate.

PCIDMX trace records
PCIDMX records represent a PCIE de-multiplexing request.

PCIDMX hhhhhhh ASCB.... hhhhhhh CPUID... hhhhhhh JOBN.... cccccccc
DEVTYP... hhhhhhh DMXAD... hhhhhhh CALLBK0... hhhhhhh
PARMS... hhhhhhh hhhhhhh

PCIDMX hhhhhhh
PCIE function identifier (PFID) for PCIE device/function.

ASCB hhhhhhh
Address of the ASCB for the address space that was active when the
de-multiplexing event occurred.
Generalized Trace Facility

**CPUID hhh**
Address of the processor on which the de-multiplexing operation occurred.

**JOBN cccccccc**
Name of the job associated with the address space that was active when the de-multiplexing event occurred.

**DEVTYPE hhhhhhhh**
PCIE device type.

**DMXAD hhhhhhhh**
Address of the program that is performing the de-multiplexing.

**CALLBK@ hhhhhhhh**
Address of the callback routine.

**PARMS hhhhhhhh hhhhhhhh**
First 8 bytes of the callback parameters.

**PCILG trace records**
A PCILG record represents a PCI load instruction.

```
PCILG... hhhhhhhh ASCB.... hhhhhhhh CPUID... hhhhh垮 JOBN.... cccccccc
         TCB...... hhhhhhhh CC...... hh REQAD... hhhhhhhh
         TRCID.... hhhhhhhh DATA.... hhhhhhhh hhhhhhhh
         STATUS.. hh PCIAS... hh LENGTH.. hh
         OFFSET.. hhhhhhhh hhhhhhhh
```

**PCILG hhhhhhhh**
PCIE function identifier (PFID) for PCIE device/function.

**ASCB hhhhhhhh**
Address of the ASCB for the address space that issued the PCI load instruction.

**CPUID hhh**
Address of the processor on which the PCI load instruction was issued.

**JOBN cccccccc**
Name of the job associated with the address space that issued the PCI load operation.

**TCB hhhhhhhh**
Address of task control block or zero if not running under a task.

**CC hh**
Condition code from the PCI load instruction.

**REQAD hhhhhhhh**
Address of the program that requested the PCI load instruction to be issued.

**TRCID hhhhhhhh**
Program defined trace identifier that can be used to determine why the PCI load instruction is being issued.

**DATA hhhhhhhh hhhhhhhh**
The data that was loaded.

**HANDLE hhhhhhhh**
PCIE function hardware handle.
Generalized Trace Facility

STATUS hh
Error status information.

PCIAS hh
PCIE address space associated with the request.

LENGTH hh
Length of the data that was loaded.

OFFSET hhhhhhhh hhhhhhhh
Offset of the data within the PCIE address space that was loaded.

PCISTG trace records
A PCISTG record represents a PCI store instruction.

| PCISTG... hhhhhhhh ASCB.... hhhhhhhh CPUID... hhhhhh JOBN.... cccccc... |
| TCB.... hhhhhhhh CC....... hh REQAD... hhhhhhhh |
| TRCID... hhhhhhhh DATA.... hhhhhhhh hhhhhhhh HANDLE.. hhhhhhhh |
| STATUS.. hh PCIAS... hh LENGTH.. hh |
| OFFSET.. hhhhhhhh hhhhhhhh |

PCILG hhhhhhhh
PCIE function identifier (PFID) for PCIE device/function.

ASCB hhhhhhhh
Address of the ASCB for the address space that issued the PCI store instruction.

CPUID hhhh
Address of the processor on which the PCI store instruction was issued.

JOBN cccccc...
Name of the job associated with the address space that issued the PCI store operation.

TCB hhhhhhhh
Address of task control block or zero if not running under a task.

CC hh
Condition code from the PCI store instruction.

REQAD hhhhhhhh
Address of the program that requested the PCI store instruction to be issued.

TRCID hhhhhhhh
Program defined trace identifier that can be used to determine why the PCI store instruction is being issued.

DATA hhhhhhhh hhhhhhhh
The data that was stored.

HANDLE hhhhhhhh
PCIE function hardware handle.

STATUS hh
Error status information.

PCIAS hh
PCIE address space associated with the request.

LENGTH hh
Length of the data that was stored.
Generalized Trace Facility

OFFSET hhhhhhh hhhhhhh
Offset of the data within the PCIE address space that was stored.

PGM and PI trace records
PGM and PI records represent program interruptions.

Minimal Trace Record Format

<table>
<thead>
<tr>
<th>PI CODE</th>
<th>ASCB</th>
<th>CPU</th>
<th>JOBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>hhh</td>
<td>hhhhhhhh</td>
<td>hhhhhh</td>
<td>cccccccc</td>
</tr>
<tr>
<td>PSW</td>
<td>hhhhhh</td>
<td>hhhhhh</td>
<td></td>
</tr>
<tr>
<td>TEB</td>
<td>hhhhhh</td>
<td>VPA</td>
<td></td>
</tr>
<tr>
<td>R15</td>
<td>hhhhhh</td>
<td>R1</td>
<td></td>
</tr>
</tbody>
</table>

Comprehensive Trace Record Format

<table>
<thead>
<tr>
<th>PGM</th>
<th>ASCB</th>
<th>CPU</th>
<th>JOBN</th>
<th>OLD-PSW</th>
<th>TCB</th>
<th>VPH</th>
<th>VPA</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>hhhh</td>
<td>hhhhhhhh</td>
<td>hhhhhh</td>
<td>cccccccc</td>
<td>hhhhhhh</td>
<td>hhhhhhh</td>
<td>hhhhhhh</td>
<td>hhhhhhh</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>R10</td>
<td>R11</td>
<td>R12</td>
<td>R13</td>
<td>R14</td>
<td>R15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PI CODE hhh
PGM hhh
Program interruption code, in decimal.

ASCB hhhhhhh
Address of ASCB for the address space in which the interruption occurred.

CPU hhhh
Address of the processor on which the interruption occurred.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the interruption

N/A Not applicable

PPPPPPPPP
A page fault occurred

******
An internal error occurred

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
OLD-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word stored when the interruption occurred.

TCB hhhhhhhh
One of the following:

hhhhhhhh
Address of the TCB for the interrupted task

N/A Not applicable as in the case of an interrupted SRB routine
Generalized Trace Facility

VPH hhhhhhhh
VPA hhhhhhhh
Virtual page address high half, in the case of a 64-bit translation exception address (TEA) value greater than X'FFFFFFFF' is stored. Virtual page address, in the case of a translation process exception resulting from a reference to the page. This area is meaningless for other types of program interruptions.

MODN cccccccc
cccccccc is one of the following:

  mod_name
  The name of a module that will receive control when the task is dispatched.

WAITTCB
  Indicates that the system wait task was interrupted.

SVC-T2
  Indicates that a type 2 SVC routine resident in the nucleus was interrupted.

SVC-RES
  Indicates that a type 2 SVC routine or the first load module of a type 4 SVC routine was interrupted. The routine is located in the pageable link pack area (PLPA).

SVC-ccc
  Indicates that the second or subsequent load module of a type 4 SVC routine was interrupted. The module is located in the fixed or pageable link pack area (LPA). The last four characters of the load module name are cccc.

**IRB***
  Indicates that an asynchronous routine with an associated interrupt request block was interrupted. No module name is available.

*cccccccc
  Indicates that an error recovery module was in control. The last seven characters of the module name are ccccccc.

*******
  An internal error occurred

Rdd hhhhhhhh
  Contents of general registers when the interruption occurred.

RNIO trace records
An RNIO record represents a VTAM remote network input/output event. For trace information, see z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures.

Minimal Trace Record Format

```
RNIO ASCB.... hhhhhhhh CPU..... hhhh RD..... hhhhhhhh
```

Comprehensive Trace Record Format
Generalized Trace Facility

<table>
<thead>
<tr>
<th>RNIO</th>
<th>ASCB...... hhhhhhhh CPU...... hhhh JOB...... cccccccc</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN...</td>
<td>hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh</td>
</tr>
<tr>
<td>RO....</td>
<td>hhhhhhhh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RNIO</th>
<th>ASCB...... hhhhhhhh CPU...... hhhh JOB...... cccccccc</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT...</td>
<td>hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh</td>
</tr>
<tr>
<td>RO....</td>
<td>hhhhhhhh</td>
</tr>
</tbody>
</table>

**ASCB hhhhhhhh**
Address of the ASCB for the address space of the application associated with the event.

**CPU hhhh**
Address of the processor that ran the I/O instruction.

**JOBN cccccccc**
One of the following:

  - cccccccc
    - Name of the job associated with the IO event
  - N/A
    - Not applicable

**PPPPPPPPP**
A page fault occurred

***********
An internal error occurred

**IN hhhhhhhh ... hhhhhhhh**
**OUT hhhhhhhh ... hhhhhhhh**
IN indicates that the I/O is from NCP to VTAM; OUT indicates that the direction of the I/O is from VTAM to NCP. The hexadecimal data is:
- For IN events: the transmission header, the response header, and the response unit.
- For OUT events: the transmission header, the request header, and the request unit.

**R0 hhhhhhhh**
Contents of general register 1 when the event occurred.

**RSCH trace records**
An RSCH record represents a resume subchannel operation.

---

**RSCH shhhh**
Device number from the UCBCHAN field of the UCB with subchannel set identifier when appropriate.

**ASCB hhhhhhhh**
Address of the ASCB for the address space that started the I/O operation.

**CPU hhhh**
Address of the processor on which the I/O operation resumed.
JOBN cccccccc
   One of the following:
   
   cccccccc  Name of the job associated with the I/O operation
   
   N/A      Not applicable

RST hhhhhhhh
   Address of the channel program. This value comes from the contents of the
   IOSRST field of the IOSB.

VST hhhhhhhh
   Virtual address of the channel program. This value comes from the contents of
   the IOSVST field of the IOSB.

DSID hhhhhhhh
   Request identifier used by purge. Contents of the IOSDID field of the IOSB
   (address of the DEB or another control block used by purge).

CC hh
   RSCH condition code in bits 2 and 3.

SEEKA hhhhhhhh hhhhhhhh
   Dynamic seek address from the IOSEEKA field of the IOSB.

GPMSK hh
   Guaranteed device path mask for GDP requests from the IOSEEKA field of the
   IOSB.

OPT hh
   IOSB options byte from the IOSOPT field of the IOSB.

FMSK hh
   Mode set/file mask from the IOSFMSK field of the IOSB.

DVRID hh
   Driver ID from the IOSDVRID field of the IOSB.

IOSLVL hh
   Function level to provide serialization of I/O requests. This value comes from
   the IOSLEVEL field of the IOSB.

UCBLVL hh
   UCB level value from the UCBLEVEL field of the UCB.

UCBWGT hh
   Flags from the UCBWGT field of the UCB.

BASE shhhh
   Device number from the UCBCHAN field of the UCB, which includes the
   subchannel set identifier when appropriate.

**SLIP trace records**

A SLIP record represents a SLIP program event interruption. GTF writes four types
of SLIP records:

- SLIP standard trace record
- SLIP stand/user trace record
- SLIP user trace record
- SLIP debug trace record
Generalized Trace Facility

SLIP standard trace record
A SLIP standard (STD) trace record represents a slip trap match when the SLIP command specifies ACTION=TRACE or ACTION=TRDUMP.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCB hhhhhhhh</td>
<td>The address of the ASCB for the current address space.</td>
</tr>
<tr>
<td>CPU hhhh</td>
<td>The processor identifier (ID).</td>
</tr>
<tr>
<td>JOBN cccccccc</td>
<td>One of the following: cccccccc Name of the job associated with the SLIP trap.</td>
</tr>
<tr>
<td>N/A</td>
<td>Not applicable</td>
</tr>
<tr>
<td>TID cccc</td>
<td>The trap ID.</td>
</tr>
<tr>
<td>ASID hhhh</td>
<td>The identifier of the current address space.</td>
</tr>
<tr>
<td>JSP cccccccc</td>
<td>One of the following: cccccccc Job step program name.</td>
</tr>
<tr>
<td>N/A</td>
<td>Not applicable</td>
</tr>
<tr>
<td>U/A</td>
<td>Unavailable</td>
</tr>
<tr>
<td>TCB hhhhhhhh</td>
<td>One of the following: hhhhhhhh TCB address.</td>
</tr>
<tr>
<td>N/A</td>
<td>Not applicable</td>
</tr>
<tr>
<td>MFLG hhh</td>
<td>System mode indicators that indicate the status of the system. The indicators correspond to the SLWACW field in the SLWA. For a description of the SLWA, see z/OS MVS Data Areas in z/OS Internet library (<a href="http://www.ibm.com/systems/z/os/zos/bkserv/">http://www.ibm.com/systems/z/os/zos/bkserv/</a>).</td>
</tr>
<tr>
<td>EFLG hhh</td>
<td>Error bytes that indicate the error status of the system. These bytes correspond</td>
</tr>
</tbody>
</table>
Generalized Trace Facility

to SDWAERRA in the SDWA. For a description of the SDWA, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

SFLD hh
SLIP status flags.

DAUN hhhhhhh
A counter representing the number of times data was unavailable for the DATA keyword test.

The following fields apply to PER interruptions only. For other than PER interruptions, these fields are not applicable and contain: N/A, N/., or N.

MODN cccccccc
One of the following:
  
  cccccccc
  Load module name in which the interruption occurred

  N/A Not applicable

  U/A Unavailable

OFFS hhhhhhhh
One of the following:
  
  hhhhhhhh
  Offset into the load module containing the instruction that caused the interruption

  N/A Not applicable

  U/A Unavailable

IADR hhhhhhhh hhhhhhhh
Address of the instruction that caused the interruption.

INS hhhhhhhh hhhh
Instruction content: the instruction that caused the PER interruption.

EXSIAD hhhhhhhh hhhhhhhh
One of the following:
  
  hhhhhhhh hhhhhhhh
  Target instruction address if the INS field is an Execute instruction

  N/A Not applicable

  U/A Unavailable

EXSINS hhhhhhhhhh hhhh
One of the following:
  
  hhhhhhhhhh hhhh
  Target instruction content if an INS field is an Execute instruction: 6 bytes of data beginning at the target instruction address

  N/A Not applicable

  U/A Unavailable

BRNGH hhhhhhhh
BRNGA hhhhhhhh
One of the following:
**Generalized Trace Facility**

```

hhhhhhhh
The beginning range virtual address if the SLIP command specified SA. BRNGH identifies the high 4 bytes. BRNGA identifies the low 4 bytes.

N/A  Not applicable

**BRNGD hhhhhhhh**
One of the following:

```

hhh
Four bytes of storage starting at the beginning range virtual address if SA was specified

N/A  Not applicable

U/A  Unavailable

**OPSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh**
The program old PSW.

**ILC/PIC hhhhhhhh**
The instruction length code and program interruption code.

**PERC hh**
The PER interruption code.

**TYP hh**
The PER trap mode.

**PKM hhh**
The PSW key mask.

**SASID hhhh**
The identifier of the secondary address space.

**AX hhhh**
The authorization index.

**PASID hhhh**
The identifier of the primary address space.

**ASC c**
The PSW ASC mode indicator:

<table>
<thead>
<tr>
<th>c</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Primary addressing mode</td>
</tr>
<tr>
<td>1</td>
<td>Access register addressing mode</td>
</tr>
<tr>
<td>2</td>
<td>Secondary addressing mode</td>
</tr>
<tr>
<td>3</td>
<td>Home addressing mode</td>
</tr>
</tbody>
</table>

**SA-SPACE ccccccccccccc**
Storage alteration space identifier, as follows:

- The ASID, for an address space
- The owning ASID and the data space name, for a data space

**DATX hh**
The DATA filter mismatch count due to an event that occurred in transactional execution mode.

```
SLIP standard/user trace record

The SLIP standard/user trace record represents a slip trap match when the SLIP command specifies ACTION=TRACE or ACTION=TRDUMP and TRDATA=parameters.

<table>
<thead>
<tr>
<th>SLIP Standard/User Trace Record</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCB hhhhhhh . . . DATX hh</td>
<td>These fields are the same as the fields in the SLIP standard trace record.</td>
</tr>
<tr>
<td><strong>GENERAL PURPOSE REGISTER VALUES</strong></td>
<td></td>
</tr>
<tr>
<td><strong>GPR HIGH HALF VALUES</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ACCESS REGISTER VALUES</strong></td>
<td>Contents of the general purpose registers and access registers at the time of the error or interruption, if REGS is specified in TRDATA on the SLIP command.</td>
</tr>
<tr>
<td><strong>SLIP user trace record</strong></td>
<td>The SLIP user record represents a SLIP trap match when the SLIP command specifies ACTION=TRACE or ACTION=TRDUMP and TRDATA=parameters.</td>
</tr>
<tr>
<td>CPU hhhh</td>
<td>Processor ID.</td>
</tr>
<tr>
<td>EXT hhhh</td>
<td>Extension number.</td>
</tr>
<tr>
<td>CNTLN hh</td>
<td>Continuation length.</td>
</tr>
</tbody>
</table>
hhhh

Length for the single range in the SLIP command. If hhhh is zero, either the range was not available or the range was not valid, so that GTF did not collect data for the range. GTF would consider the range not valid if, for example, the ending range address precedes the beginning range address.

hhhhhhhh hhhhhhh hhhhhhh hhhhhhh | cccccc cccccc cccccc | User-defined data fields that are specified by TRDATA on the SLIP command. The length and data fields may be repeated.

For a SLIP command, the trace contains as many user records and user continuation records as needed to trace the data ranges specified in the TRDATA parameter on the SLIP command. The header in each record contains the processor ID and the extension number. When a record is filled enough so that the next data range cannot fit, GTF writes the partially filled record to the GTF trace table. GTF builds another record; its extension number is increased by one and the continuation length is set to zero.

When the length of data from a range is greater than 249 bytes, the excess data is put in user continuation records. After writing the SLIP USR record, GTF builds a user continuation record. GTF increases the extension number by one and sets the continuation length to the number of bytes of data to be put in the continuation record. If more than 251 bytes of data are left, GTF copies 248 bytes into the record and places it in the GTF trace table. GTF builds user continuation records until all the data from a range is traced.

SLIP debug trace record

The SLIP debug record represents a SLIP trap match when the SLIP command specifies DEBUG.

| SLIP S+U | ASCB..... | hhhhhhh CPU..... | hhh | JOBN.... | cccccc cccccc |
| TID..... | cccc | ASID..... | hhh | JSP..... | cccccc cccccc |
| TCB..... | hhhhhhh | MFLG.... | hhh | EFLG.... | hhh |
| SFLG..... | hh | DAUN.... | hhh | MOON.... | cccccc cccccc |
| OFFS..... | hhhhhhh | 1ADR..... | hhhhhhh hhhhhhh | hhhhhhh |
| INS..... | hhhhhhh | hhh |
| EXSIAD... | hhhhhhh | hhhhhhh | EXSINS.. | hhhhhhh |
| BRNGH... | hhhhhhh | BRNGA... | hhhhhhh | BRNGD... | hhhhhhh |
| OPSW.... | hhhhhhh | hhhhhhh | hhhhhhh | hhhhhhh |
| ILC/PIC. | hhhhhhh | PERC.... | hh | TYP....... | hh |
| PKM..... | hhh | SASID... | hhh | AX....... | hhh |
| PASID... | hhh | ASC..... | h | SA-SPACE | hhh |
| hh00 | DATX.... | hh |

ASCB hhhhhhh . . DATX hh

These fields are the same as the fields in the SLIP standard trace record. The high order bit in the SFLG field is set to 1 to indicate a debug record.

hh00

Two bytes of debug-produced data. The first byte indicates which keyword failed, the second byte contains zeros.

**Byte 1 (decimal)**

**Keyword That Failed**

| ...1 | DATA test failed |
| 3 | ASID |
| 4 | JOBNAME |
SRB trace records

An SRB record represents dispatching of an asynchronous routine represented by a service request block (SRB).

Minimal Trace Record Format

```
SRB   ASCB..... hhhhhhhh CPU..... hhh
     PSW...... hhhhhhhhh hhhhhhhhh hhhhhhhhh
     R15..... hhhhhhhhh SRB..... hhhhhhhhh R1..... hhhhhhhhh
     TYPE..... cccccc ccccccccccccccccccccccccc
```

Comprehensive Trace Record Format

```
SRB   ASCB..... hhhhhhhh CPU..... hhh
     JOBN..... cccccccc
     SRB-PSW. hhhhhhhhh hhhhhhhhh hhhhhhhhh
     SRB..... hhhhhhh
     TYPE..... cccccc ccccccccccccccccccccccccc
```

ASCB hhhhhhhh
Address of the ASCB for the address space in which the SRB routine is dispatched. This may or may not be the address space in which the SRB was created.

CPU hhh
Address of the processor on which the SRB routine is dispatched.

PSW hhhhhhhhh hhhhhhhhh hhhhhhhhh hhhhhhhhh
SRB-PSW hhhhhhhhh hhhhhhhhh hhhhhhhhh hhhhhhhhh
Program status word under which the SRB routine receives control.

JOBN cccccccc
One of the following:

ccccccc
Name of the job associated with the SRB being dispatched.
Generalized Trace Facility

N/A  Not applicable, as in the case of a global SRB, which is indicated in the
TYPE field

********
An internal error occurred

SRB hhhhhhh
One of the following:

hhhhhhhh
Address of the service request block (SRB)

********
An internal error occurred

R15 hhhhhhh
R1hhhhhhhh
One of the following:

hhhhhhhh
Data that will appear in general registers 15 and 1 when the SRB
routine is dispatched

********
An internal error occurred

PARM hhhhhhh
One of the following:

hhhhhhhh
Four-byte parameter or the address of a parameter field to be passed to
the SRB routine

N/A  Not applicable, as in the case of a suspended SRB, which is indicated
in the TYPE field

TYPE cccccccccccccccccccccccccccccccccccc
Indicates the type of SRB routine, as follows:

SUSPENDED
Denotes an SRB routine that was dispatched earlier and was subsequently
interrupted (for example, by I/O operations or by a request for a lock).
The routine is about to be re-dispatched.

INITIAL DISPATCH OF SRB
Denotes an SRB routine selected from the service priority list that is about
to be dispatched for the first time.

REDISPATCH OF SUSPENDED SRB
Denotes an SRB routine that was dispatched earlier and was subsequently
interrupted (for example, by I/O operations or by a request for a lock).
The routine is about to be re-dispatched.

SRM trace records
An SRM record represents an entry to the system resources manager (SRM).

Minimal Trace Record Format

| SRM | ASCB..... | hhhhhhhhh | CPU..... | hhh | R15..... | hhhhhhhhh | RO..... | hhhhhhhhh | R1..... | hhhhhhhhh |
Comprehensive Trace Record Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCB</td>
<td>Address of the ASCB for the address space that was current when SRM was entered</td>
</tr>
<tr>
<td>CPU</td>
<td>Address of the processor used by the system resources manager.</td>
</tr>
<tr>
<td>JOBN</td>
<td>Name of the job associated with the entry to SRM.</td>
</tr>
</tbody>
</table>

**ASCB**

One of the following:

- Address of the ASCB for the address space that was current when SRM was entered.
- An internal error occurred.

**CPU**

Address of the processor used by the system resources manager.

**JOBN**

One of the following:

- Name of the job associated with the entry to SRM.
- Not applicable.

**SSCH trace records**

An SSCH record represents a start subchannel operation.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSCH</td>
<td>Device number from the UCBCHAN field of the UCB with subchannel set identifier when appropriate.</td>
</tr>
<tr>
<td>ASCB</td>
<td>Address of the ASCB for the address space that started the I/O operation.</td>
</tr>
<tr>
<td>CPU</td>
<td>Address of the processor on which the I/O operation started.</td>
</tr>
<tr>
<td>JOBN</td>
<td>Name of the job associated with the entry to SRM.</td>
</tr>
</tbody>
</table>

**SSCH**

Device number from the UCBCHAN field of the UCB with subchannel set identifier when appropriate.
Generalized Trace Facility

ccc
Name of the job associated with I/O operation

N/A Not applicable

RST hhhhhhh
Address of the channel program. This value comes from the contents of the IOSRST field of the IOSB.

VST hhhhhhh
Virtual address of the channel program. This value comes from the contents of the IOSVST field of the IOSB.

DSID hhhhhhh
Request identifier used by purge. This identifier is in the IOSDID field of the IOSB and is the address of the DEB or another control block used by PURGE.

CC hh
SSCH condition code in bits 2 and 3.

SEEKA hhhhhhh hhhhhhh
Dynamic seek address from the IOSEEKA field of the IOSB.

GPMSK hh
Guaranteed device path mask for GDP requests from the IOSGPMSK field of the IOSB.

OPT hh
IOSB options byte from the IOSOPT field of the IOSB.

FMSK hh
Mode Set/File mask from the IOSFMSK field of the IOSB.

DVRID hh
Driver ID from the IOSDVRID field of the IOSB.

IOSLVL hh
Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

UCBWGT hh
Flags from the UCBWGT field of the UCB.

BASE shhhh
Device number from the UCBCCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

ORB hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh
Contents of the operation request block (ORB).

STAE trace records
A STAE record represents return to the recovery termination manager (RTM) from a STAE or ESTAE routine.

Minimal Trace Record Format
Comprehensive Trace Record Format

<table>
<thead>
<tr>
<th>STAE</th>
<th>ASCB....</th>
<th>hhhhhhhh</th>
<th>hhhhhhhh</th>
<th>hhhhhhhh</th>
<th>hhhhhhhh</th>
<th>CPU.....</th>
<th>hhh</th>
<th>JOBN....</th>
<th>cccccccc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ESTN....</td>
<td>cccccccc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ERR-PSW.</td>
<td>hhhhhhhh</td>
<td>hhhhhhhh</td>
<td>hhhhhhhh</td>
<td>hhhhhhhh</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABCC....</td>
<td>hhhhhhhh</td>
<td>ERRT....</td>
<td>hhhhhhhh</td>
<td>FLG.....</td>
<td>hhhhhhh</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RC.......</td>
<td>hh</td>
<td>RTRY....</td>
<td>hhhhhhh</td>
<td>RTCA....</td>
<td>hhhhhhh</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ASCB hhhhhhhh**
Address of the ASCB for the address space involved in the recovery.

**CPU hhh**
Address of the processor.

**JOBN cccccccc**
One of the following:

cccccccc
Name of the job involved in the recovery

N/A Not applicable

*******
An internal error occurred

**ESTN cccccccc**
One of the following:

cccccccc
ESTAE routine name

U/A Unavailable because the routine did not supply a name

*******
An internal error occurred

**PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh**
One of the following:

hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word at the time of the error

U/A Unavailable because the system diagnostic work area (SDWA) was unavailable

*******
An internal error occurred

**CC hhhhhhhh**
**ABCC hhhhhhhh**
One of the following:

hhhhhhhh
The first four digits are the system completion code and the last four digits are the user completion code

U/A Unavailable because the system diagnostic work area (SDWA) was unavailable
An internal error occurred

TYCA hhhhhhhh hhhhhhhh
Retry address (see RTRY hhhhhhhh following) and an indication of whether
the routine was a STAE or ESTAE (see RTCA hhhhhhhh following).

RF hhhhhhhh
FLG hhhhhh
ERRT hhhhhhhh
Error flags from the SDWAFLGS field of the SDWA.

RC hh
Return code

RTRY hhhhhhhh
One of the following:

 hhhhhhhh
The address supplied by the FRR

 N/A Not applicable, indicating an FRR return code other than 4

 PPPPPPPP
A page fault occurred

An internal error occurred

RTCA hhhhhhhh
Indicates if the recovery routine was a STAE or ESTAE.

SVC and SVCR trace records

An SVC record represents a supervisor call (SVC) interruption. An SVCR record
represents an exit from a supervisor call. SDSP interruptions also build SVC exit
records with label SDSP. When both DSP and SVC options are in effect, the SVC
format of trace record is produced by IPCS.

If the trace data contains an SVC exit record, the label that appears in the
formatted output will depend on the options selected during IPCS.

1. If the SVC option is selected in the IPCS dialog, the SVC exit record and the
   SVC number will appear with the label SVCR.

2. If only the DSP option is chosen in the IPCS dialog, the formatted output
   record will remain unchanged; DSP and SDSP labels will appear in the
   formatted output and no SVC number will be present.

3. If both DSP and SVC options are active in IPCS, the SVCR along with SVC
   number will appear.

It can be concluded, if SVC is one of the options selected during IPCS formatting,
all SVC exit records will appear with label SVCR along with SVC number.

The format of an SVC and SVCR trace record depends on the SVC interruption
being traced. For a break down of the information that GTF collects for each SVC,
see the SVC Summary chapter of z/OS MVS Diagnosis: Reference. The formats shown
are typical.

Minimal Trace Record Format
**Comprehensive Trace Record Format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC CODE</td>
<td>hhh</td>
<td>For SVC, and for SVCR when not X'FFxx': SVC interruption code, which is also called the SVC number.</td>
</tr>
<tr>
<td>SVC</td>
<td>hhh</td>
<td>For SVC when X'FF00': completion of the system-initiated processing involved with ATTACH, LINK, XCTL or SYNCH processing prior to the target routine getting control.</td>
</tr>
<tr>
<td>SVC</td>
<td>hhh</td>
<td>For SVCR when X'FF01': initial system-initiated processing involved with XCTL processing prior to the target routine getting control.</td>
</tr>
<tr>
<td>ASCB</td>
<td>hhhhhhhh</td>
<td>Address of the ASCB for the address space in which the interruption occurred.</td>
</tr>
<tr>
<td>CPU</td>
<td>hhhh</td>
<td>Address of the processor on which the interruption occurred.</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>cccccccc</td>
<td>One of the following: cccccccc Name of the job associated with SVC interruption SSSSSSSSS Unavailable; GTF cannot provide data for the SVC due to security considerations. ******* An internal error occurred</td>
</tr>
<tr>
<td>PSW</td>
<td>hhhhhhhh</td>
<td>Program status word stored when the interruption occurred.</td>
</tr>
<tr>
<td>OLD-PSW</td>
<td>hhhhhhhh</td>
<td>For SVC, and for SVCR when SVC Code is not X'FFxx': Program status word stored when the interruption occurred.</td>
</tr>
<tr>
<td>DSP-PSW</td>
<td>hhhhhhhh</td>
<td>For SVCR when SVC Code is X'FF00': PSW of the target routine that will get control as a result of ATTACH, LINK, XCTL or SYNCH processing.</td>
</tr>
</tbody>
</table>

---

**Generalized Trace Facility**

---

Chapter 10. The Generalized Trace Facility (GTF)  307
Generalized Trace Facility

- For SVCR when SVC Code is X'FF01': PSW of a system routine that will get control as a result of initial processing involved with XCTL.

TCB  hhhhhhhh
Address of the TCB for the interrupted task, that is, the task that issued the SVC instruction.

R15  hhhhhhhh
R10  hhhhhhhh
R1   hhhhhhhh
Data in general registers 15, 0, and 1 when the SVC instruction ran.

MODN  cccccc

mod_name
The name of a module that will receive control when the task is dispatched.

SVC–T2
Indicates a type 2 SVC routine resident in the nucleus.

SVC–RES
Indicates a type 3 SVC routine or the first load module of a type 4 SVC routine. The routine is located in the pageable link pack area (PLPA).

SVC  cccc
Indicates the second or subsequent load module of a type 4 SVC routine. The routine is located in the fixed or pageable link pack area (LPA). The last four characters of the load module name are cccc.

**IRB****
Indicates an asynchronous routine with an associated interruption request block. No module name is available.

*cccccc
Indicates an error recovery module. The last seven characters of the load module name are cccccc.

PPPPPPPPP
A page fault occurred

********
An internal error occurred

DDNAM  cccccc
Name of the DD statement associated with the SVC, if applicable.

additional fields
Vary with the SVC number. These fields are described for the SVC in the z/OS MVS Diagnosis: Reference

TCW trace records
A TCW record represents the processing of a zHPF channel program. TCW trace records appear following INTG, IOCS, IO, SSCH, and XSCH trace records; they do not appear alone.

| FORMAT | d | cccc | TCW_CHAIN | DEVO... hhhhh | ASCB... hhhhhhh | CPU... hhhh | JOBN... hhhhh |

308  z/OS V2R2 MVS Diagnosis: Tools and Service Aids
Generalized Trace Facility

TCW at rrrrrrrr (vvvvvvvvv)
- Format........... hh
- Flag1............. hh
- Flag2............. hh
- Flag3............. hh
- TCCBL/R/W........ hh
- Output Address... rrrrrrrr rrrrrrrr
- Input Address.... rrrrrrrr rrrrrrrr
- TSB Address...... rrrrrrrr rrrrrrrr
- TCB Address...... rrrrrrrr rrrrrrrr
- Output Count..... hhhhhhhh
- Input Count...... hhhhhhhh
- Interrogate TCW.. rrrrrrrr

TSB at rrrrrrrr_rrrrrrrr
- tsbtype TSB at rrrrrrrr rr
- Length.......... hh
- Flags........... hh
- DCW Offset...... hhhh
- Count........... hhhhhhhh
- TotalDevTime.... hhhhhhhh
- DeferveTime...... hhhhhhhh
- CQQueueTime..... hhhhhhhh
- DevBusyTime..... hhhhhhhh
- DevActOnlyTime.. hhhhhhhh
- Sense Data...... hhhhhhhh
- hhhhhhhh
- hhhhhhhh
- hhhhhhhh
- hhhhhhhh

TCB TIDAW at rrrrrrrr_rrrrrrrr
- Flags... hh
- Count... hhhhhhhh
- Addr... rrrrrrrr rrrrrrrr

TCA Header at rrrrrrrr_rrrrrrrr
- Format......... hh
- TCALen.......... hh
- Serv Act Code... hhhh
- Priority....... hh

DCW at rrrrrrrr_rrrrrrrr
- Command.. hh
- Flags.. hh
- CD Count.. hh
- Count.. hhhhhhhh

DCW Control Data at rrrrrrrr_rrrrrrrr
- dddddddd dddddddd dddddddd dddddddd
- dddddddd dddddddd dddddddd dddddddd

Data TIDAW at rrrrrrrr_rrrrrrrr
- Flags... hh
- Count... hhhhhhhh
- Addr... rrrrrrrr rrrrrrrr

Data at rrrrrrrr_rrrrrrrr
- dddddddd dddddddd dddddddd dddddddd
- dddddddd dddddddd dddddddd dddddddd

**** '0020'X CONSECUTIVE BYTES ARE '00'X
*** Back half of split data ***
- dddddddd dddddddd dddddddd dddddddd
- dddddddd dddddddd dddddddd dddddddd

Chapter 10. The Generalized Trace Facility (GTF)  309
Generalized Trace Facility

**FORMAT d cccc**
Format (d) and type of trace event (cccc): INTG, IO, IOCS, SSCH, or XSCH. Format is zero.

**DEV hhhhh**
The device number qualified with the subchannel set identifier.

**ASCB hhhhhhh**
Same as the ASCB field in the INTG, IO, IOCS, SSCH, or XSCH base record.

**CPU hhhh**
Same as the CPU ID field in the INTG, IO, IOCS, SSCH, or XSCH base record.

**JOBN cccccc**
Same as the job named (JOBN) field in the INTG, IO, IOCS, SSCH, or XSCH base record.

**TCW at rrrrrrrr (vvvvvvvv)**
The Transport Control Word (TCW) at real address rrrrrrrr and virtual address vvvvvvvv. The formatted TCW follows. Fields designated as "rrrrrrrr" or "rrrrrrrr_rrrrrrrr" are real addresses.

**tsbtype TSB at rrrrrrrr_rrrrrrrr**
The Transport Status Block (TSB) at real address rrrrrrrr_rrrrrrrr. The formatted TSB follows. The TSB is only formatted for I/O interruptions (trace events INTG, IO, and IOCS).
tsbtype
Describes the type of TSB. It can be one of the following:

- I/O status - This is a TSB for an I/O completion.
- Interrogate - This is a TSB for the completion of an interrogate operation.
- Program Check - This is a TSB for an I/O completion with status indicating a device detected program check.
- Unknown - The TSB type is not recognized. In this case, the TSB is formatted as hexadecimal data.

TCCB TIDAW at rrrrrrrr_rrrrrrrr
A Transport Indirect Address Word (TIDAW) for the Transport Command Control Block (TCCB) at real address rrrrrrrr_rrrrrrrr. The formatted TIDAW follows.

TCA Header at rrrrrrrr_rrrrrrrr
The Transport Control Area Header (TCAH) at real address rrrrrrrr_rrrrrrrr. The formatted TCAH follows.

DCW at rrrrrrrr_rrrrrrrr
A Device Command Word (DCW) at real address rrrrrrrr_rrrrrrrr. The formatted DCW follows.

DCW Control Data at rrrrrrrr_rrrrrrrr
The control data (command parameters) for the preceding DCW at rrrrrrrr_rrrrrrrr. The control data is formatted as hexadecimal data.

Data TIDAW at rrrrrrrr_rrrrrrrr
A Transport Indirect Address Word (TIDAW) for the input or output data buffers at real address rrrrrrrr_rrrrrrrr. The formatted TIDAW follows.

Data at rrrrrrrr_rrrrrrrr
Data transferred by the preceding DCW at real address rrrrrrrr_rrrrrrrr.

dddddddd dddddddd dddddddd dddddddd
Data transferred by the DCW. If there is not a series of dashes in this field, then all transferred data are displayed in four byte sections.

*** Back half of split data ***
Indicates there were more bytes of information transferred than were specified on the START command. The default value is 20 bytes, but you can specify the number of bytes to be shown. The specified value is halved; for an odd number, the larger section is shown first. The first section of data displayed comes from the beginning of the buffer from which the data was transferred. The last section comes from the end of the buffer.

TCA Trailer at rrrrrrrr_rrrrrrrr
The Transport Control Area Trailer (TCAT) at real address rrrrrrrr_rrrrrrrr. The formatted TCAT follows.

Transport Count hhhhhhhh
Fullword count of total data transferred.

Write Count hhhhhhhh
Fullword count of total write data transferred.

Read Count hhhhhhhh
Fullword count of total read data transferred.
TCAX TIDAW at rrrrrrrr_rrrrrrrr
A Transport Indirect Address Word (TIDAW) for the Transport Control Area Extension (TCAX) at real address rrrrrrrr_rrrrrrrr. The formatted TIDAW follows.

DCW (TCAX) at rrrrrrrr_rrrrrrrr
A Device Command Word (DCW) in the Transport Control Area Extension (TCAX) at real address rrrrrrrr_rrrrrrrr. The formatted DCW follows.

DCW Control Data (TCAX) at rrrrrrrr_rrrrrrrr
The control data (command parameters) for the preceding DCW in the TCAX at rrrrrrrr_rrrrrrrr. The control data is formatted as hexadecimal data.

IOSB vvvvvvvv
Fullword virtual address of the IOSB followed by the formatted contents of the IOSB.

IOBE vvvvvvvv
Fullword virtual address of the IOBE followed by the formatted contents of the IOBE.

EWA vvvvvvvv
Fullword virtual address of the error recovery procedure work area (EWA), followed by the formatted contents of EWA.

USR trace records
The USR record represents processing of a GTRACE macro. A user-supplied formatting routine (AMDUSRhh) formats the record. If a routine is not supplied, GTF prints the record without formatting.

This topic shows the unformatted and formatted records, then shows the following examples of USR records created by GTRACE macros in IBM-components:
- USRF9 trace records for VSAM
- USRFE trace records for BSAM, QSAM, BPAM, and BDAM
- USRFF trace records for open, close, and end-of-volume (EOV)

The USRFD trace records for VTAM are described in z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures.

USR records contain the following information useful for identifying the user program, MVS component, or IBM product producing the record and the routine you can use to format the record:
- Event identifiers (EIDs) identify the event that produced the record. See “Event Identifiers (EIDs) for USR trace records” on page 317 for a list of the EIDs and associated products for USR trace records. Because each EID for USR records start with an E, unformatted USR records show just the last three numbers of the EID after the E.
- Format identifiers (FIDs) identify the routine that the system used to format the USR trace record. See “Format Identifiers (FIDs) for USR trace records” on page 319 for a list of the FIDs and associated routines.

Unformatted USR trace record
An unformatted user trace record represents processing of a GTRACE macro when a formatting routine is not supplied.
Generalized Trace Facility

USR AID hh FID hhhh EID hhhh hhhhhhh hhhhhhhhh ....

AID hh
Application identifier, which should always be AID FF.

FID hhhh
Format identifier of the routine (AMDUSRhh) that was to format this record. See “Format Identifiers (FIDs) for USR trace records” on page 319 for a list of the FIDs and associated formatting routines for user trace records.

EID hhhh
Event identifier, which identifies the event that produced the record. See “Event Identifiers (EIDs) for USR trace records” on page 317 for a list of the EIDs and associated products for USR trace records.

hhhhhhhh hhhhhhh ....
Recorded data (268 bytes maximum). The data are as follows:
- Bytes 0-3: ASCB address
- Bytes 4-11: jobname
- Bytes 12-256: user data

Formatted USR trace record
A formatted user trace record represents processing of a GTRACE macro when an AMDUSRhh formatting routine is supplied.

USRhh hhh ASCB hhhhhhhhh JOBN cccccc
xxxx ...

USRhh
Identifies the user-supplied formatting routine (AMDUSRhh). The following USR records are generated and formatted by system components, and are described in the following topics:
- USRF9 Trace Records for VSAM
- USRFD Trace Records for VTAM
- USRFE Trace Records for BSAM, QSAM, BPAM, and BDAM
- USRFF Trace Records for Open/Close/EOV

hhh
Last three numbers of the event identifier (EID) specified in the GTRACE macro. See “Event Identifiers (EIDs) for USR trace records” on page 317 for a list of the EIDs and associated products for USR trace records.

ASCB hhhhhhhhh
Address of the ASCB for the address space that created the record.

JOBN cccccc
Name of the job associated with the address space.

xxxx ...
User-formatted trace data.

USRF9 trace record for VSAM
The USRF9 trace record represents opening or closing of a VSAM data set.
Generalized Trace Facility

USRF9 FF5 ASCB hhhhhhh JOBN cccccc
   JOB NAME cccccc STEP NAME cccccc
   TIOT ENT hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   ACB hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   AMBL hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   AMB hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   AMDSB hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   AMB hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   AMDSB hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhhhhhh hhhhhhh hhhhhhh hhhhhhh

USRF9
   Identifies VSAM's trace-record formatting routine (AMDUSRF9).

FF5
   Last three numbers of the event identifier (EID) specified in the GTRACE macro. See "Event Identifiers (EIDs) for USR trace records" on page 317 for a list of the EIDs and associated products for USR trace records.

ASCB hhhhhhh
   Address of the ASCB for the address space in which the event occurred.

JOBN cccccc

JOB NAME cccccc
   Name of the job.

STEP NAME cccccc
   Name of the job step during which the event occurred.

TIOT ENT hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   Data set entry from the task I/O table (TIOT).

ACB hhhhhhh ...
   Contents of the data set's access method control block (ACB).

AMBL hhhhhhh ...
   Contents of the AMB list (AMBL).

AMB hhhhhhh ...
   Contents of the access method block (AMB). The first AMB is for data, the second for the index.

AMDSB hhhhhhh ...
   Contents of the access method statistics block (AMDSB). The first AMDSB is for data, the second for the index.

USRFD trace record for VTAM
   See z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures for samples of the USRFD trace records.
**USRFE trace record for BSAM, QSAM, BPAM, and BDAM**

The USRFE trace record represents abnormal termination of an access method routine for basic sequential access method (BSAM), queued sequential access method (QSAM), basic partitioned access method (BPAM), or basic direct access method (BDAM).

```
USRFE hhh ASCB hhhhhhhh JOBN cccccc
    BSAM/QSAM/BPAM/BDAM TRACE RECORD DDNAME cccccc ABEND CODE hh
     cccc...[AT LOCATION hhhhhhhh]
     hhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh ... hhhhhhh hhhhhhhh hhhhhhhh ...
```

**USRFE**
Identifies the trace-record formatting routine (AMDUSRFE).

**hhh**
Last three numbers of the event identifier (EID) specified in the GTRACE macro. See “Event Identifiers (EIDs) for USR trace records” on page 317 for a list of the EIDs and associated products for USR trace records. The event identifier (EID) corresponds to the system completion code as follows:

<table>
<thead>
<tr>
<th>EID</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF3</td>
<td>002</td>
</tr>
<tr>
<td>FF4</td>
<td>008</td>
</tr>
<tr>
<td>FF6</td>
<td>112</td>
</tr>
<tr>
<td>FF7</td>
<td>215</td>
</tr>
<tr>
<td>FF8</td>
<td>119</td>
</tr>
<tr>
<td>FF9</td>
<td>235</td>
</tr>
<tr>
<td>FFA</td>
<td>239</td>
</tr>
<tr>
<td>FFB</td>
<td>145</td>
</tr>
<tr>
<td>FFC</td>
<td>251</td>
</tr>
<tr>
<td>FFD</td>
<td>451</td>
</tr>
<tr>
<td>FFE</td>
<td>169</td>
</tr>
</tbody>
</table>

**ASCB hhhhhhhh**
Address of the ASCB for the address space in which the abnormal termination occurred.

**JOBN cccccc**
Name of the job associated with the address space.

**BSAM/QSAM/BPAM/BDAM TRACE RECORD**
Record identification provided by the AMDUSRFE formatting routine.

**DDNAME cccccc**
Name of the DD statement for the data set being processed.

**ABEND CODE hhh**
System completion code for the abnormal termination of the task.

**RETURN CODE hh**
Return code from the module that detected the error condition.

**TIME=dd.dd.dd**
Time (hour.minute.second) when the GTRACE macro was processed or blank, if the time is not available.

**ccc...[AT LOCATION hhhhhhhh]**
**Data area name, or name and address, followed by the data area contents.**

**USRFF trace record for open/close/EOV abnormal end**

This USRFF trace record represents an abnormal end during open, close, or end-of-volume (EOV).

```
USRFF  FFF  ASCB hhhhhhh  JOBN cccccc
       xxxx ... 
```

**USRFF**

Identifies the Open/Close/EOV trace record formatting routine (IMDUSRFF).

**FFF**

Last three numbers of the event identifier (EID) specified in the GTRACE macro. See "Event Identifiers (EIDs) for USR trace records" on page 317 for a list of the EIDs and associated products for USR trace records.

**xxxx ...**

Unformatted RRCBSA’s (recovery routine control block save areas).

**USRFF trace record for user requested work area**

This USRFF trace record represents a user request for a work area trace.

```
USRFF  FFF  ASCB hhhhhhh  JOBN cccccc
       DCB  xxxx ...
       WKAREA1  xxxx ...
       WKAREA2  xxxx ...
       WKAREA3  xxxx ...
       WKAREA4  xxxx ...
       WKAREA5  xxxx ...
       WTG TBL  xxxx ...
```

**USRFF**

Identifies the Open/Close/EOV trace record formatting routine (IMDUSRFF).

**FFF**

Last three numbers of the event identifier (EID) specified in the GTRACE macro. See "Event Identifiers (EIDs) for USR trace records" on page 317 for a list of the EIDs and associated products for user trace records.

**DCB**

Data control block.

**WKAREA1**

Volume labels, file labels, DSCBs or message area. See z/OS DFSMS Using Magnetic Tapes.

**WKAREA2**

Job file control block.

**WKAREA3**

Internal control blocks for Open/Close/EOV. These blocks are the data control block (DCB), data extent block (DEB), and the input/output block (IOB).

**WKAREA4**
Generalized Trace Facility

**WKAREA5**
Where-to-go-table used in transferring control among CSECTs of Open/Close/EOV.

**XSCH trace record**
An XSCH record represents a cancel subchannel operation. For zHPF I/O operations, a cancel subchannel can be used to initiate an interrogate operation to query the status of the I/O operation at the device.

<table>
<thead>
<tr>
<th>XSCH</th>
<th>Device number with the subchannel set identifier that the XSCH was issued for.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCB</td>
<td>Address of the ASCB.</td>
</tr>
<tr>
<td>CPU</td>
<td>Address of the processor.</td>
</tr>
<tr>
<td>JOBN</td>
<td>Name of the job associated with I/O operation.</td>
</tr>
<tr>
<td>SID</td>
<td>Subchannel ID from the UCBSID field of the UCB</td>
</tr>
<tr>
<td>CC</td>
<td>The condition code of the XSCH request.</td>
</tr>
<tr>
<td>DVRID</td>
<td>The IOSB driver ID field (IOSDVRID) of the request that is attempting to be cancelled.</td>
</tr>
<tr>
<td>IOSLVL</td>
<td>Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.</td>
</tr>
<tr>
<td>UCBlvl</td>
<td>UCB level value from the UCIBLEVEL field of the UCB.</td>
</tr>
<tr>
<td>UCBWGT</td>
<td>Flags from the UCBWGT field of the UCB.</td>
</tr>
<tr>
<td>BASE</td>
<td>The base device number and subchannel set id if the device is a PAV.</td>
</tr>
<tr>
<td>INTTCW</td>
<td>The virtual address of the interrogate TCW, if the XSCH was used to initiate an interrogate operation, or zero.</td>
</tr>
</tbody>
</table>

**Event Identifiers (EIDs) for USR trace records**
The event identifier (EID) in GTF trace records is a 2-byte hexadecimal number that identifies the event producing the record. You can use it to identify the product that produced the record. Table 41 on page 318 shows the full 2-byte EID, but because EIDs for USR records start with an E, often unformatted USR records
show just the last three numbers of the EID after the E. If you have a three number EID, such as FF5, look for EFF5 in the table.

Table 41. Event identifiers for USA trace records

<table>
<thead>
<tr>
<th>EID (hex)</th>
<th>Symbolic Name</th>
<th>Issued by</th>
</tr>
</thead>
<tbody>
<tr>
<td>E000-E3FF</td>
<td>GTF user program</td>
<td></td>
</tr>
<tr>
<td>E400-E5F0</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF1</td>
<td>PVM</td>
<td></td>
</tr>
<tr>
<td>ESF2-ESF3</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF4-ESF5</td>
<td>NetView® Session Monitor</td>
<td></td>
</tr>
<tr>
<td>ESF6</td>
<td>NetView</td>
<td></td>
</tr>
<tr>
<td>ESF7</td>
<td>IMDOMGM</td>
<td>Omegamon for DB2</td>
</tr>
<tr>
<td>ESF8-ESF9</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESFA-ESFB</td>
<td>ALCS</td>
<td></td>
</tr>
<tr>
<td>ESFC-ESF0</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF1C</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF1F</td>
<td>MVS Job Management - Dynamic Allocation (SVC 99)</td>
<td></td>
</tr>
<tr>
<td>ESF20-ESF3F</td>
<td>LANRES</td>
<td></td>
</tr>
<tr>
<td>ESF41</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF42</td>
<td>IMDEF42</td>
<td>IBM Client I/O Sockets</td>
</tr>
<tr>
<td>ESF43</td>
<td>IMDEF43</td>
<td>MVS System logger</td>
</tr>
<tr>
<td>ESF44-ESF45</td>
<td>RACF</td>
<td></td>
</tr>
<tr>
<td>ESF46</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF47</td>
<td>IMDEF47</td>
<td>OSI</td>
</tr>
<tr>
<td>ESF48</td>
<td>IOS</td>
<td></td>
</tr>
<tr>
<td>ESF49</td>
<td>BDT</td>
<td></td>
</tr>
<tr>
<td>ESF50</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF51</td>
<td>Netview</td>
<td></td>
</tr>
<tr>
<td>ESF52</td>
<td>OSI</td>
<td></td>
</tr>
<tr>
<td>ESF53</td>
<td>OSI</td>
<td></td>
</tr>
<tr>
<td>ESF54-ESF5D</td>
<td>FSI</td>
<td></td>
</tr>
<tr>
<td>ESF5E</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF5F</td>
<td>DB2</td>
<td></td>
</tr>
<tr>
<td>ESF60</td>
<td>JES3</td>
<td></td>
</tr>
<tr>
<td>ESF61</td>
<td>VSAM Buffer Manager</td>
<td></td>
</tr>
<tr>
<td>ESF62</td>
<td>Dynamic output SVC installation exit</td>
<td></td>
</tr>
<tr>
<td>ESF63</td>
<td>Converter/Interpreter installation exit</td>
<td></td>
</tr>
<tr>
<td>ESF64</td>
<td>APPC/VM VTAM Support (AVS)</td>
<td></td>
</tr>
<tr>
<td>ESF65</td>
<td>GETMAIN FREEMAIN STORAGE trace (MVS)</td>
<td></td>
</tr>
<tr>
<td>ESF66-ESF6B</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF6C</td>
<td>CICS</td>
<td></td>
</tr>
<tr>
<td>ESF6D-ESF8C</td>
<td>Netware</td>
<td></td>
</tr>
<tr>
<td>ESF8D-ESF8F</td>
<td>Reserved for IBM use</td>
<td></td>
</tr>
<tr>
<td>ESF90</td>
<td>IMDVTMDS</td>
<td>VTAM</td>
</tr>
</tbody>
</table>
Format Identifiers (FIDs) for USR trace records

As Table 42 on page 320 shows, the format identifier (FID) in GTF trace records is a one-byte hexadecimal number that is used to determine the name of the GTFTRACE module you can use to format USR records. See [z/OS MVS IPCS Customization] for information about the GTFTRACE formatting appendage for formatting USR trace records.
Table 42. Format identifiers for USR trace records

<table>
<thead>
<tr>
<th>FID (hex)</th>
<th>EID</th>
<th>Issued by</th>
<th>Optional format module</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>E000-EFE4</td>
<td>User/component</td>
<td>CSECT AHIFFILT in AHLFINIT</td>
</tr>
<tr>
<td>01-50</td>
<td>E000-E3FF</td>
<td>User</td>
<td>IMDUSR or AMDUSR (01-50)</td>
</tr>
<tr>
<td>57</td>
<td>EF44-EF45</td>
<td>RACF</td>
<td>AMDUSR57</td>
</tr>
<tr>
<td>81</td>
<td>VMSI</td>
<td>IMDUSR81 or AMDUSR81</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>VMSI/VTAM</td>
<td>IMDUSR84 or AMDUSR84</td>
<td></td>
</tr>
<tr>
<td>DC</td>
<td>PVM</td>
<td>IMDUSRDC or AMDUSRDC</td>
<td></td>
</tr>
<tr>
<td>E2-E3</td>
<td>PSF/MVS</td>
<td>IMDUSRE2-IMDUSER3 or AMDUSRE2-AMDUSER3</td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td>OSI</td>
<td>IMDUSRE6 or AMDUSRE6</td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>FSI</td>
<td>IMDUSRE8 or AMDUSRE8</td>
<td></td>
</tr>
<tr>
<td>E9</td>
<td>DB2/VSAM</td>
<td>IMDUSRE9 or AMDUSRE9</td>
<td></td>
</tr>
<tr>
<td>EB</td>
<td>APPC/VM</td>
<td>IMDUSREB or AMDUSREB</td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td>VTAM</td>
<td>IMDUSREC or AMDUSREC</td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>VTAM/VSCS</td>
<td>IMDUSR5 or AMDUSR5</td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>EFF5</td>
<td>VSAM</td>
<td>IMDUSR9 or AMDUSR9</td>
</tr>
<tr>
<td>FA</td>
<td>EFAB</td>
<td>DFSMS Media Manager</td>
<td>IMDUSRFA or AMDUSRFA</td>
</tr>
<tr>
<td>FD</td>
<td>EEF-EFF2</td>
<td>VTAM</td>
<td>IMDUSRFD or AMDUSRFD</td>
</tr>
<tr>
<td>FE</td>
<td>EFF3-EFF4, EFF6-EFFE</td>
<td>SAM/PAM/DAM</td>
<td>IMDUSRFE or AMDUSRFE</td>
</tr>
</tbody>
</table>

Unformatted GTF trace output

This topic describes GTF output records that are not formatted by IPCS or other routines. You can use this information to write your own formatting or analysis routines.

Note: When GTF cannot obtain the data normally placed in fields of the following records, it signals this by placing one of the following values in the field:
- C'U/A'. Blanks are added on the right to fill out the field.
- C'*'. Asterisks are replicated to fill out the field.

There are several types of output records:
- Control records, see “Control records” on page 321.
- Lost data records, see “Unformatted lost event records” on page 322.
- User data record, see “User data records” on page 323.
- System data records, see “System data records” on page 324.

The lost data, user data and system data records all contain optional fields, which are fields that only appear under certain conditions. The conditions are covered in the explanation for the fields. Make sure that your formatting or analysis routine takes these variable fields into account.

This section also describes the GTF system data records for individual events. See “CCW trace record” on page 326 through “SVC minimal trace record” on page 341.
Control records

GTF creates a control record at the start of each block of trace output. The control record can be followed by lost data, user data, and system data records. If this trace output was merged from multiple systems using the IPCS COPYTRC subcommand, then the control record reflects the combined GTF options in effect from all the systems. See z/OS MVS IPCS Commands for more information about the COPYTRC subcommand. Figure 99 shows the format of a control record.

| length | res | AID | FID   | time zone | time stamp | options | ... source descriptors ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>bytes by byte byte bytes bytes bytes bytes...</td>
</tr>
</tbody>
</table>

*Figure 99. Unformatted control record*

The fields in the control record contain the following information:

**length**
- Total length of the record, in bytes.

**res**
- Two bytes of zeroes. Reserved for IBM use.

**AID**
- Application identifier, which is always zero for control records.

**FID**
- Format identifier of the routine that will format the record, which is always X'01' for a control record.

**time zone**
- Value showing the difference between local time and Greenwich mean time (GMT) in binary units of 1.048576 seconds when tracing began.

**time stamp**
- Time stamp showing the eight-byte Greenwich mean time (GMT) when the control record was created.

**options**
- An eight-byte field containing the following: The first five bytes identify the GTF options in effect for a block of trace output. See mapping macro AHLZGTO in z/OS MVS Data Areas in z/OS Internet library [http://www.ibm.com/systems/z/os/zos/bkserv/](http://www.ibm.com/systems/z/os/zos/bkserv/)

The remaining 3 bytes contain the following important flags, in bit ranges 0-7:

**GTWCFSID - Byte 6, Bit 6**
- 1, if the individual trace records have SIDs (system identifiers) indicating that the GTF trace data from multiple systems was merged using the IPCS COPYTRC command. In this case, there is multiple source descriptors, one for each system. The source descriptors are ranged in order by system identifier (SID). Use the value in the SID field as an array index to locate the source descriptor for a particular system.

- 0, if the trace records have no SIDs.

**GTWCFNEW - Byte 6, Bit 7**
- 1.
Generalized Trace Facility

Source descriptors
One or more arrays of information about the origins of the records in this block of trace data, such as the release level of the system issuing the trace data and the GTF options in effect. If GTF trace data was merged from multiple systems, there are multiple source descriptors, one for each system. Use the value in the SID field as an array index to locate the source descriptor for a particular system.

See mapping macro AHLZGTS in z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/) and check the format of the source descriptor information.

Unformatted lost event records
A lost event record indicates that GTF lost the trace records for one or more events because of an error or overflow of the trace buffer. Figure 100 shows the format of a lost event record.

<table>
<thead>
<tr>
<th>length</th>
<th>res</th>
<th>AID</th>
<th>FID</th>
<th>time zone</th>
<th>time stamp</th>
<th>count</th>
<th>SID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>bytes</td>
<td>bytes</td>
<td>byte</td>
<td>byte</td>
<td>bytes</td>
<td>bytes</td>
<td>bytes</td>
<td>bytes</td>
</tr>
</tbody>
</table>

Figure 100. Unformatted Lost Event Record

The fields in the lost event record contain the following information:

length
Total length of the record, in bytes.

res
Two bytes of zeroes. Reserved for IBM use.

AID
Application identifier, which is always zero for lost event records.

FID
Format identifier. The value of FID is one of the following:
- X’02’, if some trace records are missing because of an error or an overflow of the trace buffer.
- X’03’, if an entire block of trace records is missing because of an error or an overflow of the trace buffer.

time zone
Value showing the difference between local time and Greenwich mean time (GMT) in binary units of 1.048576 seconds when tracing began.

time stamp
Time stamp showing the eight-byte Greenwich mean time (GMT) when the control record was created.

count
If the FID is X’02’, indicating that some trace records are missing, this field contains the number of trace events that are lost.
If the FID is X’03’, indicating that an entire block of trace data is missing, this field contains zeros.

SID
The system identifier of the system where this trace record was created. This 2-byte field only exists when GTF trace data from multiple systems was
merged using the IPCS COPYTRC command. When present, the SID is an
array index you can use to locate the source descriptor information for a
particular system. For example, if the SID value for a record is 3, the source
descriptor information for the system issuing the record is the third source
descriptor in the control record.

To check to see whether trace data for a block of output comes from multiple
systems, look in the control record for the options field and see if the
GTWCFSID bit is set on. See “Control records” on page 321 for the options
field.

**User data records**

This topic describes the format of user trace records requested using the GTRACE
macro.

If the application using GTRACE specifies more than 256 bytes of data, the user
records may be split. If a user trace record is a split record, the AID will contain a
value of X'F0', X'F1', or X'F2'. Split records contain the optional sequence and total
length fields.

The records have the general format shown in [Figure 101].

```
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
| length | res    | AID    | FID    | time stamp | EID    | SID    | sequence | total length | ASCB | job name | ... data ...
| 2      | 2      | 1      | 1      | 8          | 2      | 2      | 2        | 4          | 4    | 8        | ...
```

*Figure 101. Unformatted User trace record Format*

The fields in the record contain the following information:

- **length**
  Total length of the record, in bytes.

- **res**
  Two bytes of zeros. Reserved for IBM use.

- **AID**
  Application identifier, which is one of the following:
  - X'FF'-- Non-split record
  - X'F0'-- The first record of a series of split records
  - X'F1'-- A middle record in a series of split records
  - X'F2'-- The last record in a series of split records

- **FID**
  Format identifier of the routine that will format the trace record. See “Format
  Identifiers (FIDs) for USR trace records” on page 319 for a list of FIDs and
  associated formatting routines.

- **time stamp**
  Time stamp showing the eight-byte Greenwich mean time (GMT) when the
  record was created.

- **EID**
  Event identifier, which identifies the event that produced the trace record. See
  “Event Identifiers (EIDs) for USR trace records” on page 317 for a list of the
  EIDs and associated products for user trace records.
SID
System identifier, which identifies the system where the record was produced. This 2-byte field only exists in the following cases:

- GTF trace data from multiple systems was merged using the IPCS COPYTRC command.
- The record is a split one. If the trace data containing this split record was not merged from multiple systems, the SID field for the split record contains zeros.

You can use the SID from merged trace data as an array index to locate the source descriptor information for a particular system. For example, if the SID value for a record is 3, the source descriptor information for the system issuing the record is the third source descriptor in the control record.

To check to see whether trace data for a block of output comes from multiple systems, look in the control record for the options field and see if the GTWCFSID bit is set on. See "Control records" on page 321 for the options field.

sequence
Sequence number, in hexadecimal, of this split record. This field only exists for split records.

total length
Total length of the split trace data. This field only exists for split records.

ASCB
The address of the address space control block (ASCB) for the address space where the GTRACE macro was issued.

jobname
The name of the job associated with the task where the GTRACE macro was issued.

data
Contains the trace data gathered for the requested event. The length of this field varies according to the event being traced. The number of bytes of data in the data field for user records is equal to the number of bytes specified on the GTRACE macro.

System data records
GTF creates trace records for each system event you select when requesting GTF tracing. The header portion of system data records for events is shown in Figure 102. Individual event record formats The format of individual system data records are shown in "Unformatted trace records for events" on page 325 in alphabetical order. Note that this section does not include all system events.

The fields in the record contain the following information:

length
Total length of the record, in bytes.
Two bytes of zeros. Reserved for IBM use.

**AID**
Application identifier, which is always X'FF' for system data records.

**FID**
Format identifier of the routine that will format the trace record.

**time stamp**
Time stamp showing the eight-byte Greenwich mean time (GMT) when the record was created.

**EID**
Event identifier, which identifies the event that produced the trace record.

**SID**
System identifier, which identifies the system where the record was produced. The SID field contains zeros when the record is a split record. This 2-byte field is only created when GTF trace data from multiple systems was merged using the IPCS COPYTRC command. When present, the SID is an array index you can use to locate the source descriptor information for a particular system. For example, if the SID value for a record is 3, the source descriptor information for the system issuing the record is the third source descriptor in the control record.

To check to see whether trace data for a block of output comes from multiple systems, look in the control record for the **options** field and see if the GTWCFSID bit is set on. See "Control records" on page 321 for the **options** field.

**data**
Trace data gathered for the requested event. The length of this field varies according to the event being traced. The data portions for individual system trace records are shown starting on "Unformatted trace records for events."

---

### Unformatted trace records for events

This topic presents the records for a selection of system events in alphabetical order. It shows the unformatted layout of the data for individual event records. See "System data records" on page 324 to see the header section for the records. Note that not all system events are included in this topic. Fields in a trace record may contain the following special indicators:

- **N/A** Not applicable. The field does not apply in this record. In a 2-byte field, not applicable appears as N/.
- **U/A** Unavailable. GTF could not gather the information. In a 2-byte field, unavailable appears as U/.

The offsets for all the data records are relative and do not reflect the actual number of bytes into the record for each field. The offsets begin at the start of the data portion of the each record because the header section varies in length, depending on whether the optional SID field is present.

### ADINT trace record

For a complete mapping of the adapter interruption trace record, see the AHLPCIE (PCIE trace record formats) data area in `z/OS MVS Data Areas` in [z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/).
**Generalized Trace Facility**

**CCW trace record**
For a complete mapping of the AHLMCWRC data area, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

**DSP comprehensive trace record**
GTF builds a DSP record when an entry is made to the dispatcher to dispatch a unit of work and TRACE=DSP is the GTF option in effect.

The FID for the DSP comprehensive trace record is X'00'. The EID is one of the following:
- X'0001' - indicates SRB dispatching.
- X'0002' - indicates LSR dispatching.
- X'0003' - indicates TCB dispatching.
- X'0004' - indicates exit prolog dispatching.

*Table 43. DSP trace record offset, size, and description*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>8</td>
<td>Jobname.</td>
</tr>
<tr>
<td>14 (E)</td>
<td>16</td>
<td>Resume PSW for new task.</td>
</tr>
<tr>
<td>30 (1E)</td>
<td>4</td>
<td>New TCB (for LSR and TCB, SRB for SRB).</td>
</tr>
</tbody>
</table>

For TCB only:
- 34 (22) 8 CDE name.

For SRB only:
- 34 (22) 4 Parm address.
- 38 (26) 1 For SRB only, SRB routine type indicator.
  - S for a suspended SRB that is about to be re-dispatched.
  - I for an SRB that is about to be dispatched for the first (initial) time.

**DSP minimal trace record**
GTF builds a DSP minimal record when an entry is made to the dispatcher to dispatch a unit of work and both TRACE=SYSM,DSP are the GTF options in effect.

The FID for the DSP minimal trace record is X'03'. The EID is one of the following values:
- X'0001' - indicates SRB dispatching.
- X'0002' - indicates LSR dispatching.
- X'0003' - indicates TCB dispatching.
- X'0004' - indicates exit prolog dispatching.

*Table 44. Values for DSP minimal trace record*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
</tbody>
</table>
### Generalized Trace Facility

#### Table 44. Values for DSP minimal trace record (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (6)</td>
<td>16</td>
<td>Resume PSW for work unit that is being dispatched.</td>
</tr>
<tr>
<td>22 (16)</td>
<td>4</td>
<td>Current TCB or N/A (for TCB and LSR only).</td>
</tr>
<tr>
<td>26 (1A)</td>
<td>4</td>
<td>Register 15.</td>
</tr>
<tr>
<td>30 (1E)</td>
<td>4</td>
<td>Register 0 or SRB.</td>
</tr>
<tr>
<td>34 (22)</td>
<td>4</td>
<td>Register 1.</td>
</tr>
<tr>
<td>38 (26)</td>
<td>1</td>
<td>For SRB only. The SRB routine type indicator.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S for a suspended SRB that is about to be redispached.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I for an SRB that is about to be dispatched for the first (initial) time.</td>
</tr>
</tbody>
</table>

#### EXT comprehensive trace record

GTF builds an EXT comprehensive record when an external interruption occurs and either TRACE=SYS or TRACE=EXT are the GTF options in effect.

The FID for the EXT comprehensive trace record is X'02'. The EID is X'6201'.

#### Table 45. EXT comprehensive trace record offset, size, and description

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>8</td>
<td>Jobname.</td>
</tr>
<tr>
<td>14 (E)</td>
<td>16</td>
<td>External old PSW.</td>
</tr>
<tr>
<td>30 (1E)</td>
<td>4</td>
<td>Old TCB.</td>
</tr>
</tbody>
</table>

For SIGP interrupt:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34 (22)</td>
<td>4</td>
<td>PARMFIELD.</td>
</tr>
<tr>
<td>38 (26)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
</tbody>
</table>

For clock comparator interrupt:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34 (22)</td>
<td>2</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>4</td>
<td>TQE exit.</td>
</tr>
<tr>
<td>40 (28)</td>
<td>4</td>
<td>TQE ASCB.</td>
</tr>
</tbody>
</table>

For CPU timer interrupt:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34 (22)</td>
<td>2</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>4</td>
<td>TQE exit.</td>
</tr>
</tbody>
</table>

#### EXT minimal trace record

GTF builds an EXT minimal record when an external interruption occurs and TRACE=SYSM is the GTF option in effect.

The FID for the EXT minimal trace record is X'03'. An EID of X'6201' indicates an external interruption.
### Generalized Trace Facility

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>16</td>
<td>External old PSW.</td>
</tr>
<tr>
<td>22 (16)</td>
<td>4</td>
<td>TCB of interrupted task or N/A.</td>
</tr>
<tr>
<td>26 (1A)</td>
<td>4</td>
<td>NTQE TCB or INT CPU or N/A.</td>
</tr>
</tbody>
</table>

### I/O summary trace record

GTF builds an I/O summary record when an I/O interruption occurs and TRACE=IOX or TRACE=IOXP is a GTF option in effect. To trace PCI I/O interruptions, TRACE=PCI must also be in effect.

The FID for the I/O summary record is 'X'08'. The EID is one of the following:

- 'X'2100' - indicates a PCI I/O interruption
- 'X'5107' - indicates an EOS I/O interruption
- 'X'5202' - indicates an I/O interruption with a valid UCB
- 'X'5203' - indicates a I/OCS I/O interruption. It indicates an I/O interruption that also contains concurrent sense information, for devices that support the concurrent sense facility.

The I/O summary record will always contain a header section, followed by a section header and a common section. The section header describes the type and length of the following section and an indicator if this is the last section of the record.

A typical I/O summary record for a dasd device would have a header section, a common section, a data set section, a CMB section and, probably, one or more CCW sections. If an I/O summary record has to be extended the following extension records would consist of a header section with the header record number greater than 1, a common section and one or more CCW sections.

### Header section:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>8</td>
<td>Jobname.</td>
</tr>
<tr>
<td>14 (E)</td>
<td>2</td>
<td>Device number.</td>
</tr>
<tr>
<td>16 (10)</td>
<td>4</td>
<td>System ID.</td>
</tr>
<tr>
<td>20 (14)</td>
<td>1</td>
<td>Driver ID.</td>
</tr>
<tr>
<td>21 (15)</td>
<td>1</td>
<td>Trace version.</td>
</tr>
<tr>
<td>22 (16)</td>
<td>2</td>
<td>Record count.</td>
</tr>
</tbody>
</table>
### Generalized Trace Facility

#### Offset | Size | Description
---|---|---
0 (0) | 1 | Section type:
  - FL1'0': Common section
  - FL1'1': CMB section
  - FL1'3': CCW Orientation section
  - FL1'4': Data set section
1 (1) | 1 | Flag identifiers.
1... ... | Last section of this record.
2 (2) | 2 | Section length.

#### Common section:

Offset | Size | Description
---|---|---
0 (0) | 1 | Device class.
1 (1) | 1 | Device status.
2 (2) | 1 | Error codes: indicate errors found during CCW analysis. See "CCW error codes" on page 331 for a description.
3 (3) | 1 | Flag byte.
1... ... | Last trace record of this I/O event.
1... ... | Reserve (conditional or unconditional).
1... ... | Release.
1... ... | At least one search CCW.
4 (4) | 6 | Volume serial.
10 (A) | 4 | Device type.

#### Data set section:

Offset | Size | Description
---|---|---
0 (0) | 1 | Data set type.
1 (1) | 1 | Name length.
2 (2) | 44 | Data set name.

#### CMB Section:

Offset | Size | Description
---|---|---
0 (0) | 2 | Number of SSCH instructions.
2 (2) | 2 | Number of SSCH instructions for which data was collected.
4 (4) | 4 | Sum of device connect times.
8 (4) | 4 | Sum of SSCH request pending times.
12 (C) | 4 | Sum of subchannel disconnect times.
16 (10) | 4 | Sum of control unit queueing times.
20 (14) | 8 | Time stamp for the start of this I/O request.
28 (1C) | 4 | Device active only time.
### Generalized Trace Facility

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 (20)</td>
<td>4</td>
<td>Number of SSCH instructions</td>
</tr>
<tr>
<td>36 (24)</td>
<td>4</td>
<td>Number of SSCH instructions for which data was collected</td>
</tr>
<tr>
<td>40 (28)</td>
<td>4</td>
<td>Sum of device busy times</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>4</td>
<td>Sum of initial command response times</td>
</tr>
</tbody>
</table>

### CCW orientation section:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>1</td>
<td>Orientation sequence number.</td>
</tr>
<tr>
<td>1 (1)</td>
<td>1</td>
<td>Flag byte 1.</td>
</tr>
<tr>
<td>1... ....</td>
<td>At least one SILI bit on.</td>
<td></td>
</tr>
<tr>
<td>..1. ....</td>
<td>At least one suspend bit on.</td>
<td></td>
</tr>
<tr>
<td>..1. ....</td>
<td>At least one PCI.</td>
<td></td>
</tr>
<tr>
<td>...1 ....</td>
<td>At least one skip bit on.</td>
<td></td>
</tr>
<tr>
<td>.... 1...</td>
<td>Read record zero or read home address.</td>
<td></td>
</tr>
<tr>
<td>.... 1..</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>.... ..1.</td>
<td>Read multiple CKD or track.</td>
<td></td>
</tr>
<tr>
<td>.... ...1</td>
<td>At least one erase CCW.</td>
<td></td>
</tr>
<tr>
<td>2 (2)</td>
<td>1</td>
<td>Flag byte 2.</td>
</tr>
<tr>
<td>1... ....</td>
<td>End of file read or written.</td>
<td></td>
</tr>
<tr>
<td>..1. ....</td>
<td>At least one format write.</td>
<td></td>
</tr>
<tr>
<td>..1. ....</td>
<td>This record is for an FCX (zHPF) channel program</td>
<td></td>
</tr>
<tr>
<td>3 (3)</td>
<td>1</td>
<td>Count of erase, read MCKD.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>Number of blocks read.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>2</td>
<td>Number of blocks written.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>4</td>
<td>Number of bytes read.</td>
</tr>
<tr>
<td>12 (C)</td>
<td>4</td>
<td>Number of bytes written.</td>
</tr>
<tr>
<td>16 (10)</td>
<td>2</td>
<td>Number of data chain CCWs.</td>
</tr>
<tr>
<td>18 (12)</td>
<td>2</td>
<td>Number of COM chain CCWs.</td>
</tr>
<tr>
<td>20 (14)</td>
<td>1</td>
<td>External global attribute.</td>
</tr>
<tr>
<td>11... ....</td>
<td>'11' (only allowed value).</td>
<td></td>
</tr>
<tr>
<td>..1. ....</td>
<td>CKD conversion mode.</td>
<td></td>
</tr>
<tr>
<td>...1 ....</td>
<td>Subsystem operation mode.</td>
<td></td>
</tr>
<tr>
<td>..... 1..</td>
<td>Cache fast write.</td>
<td></td>
</tr>
<tr>
<td>..... ...1</td>
<td>Inhibit DASD fast write.</td>
<td></td>
</tr>
<tr>
<td>21 (15)</td>
<td>1</td>
<td>External global attribute extended.</td>
</tr>
<tr>
<td>22 (16)</td>
<td>4</td>
<td>External end of extent CCH.</td>
</tr>
<tr>
<td>26 (1A)</td>
<td>1</td>
<td>Seek/locate code.</td>
</tr>
<tr>
<td>27 (1B)</td>
<td>5</td>
<td>CCHHR (search ID equal).</td>
</tr>
<tr>
<td>32 (20)</td>
<td>1</td>
<td>Operation code from locate parameter.</td>
</tr>
</tbody>
</table>
Generalized Trace Facility

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 (21)</td>
<td>1</td>
<td>Sector number from parameter.</td>
</tr>
<tr>
<td>34 (22)</td>
<td>1</td>
<td>Extended operation code.</td>
</tr>
<tr>
<td>35 (23)</td>
<td>2</td>
<td>Extended parameters.</td>
</tr>
</tbody>
</table>

**CCW error codes:**

Table 46. CCW error codes

<table>
<thead>
<tr>
<th>Error code (Hex)</th>
<th>Error Code Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>IONOCCW</td>
<td>Bad CCW</td>
</tr>
<tr>
<td>04</td>
<td>IOINSPAC</td>
<td>Insufficient space in table</td>
</tr>
<tr>
<td>05</td>
<td>IOX4K</td>
<td>Target crosses 4K boundary</td>
</tr>
<tr>
<td>06</td>
<td>IOLASTCC</td>
<td>Last CCW not trace</td>
</tr>
<tr>
<td>07</td>
<td>IOSRCHCD</td>
<td>Search CCW with data chain</td>
</tr>
<tr>
<td>08</td>
<td>IOINSCNT</td>
<td>Insufficient count</td>
</tr>
<tr>
<td>09</td>
<td>IOCPUNAV</td>
<td>Channel program unavailable</td>
</tr>
<tr>
<td>0A</td>
<td>IORCKD8</td>
<td>Read CKD with count &lt;8</td>
</tr>
<tr>
<td>0B</td>
<td>IONOIDAW</td>
<td>No IDAW where needed</td>
</tr>
<tr>
<td>0C</td>
<td>IOCSWINV</td>
<td>Invalid CSW</td>
</tr>
<tr>
<td>0D</td>
<td>IOTICBAD</td>
<td>Invalid TIC address</td>
</tr>
<tr>
<td>0E</td>
<td>IOEBIDA</td>
<td>IDAL across page bound</td>
</tr>
<tr>
<td>0F</td>
<td>IOEBIDAL</td>
<td>IDAL not aligned on correct boundary</td>
</tr>
<tr>
<td>10</td>
<td>IOMIDNOQ</td>
<td>MIDAW not aligned on correct boundary</td>
</tr>
<tr>
<td>11</td>
<td>IOMIDZCT</td>
<td>MIDAW contains a zero count</td>
</tr>
<tr>
<td>12</td>
<td>IOMIDPBD</td>
<td>MIDAW crosses a page boundary (ignored if the skip bit is on)</td>
</tr>
<tr>
<td>13</td>
<td>IOMIDIDA</td>
<td>Both the IDAW and MIDAW bits are on</td>
</tr>
<tr>
<td>14</td>
<td>IONOTCW</td>
<td>Failure accessing TCW</td>
</tr>
<tr>
<td>15</td>
<td>IONOTCAH</td>
<td>Failure accessing TCA header</td>
</tr>
<tr>
<td>16</td>
<td>IONOTSB</td>
<td>Failure accessing TSB</td>
</tr>
<tr>
<td>17</td>
<td>IONOTIDA</td>
<td>Failure accessing TIDAW</td>
</tr>
<tr>
<td>18</td>
<td>IONODCW</td>
<td>Failure accessing DCW</td>
</tr>
<tr>
<td>19</td>
<td>IOINCMDL</td>
<td>TCW command length in error</td>
</tr>
<tr>
<td>1A</td>
<td>IOCmdParmInvalid</td>
<td>Command dependent parameters contain invalid information</td>
</tr>
</tbody>
</table>

**I/O trace record**

GTF builds an I/O record when an I/O interruption occurs and TRACE=SYSM, TRACE=SYS, TRACE=IO, or TRACE=IOP are the GTF options in effect. To trace PCI I/O interruptions, TRACE=PCI must also be in effect.

The FID for the I/O trace record is X'07'. The EID is one of the following:

- X'2100' - indicates a PCI I/O interruption.
- X'5101' - indicates an EOS I/O interruption.
- X'5200' - indicates an I/O interruption with a valid UCB.
**Generalized Trace Facility**

- X'5201' - indicates a IOCS I/O interruption. It indicates an I/O interruption that also contains concurrent sense information, for devices that support the concurrent sense facility.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>8</td>
<td>Jobname.</td>
</tr>
<tr>
<td>14 (E)</td>
<td>2</td>
<td>Device number.</td>
</tr>
<tr>
<td>16 (10)</td>
<td>16</td>
<td>I/O old PSW.</td>
</tr>
<tr>
<td>32 (20)</td>
<td>20</td>
<td>IRB words 0-4.</td>
</tr>
<tr>
<td>52 (34)</td>
<td>4</td>
<td>TCB.</td>
</tr>
<tr>
<td>56 (38)</td>
<td>2</td>
<td>Sense bytes.</td>
</tr>
<tr>
<td>58 (3A)</td>
<td>1</td>
<td>IOSB Flag (IOSFLA).</td>
</tr>
<tr>
<td>59 (3B)</td>
<td>1</td>
<td>IOSB Option (IOSOPT).</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>1</td>
<td>IOS Driver ID.</td>
</tr>
<tr>
<td>61 (3D)</td>
<td>1</td>
<td>IOS level.</td>
</tr>
<tr>
<td>62 (3E)</td>
<td>1</td>
<td>UCB level.</td>
</tr>
<tr>
<td>63 (3F)</td>
<td>1</td>
<td>Flags (UCBGWT).</td>
</tr>
<tr>
<td>64 (40)</td>
<td>2</td>
<td>Base device number.</td>
</tr>
<tr>
<td>66 (42)</td>
<td>44</td>
<td>IRB words 5-15 (for EID X'5201' only).</td>
</tr>
</tbody>
</table>

**PCIDMX trace record**
For a complete mapping of the PCIE de-multiplexing trace record, see the AHLPCIE (PCIE trace record formats) data area in z/OS MVS Data Areas in [z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/).

**PCILG trace record**
For a complete mapping of the PCILG trace record, see the AHLPCIE (PCIE trace record formats) data area in z/OS MVS Data Areas in [z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/).

**PCISTG trace record**
For a complete mapping of the PCISTG trace record, see the AHLPCIE (PCIE trace record formats) data area in z/OS MVS Data Areas in [z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/).

**PI comprehensive trace record**
GTF builds a PI comprehensive record when a program interruption occurs and either TRACE=PI or TRACE=SYS are the GTF options in effect.

The FID for the PI comprehensive trace record is X'00'. The EID is one of the following:
- X'6101' - indicates a program interruption with codes 1-17, 19, and 128.
- X'6200' - indicates a program interruption with code 18.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
</tbody>
</table>
## Generalized Trace Facility

### PI minimal trace record

GTF builds a PI minimal record when a program interruption occurs and TRACE=SYSM is the GTF option in effect.

The FID for the PI minimal trace record is X'03'. The EID is one of the following:
- X'6101' - indicates a program interruption with codes 1-17, 19, and 128.
- X'6200' - indicates a program interruption with code 18.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (6)</td>
<td>8</td>
<td>Jobname.</td>
</tr>
<tr>
<td>14 (E)</td>
<td>16</td>
<td>Program old PSW.</td>
</tr>
<tr>
<td>30 (1E)</td>
<td>4</td>
<td>INT TCB.</td>
</tr>
<tr>
<td>34 (22)</td>
<td>4</td>
<td>Virtual page address low half.</td>
</tr>
<tr>
<td>38 (26)</td>
<td>8</td>
<td>RB or CDE name.</td>
</tr>
<tr>
<td>46 (2E)</td>
<td>64</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>110 (6E)</td>
<td>4</td>
<td>Virtual page address high half.</td>
</tr>
</tbody>
</table>

### RR comprehensive trace record

GTF builds an RR comprehensive record when a recovery routine is invoked and TRACE=SYS or TRACE=RR are the GTF options in effect.

The FID for the RR comprehensive trace record is X'04'. The EID is one of the following:
- X'4002' - indicates STAE/ESTAE invocation.
- X'4003' - indicates FRR invocation.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>16</td>
<td>Program old PSW.</td>
</tr>
<tr>
<td>22 (16)</td>
<td>4</td>
<td>Old TCB.</td>
</tr>
<tr>
<td>26 (1A)</td>
<td>4</td>
<td>Virtual page address low half.</td>
</tr>
<tr>
<td>30 (1E)</td>
<td>4</td>
<td>Register 15.</td>
</tr>
<tr>
<td>34 (22)</td>
<td>4</td>
<td>Register 1.</td>
</tr>
<tr>
<td>38 (26)</td>
<td>4</td>
<td>Virtual page address high half.</td>
</tr>
</tbody>
</table>

---

Chapter 10. The Generalized Trace Facility (GTF) 333
RR minimal trace record
GTF builds an RR minimal record when a recovery routine is invoked and
TRACE=SYS is the GTF option in effect.

The FID for the RR minimal trace record is X'03'. The EID is one of the following:
• X'4002' - indicates STAE/ESTAE invocation.
• X'4003' - indicates FRR invocation.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID</td>
</tr>
<tr>
<td>6 (6)</td>
<td>16</td>
<td>Error PSW.</td>
</tr>
<tr>
<td>22 (16)</td>
<td>4</td>
<td>Completion code.</td>
</tr>
<tr>
<td>26 (1A)</td>
<td>4</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>30 (1E)</td>
<td>3</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>33 (21)</td>
<td>1</td>
<td>Return code from recovery routine (STAE/ESTAE only).</td>
</tr>
<tr>
<td>34 (22)</td>
<td>4</td>
<td>Retry address or N/A. <strong>Note:</strong> If no return code at offset 33, begin retry address at offset 33.</td>
</tr>
<tr>
<td>38 (26)</td>
<td>4</td>
<td>Address of SDWA (STAE/ESTAE only). <strong>Note:</strong> If retry address begins at offset 33, SDWA address begins at offset 37.</td>
</tr>
</tbody>
</table>

SLIP trace records
GTF builds a SLIP trace record when TRACE=SLIP is the GTF option in effect and:
• A SLIP trap has matched and either TRACE or TRDUMP has been specified on the SLIP command.
• A SLIP trap is in DEBUG mode (specified on the SLIP command) and is inspected by the SLIP processor as a result of any SLIP event.

The SLIP trace records are:
• SLIP Standard Trace Record
• SLIP Standard/User Trace Record
• SLIP User Trace Record
• SLIP DEBUG Trace Record

SLIP standard trace record: The FID for the SLIP standard trace record is X'04'. The EID is X'4004'.

A field will contain asterisks if an error occurred when attempting to obtain data or the data is unavailable because it is paged out.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB address.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID. <strong>Note:</strong> When SLIP is entered from RTM2, the CPUID recorded may be different from the CPUID when RTM2 was running.)</td>
</tr>
</tbody>
</table>
### Generalized Trace Facility

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (6)</td>
<td>8</td>
<td>Jobname from current address space (or N/A).</td>
</tr>
<tr>
<td>14 (E)</td>
<td>4</td>
<td>SLIP trap ID.</td>
</tr>
<tr>
<td>18 (12)</td>
<td>2</td>
<td>ASID of current address space.</td>
</tr>
<tr>
<td>20 (14)</td>
<td>8</td>
<td>Job step program name (or U/A or N/A).</td>
</tr>
<tr>
<td>28 (1C)</td>
<td>4</td>
<td>TCB address (or N/A).</td>
</tr>
<tr>
<td>32 (20)</td>
<td>1</td>
<td>System mode indicators, byte 1:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1... .... Supervisor control mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1... .... Disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>..1 ... Global spin lock held.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...1... Global suspend lock held.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... 1... Local lock held.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... 1 Type 1 SVC in control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... ..1 SRB mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... .... TCB mode.</td>
</tr>
<tr>
<td>33 (21)</td>
<td>1</td>
<td>System mode indicators, byte 2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1... .... Recovery routine in control (always zero if a PER interrupt).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1... .... Problem program state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>..1 ... Supervisor state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...1 ... System key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... 1... Problem program key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... 1... Any global lock held.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... ...1 Any lock held.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>1</td>
<td>Error byte 1 (or zeros if a PER interrupt):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1... .... Program check interrupt.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1... .... Restart interrupt.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>..1 ... SVC error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...1 ... Abend; task issued SVC 13.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... 1... Paging I/O error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... ..1 Dynamic address translation error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... ...1 Software error caused by machine check.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... .... Abnormal address space termination.</td>
</tr>
<tr>
<td>35 (23)</td>
<td>1</td>
<td>Error byte 2 (or zeros if a PER interrupt):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1... .... Memterm.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>1</td>
<td>SLIP flags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1... .... DEBUG record.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1... .... Registers collected.</td>
</tr>
<tr>
<td>37 (25)</td>
<td>2</td>
<td>Data unavailable counter (or zeros if DATA was not specified for the trap).</td>
</tr>
</tbody>
</table>

The following fields apply only to PER interrupts otherwise set to N/A (or N for one-byte fields).
### Generalized Trace Facility

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39 (27)</td>
<td>8</td>
<td>Load module name in which the interrupt occurred (or U/A or N/A).</td>
</tr>
<tr>
<td>47 (2F)</td>
<td>4</td>
<td>Offset in load module (or U/A or N/A).</td>
</tr>
<tr>
<td>51 (33)</td>
<td>8</td>
<td>Address of the instruction that caused the PER interrupt.</td>
</tr>
<tr>
<td>59 (3D)</td>
<td>6</td>
<td>Instruction content (six bytes of data beginning at the address of the instruction that caused the PER interrupt).</td>
</tr>
<tr>
<td>65 (41)</td>
<td>8</td>
<td>Target instruction address if EXECUTE instruction (or N/A or U/A).</td>
</tr>
<tr>
<td>73 (49)</td>
<td>6</td>
<td>Target instruction content if EXECUTE instruction (six bytes of data beginning at the target instruction address), or (N/A or U/A).</td>
</tr>
<tr>
<td>79 (4F)</td>
<td>4</td>
<td>Beginning range virtual address if SA (storage-alteration) specified on SLIP command (or N/A).</td>
</tr>
<tr>
<td>83 (53)</td>
<td>4</td>
<td>Four bytes of storage starting at beginning range virtual address if SA specified (or N/A or U/A).</td>
</tr>
<tr>
<td>87 (57)</td>
<td>16</td>
<td>Program old PSW.</td>
</tr>
<tr>
<td>103 (67)</td>
<td>4</td>
<td>Program interrupt code (PIC) and instruction length code.</td>
</tr>
<tr>
<td>107 (6B)</td>
<td>1</td>
<td>PER interrupt code:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1..    Successful-branch event (SB).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1..    Instruction-fetch event (IF).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>..1.    Storage-alteration event (SA).</td>
</tr>
<tr>
<td>108 (6C)</td>
<td>1</td>
<td>PER trap mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1..    Successful-branch monitoring (SB).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1..    Instruction-fetch monitoring (IF).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>..1.    Storage-alteration monitoring (SA).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...x    Reserved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... 1.. PER trap.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... .1.. Recovery specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... ..1. Message flag.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.... ...1 Message flag.</td>
</tr>
<tr>
<td>109 (6D)</td>
<td>2</td>
<td>Key mask.</td>
</tr>
<tr>
<td>111 (6F)</td>
<td>2</td>
<td>SASID.</td>
</tr>
<tr>
<td>113 (71)</td>
<td>2</td>
<td>Authorization index.</td>
</tr>
<tr>
<td>115 (73)</td>
<td>2</td>
<td>PASID.</td>
</tr>
<tr>
<td>117 (75)</td>
<td>1</td>
<td>PSW ASC mode indicator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• F0: primary addressing mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• F1: access register addressing mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• F2: secondary addressing mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• F3 home addressing mode</td>
</tr>
<tr>
<td>118 (76)</td>
<td>13</td>
<td>Storage Alteration Space Identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For an address space: contains the ASID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For a data space: contains the owning ASID and the dataspace name</td>
</tr>
<tr>
<td>131 (83)</td>
<td>4</td>
<td>High-half of begin range.</td>
</tr>
</tbody>
</table>
**Generalized Trace Facility**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>135 (87)</td>
<td>1</td>
<td>Transactional execution DATA filter mismatch count</td>
</tr>
</tbody>
</table>

**SLIP standard + user trace record:** The FID for the SLIP Standard + User trace record is X'04'. The EID is X'4005'.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>136</td>
<td>Fields are identical to the SLIP standard record.</td>
</tr>
<tr>
<td>136 (88)</td>
<td>2</td>
<td>Length of user-defined data.</td>
</tr>
<tr>
<td>138 (8A)</td>
<td>variable</td>
<td>User-defined data (specified through the TRDATA parameter on the SLIP command).</td>
</tr>
</tbody>
</table>

**SLIP user trace record:** The FID for the SLIP user trace record is X'04'. The EID is X'4006'.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>2 (2)</td>
<td>2</td>
<td>Extension number.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>1</td>
<td>Continuation length.</td>
</tr>
<tr>
<td>5 (5)</td>
<td>2</td>
<td>Length of the user defined data.</td>
</tr>
<tr>
<td>7 (7)</td>
<td>variable</td>
<td>User-defined data (specified through the TRDATA parameter on the SLIP command).</td>
</tr>
</tbody>
</table>

**Note:**

1. If the SLIP user requests registers to be placed in the SLIP user record, they are the first field in the record.
2. A length field of zero indicates that the user-defined data was not available (for example, the data is paged out).
3. The TRDATA parameter on the SLIP command specifies one or more data ranges. The number of records needed to trace these ranges depends on the size of the ranges specified. The trace contains a standard plus (+) user record from the next range or a user record followed by as many user records and user continuation records as needed to trace the ranges specified. The header for each record contains the CPUID and extension number to help correlate the output (extension numbers apply only to user and user continuation records).

When a record is partially filled and the data from the next range will not fit in the remaining space; the partially filled record is written to the trace data set. Another user record is built, the extension number is increased by one, and the continuation length is set to zero. The data length and data is then copied into this record.

When the length of the data from a range is greater than 249 bytes, the excess data is put in user continuation records in the following manner. The data length and first 248 bytes are put in a user record. After writing that record a user continuation record is built. The extension number is increased by one and the continuation length is set to the number of bytes of data to be put in this record. If more than 251 bytes of data are left, 248 bytes are copied into record, and it is written. User continuation records are built until all the data in from that range is traced.
### SLIP DEBUG trace record
The FID for the SLIP DEBUG trace record is X'04'. The EID is X'4005'.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>136</td>
<td>Fields are identical to the SLIP standard record.</td>
</tr>
<tr>
<td>through 135</td>
<td></td>
<td>DEBUG byte; indication of which keyword failed:</td>
</tr>
<tr>
<td>136 (88)</td>
<td>1</td>
<td>- Decimal 2 indicates COMP keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 3 indicates ASID keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 4 indicates JOBNAME keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 5 indicates JSPGM keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 6 indicates PVTMOD keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 7 indicates LPAMOD keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 8 indicates ADDRESS keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 9 indicates MODE keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 10 indicates ERRTYP keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 13 indicates RANGE keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 14 indicates DATA keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 20 indicates ASIDSA keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 22 indicates REASON CODE keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 23 indicates NUCMOD keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 24 indicates PSWASC keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Decimal 26 indicates DSSA keyword</td>
</tr>
<tr>
<td>137 (89)</td>
<td>1</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**Note:** The high-order bit in the SLIP flags (SFLG) field (at offset X'34') is set on to indicate a DEBUG record.

### SRM comprehensive trace record
GTF builds an SRM comprehensive record when system resource manager is invoked and TRACE=SYS or TRACE=SRM are the trace options in effect.

The FID for the SRM trace record is X'04'. The EID is X'4001'.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>8</td>
<td>Jobname</td>
</tr>
<tr>
<td>14 (E)</td>
<td>4</td>
<td>Register 15.</td>
</tr>
<tr>
<td>18 (12)</td>
<td>4</td>
<td>Register 0.</td>
</tr>
<tr>
<td>22 (16)</td>
<td>4</td>
<td>Register 1.</td>
</tr>
</tbody>
</table>

### SRM minimal trace record
GTF builds an SRM minimal record when system resource manager is invoked and TRACE=SYSM is the GTF option in effect.

The FID for the SRM minimal trace record is X'03'. The EID is X'4001'.
SSCH trace record
GTF builds an SSCH record when an SSCH event occurs and TRACE=SYSM, TRACE=SYS, TRACE=SYSP, TRACE=SSCH or TRACE=SSCHP are the GTF options in effect.

The FID for the SSCH trace record is X'00'. The EID is X'5105'.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>4</td>
<td>Register 15.</td>
</tr>
<tr>
<td>10 (A)</td>
<td>4</td>
<td>Register 0.</td>
</tr>
<tr>
<td>14 (E)</td>
<td>4</td>
<td>Register 1.</td>
</tr>
</tbody>
</table>

SVC comprehensive trace records
GTF builds SVC comprehensive records when an SVC interruption occurs and either the TRACE=SYS or TRACE=SVC GTF option is in effect. All SVC records contain the basic data described below; however, many SVC numbers invoke additional data recording, described following the basic data.

The FID for the SVC comprehensive trace record is X'010'. The EID is X'1000'.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>8</td>
<td>Jobname.</td>
</tr>
<tr>
<td>14 (E)</td>
<td>2</td>
<td>Device number.</td>
</tr>
<tr>
<td>16 (10)</td>
<td>4</td>
<td>Real address of channel program.</td>
</tr>
<tr>
<td>20 (14)</td>
<td>4</td>
<td>Virtual address of channel program.</td>
</tr>
<tr>
<td>24 (18)</td>
<td>4</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>28 (1C)</td>
<td>1</td>
<td>Condition code.</td>
</tr>
<tr>
<td>29 (1D)</td>
<td>12</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>41 (29)</td>
<td>8</td>
<td>Dynamic seek address.</td>
</tr>
<tr>
<td>49 (31)</td>
<td>1</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>50 (32)</td>
<td>1</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>51 (33)</td>
<td>1</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>52 (34)</td>
<td>1</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>53 (35)</td>
<td>1</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>54 (36)</td>
<td>1</td>
<td>Reserved for IBM use.</td>
</tr>
</tbody>
</table>
Table 47. Basic SVC comprehensive trace record (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 (E)</td>
<td>16</td>
<td>SVC old PSW. The seventh and eighth bytes contain the SVC number.</td>
</tr>
<tr>
<td>30 (1E)</td>
<td>4</td>
<td>Old TCB.</td>
</tr>
<tr>
<td>34 (22)</td>
<td>8</td>
<td>CDE name.</td>
</tr>
<tr>
<td>42 (2A)</td>
<td>4</td>
<td>Register 15.</td>
</tr>
<tr>
<td>46 (2E)</td>
<td>4</td>
<td>Register 0.</td>
</tr>
<tr>
<td>50 (32)</td>
<td>4</td>
<td>Register 1.</td>
</tr>
</tbody>
</table>

GTF builds only a basic comprehensive trace record for the following SVCs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Name</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>3</td>
<td>TEST</td>
<td>97</td>
</tr>
<tr>
<td>GETMAIN/FREEMAIN</td>
<td>10</td>
<td>SUBMIT</td>
<td>100</td>
</tr>
<tr>
<td>TIME</td>
<td>11</td>
<td>QTIP</td>
<td>101</td>
</tr>
<tr>
<td>SYNCH</td>
<td>12</td>
<td>XSTATE</td>
<td>103</td>
</tr>
<tr>
<td>MGCR</td>
<td>34</td>
<td>TOPCTL</td>
<td>104</td>
</tr>
<tr>
<td>WTL</td>
<td>36</td>
<td>IMBLIB</td>
<td>105</td>
</tr>
<tr>
<td>TSTIO</td>
<td>38</td>
<td>REQUEST</td>
<td>106</td>
</tr>
<tr>
<td>CIRB</td>
<td>43</td>
<td>MODESET</td>
<td>107</td>
</tr>
<tr>
<td>TTOUTER</td>
<td>46</td>
<td>None</td>
<td>109</td>
</tr>
<tr>
<td>TSTOP</td>
<td>49</td>
<td>DSTATUS</td>
<td>110</td>
</tr>
<tr>
<td>NOP</td>
<td>50</td>
<td>JECS</td>
<td>111</td>
</tr>
<tr>
<td>OLET</td>
<td>59</td>
<td>RELEASE</td>
<td>112</td>
</tr>
<tr>
<td>TSTOP</td>
<td>61</td>
<td>JX</td>
<td>113</td>
</tr>
<tr>
<td>CHATR</td>
<td>72</td>
<td>BLKPAGE</td>
<td>115</td>
</tr>
<tr>
<td>(IFBSTAT)</td>
<td>76</td>
<td>None</td>
<td>116</td>
</tr>
<tr>
<td>STATUS</td>
<td>79</td>
<td>None</td>
<td>117</td>
</tr>
<tr>
<td>SMFWTM</td>
<td>83</td>
<td>DSSPATCH</td>
<td>118</td>
</tr>
<tr>
<td>(IGC084)</td>
<td>84</td>
<td>TESTAUTH</td>
<td>119</td>
</tr>
<tr>
<td>SWAP</td>
<td>85</td>
<td>GETMAIN/FREEMAIN</td>
<td>120</td>
</tr>
<tr>
<td>EMSERV</td>
<td>89</td>
<td>None</td>
<td>121</td>
</tr>
<tr>
<td>VOLSTAT</td>
<td>91</td>
<td>LINK, LOAD, XCTL</td>
<td>122</td>
</tr>
<tr>
<td>TPUT/TGET</td>
<td>93</td>
<td>PURGEDQ</td>
<td>123</td>
</tr>
<tr>
<td>TSO terminal control</td>
<td>94</td>
<td>TPIO</td>
<td>124</td>
</tr>
<tr>
<td>SYSEVENT</td>
<td>95</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Basic SVC comprehensive trace record with parameter list information: For detailed information about data gathered for the following SVCs, see z/OS MVS Diagnosis: Reference.
## Generalized Trace Facility

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Name</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCP</td>
<td>0</td>
<td>STIMER</td>
<td>47</td>
</tr>
<tr>
<td>WAIT/WAITR</td>
<td>1</td>
<td>DEQ</td>
<td>48</td>
</tr>
<tr>
<td>POST</td>
<td>2</td>
<td>SNAP</td>
<td>51</td>
</tr>
<tr>
<td>GETMAIN</td>
<td>4</td>
<td>RESTART</td>
<td>52</td>
</tr>
<tr>
<td>FREEMAIN</td>
<td>5</td>
<td>RELEX</td>
<td>53</td>
</tr>
<tr>
<td>LINK</td>
<td>6</td>
<td>DISABLE</td>
<td>54</td>
</tr>
<tr>
<td>XCTL</td>
<td>7</td>
<td>EOV</td>
<td>55</td>
</tr>
<tr>
<td>LOAD</td>
<td>8</td>
<td>ENQ/RESERVE</td>
<td>56</td>
</tr>
<tr>
<td>DELETE</td>
<td>9</td>
<td>FREEDBUF</td>
<td>57</td>
</tr>
<tr>
<td>ABEND</td>
<td>13</td>
<td>RELBUF/REQBUF</td>
<td>58</td>
</tr>
<tr>
<td>SPIE</td>
<td>14</td>
<td>STAE</td>
<td>60</td>
</tr>
<tr>
<td>ERREXCP</td>
<td>15</td>
<td>DETACH</td>
<td>62</td>
</tr>
<tr>
<td>PURGE</td>
<td>16</td>
<td>CHKPT</td>
<td>63</td>
</tr>
<tr>
<td>RESTORE</td>
<td>17</td>
<td>RDJFCB</td>
<td>64</td>
</tr>
<tr>
<td>BLDL/FIND</td>
<td>18</td>
<td>BTAMTEST</td>
<td>66</td>
</tr>
<tr>
<td>OPEN</td>
<td>19</td>
<td>BSP</td>
<td>69</td>
</tr>
<tr>
<td>CLOSE</td>
<td>20</td>
<td>GSERV</td>
<td>70</td>
</tr>
<tr>
<td>STOW</td>
<td>21</td>
<td>ASGNBFR/BUFQNQ/RLSEBFR</td>
<td>71</td>
</tr>
<tr>
<td>OPEN TYPE = J</td>
<td>22</td>
<td>SPAR</td>
<td>73</td>
</tr>
<tr>
<td>CLOSE TYPE = T</td>
<td>23</td>
<td>DAR</td>
<td>74</td>
</tr>
<tr>
<td>DEVTYPE</td>
<td>24</td>
<td>DQUEUE</td>
<td>75</td>
</tr>
<tr>
<td>TRKBAL</td>
<td>25</td>
<td>LSPACE</td>
<td>78</td>
</tr>
<tr>
<td>CATLG</td>
<td>26</td>
<td>GJP</td>
<td>80</td>
</tr>
<tr>
<td>OBTAIN</td>
<td>27</td>
<td>SETPRT</td>
<td>81</td>
</tr>
<tr>
<td>SCRATCH</td>
<td>29</td>
<td>ATLAS</td>
<td>86</td>
</tr>
<tr>
<td>RENAME</td>
<td>30</td>
<td>DOM</td>
<td>87</td>
</tr>
<tr>
<td>FEOB</td>
<td>31</td>
<td>MOD88</td>
<td>88</td>
</tr>
<tr>
<td>ALLOC</td>
<td>32</td>
<td>TCBEXCP</td>
<td>92</td>
</tr>
<tr>
<td>WTO/WROR</td>
<td>35</td>
<td>PROTECT</td>
<td>98</td>
</tr>
<tr>
<td>SEGLD/SEGWT</td>
<td>37</td>
<td>Dynamic allocation</td>
<td>99</td>
</tr>
<tr>
<td>LABEL</td>
<td>39</td>
<td>EXCPVR</td>
<td>114</td>
</tr>
<tr>
<td>EXTRACT</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDENTIFY</td>
<td>41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTACH</td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAP</td>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVLYBRCH</td>
<td>45</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SVC minimal trace record

GTF builds an SVC minimal record when an SVC interruption occurs and TRACE=SYSM is the GTF option in effect. The FID for the SVC minimal trace record is X'010'. The EID is X'1000'.
Generalized Trace Facility

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>4</td>
<td>ASCB.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>2</td>
<td>CPUID.</td>
</tr>
<tr>
<td>6 (6)</td>
<td>16</td>
<td>SVC old PSW. The seventh and eighth bytes contain the SVC number.</td>
</tr>
<tr>
<td>22 (16)</td>
<td>4</td>
<td>Old TCB.</td>
</tr>
<tr>
<td>26 (1A)</td>
<td>4</td>
<td>Register 15.</td>
</tr>
<tr>
<td>30 (1E)</td>
<td>4</td>
<td>Register 0.</td>
</tr>
<tr>
<td>34 (22)</td>
<td>4</td>
<td>Register 1.</td>
</tr>
</tbody>
</table>

**TCW trace record**

For a complete mapping of the AHLFCXG (FCX/zHPF channel program) data area, see *z/OS MVS Data Areas* in [z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/).
Chapter 11. The generic tracker facility

The intended purpose of generic tracker is to be a migration aid and to help assess exploitation of old and new function. For example, you can assess use of interfaces you intend to make obsolete, and assess exploitation of old and new function in general. For functions that are already obsolete in a new release, you can use generic tracker to assess their use on previous releases.

The generic tracker facility is a tracking facility that provides these services.

• A callable tracking service that users can instrument code with:
  – The caller passes an event address and other related information to the service.
  – The service attempts to resolve the event address to a program name and store it with the other information for later analysis.
• Provides a callable query service that extracts previously stored track records and tracking facility information.
• Provides operator commands to display and maintain tracking facility information and configuration details.
• Supports parmlib members for easy reuse of tracking facility configuration statements.
• Provides more tools to aid in the use of the tracking facility and the information it stores.

The generic tracker Facility replaces the Console Tracking Facility, which was available in releases before z/OS V2R1. The Console Tracking Facility operator commands are available only in releases before z/OS V2R1. Macro CNZTRKR continues to be supported, but the recommendation is to use macro GTZTRACK instead. Any information CNZTRKR collects is stored in the new tracking facility. Do not continue to use service CNZTRKR. When possible, replace existing CNZTRKR invocations with GTZTRACK invocations.

The generic tracker consists of the following parts:

• A system parameter named GTZ - see z/OS MVS Initialization and Tuning Guide.
• Support for GTZPRMxx parmlib members. See z/OS MVS Initialization and Tuning Guide – Shipped with GTZPRM00, which includes an exclusion list.
• Callable macro services. See z/OS MVS Programming: Assembler Services Reference – GTZTRACK – track an event – Associated mapping macro GTZZTRK – GTZQUERY – report tracking facility information – Associated mapping macros for Assembler (GTZZQRY) and METAL-C (GTZHQRY).
• REXX callable functions. See “GTZLQRY REXX callable function” on page 346.
- GTZLQRY – report tracking facility information

• Utility programs
  - GTZPRINT – report tracking facility information
    - Associated mapping macro GTZZPRT
  - GTZCNIDT – create GTZPRMxx from CNIDTRxx
    - Associated mapping macro GTZZCNI
  - GTZSMFU2 and GTZSMFU3 – retrieve GTZ data from standard SMF record
    Backend stores and format the data in text form

• Samples – see SYS1.SAMPLIB
  - GTZCNIDJ – start utility program GTZCNIDT
  - GTZPRNTJ – start utility program GTZPRINT
  - GTZSHCK – a sample health check to report tracked events
  - GTZSHCKJ – build health check GTZSHCK
  - GTZSMFJ – use of utility programs GTZSMFU2 and GTZSMFU3

The generic tracker is a system address space that starts automatically during the
IPL of the system. A restart is not required or recommended unless it is explicitly
requested, for example to apply service. The source of the IBM supplied started
procedure GTZ can be modified to configure the MEMLIMIT, which determines
how much information the tracking facility can store.

Actual tracking (that is, recording track events) is disabled by default. When
tracking is disabled, invocations of GTZTRACK are allowed, but ignored by the
system. Use the SETGTZ TRACKING=ON operator command to enable tracking.

Besides storing track data in the dynamic storage of the GTZ address space, you
can also enable the tracking facility to persist track data using SMF records. For
more information, see “GTZLQRY "TRACKDATA" interface” on page 351.

Before you enable tracking and allow the tracking facility to store data and report
it back to the users, consider protecting the DISPLAY GTZ and SETGTZ operator
commands. As mentioned in IBM VTAM User's Guide, these GTZ commands
can be protected by RACF resource profiles for the OPERCMDS class. The security
administrator can for example take the following steps to give a user profile access
to the MVS.SETGTZ.GTZ resource:

1. Ensure that the OPERCMDS class is active:
   SETROPTS CLASSACT(OPERCMDS) RACLIST(OPERCMDS)

2. Create the MVS.SETGTZ.GTZ resource profile and require explicit access to be
   given:
   RDEFINE OPERCMDS MVS.SETGTZ.GTZ UACC(NONE)

3. Grant individual access to the resource:
   PERMIT MVS.SETGTZ.GTZ CLASS(OPERCMDS) ID(user) ACCESS(UPDATE)

4. Refresh the OPERCMDS class to activate the changes:
   SETROPTS RACLIST(OPERCMDS) REFRESH

All SETGTZ operator commands can be used in GTZPRMxx parmlib members. For
example, you can put a TRACKING=ON statement into a GTZPRMxx parmlib
member.

System parameter GTZ can be set to one or more suffixes, which identify the
GTZPRMxx parmlib members that the tracking facility must process at startup. See
the description of system parameter GTZ in \textit{z/OS MVS Initialization and Tuning Reference}. It contains requirements and details on how to grant the tracking facility permission to access the parmlib members at start-up. If you do not specify system parameter GTZ, or specify PARM='GTZPRM=*NONE' in procedure GTZ, the tracking facility will start without reading any parmlib members initially.

You can add more GTZPRMxx parmlib members later with the operator command SET GTZ.

You can use the SETGTZ CLEAR operator command to remove configuration data previously added via GTZPRMxx members and to remove previously recorded track data.

To prevent certain known track events from recording any data, you can add EXCLUDE statements. The tracking facility will ignore any future track data that matches such EXCLUDE filters and it will remove any matching previously recorded track data. Typically EXCLUDE statements are provided in GTZPRMxx parmlib members and can be specified via the SETGTZ EXCLUDE operator command as well.

A standard list of EXCLUDE statements for known track events is shipped in parmlib member GTZPRM00. You can download it from the z/OS Downloads website. It is recommended that you use the GTZPRMxx parmlib member as is and enable it by default by adding the 00 suffix to the system parameter GTZ in IEASYSxx using GTZ=(00,your_suffixes). Keeping GTZPRM00 separate allows for easier updates when z/OS releases a new GTZPRM00 version. More GTZPRMxx parmlib members would then have any additional EXCLUDEs and for example a TRACKING=ON, if wanted.

If you used the Console Tracking Facility and its CNIDTRxx parmlib members for exclusion lists before, you can use the GTZCNIDT utility program to create GTZPRMxx parmlib members from existing CNIDTRxx parmlib members. The sample JCL GTZCNIDJ explains how to start GTZCNIDT.

A unique tracked event is identified by the following fields, which are referenced by the EXCLUDE filters. The tracking facility increments an associated occurrence count if a GTZTRACK call results in an identical set of unique fields:

- The program name and program offset as resolved from the passed in event address.
- The name of the owner and source of the track event.
- The event description text.
- The (binary) event data.
- Job names and ASIDs associated with the track event.
- The authorization state that is associated with the track event.

For events that are difficult to track, where the program name cannot be determined by the system from the passed in event address, you can define DEBUG filters using the SETGTZ DEBUG operator command or in a GTZPRMxx parmlib member. The DEBUG statement allows for example to trigger a non-percolating ABEND, code E77, with custom reason codes, that you can define SLIP traps for to collect dumps.

You can use the DISPLAY GTZ operator command to display:
• General status and statistics for the tracking facility – DISPLAY GTZ,STATUS.
• All or a subset of the track data that is currently stored in the tracking facility – DISPLAY GTZ,TRACKDATA[(filters)].
• All the EXCLUDE statements – DISPLAY GTZ,EXCLUDE.
• All the DEBUG statements – DISPLAY GTZ,DEBUG.

Using the utility program GTZPRINT, you can collect information in a data set instead of reading it on the console or in the system LOG. The sample JCL GTZPRNTJ explains how to use GTZPRINT.

Here are some typical scenarios
• Use GTZPRINT to preserve tracking information for later analysis. This helps when you must submit the information to IBM, or an independent software vendor (ISV) product to have them analyze the output or diagnose problems.
• Use DISPLAY GTZ for quick interactive access. Certain command line and output length limits might apply.
• Use callable macro service GTZQUERY when you want to access tracking facility information from a program. For example, for a custom reporting tool or for a health check for the IBM Health Checker for z/OS. See the sample health check GTZSHCK shipped in SAMPLIB. This can be built with the sample JCL GTZSHCKJ. While local health checks run authorized, other uses of GTZQUERY require the calling user ID to have access to the GTZ.sysname.QUERY RACF profile that is used similar to how the GTZ operator commands are protected.

**GTZLQRY REXX callable function**

REXX callable function GTZLQRY allows REXX programs access to data that is provided by the existing macro service GTZQUERY.

The GTZLQRY function uses corresponding request types that are provided by REXX (input) variable GTZLQRY_REQUEST to support all four published request types of the GTZQUERY macro service:

• GTZLQRY_REQUEST = "STATUS"
• GTZLQRY_REQUEST = "TRACKDATA"
• GTZLQRY_REQUEST = "EXCLUDE"
• GTZLQRY_REQUEST = "DEBUG"

The requested data is provided as REXX STEM output variables.

Like the GTZQUERY macro service, the TRACKDATA request supports additional filters that allow you to limit the resulting data to only a subset of the potentially very many available track data entries.

For TRACKDATA entries, EXCLUDE statements, and DEBUG statements, which are the three request types that return lists of items, GTZLQRY supports additional OPTION input parameters to further limit the result sets.

**GTZLQRY "STATUS" interface**

Accepts input using a single variable. The following required statement requests to return status information:

GTZLQRY_REQUEST="STATUS"
GTZLQRY with REQUEST="STATUS" returns output with STEM variable GTZQUAAS. Diagnostic information such as the return code is provided with separate variables.

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GtzQuaaS.TrackEnabled</td>
<td>1, if tracking is currently enabled, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.Full</td>
<td>1, if the tracking facility is full, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.ExcludeNoPrm</td>
<td>1, if some EXCLUDE statements did not originate from GTZPRMxx, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.DebugNoPrm</td>
<td>1, if some DEBUG statements did not originate from GTZPRMxx, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.GtzPrmFull</td>
<td>1, if some GTZPRMxx could not be recorded centrally, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.ClearedALL</td>
<td>1, if a CLEAR=ALL has been used at some time for this tracker instance, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.ClearedTRACKDATA</td>
<td>1, if a CLEAR=TRACKDATA or a CLEAR=ALL removed trackdata at some time for this tracker instance, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.ClearedEXCLUDE</td>
<td>1, if a CLEAR=EXCLUDE or a CLEAR=ALL removed EXCLUDE statements at some time for this tracker instance, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.ClearedDEBUG</td>
<td>1, if a CLEAR=DEBUG or a CLEAR=ALL removed DEBUG statements at some time for this tracker instance, otherwise &quot;0&quot;</td>
</tr>
<tr>
<td>GtzQuaaS.PersistSMF</td>
<td>ON ('1'b), if data persistence via SMF records is enabled.</td>
</tr>
<tr>
<td>GtzQuaaS.EnabledCount</td>
<td>Number of times the tracking facility moved from disabled to enabled. An unsigned number in the single-word (32-bit) range.</td>
</tr>
<tr>
<td>GtzQuaaS.TrackDataEntriesAvailable</td>
<td>Total number of unique tracked instances currently known to the tracking facility. An unsigned number in the double-word (64-bit) range.</td>
</tr>
<tr>
<td>GtzQuaaS.ExcludeEntriesAvailable</td>
<td>Total number of exclusion statements currently known to the tracking facility. An unsigned number in the double-word (64-bit) range.</td>
</tr>
<tr>
<td>GtzQuaaS.DebugEntriesAvailable</td>
<td>Total number of DEBUG statements currently known to the tracking facility. An unsigned number in the double-word (64-bit) range.</td>
</tr>
<tr>
<td>GtzQuaaS.TrackDataEntriesEncountered</td>
<td>Total number of non-unique tracked instances currently known to the tracking facility. An unsigned number in the double-word (64-bit) range.</td>
</tr>
<tr>
<td>GtzQuaaS.ExcludeRejectCount</td>
<td>Total number of GTZTRACK requests rejected due to a matching EXCLUDE statement. This counter is reset when the EXCLUDE statements are cleared. An unsigned number in the double-word (64-bit) range.</td>
</tr>
</tbody>
</table>
Table 48. REQUEST="STATUS" output variable names and descriptions (continued)

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GtzQuaaS.DebugActionCount</td>
<td>Total number of GTZTRACK requests which triggered a DEBUG action as specified by a matching DEBUG statement (with its LIMIT not exceeded yet). This counter is reset when the DEBUG statements are cleared. An unsigned number in the double-word (64-bit) range.</td>
</tr>
<tr>
<td>GtzQuaaS.GtzPrmSuffixes.0</td>
<td>Number of GTZPRMxx members currently known to the tracking facility. An unsigned number in the half-word (16-bit) range.</td>
</tr>
<tr>
<td>GtzQuaaS.GtzPrmSuffixes</td>
<td>The i=1..GtzQuaaS.GtzPrmSuffixes.0 GTZPRMxx 2-character suffixes currently known to the tracking facility.</td>
</tr>
<tr>
<td>GtzQuaaS.GtzPrmIplSuffixes.0</td>
<td>Number of GTZPRMxx members specified at IPL time via system parameter GTZ. The currently known GTZPRMxx suffix list might be different than this IPL-time list, if suffixes have been added or cleared in between. An unsigned number in the half-word (16-bit) range.</td>
</tr>
<tr>
<td>GtzQuaaS.GtzPrmIplSuffixes</td>
<td>The i=1..GtzQuaaS.GtzPrmIplSuffixes.0 GTZPRMxx 2-character suffixes specified at IPL time.</td>
</tr>
<tr>
<td>GtzQuaaS.MemAvailPercent</td>
<td>A number between 0 and 100, indicating the percentage of how much of the tracking facility's total dynamic memory is still available to store data for tracking entries, EXCLUDE statements, etc. When GtzQuaaS.Full is &quot;1&quot; then GtzQuaaS.MemAvailPercent will be zero.</td>
</tr>
<tr>
<td>GtzQuaaS.SystemName</td>
<td>The name of the system the (unique per system) tracking facility is running on. An up to 8 character long string.</td>
</tr>
<tr>
<td>GtzQuaaS.EnabledTOD</td>
<td>The timestamp when tracking was last enabled (if GtzQuaaS.TrackEnabled=&quot;1&quot;), or when tracking was last disabled (if GtzQuaaSTrackEnabled = &quot;0&quot;), where for the latter this might be the time of when the facility started, if tracking has not been enabled since. See also GtzQuaaS.EnabledCount. This value is provided in STCK format. See for example BLSUXTOD to format this as human readable timestamp.</td>
</tr>
</tbody>
</table>
Table 48. REQUEST="STATUS" output variable names and descriptions (continued)

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTZLQRY_RC</td>
<td>Return code. Same as the REXX supplied output variable RESULT which REXX sets if the function result is not explicitly captured via for example RC = GTZLQRY(). This is a decimal text string so that for example RC=12 is reported as &quot;12&quot;, not &quot;0C&quot;. The possible return codes are as follows:</td>
</tr>
<tr>
<td></td>
<td>0 Meaning: The GTZLQRY request completed successfully.</td>
</tr>
<tr>
<td></td>
<td>4 Meaning: The GTZLQRY request completed successfully, but issued a warning.</td>
</tr>
<tr>
<td></td>
<td>Action: Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td>8 Meaning: The GTZLQRY request encountered an error and did not complete successfully.</td>
</tr>
<tr>
<td></td>
<td>Action: Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td>12 (X'C') Meaning: The GTZLQRY request encountered a severe error, typically related to the environment in which GTZLQRY was invoked, and did not complete successfully.</td>
</tr>
<tr>
<td></td>
<td>Action: Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td>16 (X'10') Meaning: The GTZLQRY request encountered an internal, component error and did not complete successfully.</td>
</tr>
<tr>
<td></td>
<td>Action: Contact IBM Service.</td>
</tr>
<tr>
<td>GTZLQRY_RSN</td>
<td>(See the description following this table.)</td>
</tr>
<tr>
<td>GTZLQRY_SYSTEMDIAG</td>
<td>Additional diagnostic information for some non-zero return codes. An up to 100-character string.</td>
</tr>
</tbody>
</table>

Description for output variable GTZLQRY_RSN: Reason code. For non-zero return codes. This is a hexadecimal text string so that for example RSN=X'00000841' is reported as "00000841", not "2113". The reason codes are as follows:

"rrrr0401"

**Meaning:** The underlying GTZQUERY service reported a return code of 4 and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.
"rrrr0801"

**Meaning:** The underlying GTZQUERY service reported a return code of 8 and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrLL0802"

**Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

**Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0803"

**Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

**Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "LLLL" is not zero then it is the last two bytes of the length as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"rrrr0C01"

**Meaning:** The underlying GTZQUERY service reported a return code of 12 (X'C') and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"xxxx0C02"

**Meaning:** GTZLQRY was invoked with a NUL REXX ENVBLOCK address in register 0 on entry.

**Action:** Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

"xxxx0C03"

**Meaning:** GTZLQRY was invoked with an unrecognized ENVBLOCK, pointed to by register 0 on entry.

**Action:** Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

In addition, GTZLQRY can issue an ABEND with system completion code E77 with the following possible ABEND reason codes:

<table>
<thead>
<tr>
<th>ABEND reason code</th>
<th>Reason code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'rrrr0C27'</td>
<td><strong>Meaning:</strong> GTZLQRY could not write any output REXX variables.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Ensure that GTZLQRY is called within a valid REXX environment, which</td>
</tr>
<tr>
<td></td>
<td>includes GTZLQRY being called with a valid ENVBLOCK in register 0 on entry. A</td>
</tr>
<tr>
<td></td>
<td>non-zero &quot;rrrr&quot; value provides additional diagnostic information. Check the</td>
</tr>
<tr>
<td></td>
<td>GTZLQRY_RSN variable description for possibly related values.</td>
</tr>
</tbody>
</table>
Table 49. REQUEST="STATUS" ABEND reason codes and descriptions (continued)

<table>
<thead>
<tr>
<th>ABEND reason code</th>
<th>Reason code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'xxxx0C29'</td>
<td>Meaning: An error occurred while accessing the internal GTZLQRY parameter list.</td>
</tr>
<tr>
<td></td>
<td>Action: Ensure that GTZLQRY is called within a valid REXX environment, ensuring standard REXX linkage conventions.</td>
</tr>
<tr>
<td>X'xxxx10xx'</td>
<td>Meaning: An unexpected error occurred.</td>
</tr>
<tr>
<td></td>
<td>Action: Contact IBM Service.</td>
</tr>
</tbody>
</table>

GTZLQRY "TRACKDATA" interface

Accepts input using multiple variables. See also the description of corresponding parameters on the GTZQUERY macro service for more details on single parameters. The following required statement requests to return trackdata entries:

GTZLQRY_REQUEST="TRACKDATA"

The following optional statement requests to return only up to maxresults number of trackdata entries values:

GTZLQRY_OPTION.MAXRESULTS = { "ALL" | "maxresults" }

Enter special value "ALL" or an unsigned number in the double-word (64-bit) range.

Optionally enter one or more filter variables, all starting with GTZLQRY_FILTER. Note that all filter values, except for SOURCETYPE and PROGRAMTYPE, also accept values including wild card characters. For text values, an asterisk (\*) will match zero or more characters of any type, and a question mark (?) will match a single character of any type. For numeric or binary values, such as EVENTDATA or EVENTASID, only a single asterisk (\*) is supported, as a way to explicitly specify the otherwise implied default value of "match all or match any value".

The following optional statement requests to return only such tracked instances that have a matching OWNER value.

GTZLQRY_FILTER.OWNER = "owner"

The value is a 1-16 character string.

The following optional statement identifies what type of tracked instance source values to match.

GTZLQRY_FILTER.SOURCETYPE = { "ALL" | "NOPATH" | "PATH" }

The following optional statement requests to return only those tracked instances that have a matching SOURCE value.

GTZLQRY_FILTER.SOURCE = "source"

The value is a 1-8 character string that also requires the specification of GTZLQRY_FILTER.SOURCETYPE = "NOPATH".

The following optional statement requests to return only that tracked instances that have a matching SOURCEPATH value.

GTZLQRY_FILTER.SOURCEPATH = "sourcepath"
The value is a 1-1024 character string that also requires the specification of
GTZLQRY_FILTER.SOURCETYPE = "PATH".

The following optional statement requests to return only those tracked instances
that have a matching EVENTDESC value.
GTZLQRY_FILTER.EVENTDESC = "eventdesc"

The value is a 0-64 character string.

The following optional statement requests to return only those tracked instances
that have a matching EVENTDATA value.
GTZLQRY_FILTER.EVENTDATA = {"eventdata" | "*"}

The value, if not "*", is a 16 byte string with a binary representation that is used for
comparisons.

The following optional statement requests to return only those tracked instances
that have a matching EVENTASID value.
GTZLQRY_FILTER.EVENTASID = {"eventASID" | "*"}

The value is a number between 0 and 65535 (FFFFX)

Tip: REXX function X2D can help specify the number in hex format in the REXX
source.

The following optional statement requests to return only those tracked instances
that were associated with a matching job name of the address space identified by
the EVENTASID value.
GTZLQRY_FILTER.EVENTJOB = "eventjob"

The value is a 1-8 character string.

The following optional statement identifies the type of tracked instance program
values to match.
GTZLQRY_FILTER.PROGRAMTYPE = { "ALL" | "NOPATH" | "PATH" }

The following optional statement requests to return only those tracked instances
that have a matching PROGRAM value.
GTZLQRY_FILTER.PROGRAM = "program"

The value is a 1-8 character string that also requires the specification of
GTZLQRY_FILTER.PROGRAMTYPE = "NOPATH".

The following optional statement requests to return only those tracked instances
that have a matching PROGRAMPATH values.
GTZLQRY_FILTER.PROGRAMPATH = "programpath"

The value is a 1-1024 character string that also requires the specification of
GTZLQRY_FILTER.PROGRAMTYPE = "PATH".

The following optional statement requests to return only those tracked instances
that have a matching PROGRAMOFFSET value.
GTZLQRY_FILTER.PROGRAMOFFSET = { "programoffset" | "*"}

The value is an unsigned number in the double-word (64-bit) range.
**Tip:** REXX function X2D can help specify the number in hex format in the REXX source.

The following optional statement requests to return only those tracked instances that have a matching HOMEASID value.

```bash
GTZLQRY_FILTER.HOMEASID = { "homeASID" | "*"}
```

The value is a number between 0 and 65535 ('FFFF'X).

**Tip:** REXX function X2D can help specify the number in hex format in the REXX source.

The following optional statement requests to return only those tracked instances that have a matching HOMEJOB value.

```bash
GTZLQRY_FILTER.HOMEJOB = "homejob"
```

The value is a 1-8 character string.

Specifying GTZLQRY with REQUEST="TRACKDATA" returns its output with the variables described in the following table.

**Table 50. REQUEST="TRACKDATA" output variable names and descriptions**

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTZQUAA.TrackDataEntriesAvailable</td>
<td>Number of TRACKDATA entries which are known to the tracking facility and which match the (optional) filter.</td>
</tr>
<tr>
<td>GTZQUAA.TrackDataEntriesProvided</td>
<td>Number (n) of TRACKDATA entries provided, if n &gt; 0, via variables GTZQUAA.i. with i=1..n. This number n is determined as MIN(GTZQUAA.TrackDataEntriesAvailable, GTZLQRY_OPTION.MAXRESULTS).</td>
</tr>
<tr>
<td>GtzQuaaT.i.isSourcePath</td>
<td>1, if GtzQuaaT.i.Source represents a SOURCE, otherwise &quot;0&quot;, to indicate a SOURCEPATH</td>
</tr>
<tr>
<td>GtzQuaaT.i.isProgramPath</td>
<td>1, if GtzQuaaT.Program represents a PROGRAM, otherwise &quot;0&quot;, to indicate a PROGRAMPATH</td>
</tr>
<tr>
<td>GtzQuaaT.i.isAuthorized</td>
<td>1, if the tracked event ran authorized, otherwise '0'</td>
</tr>
<tr>
<td>GtzQuaaT.i.FirstTOD</td>
<td>The timestamp when the first instance of this (unique) tracked instance was recorded (all others just had the occurrence count incremented). This value is provided in STCK format. See for example BLSUXTOD to format this as human readable timestamp.</td>
</tr>
<tr>
<td>GtzQuaaT.i.Count</td>
<td>How often this (unique) tracked instance was recorded.</td>
</tr>
<tr>
<td>GtzQuaaT.i.Owner</td>
<td>OWNER value, an up to 16-character string</td>
</tr>
<tr>
<td>GtzQuaaT.i.Source</td>
<td>SOURCE or SOURCEPATH value, depending on GtzQuaaT.i.isSourcePath, an up to 8-character resp. 1024-character string</td>
</tr>
<tr>
<td>GtzQuaaT.i.EventDesc</td>
<td>EVENTDESC value, an up to 64-character string</td>
</tr>
<tr>
<td>Output variable name</td>
<td>Variable description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>GtzQuaaT.iEventData</td>
<td>EVENTDATA value, a 16-byte binary string</td>
</tr>
<tr>
<td>GtzQuaaT.iEventJob</td>
<td>The derived EVENTJOB-name value, an up to 8-character string</td>
</tr>
<tr>
<td>GtzQuaaT.iHomeJob</td>
<td>The derived HOMEJOB-name value, an up to 8-character string</td>
</tr>
<tr>
<td>GtzQuaaT.iProgram</td>
<td>The derived PROGRAM or PROGRAMPATH value, depending on GtzQuaaT.i.isProgramPath, an up to 8-character resp. 1024-character string</td>
</tr>
<tr>
<td>GtzQuaaT.iProgramOffset</td>
<td>The derived PROGRAMOFFSET value, a decimal unsigned number in the double-word (64-bit) range</td>
</tr>
<tr>
<td>GtzQuaaT.iEventASID</td>
<td>EVENTASID value, a decimal number in the range 0 through 65535 (X'FFFF')</td>
</tr>
<tr>
<td>GtzQuaaT.iHomeASID</td>
<td>HOMEASID value, a decimal number in the range 0 through 65535 (X'FFFF')</td>
</tr>
</tbody>
</table>
### Table 50. REQUEST="TRACKDATA" output variable names and descriptions (continued)

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTZLQRY_RC</td>
<td>Return code. Same as the REXX supplied output variable RESULT which REXX sets if the function result is not explicitly captured via for example RC = GTZLQRY(). This is a decimal text string so that for example RC=12 is reported as &quot;12&quot;, not &quot;0C&quot;. The possible return codes are as follows:</td>
</tr>
<tr>
<td></td>
<td><strong>0</strong> <strong>Meaning:</strong> The GTZLQRY request completed successfully.</td>
</tr>
<tr>
<td></td>
<td><strong>4</strong> <strong>Meaning:</strong> The GTZLQRY request completed successfully, but issued a warning.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td><strong>8</strong> <strong>Meaning:</strong> The GTZLQRY request encountered an error and did not complete successfully.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td><strong>12 (X'C')</strong> <strong>Meaning:</strong> The GTZLQRY request encountered a severe error, typically related to the environment in which GTZLQRY was invoked, and did not complete successfully.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td><strong>16 (X'10')</strong> <strong>Meaning:</strong> The GTZLQRY request encountered an internal, component error and did not complete successfully.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Contact IBM Service.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GTZLQRY_RSN</th>
<th>(See the description following this table.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTZLQRY_SYSTEMDIAG</td>
<td>Additional diagnostic information for some non-zero return codes. An up to 100-character string.</td>
</tr>
</tbody>
</table>

Description for output variable GTZLQRY_RSN: Reason code. For non-zero return codes. This is a hexadecimal text string so that for example RSN=X'00000841' is reported as "00000841", not "2113". The reason codes are as follows:

"rrrr0401"

**Meaning:** The underlying GTZQUERY service reported a return code of 4 and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.
"rrrr0801"

**Meaning:** The underlying GTZQUERY service reported a return code of 8 and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrLL0802"

**Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

**Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0803"

**Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

**Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "LLLL" is not zero then it is the last two bytes of the length as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"rrLL0804"

**Meaning:** An unrecognized GTZLQRY_OPTION.MAXRESULTS value has been found.

**Action:** Ensure you only specify valid GTZLQRY_OPTION.MAXRESULTS values ("ALL" or a valid number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_OPTION.MAXRESULTS input variable.

"xxxx0805"

**Meaning:** No SOURCETYPE or an invalid SOURCETYPE has been specified via input variable GTZLQRY_FILTER.SOURCETYPE.

**Action:** When GTZLQRY_FILTER.SOURCE is specified on input, ensure that you also specify a matching GTZLQRY_FILTER.SOURCETYPE of "NOPATH".

"xxxx0806"

**Meaning:** No PROGRAMTYPE or an invalid PROGRAMTYPE has been specified via input variable GTZLQRY_FILTER.PROGRAMTYPE.

**Action:** When GTZLQRY_FILTER.PROGRAM is specified on input, ensure that you also specify a matching GTZLQRY_FILTER.PROGRAMTYPE of "NOPATH".

"xxxx0807"

**Meaning:** No SOURCETYPE or an invalid SOURCETYPE has been specified via input variable GTZLQRY_FILTER.SOURCETYPE.

**Action:** When GTZLQRY_FILTER.SOURCEPATH is specified on input, ensure that you also specify a matching GTZLQRY_FILTER.SOURCETYPE of "PATH".

"xxxx0808"

**Meaning:** No PROGRAMTYPE or an invalid PROGRAMTYPE has been specified via input variable GTZLQRY_FILTER.PROGRAMTYPE.
Action: When GTZLQRY_FILTER.PROGRAMPATH is specified on input, ensure that you also specify a matching GTZLQRY_FILTER.PROGRAMTYPE of "PATH".

"rrLL0809"

Meaning: An unrecognized GTZLQRY_FILTER.SOURCETYPE value has been found.

Action: Ensure you only specify valid GTZLQRY_FILTER.SOURCETYPE values ("ALL", "PATH", or "NOPATH"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.SOURCETYPE input variable.

"rrLL080A"

Meaning: An unrecognized GTZLQRY_FILTER.PROGRAMTYPE value has been found.

Action: Ensure you only specify valid GTZLQRY_FILTER.PROGRAMTYPE values ("ALL", "PATH", or "NOPATH"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.PROGRAMTYPE input variable.

"rrLL080B"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.SOURCE input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.SOURCE values (up to eight characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.SOURCE input variable.

"rrLL080C"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.PROGRAM input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.PROGRAM values (up to eight characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.PROGRAM input variable.

"rrLL080D"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.SOURCEPATH input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.SOURCEPATH values (up to 1024 characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.SOURCEPATH input variable.

"rrLL080E"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.PROGRAMPATH input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.PROGRAMPATH values (up to 1024 characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as
reported by service IRXEXCOM used to retrieve the
GTZLQRY_FILTER.PROGRAMPATH input variable.

"rrLL080F"

**Meaning:** When attempting to retrieve the
GTZLQRY_FILTER.EVENTDESC input value an error was encountered.

**Action:** Ensure you only specify valid GTZLQRY_FILTER.EVENTDESC
values (up to 64 characters long). If "rrLL" is not zero then "rr" is the return
code and "LL" is the last byte of the length of the variable as reported by
service IRXEXCOM used to retrieve the GTZLQRY_FILTER.EVENTDESC
input variable.

"rrLL0810"

**Meaning:** When attempting to retrieve the
GTZLQRY_FILTER.EVENTDATA input value an error was encountered.

**Action:** Ensure you only specify valid GTZLQRY_FILTER.EVENTDATA
values (16 bytes long, or "."). If "rrLL" is not zero then "rr" is the return
code and "LL" is the last byte of the length of the variable as reported by
service IRXEXCOM used to retrieve the GTZLQRY_FILTER.EVENTDATA
input variable.

"rrLL0811"

**Meaning:** When attempting to retrieve the GTZLQRY_FILTER.EVENTJOB
input value an error was encountered.

**Action:** Ensure you only specify valid GTZLQRY_FILTER.EVENTJOB
values (up to eight characters long). If "rrLL" is not zero then "rr" is the return
code and "LL" is the last byte of the length of the variable as
reported by service IRXEXCOM used to retrieve the
GTZLQRY_FILTER.EVENTJOB input variable.

"rrLL0812"

**Meaning:** When attempting to retrieve the GTZLQRY_FILTER.HOMEJOB
input value an error was encountered.

**Action:** Ensure you only specify valid GTZLQRY_FILTER.HOMEJOB
values (up to eight characters long). If "rrLL" is not zero then "rr" is the return
code and "LL" is the last byte of the length of the variable as
reported by service IRXEXCOM used to retrieve the
GTZLQRY_FILTER.HOMEJOB input variable.

"rrLL0813"

**Meaning:** When attempting to retrieve the GTZLQRY_FILTER.OWNER
input value an error was encountered.

**Action:** Ensure you only specify valid GTZLQRY_FILTER.OWNER values
(up to 16 characters long). If "rrLL" is not zero then "rr" is the return code
and "LL" is the last byte of the length of the variable as reported by service
IRXEXCOM used to retrieve the GTZLQRY_FILTER.OWNER input variable.

"rrLL0814"

**Meaning:** An unrecognized GTZLQRY_FILTER.PROGRAMOFFSET value
has been found.

**Action:** Ensure you only specify valid
GTZLQRY_FILTER.PROGRAMOFFSET values (double-word number). If
"rrLL" is not zero then "rr" is the return code and "LL" is the last byte of
the length of the variable as reported by service IRXEXCOM used to
retrieve the GTZLQRY_FILTER.PROGRAMOFFSET input variable.
"rrLL0815"

**Meaning:** An unrecognized GTZLQRY_FILTER.EVENTASID value has been found.

**Action:** Ensure you only specify valid GTZLQRY_FILTER.EVENTASID values (half-word number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.EVENTASID input variable.

"rrLL0816"

**Meaning:** An unrecognized GTZLQRY_FILTER.HOMEASID value has been found.

**Action:** Ensure you only specify valid GTZLQRY_FILTER.HOMEASID values (half-word number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.HOMEASID input variable.

"rrrr0C01"

**Meaning:** The underlying GTZQUERY service reported a return code of 12 (X'C') and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"xxxx0C02"

**Meaning:** GTZLQRY was invoked with a NUL REXX ENVBLOCK address in register 0 on entry.

**Action:** Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

"xxxx0C03"

**Meaning:** GTZLQRY was invoked with an unrecognized ENVBLOCK, pointed to by register 0 on entry.

**Action:** Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

In addition, GTZLQRY can issue an ABEND with system completion code E77 with the following possible ABEND reason codes:

**Table 51. REQUEST="TRACKDATA" ABEND reason codes and descriptions**

<table>
<thead>
<tr>
<th>ABEND reason code</th>
<th>Reason code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'rrrr0C27'</td>
<td><strong>Meaning:</strong> GTZLQRY could not write any output REXX variables.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Ensure that GTZLQRY is called within a valid REXX environment, which includes GTZLQRY being called with a valid ENVBLOCK in register 0 on entry. A non-zero &quot;rrrr&quot; value provides additional diagnostic information. Check the GTZLQRY_RSN variable description for possibly related values.</td>
</tr>
</tbody>
</table>
Table 51. REQUEST="TRACKDATA" ABEND reason codes and descriptions (continued)

<table>
<thead>
<tr>
<th>ABEND reason code</th>
<th>Reason code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'xxxx0C29'</td>
<td><strong>Meaning:</strong> An error occurred while accessing the internal GTZLQRY parameter list.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Ensure that GTZLQRY is called within a valid REXX environment, ensuring</td>
</tr>
<tr>
<td></td>
<td>standard REXX linkage conventions.</td>
</tr>
<tr>
<td>X'xxxx10xx'</td>
<td><strong>Meaning:</strong> An unexpected error occurred.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Contact IBM Service.</td>
</tr>
</tbody>
</table>

**GTZLQRY "EXCLUDE" interface**

Accepts input using multiple variables. See also the description of corresponding parameters on the GTZQUERY macro service for more details on single parameters. The following required statement requests to return EXCLUDE statements:

```
GTZLQRY_REQUEST="EXCLUDE"
```

The following optional statement requests to return only up to maxresults number of EXCLUDE statements:

```
GTZLQRY_OPTION.MAXRESULTS = { "ALL" | "maxresults" }
```

Enter special value "ALL" or an unsigned number in the double-word (64-bit) range.

**Note:** This is provided mainly for consistency among the list requests such as REQUEST="TRACKDATA". Unlike REQUEST="TRACKDATA", which potentially can return tens of thousands or more entries, REQUEST="EXCLUDE" is expected to return at most a few dozen entries in a typical configuration.

Specifying GTZLQRY with REQUEST="EXCLUDE" returns its output with the variables described in the following table.

Table 52. REQUEST="EXCLUDE" output variable names and descriptions

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTZQUAAE.ExcludeEntriesAvailable</td>
<td>Number of EXCLUDE statements known to the tracking facility.</td>
</tr>
<tr>
<td>GTZQUAAE.ExcludeEntriesProvided</td>
<td>Number (n) of EXCLUDE statements provided, if n &gt; 0, via variables GTZQUAAE.i with i=1..n. This number n is determined as MIN(GTZQUAAE.ExcludeEntriesAvailable, GTZLQRY_OPTION.MAXRESULTS).</td>
</tr>
<tr>
<td>GtzQuaaE.i.OriginType</td>
<td>Indicates where this EXCLUDE originated from: PARMLIB - A GTZPRMxx parmlib member, COMMAND - A SETGTZ EXCLUDE command, PROGRAM - A program interface</td>
</tr>
<tr>
<td>GtzQuaaE.i.OriginSuffix</td>
<td>If this EXCLUDE was specified via a GTZPRMxx parmlib member, as indicated by a GtzQuaaE.i.OriginType value of 'PARMLIB', this field here contains the 2-character xx suffix.</td>
</tr>
</tbody>
</table>
Table 52. REQUEST="EXCLUDE" output variable names and descriptions (continued)

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GtzQuaaE.i.Filter.</td>
<td>A set of variables indicating what TRACKDATA entries this EXCLUDE statement is supposed to filter out. For all but GtzQuaaE.i.Filter.SourceType and GtzQuaaE.i.Filter.ProgramType this might be a single wildcard character &quot;*&quot;, when not filtering for a specific value, indicating that all values for the respective field will match this filter.</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.SourceType</td>
<td>Indicates what type of source this filter is defined to match: SOURCE, &quot;SOURCEPATH&quot;, or &quot;ALL&quot;</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.ProgramType</td>
<td>Indicates what type of program this filter is defined to match: PROGRAM, &quot;PROGRAMPATH&quot;, or &quot;ALL&quot;</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.Owner</td>
<td>OWNER filter value. An up to 16-character string.</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.Source</td>
<td>SOURCE or SOURCEPATH filter value, depending on the value of GtzQuaaE.i.Filter.SourceType: For SourceType=&quot;SOURCE&quot; it is a source filter, for SourceType=&quot;SOURCEPATH&quot; it is a source path filter, and for SourceType=&quot;ALL&quot; it will always be &quot;*&quot; (match all). An up to 8-character resp. 1024-character string</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.EventData</td>
<td>Event data filter value. A 16-byte binary string or &quot;*&quot;.</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.EventJob</td>
<td>Event job filter value. An up to 8-character string.</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.HomeJob</td>
<td>Home job filter value. An up to 8-character string.</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.Program</td>
<td>PROGRAM or PROGRAMPATH filter value, depending on the value of GtzQuaaE.i.Filter.ProgramType: For ProgramType=&quot;PROGRAM&quot; it is a source filter, for ProgramType=&quot;PROGRAMPATH&quot; it is a program path filter, and for ProgramType=&quot;ALL&quot; it will always be &quot;*&quot; (match all). An up to 8-character resp. 1024-character string</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.ProgramOffset</td>
<td>PROGRAMOFFSET filter value. A decimal unsigned number in the double-word (64-bit) range</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.EventASID</td>
<td>Event ASID filter value. A decimal number in the range 0 through 65535 (X'FFFF')</td>
</tr>
<tr>
<td>GtzQuaaE.i.Filter.HomeASID</td>
<td>Home ASID filter value. A decimal number in the range 0 through 65535 (X'FFFF')</td>
</tr>
</tbody>
</table>
### Table 52. REQUEST="EXCLUDE" output variable names and descriptions (continued)

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
</table>
| GTZLQRY_RC           | Return code. Same as the REXX supplied output variable RESULT which REXX sets if the function result is not explicitly captured via for example RC = GTZLQRY(). This is a decimal text string so that for example RC=12 is reported as "12", not "0C". The possible return codes are as follows:
|                      | 0  **Meaning:** The GTZLQRY request completed successfully.  
|                      | 4  **Meaning:** The GTZLQRY request completed successfully, but issued a warning.  
|                      |   **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN.  
|                      | 8  **Meaning:** The GTZLQRY request encountered an error and did not complete successfully.  
|                      |   **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN.  
|                      | 12 (X'C') **Meaning:** The GTZLQRY request encountered a severe error, typically related to the environment in which GTZLQRY was invoked, and did not complete successfully.  
|                      |   **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN.  
|                      | 16 (X'10') **Meaning:** The GTZLQRY request encountered an internal, component error and did not complete successfully.  
|                      |   **Action:** Contact IBM Service.  
| GTZLQRY_RSN          | (See the description following this table.)  
| GTZLQRY_SYSTEMDIAG   | Additional diagnostic information for some non-zero return codes. An up to 100-character string.  

Description for output variable GTZLQRY_RSN: Reason code. For non-zero return codes. This is a hexadecimal text string so that for example RSN=X'00000841' is reported as "00000841", not "2113". The reason codes are as follows:

"rrrr0401"

**Meaning:** The underlying GTZQUERY service reported a return code of 4 and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.
"rrrr0801"
  **Meaning:** The underlying GTZQUERY service reported a return code of 8 and reason code "rrrr".
  **Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrLL0802"
  **Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.
  **Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0803"
  **Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.
  **Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "LLLL" is not zero then it is the last two bytes of the length as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0804"
  **Meaning:** An unrecognized GTZLQRY_OPTION.MAXRESULTS value has been found.
  **Action:** Ensure you only specify valid GTZLQRY_OPTION.MAXRESULTS values ("ALL" or a valid number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_OPTION.MAXRESULTS input variable.

"rrrr0C01"
  **Meaning:** The underlying GTZQUERY service reported a return code of 12 (X'C') and reason code "rrrr".
  **Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"xxxx0C02"
  **Meaning:** GTZLQRY was invoked with a NUL REXX ENVBLOCK address in register 0 on entry.
  **Action:** Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

"xxxx0C03"
  **Meaning:** GTZLQRY was invoked with an unrecognized ENVBLOCK, pointed to by register 0 on entry.
  **Action:** Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

In addition GTZLQRY can issue an ABEND with system completion code of E77 with the following possible ABEND reason codes:
Table 53. REQUEST="EXCLUDE" ABEND reason codes and descriptions

<table>
<thead>
<tr>
<th>ABEND reason code</th>
<th>Reason code description</th>
</tr>
</thead>
</table>
| X'rrrt0C27'       | **Meaning:** GTZLQRY could not write any output REXX variables.  
                  | **Action:** Ensure that GTZLQRY is called within a valid REXX environment, which includes GTZLQRY being called with a valid ENVBLOCK in register 0 on entry. A non-zero "rrrt" value provides additional diagnostic information. Check the GTZLQRY_RSN variable description for possibly related values. |
| X'xxxx0C29'       | **Meaning:** An error occurred while accessing the internal GTZLQRY parameter list.  
                  | **Action:** Ensure that GTZLQRY is called within a valid REXX environment, ensuring standard REXX linkage conventions. |
| X'xxxx10xx'       | **Meaning:** An unexpected error occurred.  
                  | **Action:** Contact IBM Service. |

**GTZLQRY "DEBUG" interface**

Accepts input using multiple variables. See also the description of corresponding parameters on the GTZQUERY macro service for more details on single parameters. The following required statement requests to return DEBUG statements:

```plaintext
GTZLQRY_REQUEST="DEBUG"
```

The following optional statement requests to return only up to maxresults number of DEBUG statements:

```plaintext
GTZLQRY_OPTION.MAXRESULTS = { "ALL" | "maxresults" }
```

Enter special value "ALL" or an unsigned number in the double-word (64-bit) range.

**Note:** This is provided mainly for consistency among the list requests such as REQUEST="TRACKDATA". Unlike REQUEST="TRACKDATA", which potentially can return tens of thousands or more entries, REQUEST="DEBUG" is expected to return at most a few dozen entries in a typical configuration.

Specifying GTZLQRY with REQUEST="DEBUG" returns its output with the variables described in the following table.

Table 54. REQUEST="DEBUG" output variable names and descriptions

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTZQUAAD.DebugEntriesAvailable</td>
<td>Number of DEBUG statements known to the tracking facility.</td>
</tr>
<tr>
<td>GTZQUAAD.DebugEntriesProvided</td>
<td>Number (n) of DEBUG statements provided, if n &gt; 0, via variables GTZQUAAD.i with i=1..n. This number n is determined as MIN(GTZQUAAD.DebugEntriesAvailable, GTZLQRY_OPTION.MAXRESULTS).</td>
</tr>
<tr>
<td>Output variable name</td>
<td>Variable description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>GTZQUAAD.i.OriginType</td>
<td>Indicates where this DEBUG originated from: PARMLIB - A GTZPRMxx parmlib member, COMMAND - A SETGTZ DEBUG command, PROGRAM - A program interface</td>
</tr>
<tr>
<td>GTZQUAAD.i.OriginSuffix</td>
<td>If this DEBUG was specified via a GTZPRMxx parmlib member, as indicated by a GtzQuaaD.i.OriginType value of &quot;PARMLIB&quot;, this field here contains the 2-character xx suffix.</td>
</tr>
<tr>
<td>GtzQuaaD.i.Reason</td>
<td>The DEBUG REASON code. A decimal number in the range 0 through 65535 (X'FFFF')</td>
</tr>
<tr>
<td>GtzQuaaD.i.Action</td>
<td>The ACTION requested when this DEBUG is matched: &quot;ABEND&quot; or &quot;DUMP&quot;.</td>
</tr>
<tr>
<td>GtzQuaaD.i.ActionLimit</td>
<td>How often the system is allowed to trigger the action for this DEBUG statement (when matched by a new tracked instance candidate). A value of &quot;NOLIMIT&quot; means &quot;no limit&quot;, otherwise a decimal number in the range 1 through 65535.</td>
</tr>
<tr>
<td>GtzQuaaD.i.ActionCount</td>
<td>How often this DEBUG statement triggered an action so far. A decimal number in the range 0 through 65535 which counts towards the ActionLimit, unless &quot;NOLIMIT&quot;.</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.</td>
<td>A set of variables indicating what TRACKDATA entries this DEBUG statement is supposed to trigger for. For all but GtzQuaaD.i.Filter.SourceType and GtzQuaaD.i.Filter.ProgramType this might be a single wildcard character &quot;*&quot;, when not filtering for a specific value, indicating that all values for the respective field will match this filter.</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.Owner</td>
<td>OWNER filter value. An up to 16-character string.</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.Source</td>
<td>SOURCE or SOURCEPATH filter value, depending on the value of GtzQuaaD.i.Filter.SourceType: For SourceType=&quot;SOURCE&quot; it is a source filter, for SourceType=&quot;SOURCEPATH&quot; it is a source path filter, and for SourceType=&quot;ALL&quot; it will always be &quot;*&quot; (match all). An up to 8-character resp. 1024-character string.</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.EventData</td>
<td>Event data filter value. A 16-byte binary string or &quot;*&quot;.</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.EventJob</td>
<td>Event job filter value. An up to 8-character string.</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.HomeJob</td>
<td>Home job filter value. An up to 8-character string.</td>
</tr>
<tr>
<td>Output variable name</td>
<td>Variable description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.Program</td>
<td>PROGRAM or PROGRAMPATH filter value, depending on the value of GtzQuaaD.i.Filter.ProgramType: For ProgramType=&quot;PROGRAM&quot; it is a source filter, for ProgramType=&quot;PROGRAMPATH&quot; it is a program path filter, and for ProgramType=&quot;ALL&quot; it will always be &quot;*&quot; (match all). An up to 8-character resp. 1024-character string</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.ProgramOffset</td>
<td>PROGRAMOFFSET filter value. A decimal unsigned number in the double-word (64-bit) range</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.EventASID</td>
<td>Event ASID filter value. A decimal number in the range 0 through 65535 (X'FFFF')</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.HomeASID</td>
<td>Home ASID filter value. A decimal number in the range 0 through 65535 (X'FFFF')</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.Owner</td>
<td>OWNER filter value. An up to 16-character string.</td>
</tr>
<tr>
<td>GtzQuaaD.i.Filter.Source</td>
<td>SOURCE or SOURCEPATH filter value, depending on the value of GtzQuaaD.i.FilterSourceType: For SourceType=&quot;SOURCE&quot; it is a source filter, for SourceType=&quot;SOURCEPATH&quot; it is a source path filter, and for SourceType=&quot;ALL&quot; it will always be &quot;*&quot; (match all). An up to 8-character resp. 1024-character string</td>
</tr>
</tbody>
</table>
Table 54. REQUEST="DEBUG" output variable names and descriptions (continued)

<table>
<thead>
<tr>
<th>Output variable name</th>
<th>Variable description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTZLQRY_RC</td>
<td>Return code. Same as the REXX supplied output variable RESULT which REXX sets if the function result is not explicitly captured via for example RC = GTZLQRY(). This is a decimal text string so that for example RC=12 is reported as &quot;12&quot;, not &quot;0C&quot;. The possible return codes are as follows:</td>
</tr>
<tr>
<td></td>
<td>0 Meaning: The GTZLQRY request completed successfully.</td>
</tr>
<tr>
<td></td>
<td>4 Meaning: The GTZLQRY request completed successfully, but issued a warning. Action: Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td>8 Meaning: The GTZLQRY request encountered an error and did not complete successfully. Action: Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td>12 (X'C') Meaning: The GTZLQRY request encountered a severe error, typically related to the environment in which GTZLQRY was invoked, and did not complete successfully. Action: Refer to the action under the individual reason code returned in GTZLQRY_RSN.</td>
</tr>
<tr>
<td></td>
<td>16 (X'10') Meaning: The GTZLQRY request encountered a component error and did not complete successfully. Action: Contact IBM Service.</td>
</tr>
</tbody>
</table>

100-character string (see the description following this table)

**GTZLQRY_RSN**

GTZLQRY_SYSTEMDIAG Additional diagnostic information for some non-zero return codes. An up to 100-character string.

**Description for output variable GTZLQRY_RSN:** Reason code. For non-zero return codes, this is a hexadecimal text string so that for example RSN=X'00000841' is reported as "00000841", not "2113". The possible reason codes are as follows:

- "rrrr0401" Meaning: The underlying GTZQUERY service reported a return code of 4 and reason code "rrrr". Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.
"rrrr0801"

**Meaning:** The underlying GTZQUERY service reported a return code of 8 and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrLL0802"

**Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

**Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0803"

**Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

**Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "LLLL" is not zero then it is the last two bytes of the length as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0804"

**Meaning:** An unrecognized GTZLQRY_OPTION.MAXRESULTS value has been found.

**Action:** Ensure you only specify valid GTZLQRY_OPTION.MAXRESULTS values ("ALL" or a valid number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_OPTION.MAXRESULTS input variable.

"rrrr0C01"

**Meaning:** The underlying GTZQUERY service reported a return code of 12 (X'C') and reason code "rrrr".

**Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"xxxx0C02"

**Meaning:** GTZLQRY was invoked with a NUL REXX ENVBLOCK address in register 0 on entry.

**Action:** Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

"xxxx0C03"

**Meaning:** GTZLQRY was invoked with an unrecognized ENVBLOCK, pointed to by register 0 on entry.

**Action:** Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

In addition GTZLQRY can issue an ABEND with system completion code of E77 with the following possible ABEND reason codes:
Table 55. REQUEST="DEBUG" ABEND reason codes and descriptions

<table>
<thead>
<tr>
<th>ABEND reason code</th>
<th>Reason code description</th>
</tr>
</thead>
</table>
| X'rrrt0C27'       | **Meaning:** GTZLQRY could not write any output REXX variables.  
**Action:** Ensure that GTZLQRY is called within a valid REXX environment, which includes GTZLQRY being called with a valid ENVBLOCK in register 0 on entry. A non-zero "rrrr" value provides additional diagnostic information. Check the GTZLQRY_RSN variable description for possibly related values. |
| X'xxxx0C29'       | **Meaning:** An error occurred while accessing the internal GTZLQRY parameter list.  
**Action:** Ensure that GTZLQRY is called within a valid REXX environment, ensuring standard REXX linkage conventions. |
| X'xxxx10xx'       | **Meaning:** An unexpected error occurred.  
**Action:** Contact IBM Service. |

Data persistence and simplified generic tracking services

Use generic tracker service GTZTRACK to provide persisted records of tracked events in SMF type 117 records. With this function, you can retrieve historical information about tracked events whenever needed.

Generic tracker keeps track of many different types of data, including the following data types:

- An "exclusion list" made up of one or more EXCLUDE statements
- A set of DEBUG statements
- A set of TRACKDATA instances
- STATUS information

Of all of these, only TRACKDATA is enabled for persistence using SMF records.

This persistence of TRACKDATA is done in parallel to the TRACKDATA that is kept in the dynamic memory of the GTZ address space. The TRACKDATA that is persisted as SMF records is meant to serve as a backup, or long-term storage for later analysis, of such data. This is because the data that is kept in dynamic memory is only available until the GTZ address space shuts down, at the latest before the next system restart. You can still achieve GTZ data persistence across those system restarts without using SMF records by instead using the existing tools GTZPRINT, GTZQUERY, or DISPLAY GTZ. These tools can explicitly extract the data from the live GTZ address space.

The external interface for the TRACKDATA persistence using SMF records consists of the following pieces:

- SETGTZ operator sub-command PERSIST. SETGTZ PERSIST enables or disables recording using SMF records.
- GTZPRMxx parmlib member statement PERSIST, which is equivalent to the SETGTZ PERSIST operator command.
• Operator command DISPLAY GTZ,STATUS reports the current enablement status of SMF recording for GTZ data.
• Macro service GTZQUERY REQUEST=STATUS reports the current enablement status of SMF recording for GTZ data.
• REXX callable function GTZLQRY with GTZLQRY_REQUEST=‘STATUS’ reports the current enablement status of SMF recording for GTZ data among other information.
• Programs GTZSMFU2 and GTZSMFU3 allow for the retrieval of GTZ data from standard SMF record backend stores (dataset or logstream).
• Mapping macro GTZZSMF for the new top-level SMF record type 117, which is assigned to GTZ for its data persistence.
• Macro GTZZSMFU for return and reason codes from programs GTZSMFU2 and GTZSFMU3.
• Sample member GTZSMFJ in SYS1.PARMLIB demonstrates the use of SMF dump program exit routines GTZSMFU2 and GTZSMFU3.

SMF already provides the following dump programs to retrieve raw binary data from different SMF backend stores for SMF records:
• Program IFASMFDP for SMF records that are stored in data sets
• Program IFASMFDL for SMF data records that are stored in log streams.

However, these existing dump programs also support the use of user exit routines to filter and process individual SMF records from the overall set of SMF records that are stored in the targeted SMF record store (dataset or logstream).

Generic Tracker provides exit routines for two of those dump program user exits in order to extract and format, in text form that is very similar to the existing DISPLAY GTZ,TRACKDATA output format, any GTZ SMF records that are contained in the SMF record source:
• Program GTZSMFU2, to be used as an exit routine for exit USER2, is given control only when the SMF data set dump program selects a record to be written. GTZSMFU2 gains control for each record that is selected by any optional general filtering that is specified through the existing supported input to the SMF dump programs. GTZSMFU2 ignores any non-GTZ SMF records and formats GTZ SMF records as text in a new output DD, by the name of GTZOUT, of the SMF dump program.
• Program GTZSMFU3, to be used as an exit routine for exit USER3, is given control after the output data set is closed. Pair it with USER2 exit routine GTZSMFU2 so that GTZSMFU3 can close any GTZ output data sets that GTZSMFU2 opened initially and shared across multiple calls for different SMF records that are fed into GTZSMFU2.

In addition to GTZOUT as output DD, the exit routines GTZSMFU2 and GTZSMFU3 also expect a DD GTZPRINT as target for additional messages from the GTZ processing. Both GTZ DDs require LRECL=80.

Note: Usage details are also described in the prologue of the GTZSMFJ sample job.

On exit, the GTZOUT DD contains text similar to that in the following example:
/* GENERATED BY GTZSMFU2 (H8877AZ-14260) 2014-09-17 19:40:09 */

| INSTANCE: | 1 |
| EVENTDESC: | 'GTZFCT neutral event description' |
| OWNER: | IBM_GTZ_FCT |
| SOURCE: | MYSOURCE |
Mapping macro GTZZSMF provides the mapping for SMF records of type 117 and all supported subtypes. There is currently only one subtype, for TRACKDATA.

Mapping macro GTZZSMF is also referenced in the standard SMF record mapping macro IFASMFRC, which covers record types 114-123 and is referenced by the main include IFASMFR. That way customers can also create mappings for the GTZ SMF record type 117 using standard SMF include protocol 'IFASMFR 117'.

Use new operator command SETGTZ PERSIST or new statement PERSIST in the GTZPRMxx parmlib member to enable or disable track data persistence. For more information, see the SETGTZ PERSIST command in z/OS MVS System Commands and GTZPRMxx (Generic Tracker parameters) in z/OS MVS Initialization and Tuning Reference.

**DFSMs tracking**

- Event description starting with **GDGLIMIT**
  - EVENTDESC: ‘GDGLIMIT jobname stepname proname’
    
    The field name GDGLIMIT was used in a call to Catalog Management and did not also include GDGLIMTE. This may indicate that a program or utility is not capable of handling extended Generation Data Groups (GDGs). *jobname* is the name of the user application job, *stepname* and *proname* are also listed if they are not blanks.

    An example of tracked usage reported by the catalog for using GDGLIMIT (and not also using GDGLIMTE) could appear as the following:

    ```plaintext
    INSTANCE: 1 COUNT: 1
    EVENTDESC: 'GDGLIMIT CSIRXJCL STEP1 '
    OWNER: IBMDFSMS SOURCE: IGDFCLFA
    EVENTDATA: x0000000000000000 x0000000000000000
    PROGRAM: *UNKNOWN PROGRAMOFFSET: x0000000000000000
    HOMEJOB: CATALOG HOMEASID: x0002
    EVENTJOB: CATALOG EVENTASID: x0002
    AUTHORIZED: YES FIRST TIME: 2014-10-29 09:06:46
    ```
    
    *Figure 103. Example of tracked usage reported by the catalog for using GDGLIMIT (and not also using GDGLIMTE)*
- Event description starting with **SMS-**
  - See the “EAV migration assistance tracker” section in *z/OS DFSMSdfp Advanced Services*
- Event description 'IEFALC 01: <step name> <DD name>'
  - See message IEF384I “WARNING: VOLUME NOT RETRIEVED FROM catalog”.

## MVS Allocation tracking

- Event description 'IEFALC 01: <step name> <DD name>'
  - See message IEF384I “WARNING: VOLUME NOT RETRIEVED FROM catalog”.

## JES3 control statement tracking

JES3 is instrumented to report the use of JES3 control statements (JECL) in jobs that have been submitted to the system. Occurrences of JES3 JECL statements in a job will be reported during input service using the Generic Tracker macro, GTZTRACK. When GTZ tracking is enabled, JES3 will record GTZ data identifying the JES3 JECL statements found within a job stream.

JES3 generated GTZ records are identified with the OWNER field set to IBMJES3. Information in the JES3 record will aid in identifying the source of the job stream and the JES3 control statements found in the job stream.
Figure 104. Examples of JES3 control statement GTZ records

Description of the GTZ record fields with data provided by JES3:

- **OWNER** is the string IBMJES3 and identifies the JES3 subsystem as the source of the GTZ record.
- **SOURCE** identifies the JES3 module which identified the occurrence of a JES3 control statement in the job stream. This will either be IATISLG or IATISRI.
- **EVENTDATA** will be set to zeros.
- **PROGRAM** will be *UNKNOWN and **PROGRAMOFFSET** will be zeros as JES3 provides no program specific information.
- **EVENTDESC** is a 46 character string in which JES3 provides information about the job stream and the JES3 control statement usage within the job stream. The contents of EVENTDESC by character position is:

  1 = A starting delimiter, which is the vertical bar character, for the JES3 control statement usage indicators.
  2-18 = Each character identifies whether or not a specific JES3 control statement is used in the job stream. 0=Not used. 1=Used.

        2 = JES3 command statement
        3 = /*DATASET statement
        4 = /*FORMAT statement
        5 = /*MAIN statement
        6 = /*NET statement
        7 = /*NETACCT statement
        8 = /*OPERATOR statement
9 = /*PAUSE statement
10 = Blank character
11 = /*PROCESS statement
12 = /*ROUTE statement
13 = /*SIGNON statement
14-18 = Reserved
19 = An ending delimiter which is the vertical bar character.
20 = Blank character
21-28 = JES3 point of entry for the job stream.
29 = Blank character
30-37 = Job name when SOURCE=IATISLG.
       DDNAME of the reader when SOURCE=IATISRI.
38 = Blank character
39-46 = Submitting TSO userid when SOURCE=IATISLG.
       DDNAME of the reader when SOURCE=IATISRI.

Data for the remaining GTZ record fields is supplied by the Generic Tracker Facility.
Chapter 12. Component trace

The component trace service provides a way for MVS components to collect problem data about events. Each component that uses the component trace service has set up its trace in a way that provides the unique data needed for the component. A component trace provides data about events that occur in the component. The trace data is used by the IBM Support Center, which uses the trace data to:

- Diagnose problems in the component
- See how the component is running.

You will typically use component trace while recreating a problem.

Usage of System Resources: Some component traces use minimal system resources, especially while tracing a small number of events. These minimal traces often run anytime the component is running. Other traces use significant system resources, especially when many kinds of events are traced. These large traces are usually requested only when the IBM Support Center asks for them.

Run concurrent traces: You can run more than one component trace at a time; you can run component traces:

- Concurrently for several components on one system.
- Concurrently for one or more components on some or all of the systems in a sysplex.
- Concurrently for one component on a system. Multiple concurrent traces for a component are sublevel traces.
- Concurrently for several components on some or all of the systems in a sysplex and with sublevel traces.

The following topics describe tasks for component traces:

- “Planning for component tracing” on page 377 tells you about the tasks needed to plan component tracing.
- “Obtaining a component trace” on page 385 tells you how to request a specific component trace that is needed to diagnose a problem. The tasks depend on where you plan to put trace output and if you are running traces on multiple systems in a sysplex; therefore, requesting traces is presented in three topics:
  - “Request component tracing to address space or data space trace buffers” on page 385
  - “Request writing component trace data to trace data sets” on page 388
  - “Request component tracing for systems in a sysplex” on page 393
- “Verifying component tracing” on page 395 tells how an operator can check that a requested trace is running and that a component trace writer is running.
- “Viewing the component trace data” on page 398 tells you how to format the trace output.

This topic uses tables to show the similarities and differences in the individual traces from different components. Table 56 on page 376 summarizes each BCP component trace that uses the component trace service.
### Component Trace

**Table 56. Summary of BCP component traces that use the component trace service**

<table>
<thead>
<tr>
<th>Component</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced Program-to-Program Communication/MVS (APPC/MVS)</td>
<td>“SYSAPPC component trace” on page 399</td>
</tr>
<tr>
<td>Base control program internal interface (BCPi)</td>
<td>“SYSBCPii component trace” on page 417</td>
</tr>
<tr>
<td>Basic HyperSwap® socket support</td>
<td>“SYSBHS component trace” on page 420</td>
</tr>
<tr>
<td>Common Event Adapter (CEA)</td>
<td>“SYSCEA component trace” on page 425</td>
</tr>
<tr>
<td>Cross-system coupling facility (XCF)</td>
<td>“SYSXCF component trace” on page 526</td>
</tr>
<tr>
<td>Cross-system extended services (XES)</td>
<td>“SYSXES component trace” on page 531</td>
</tr>
<tr>
<td>Data lookaside facility (DLF) of VLF</td>
<td>“SYSDLF component trace” on page 427</td>
</tr>
<tr>
<td>Distributed function of SOMobjects®</td>
<td>“SYDSOM component trace” on page 429</td>
</tr>
<tr>
<td>Global resource serialization</td>
<td>“SYSGRS component trace” on page 436</td>
</tr>
<tr>
<td>Allocation Component</td>
<td>“SYSLEFAL component trace” on page 445</td>
</tr>
<tr>
<td>IOS Component Trace</td>
<td>“SYSIOS component trace” on page 451</td>
</tr>
<tr>
<td>JES common coupling services</td>
<td>“SYIJES component trace” on page 457</td>
</tr>
<tr>
<td>JES2 rolling trace table</td>
<td>“SYJES2 component trace” on page 467</td>
</tr>
<tr>
<td>Library lookaside (LLA) of contents supervision</td>
<td>“SYLLA component trace” on page 469</td>
</tr>
<tr>
<td>z/OS UNIX System Services (z/OS UNIX)</td>
<td>“SYSOMVS component trace” on page 478</td>
</tr>
<tr>
<td>Operations services (OPS)</td>
<td>“Requesting a SYSOPS trace” on page 490</td>
</tr>
<tr>
<td>Resource recovery services (RRS)</td>
<td>“SYSRRS component trace” on page 494</td>
</tr>
<tr>
<td>Real storage manager (RSM)</td>
<td>“SYRSRM component trace” on page 501</td>
</tr>
<tr>
<td>SDUMP</td>
<td>“SYSDUMP component trace” on page 432</td>
</tr>
<tr>
<td>Service processor interface (SPI)</td>
<td>“SYSSPI component trace” on page 518</td>
</tr>
<tr>
<td>System logger</td>
<td>“SYSLOGR component trace” on page 470</td>
</tr>
<tr>
<td>System REXX</td>
<td>“SYSAXR component trace” on page 413</td>
</tr>
<tr>
<td>Transaction trace (TTRC)</td>
<td>“SYSTTRC transaction trace” on page 520</td>
</tr>
<tr>
<td>Virtual lookaside facility (VLF)</td>
<td>“SYSVLF component trace” on page 520</td>
</tr>
<tr>
<td>Workload manager (WLM)</td>
<td>“SYSWLM component trace” on page 523</td>
</tr>
</tbody>
</table>

A program product or application, if authorized, can also use the component trace service to provide an **application trace**. See the documentation for the program product or application for information about its trace.

- See [z/OS MVS Initialization and Tuning Reference](https://www.ibm.com) for the CTnccxx parmlib member.
- See [z/OS MVS System Commands](https://www.ibm.com) for the TRACE CT command.
- See [z/OS MVS IPCS Commands](https://www.ibm.com) for the COPYDUMP, COPYTRC, CTRACE, GTFTTRACE, and MERGE subcommands.
- For a description of these messages, use LookAt or see [MVS System Messages](https://www.ibm.com).
- See [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com) for information on creating an **application trace**.
Planning for component tracing

Planning for component tracing consists of the following tasks, which are performed by the system programmer:

- "Create CTnccxx parmlib members for some components"
  - "Specify buffers" on page 379
- "Select the trace options for the component trace" on page 383
- "Decide where to collect the trace records" on page 384

Create CTnccxx parmlib members for some components

Table 57 shows if a component has a parmlib member, if the member is a default member needed at system or component initialization, and if the component has default tracing. Some components run default tracing at all times when the component is running; default tracing is usually minimal and covers only unexpected events. Other components run traces only when requested.

When preparing your production SYS1.PARMLIB system library, do the following:

1. Make sure the parmlib contains all default members identified in the table. If parmlib does not contain the default members at initialization, the system issues messages.

   Make sure that the IBM-supplied CTIITT00 member is in the parmlib. PARM=CTIITT00 can be specified on a TRACE CT command for a component trace that does not have a parmlib member; CTIITT00 prevents the system from prompting for a REPLY after the TRACE CT command. In a sysplex, CTIITT00 is useful to prevent each system from requesting a reply.

2. Decide if each default member meets the needs of your installation. If it does not, customize it.

3. Decide if the buffer size specified in the default members meets the needs of your installation. Some component traces do not allow buffer size change after initialization. Change the buffer size, if needed.

Most components can run only one component trace at a time; some components can run concurrent traces, called sublevel traces. Each sublevel trace is identified by its sublevel trace name. For some components, you need to identify the component's CTnccxx member in another parmlib member; the components with this requirement have the other parmlib member listed in the default member column in Table 57.

For example, for XCF specify CTIXCF00 on the CTRACE parameter in the COUPLExx parmlib member.

```
COUPLE SYSPLEX( ... 
    CTRACE(CTIXCF00)
    ...
```

Table 57. Determining if a component has a parmlib member

<table>
<thead>
<tr>
<th>Trace</th>
<th>Parmlib member</th>
<th>Default member</th>
<th>Default tracing beginning at initialization</th>
<th>Sublevel traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSAPPC</td>
<td>CTnAPPx (see &quot;CTnAPPx parmlib member&quot; on page 400)</td>
<td>No</td>
<td>No; cannot turn trace ON or OFF in CTnAPPx</td>
<td>No</td>
</tr>
<tr>
<td>SYSAXR</td>
<td>CTIAXRnn (see &quot;CTIAXRnn parmlib member&quot; on page 413)</td>
<td>CTIAXR00</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
## Component Trace

**Table 57. Determining if a component has a parmlib member (continued)**

<table>
<thead>
<tr>
<th>Trace</th>
<th>Parmlib member</th>
<th>Default member</th>
<th>Default tracing beginning at initialization</th>
<th>Sublevel traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSBCPII</td>
<td>CTIHWI00</td>
<td>CTIHWI00</td>
<td>Yes; minimal diagnostic tracing is always in effect. The presence of a valid CTIHWI00 parmlib at BCPi startup can modify these default trace options.</td>
<td>No</td>
</tr>
<tr>
<td>SYSBHI</td>
<td>CTIBHIxx</td>
<td>CTIBHI00</td>
<td>Yes; minimal; unexpected events</td>
<td>No</td>
</tr>
<tr>
<td>SYSCEA</td>
<td>CTICEA00 (see &quot;CTICEAnn parmlib member&quot; on page 423)</td>
<td>CTICEA00</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYSDLF</td>
<td>None</td>
<td>N/A</td>
<td>Yes; always on when DLF is running</td>
<td>No</td>
</tr>
<tr>
<td>SYSDOM</td>
<td>None</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSDUMP</td>
<td>CTIDMPxx</td>
<td>CTIDMP00</td>
<td>Yes, full tracing</td>
<td>No</td>
</tr>
<tr>
<td>SYSGRS</td>
<td>CTnGRSxx (see &quot;CTnGRSxx parmlib member&quot; on page 436)</td>
<td>CTIGRS00, which is specified in GRSCNF00 member</td>
<td>Yes, if global resource serialization is active; CONTROL and MONITOR options</td>
<td>No</td>
</tr>
<tr>
<td>SYSIEFAL</td>
<td>CTIIEFxx (see &quot;CTIIEFxx parmlib member&quot; on page 446)</td>
<td>CTIIEFAL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYSIOS</td>
<td>CTnIOSxx (see &quot;CTnIOSxx parmlib member&quot; on page 453)</td>
<td>No</td>
<td>Yes; minimal; unexpected events</td>
<td>No</td>
</tr>
<tr>
<td>SYSJES</td>
<td>CTnJESxx (see &quot;CTnJESxx parmlib member&quot; on page 459)</td>
<td>CTIJES01, CTIJES02, CTIJES03, CTIJES04 You should also receive and rename members IXZCTION and IXZCTIOF supplied in SYS1.SAMPLIB to CTIJES0N and CTIJESOF.</td>
<td>Yes; full tracing for sublevels XCFEVT and FLOW; minimal tracing of unexpected events for sublevels USRXIT and MSGTRC</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSjes2</td>
<td>None</td>
<td>N/A</td>
<td>Yes; always on when JES2 is running</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSLLA</td>
<td>None</td>
<td>N/A</td>
<td>Yes; always on when LLA is running</td>
<td>No</td>
</tr>
<tr>
<td>SYSLOGR</td>
<td>CTnLOGxx (see &quot;CTnLOGxx parmlib member&quot; on page 472)</td>
<td>CTILOG00, which can be specified in IXGCNFxx member.</td>
<td>Yes; activated during system logger (IXGLOGR) address space initialization</td>
<td>No</td>
</tr>
<tr>
<td>SYSOMVS</td>
<td>CTnBPXxx (see &quot;CTnBPXxx parmlib member&quot; on page 479)</td>
<td>CTIBPX00, which must be specified in BPXPRM00 member</td>
<td>Yes; minimal; unexpected events</td>
<td>No</td>
</tr>
<tr>
<td>SYSOPS</td>
<td>CTnOPSxx (see &quot;CTnOPSxx parmlib member&quot; on page 490)</td>
<td>CTIOPS00, which must be specified in CONSOLxx member</td>
<td>Yes; minimal; unexpected events</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 57. Determining if a component has a parmlib member (continued)

<table>
<thead>
<tr>
<th>Trace</th>
<th>Parmlib member</th>
<th>Default member</th>
<th>Default tracing beginning at initialization</th>
<th>Sublevel traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSRRS</td>
<td>CTnRRSxx (see “CTnRRSxx parmlib member” on page 495)</td>
<td>None, but member ATRCTRRS supplied in SYS1.SAMPLIB can be used. See SET-UP and ACTIVATION instructions in the ATRCTRRS sample.</td>
<td>Yes; minimal; unexpected events</td>
<td>No</td>
</tr>
<tr>
<td>SYSRSM</td>
<td>CTnRSMxx (see “CTnRSMxx parmlib member” on page 502)</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYSSPI</td>
<td>None</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYSTTRC</td>
<td>N/A</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYSVLF</td>
<td>None</td>
<td>N/A</td>
<td>Yes; minimal; unexpected events</td>
<td>No</td>
</tr>
<tr>
<td>SYSWLM</td>
<td>None</td>
<td>N/A</td>
<td>Yes; minimal; unexpected events</td>
<td>No</td>
</tr>
<tr>
<td>SYSXCF</td>
<td>CTnXCFxx (see “CTnXCFxx parmlib member” on page 526)</td>
<td>CTIXCF00, which can be specified in COUPLE00 member</td>
<td>Yes; minimal; unexpected events</td>
<td>No</td>
</tr>
<tr>
<td>SYSXES</td>
<td>CTnXESxx (see “CTnXESxx parmlib member” on page 533)</td>
<td>CTIXES00, which can be specified in COUPLE00 member</td>
<td>Yes; minimal; unexpected events</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Specify buffers

Each component determines the buffer size and how it is specified. Depending on the component, you may or may not be able to change the buffer size. You may be able to change the size only at system or component initialization, or when the trace is started, or at any time, including while the trace is running. Table 58 shows how the buffers specifications.

The buffer size determines whether you get all the records needed for diagnosis; when the buffer is full, the system wraps the buffer, overwriting the oldest records. To change the size of the buffer, specify an nnnnK or nnnnM parameter on the TRACE CT operator command or a BUFSIZE parameter in the parmlib member.

Usually you should increase the size of the trace buffer when you increase the amount of tracing. However, if you plan to place a component's trace records in a trace data set, you can probably leave the buffer at its original size. Many component traces do not allow you to change the buffer size after initialization; the table indicates those component traces. If you increase the amount of tracing for one of these traces, specify use of a trace data set, if the component supports its use.

Table 58. Component trace options

<table>
<thead>
<tr>
<th>Trace</th>
<th>Default size and size range</th>
<th>Size set by</th>
<th>Change size after IPL</th>
<th>Buffer location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSAPPCC</td>
<td>512KB</td>
<td>CTnAPPxx member or REPLY for TRACE CT command</td>
<td>Yes, while a trace is running</td>
<td>Data space. A TRACE CT,OFF command requests a dump, which includes the trace buffers</td>
</tr>
</tbody>
</table>
## Component Trace

Table 58. Component trace options (continued)

<table>
<thead>
<tr>
<th>Trace</th>
<th>Default size and size range</th>
<th>Size set by</th>
<th>Change size after IPL</th>
<th>Buffer location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSAXR</td>
<td>2MB</td>
<td>CTIAXRnn parmlib member or TRACE CT command</td>
<td>Yes, when restarting a trace after stopping it</td>
<td>AXR private; AXR trace dataspace</td>
</tr>
<tr>
<td></td>
<td>1MB - 2GB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSBPII</td>
<td>4M</td>
<td>MVS system</td>
<td>No</td>
<td>Data space. A SLIP or DUMP command can always be issued to capture the trace buffers for the BCPii address space. (Specify ASID=(BCPii's ASID),DSPNAME=('HWIBCPII.'*)) In addition, if CTrace for SYSBPII is ON, a Trace CT,OFF command requests a dump, which includes the trace buffers.</td>
</tr>
<tr>
<td></td>
<td>4M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSBHI</td>
<td>4MB</td>
<td>CTIBHxx parmlib member or TRACE CT command</td>
<td>Yes, while trace is running</td>
<td>64-bit Common Service Area (ECSA)</td>
</tr>
<tr>
<td></td>
<td>4MB - 64MB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSCSA</td>
<td>2MB</td>
<td>CTICEAnn parmlib member or TRACE CT command</td>
<td>Yes, when restarting a trace after stopping it</td>
<td>CEA private; CEA trace dataspace</td>
</tr>
<tr>
<td></td>
<td>1MB - 2GB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSDLF</td>
<td>N/A</td>
<td>MVS system</td>
<td>No</td>
<td>Data space. In the REPLY for the DUMP command, specify DSPNAME=('DLF.CC0FGSDO')</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSDSM</td>
<td>N/A</td>
<td>MVS system</td>
<td>No</td>
<td>Private address space</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSDUMP</td>
<td>4MB</td>
<td>CTIDMPxx parmlib member or TRACE CT command</td>
<td>Yes</td>
<td>64-bit common storage area</td>
</tr>
<tr>
<td></td>
<td>4MB - 32MB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Component Trace

### Table 58. Component trace options (continued)

<table>
<thead>
<tr>
<th>Trace</th>
<th>Default size and size range</th>
<th>Size set by</th>
<th>Change size after IPL</th>
<th>Buffer location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSGRS</td>
<td>16M 128KB - 2047MB (System rounds size up to nearest 64KB boundary.)</td>
<td>CTnGRSxx member</td>
<td>Yes</td>
<td>In the GRS address space above the bar which means it will not constrain GRS virtual storage. Options such as FLOW, REQUEST, and MONITOR produce a large number of entries in a short period of time. When dumping by SDUMP, specify ASID=GRS's asid and SDATA=(RGN,NUC). The RGN is needed for blocks that address the ctrace buffer. Note that SDATA=GRSQ does not collect GRS CTRACE.</td>
</tr>
<tr>
<td>SYSIEFAL</td>
<td>4M 256KB - 8MB</td>
<td>CTnIEFxx member</td>
<td>Yes</td>
<td>In the component address space</td>
</tr>
<tr>
<td>SYSIOS</td>
<td>324KB 324KB-1.5MB</td>
<td>CTnIOSxx member or REPLY for TRACE CT command</td>
<td>Yes</td>
<td>Extended system queue area (ESQA). Note: Full buffers are copied to an IOS data space to allow for more data capture. For information about specifying an IOS data space size at IPL, see &quot;OPTIONS parameter&quot; on page 454.</td>
</tr>
<tr>
<td>SYSJES</td>
<td>N/A</td>
<td>MVS system</td>
<td>No</td>
<td>In the component address space</td>
</tr>
<tr>
<td>SYSjes2</td>
<td>N/A</td>
<td>JES2</td>
<td>No</td>
<td>In the component address space</td>
</tr>
<tr>
<td>SYSLLA</td>
<td>N/A</td>
<td>MVS system</td>
<td>No</td>
<td>In the component address space</td>
</tr>
<tr>
<td>SYSLOGR</td>
<td>2MB 2MB - 2047MB</td>
<td>MVS system, CTnLOGxx member, or REPLY for TRACE CT command.</td>
<td>Yes</td>
<td>Data space. In the REPLY for the DUMP command, specify DSPNAME=('IXLOGR.*'). See &quot;Obtaining a dump of system logger information&quot; on page 471.</td>
</tr>
</tbody>
</table>
### Component Trace

Table 58. Component trace options (continued)

<table>
<thead>
<tr>
<th>Trace</th>
<th>Default size and size range</th>
<th>Size set by</th>
<th>Change size after IPL</th>
<th>Buffer location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSOMVS</td>
<td>4MB 16KB - 64MB</td>
<td>CTxBPXxx member or REPLY for TRACE CT command</td>
<td>Yes, when initializing z/OS UNIX.</td>
<td>Data space. In the REPLY for the DUMP command, specify DSPNAME=(asid.SYSZBPX2) where asid is the ASID for z/OS UNIX.</td>
</tr>
<tr>
<td>SYSOPS</td>
<td>2M 64KB - 16MB</td>
<td>CTnOPSxx member or REPLY for TRACE CT command</td>
<td>Yes, when restarting a trace after stopping it</td>
<td>Console address space (private).</td>
</tr>
<tr>
<td>SYSRRS</td>
<td>1MB 1MB - 2045MB</td>
<td>CTxRRSxx member or REPLY for TRACE CT command</td>
<td>Yes, when restarting a trace after stopping it</td>
<td>Data space and component address space. In the REPLY for the DUMP command, specify DSPNAME=('RRS_ATRTRACE) and SDATA=RGN.</td>
</tr>
<tr>
<td>SYSRSM</td>
<td>3 buffers of 132 pages</td>
<td>CTnRSMxx member or REPLY for TRACE CT command</td>
<td>Yes, when starting a trace</td>
<td>Common service area (LIKECSA) or, if specified in CTnRSMxx, high virtual private storage of the RASP address space.</td>
</tr>
<tr>
<td>SYSSPI</td>
<td>64KB</td>
<td>MVS system</td>
<td>Yes, when starting a trace</td>
<td>In the component address space</td>
</tr>
<tr>
<td>SYSTTRC</td>
<td>1 MB 16K - 999K 1MB - 32MB</td>
<td>MVS system</td>
<td>Yes</td>
<td>Data space owned by the system trace address space</td>
</tr>
<tr>
<td>SYSVLF</td>
<td>N/A</td>
<td>MVS system</td>
<td>No</td>
<td>Data space. Enter DISPLAY J,VLF to identify the VLF data spaces. In the REPLY for the DUMP command, specify DSPNAME=('VLF.Dclsname, VLF.Cclsname), where clsname is a VLF class name.</td>
</tr>
<tr>
<td>SYSWLM</td>
<td>64KB 64KB - 16M</td>
<td>MVS system</td>
<td>Yes, when starting a trace</td>
<td>Extended common service area (ECSA)</td>
</tr>
<tr>
<td>SYSXCF</td>
<td>4MB 16KB - 16MB (System rounds size up to a multiple of 72 bytes.)</td>
<td>CTnXCFxx member</td>
<td>No</td>
<td>Extended local system queue area (ELSAQ) of the XCF address space</td>
</tr>
</tbody>
</table>
Component Trace

Table 58. Component trace options (continued)

<table>
<thead>
<tr>
<th>Trace</th>
<th>Default size and size range</th>
<th>Size set by</th>
<th>Change size after IPL</th>
<th>Buffer location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSXES</td>
<td>336KB</td>
<td>CTnXESxx member or REPLY for TRACE CT command</td>
<td>Yes, while a trace is running.</td>
<td>64-bit common storage above the 2 GB bar. In the REPLY for the DUMP command, specify SDATA=XESDATA.</td>
</tr>
<tr>
<td></td>
<td>16KB - 16MB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Select the trace options for the component trace

If the IBM Support Center requests a trace, the Center might specify the options, if the component trace uses an OPTIONS parameter in its parmlib member or REPLY for the TRACE CT command. The options are:

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Request OPTIONS parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSAPPC</td>
<td>See “OPTIONS parameter” on page 401</td>
</tr>
<tr>
<td>SYSAXR</td>
<td>See “OPTIONS parameter” on page 414</td>
</tr>
<tr>
<td>SYSBCPII</td>
<td>See “OPTIONS parameter” on page 418</td>
</tr>
<tr>
<td>SYSBHI</td>
<td>See “OPTIONS parameter” on page 422</td>
</tr>
<tr>
<td>SYSCEA</td>
<td>See “OPTIONS parameter” on page 426</td>
</tr>
<tr>
<td>SYSDLF</td>
<td>None</td>
</tr>
<tr>
<td>SYSDOM</td>
<td>None</td>
</tr>
<tr>
<td>SYSDUMP</td>
<td>See “OPTIONS parameter” on page 426</td>
</tr>
<tr>
<td>SYSGRS</td>
<td>See “OPTIONS parameter” on page 438</td>
</tr>
<tr>
<td>SYSIEFAL</td>
<td>See “OPTIONS parameter” on page 447</td>
</tr>
<tr>
<td>SYSIOS</td>
<td>See “OPTIONS parameter” on page 454</td>
</tr>
<tr>
<td>SYSJES</td>
<td>None</td>
</tr>
<tr>
<td>SYSjes2</td>
<td>None</td>
</tr>
<tr>
<td>SYSLLA</td>
<td>None</td>
</tr>
<tr>
<td>SYSLOGR</td>
<td>See “OPTIONS parameter” on page 475</td>
</tr>
<tr>
<td>SYSOMVS</td>
<td>See “OPTIONS parameter” on page 479</td>
</tr>
<tr>
<td>SYSOPS</td>
<td>See “OPTIONS parameter” on page 491</td>
</tr>
<tr>
<td>SYSRRS</td>
<td>See “OPTIONS parameter” on page 496</td>
</tr>
<tr>
<td>SYSRSM</td>
<td>See “OPTIONS parameter” on page 504</td>
</tr>
<tr>
<td>SYSTTRC</td>
<td>None</td>
</tr>
<tr>
<td>SYSSPI</td>
<td>None</td>
</tr>
<tr>
<td>SYSVLF</td>
<td>None</td>
</tr>
<tr>
<td>SYSWLM</td>
<td>None</td>
</tr>
<tr>
<td>SYSXCF</td>
<td>See “OPTIONS parameter” on page 527</td>
</tr>
<tr>
<td>SYSXES</td>
<td>See “OPTIONS parameter” on page 534</td>
</tr>
</tbody>
</table>

You must specify all options you would like to have in effect when you start a trace. Options specified for a previous trace of the same component do not continue to be in effect when the trace is started again.
Component Trace

If the component has default tracing started at initialization by a parmlib member without an OPTIONS parameter, you can return to the default by doing one of the following:

- Stopping the tracing with a TRACE CT,OFF command.
- Specifying OPTIONS() in the REPLY for the TRACE CT command or in the CTnccxxx member.

For XCF, the IBM Support Center identifies the options needed to diagnose a particular problem as both of the following:

SERIAL
STATUS

Decide where to collect the trace records

As Table 59 shows, depending on the component, the potential locations of the trace data are:

- In address-space buffers, which are obtained in a dump
- In data-space buffers, which are obtained in a dump
- In a trace data set or sets, if supported by the component trace

Table 59. Location of trace buffers for components

<table>
<thead>
<tr>
<th>Component</th>
<th>Address-Space Buffer</th>
<th>Data-Space Buffer</th>
<th>Trace Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSAPPC</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYSAXR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSBCPII</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYSBHI</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSCEA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSDLF</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYSDSOM</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSDUMP</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSGRS</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSEEAL</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSIOS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSJES</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSjcs2</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYSLLA</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYSLOGR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSOMV</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSOPS</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSRRS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSRSM</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSTTRC</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYSSPI</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYSVLF</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYSWLM</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSXCF</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Table 59. Location of trace buffers for components (continued)

<table>
<thead>
<tr>
<th>Component</th>
<th>Address-Space Buffer</th>
<th>Data-Space Buffer</th>
<th>Trace Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSXES</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

If the trace records of the trace you want to run can be placed in more than one location, you need to select the location. For a component that supports trace data sets, you should choose trace data sets for the following reasons:

- Because you expect a large number of trace records
- To avoid interrupting processing with a dump of the trace data
- To keep the buffer size from limiting the amount of trace data
- To avoid increasing the buffer size

Depending on the component, you might also want to dump the address-space trace buffers and data-space trace buffers.

Note: You may need to consider the amount of auxiliary storage required to back data space buffers. In general, most components which use data space buffers establish a small default value less than 500 kilobytes of virtual storage. Some components allow you to specify values up to 2 gigabytes. The SYSIOS component trace uses a default of 512 megabytes for data space buffers. You should consider SYSIOS and other component data space buffers to ensure that the potential cumulative effect of all CTRACE data space buffers for your system can be accommodated by the local page data sets that you have allocated. For more information on auxiliary storage, refer to z/OS MVS Initialization and Tuning Guide.

Obtaining a component trace

To obtain a specific component trace, use one of the following procedures:

- “Request component tracing to address space or data space trace buffers”
- “Request writing component trace data to trace data sets” on page 388
- “Request component tracing for systems in a sysplex” on page 393

Request component tracing to address space or data space trace buffers

This topic describes how to obtain component trace records in dumps. The trace records are in address-space or data-space trace buffers. The topic contains information about how to:

- “Prepare for a specific component trace to trace buffers”
- “Perform component tracing to trace buffers” on page 387

Prepare for a specific component trace to trace buffers

The system programmer performs the tasks.

1. Select How the Operator Is to Request the Trace: For most component traces, the request is made by:

- A TRACE CT operator command without a PARM parameter, followed by a reply containing the options
- A TRACE CT operator command with a PARM parameter that specifies a CTnccxxx parmlib member containing the options
Component Trace

If you do not use a parmlib member, tell the operator the options.

2. Create a parmlib member, if used: If you use a parmlib member, create the member and place it on SYS1.PARMLIB. Use a parmlib member if the options are complicated and you have access to the SYS1.PARMLIB data set, or if a parmlib member is required by the component, or if you had already set up a parmlib member with the needed options. Use a REPLY for simple options. See “Create CTnccxx parmlib members for some components” on page 377. For XCF, for example, you can create CTWXCF03 to specify the options.

TRACEOPTS
ON
OPTIONS('SERIAL', 'STATUS')

3. Determine the dump to be used to obtain the trace records: Table 60 shows how to request SVC dumps for the component traces. Possible ways of requesting SVC dumps are:

- By a DUMP operator command
- By a SLIP trap
- By the component

For the following failures, use another type of dump:

- Failure of an application program or program product: The program requests a SYSMDUMP dump.
- The system waits, hangs, or enters a loop: The operator requests a stand-alone dump.

Table 60. How to request SVC dumps for component traces

<table>
<thead>
<tr>
<th>Trace</th>
<th>Request of SVC Dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSAPPC</td>
<td>By the component when the operator stops SYSAPPC tracing with a TRACE CT,OFF command</td>
</tr>
<tr>
<td>SYSAXR</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSBCPII</td>
<td>By DUMP or SLIP command, or by the component, if Ctrace is ON for SYSBCPII and a TRACE CT,OFF command is issued.</td>
</tr>
<tr>
<td>SYSBHI</td>
<td>By DUMP or SLIP command if the Basic HyperSwap Management address space or one of the BHIHSRV address spaces are to be included in the dump.</td>
</tr>
<tr>
<td>SYSCEA</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSDLF</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSDSOM</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSGRS</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSEIFAL</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSIOS</td>
<td>By DUMP or SLIP command, or by the component</td>
</tr>
<tr>
<td></td>
<td>• In the REPLY for the DUMP command, specify the IOS address space to be dumped</td>
</tr>
<tr>
<td>SYSJES</td>
<td>By the component</td>
</tr>
<tr>
<td>SYSjes2</td>
<td>By DUMP or SLIP command or component</td>
</tr>
<tr>
<td>SYSLLA</td>
<td>By the component</td>
</tr>
<tr>
<td>SYSLOGR</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSOMVS</td>
<td>By DUMP or SLIP command</td>
</tr>
</tbody>
</table>
Table 60. How to request SVC dumps for component traces (continued)

<table>
<thead>
<tr>
<th>Trace</th>
<th>Request of SVC Dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSOPS</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSRRS</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYSRRSM</td>
<td>• By DUMP or SLIP command&lt;br&gt;• Through the DMPREC option on the CTnRSMxx parmlib member or on the REPLY for the TRACE CT command when RSM enters recovery processing (default)&lt;br&gt;• Through the DMPOFF option of CTnRSMxx or the TRACE CT reply when SYSRSM tracing is turned off</td>
</tr>
<tr>
<td>SYSTTRC</td>
<td>Automatically dumped by the Tailored SVC Dump Exits function</td>
</tr>
<tr>
<td>SYSSPI</td>
<td>By the component</td>
</tr>
<tr>
<td>SYSVLF</td>
<td>By DUMP or SLIP command or when SYSVLF full tracing is turned off</td>
</tr>
<tr>
<td>SYSWLM</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYXCF</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>SYXES</td>
<td>By DUMP or SLIP command</td>
</tr>
</tbody>
</table>

4. Make Sure the component trace Buffers Will Be Dumped: The location of the address-space and data-space buffers depends on the component being traced. See the table in “Specify buffers” on page 379 for the location of the buffers. When the component being traced requests an SVC dump, the dump will contain the address-space and/or data-space trace buffers.

Perform component tracing to trace buffers
The operator performs the tasks. Note that these tasks are for a specific component trace, rather than for a trace started by the system at initialization.

1. Start the component trace: The operator enters a TRACE operator command on the console with MVS master authority. The operator replies with the options that you specified.

   In the following example, the TRACE CT command does not specify a parmlib member.

   trace ct, on, comp=sysxcf
   * 21 ITT006A ....
   r 21, options=(serial, status),end

   This next example requests the same trace using parmlib member CTWXCF03. When TRACE CT specifies a parmlib member, the system does not issue message ITT006A.

   trace ct, on, comp=sysxcf, parm=ctwxcf03

2. Verify that the Trace Is Running: See “Verifying component tracing” on page 395.

3. Obtain the Dump Containing the component trace Records: The operator obtains the dump that contains the component trace records: an SVC dump, a stand-alone dump, or a dump requested by the component when a problem occurs or when the operator stops the tracing.
This example shows a DUMP operator command entered on the console with MVS master authority. The system issues message IEE094D in response to the DUMP command. If you requested, the operator enters dump options in the reply to IEE094D. SDATA options are needed to obtain the trace buffers. An address space identifier (ASID) should be specified for the XCF address space; in the example, XCF is in address space 6.

dump comm=(dump for xcf component trace)
* 32 IEE094D ...
  r 32,sdata=(couple,sqa,lsqa),asid=6,end
  .
  .
IEA911E ...

The system identifies the data set containing the dump in message IEA911E. If an installation exit moves the dump, the operator should look for a message identifying the data set containing the moved dump and tell you the name of the dump and the data set containing it.

If desired, the operator can request more than one dump while a component trace is running.

4. **Stop the component trace:** The operator enters a TRACE CT,OFF command on the console with MVS master authority. For some component traces, the command requests a dump, which contains the trace records.

    The following example shows how to specify the TRACE CT,OFF command.
    
    trace ct,off,comp=sysxcf

### Request writing component trace data to trace data sets

The following topics describe only the component traces that can write to trace data sets. You can also change the trace data sets that are in use without stopping the trace. See “Change trace data sets” on page 393.

#### Prepare for a specific component trace to trace data sets

The system programmer performs the following tasks:

1. **Determine the dispatching priority required for the external writer started task, the server address space for the component’s trace:**

    While component trace runs under the master scheduler address space, you need to verify that the priority of the external writer is at least equal to, and preferably greater than the priority of the component being traced. For example, if you are tracing COMP(SYSXES) for JOBNMEX(IRLMA), the dispatching priority of the external writer should be equal to or greater than that assigned to IRLMA. See [z/OS MVS Initialization and Tuning Guide](https://www.ibm.com/support/docview.wss?uid=swg21122610) for more information on setting priorities.

2. **Select How the Operator Is to Request the Trace:**

    For most component traces, the request can be made by:
    
    - A TRACE CT operator command without a PARM parameter, followed by a reply containing the options
    - A TRACE CT operator command with a PARM parameter that specifies a CTnccxxx parmlib member containing the options

    If you do not use a parmlib member, tell the operator the options.
3. Create Source JCL for the External Writer:

Create source JCL to invoke an external writer, which will send the component trace output to one or more trace data sets. Add a procedure to the SYS1.PROCLIB system library or a job as a member of the data set in the IEFJOBS or IEFPDSI concatenation.

An external writer is not specific for a component but can be used by any application. So you can use the same source JCL again for other tracing later, if needed.

Concurrent traces for different components must use separate source JCL to place their traces in separate data sets.

Because the writer source JCL specifies data sets, use a different set of source JCL for each system in a sysplex. Several systems cannot share the same data set; attempting to share the same data set will lead to contention problems. If your sysplex uses a common SYS1.PROCLIB, you need to specify a unique writer procedure for each system or use a unique job as the source JCL, when tracing the same component on several systems.

The procedure shown in Figure 105 places trace data on two DASD data sets. The procedure is placed in member WTRD2 of SYS1.PROCLIB.

```
//WTDA2D2  PROC
//IEFPROC EXEC PGM=ITTTRCWR,REGION=32M
//TRCOUT01 DD DSNAM=SYS1.CTRACE1, VOL=SER=TRACE6,UNIT=DASD,
//     SPACE=(CYL,10), DISP=(NEW,KEEP),DSORG=PS
//TRCOUT02 DD DSNAM=SYS1.CTRACE2, VOL=SER=TRACE7,UNIT=DASD,
//     SPACE=(CYL,10), DISP=(NEW,KEEP),DSORG=PS
```

Figure 105. Example: Cataloged Procedure for an External Writer

Rules for the Source JCL for an External Writer:

- The name specified on the TRACE CT command or CTnccxxx parmlib member is the member name of the source JCL; in the preceding example, WTRD2. The name is 1 to 7 alphanumeric or national characters, with the first character alphabetic or national. National characters are represented by the following hexadecimal codes (in other languages, the codes represent different characters):

<table>
<thead>
<tr>
<th>Code</th>
<th>U.S. English EBCDIC Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'5B'</td>
<td>$</td>
</tr>
<tr>
<td>X'7B'</td>
<td>#</td>
</tr>
<tr>
<td>X'7C'</td>
<td>@</td>
</tr>
</tbody>
</table>

- The procedure must invoke the external writer program ITTTRCWR. Code the REGION= keyword on the EXEC statement to specify the maximum storage size required by the external writer.

- The source JCL can specify up to 16 trace data sets. The DD statements have ddnames of TRCOUTxx, where xx is 01 through 16.

- The trace data sets must be sequential data sets. You can use extended format sequential data sets as dump data sets for trace output. Extended format sequential data sets have the following features:
  - Have a greater capacity than sequential data sets
Component Trace

- Support striping
- Support compression

To help you manage the trace data sets, establish a naming convention so that the data set name indicates the component trace, the date, and so on.

All of the data sets must be on DASD or tape. Do not mix device classes, such as tape and DASD.

Within a device class, IBM recommends that you do not mix several types of devices, such as 3380 and 3390. In a mix of device types, the system would use the smallest block size for all the data sets.

Do not specify the following DCB parameters:
- BLKSIZE. The system uses the optimal block size, which is 4096 or larger.
- RECFM. The system uses VB.
- LRECL. The system uses BLKSIZE minus 4.

Do not specify DISP=SHR.

Do not concatenate trace data sets.

Use a separate member for each component’s trace, even though you can connect more than one trace to the same member.

Use the same member for all the sublevel traces for a component. This approach reduces the number of data sets you must manage.

Use a separate member for each system’s component trace, when a component trace runs in two or more of the systems of a sysplex. If the component traces specify the same cataloged procedure in a shared SYS1.PROCLIB, they will use the same data set or group of data sets; in this case, contention might develop for the data set or sets.

System security may require that you have RACF SYSHIGH authority to access the trace data sets.

z/OS V1R7 supports the following types of external media:
- DSNTYPE=LARGE data sets
- VSAM linear data sets

GTF and CTRACE accept a single VSAM linear data set as output. VSAM’s support for striping can increase data rate without the complexity associated with the use of distinct data sets.

For the details of the external data sets guidelines, see "Guidelines for defining GTF trace output data sets in a cataloged procedure" on page 236.

Wrapping DASD Trace Data Sets: If the WTRSTART parameter on the CTnccxx parmlib member or TRACE CT operator command specifies:

- WRAP or omits the parameter: When the system reaches the end of the data set or group of data sets, it writes over the oldest data at the start of the data set or first data set.

The system also uses only the primary extent or extents for the data set or sets. To obtain the maximum degree of control over the number of trace entries for which space will be allocated, specify space allocation in units of the BLKSIZE of your trace data set, no secondary space, and use the option for contiguous allocation. For example, if your BLKSIZE is 8192, code the SPACE keyword as follows:

\[ \text{SPACE=(8192, \{500,0\}, , CONTIG)} \]
Component Trace

- NOWRAP: When the data set or sets are full, the system stops writing trace records to the data set or sets. The system continues writing trace records in the address-space buffers.

The system also uses the primary and secondary extents of the data set or sets.

**Note**: Wrapping is not supported for Extended Format Sequential data sets, which are treated as NOWRAP even if WRAP is specified.

**Tape Data Sets**: CTRACE writes an end-of-file record. The tape is rewound and unloaded, then a new volume is mounted on the tape unit. If CTRACE has only one tape data set and only one unit for the data set, CTRACE does not write trace records while the tape is unavailable, thus losing trace data. CTRACE can write to multiple tape units in the way that multiple TRCOUTxx DD statements can specify tape data sets. When CTRACE fills one data set, it changes to the next data set.

**Note**: GTF and CTRACE support placement of NOWRAP traces in cylinder-managed space. WRAP traces placed in VSAM linear data sets can reside in cylinder-managed space too. WRAP traces in non-VSAM data sets cannot be placed in large format data sets, extended format data sets, or cylinder-managed space.

**Multiple Trace Data Sets**: Use multiple data sets to capture all the trace records, even during spikes of activity. For a SYSRSM trace, which typically produces large numbers of trace records, use multiple data sets to keep from losing records. Multiple trace data sets using different DASD devices can improve performance.

To view the trace records in chronological sequence, the system programmer can:
- Combine the trace records into one data set, using an IPCS COPYTRC subcommand, then use the CTRACE subcommand to format the records from the data set.
- Use an IPCS MERGE subcommand to format the records from multiple data sets.

The system places component trace records into each trace data set in sequence.

For example, for three data sets, the system places:
- Record 1 into data set 1
- Record 2 into data set 2
- Record 3 into data set 3
- Record 4 into data set 1
- Record 5 into data set 2
- And so on

**Lost Trace Data**: Ctrace will give an indication in the next successfully written record of any trace data that did not reach the output medium. If no further records are written, the following message is displayed when the external writer is stopped:

```
ITT120I SOME CTRACE DATA HAS BEEN LOST.
LAST nnn BUFFERS NOT WRITTEN.
```

**Create a parmlib member**

If you use a parmlib member, create the member and place it on SYS1.PARMLIB. Use a parmlib member if the options are complicated and you have access to the SYS1.PARMLIB data set, or if a parmlib member is required by the component, or if you had already set up a parmlib member with the needed options. Use a
Component Trace

REPLY for simple options. See "Create CTnccxx parmlib members for some components" on page 377 for more information. **Example: CTWXCF04 parmlib member** For XCF, create CTWXCF04. Notice the two statements for the writer; the WTRSTART statement starts the writer and the WTR statement connects the writer to the component.

```
TRACEOPTS
  WTRSTART(WTDASD2)
  ON
  WTR(WTDASD2)
  OPTIONS('SERIAL','STATUS')
```

**Perform component tracing to trace data sets**

The operator performs the tasks. Note that these tasks are for a specific component trace, rather than for a trace started by the system at initialization.

1. **Start the Writer and component trace:** The operator enters TRACE operator commands on the console with MVS master authority and replies with the options specified by the system programmer.

   **Example: TRACE CT command not specifying a parmlib member:** The second TRACE CT command starts the SYSXCF trace; the trace options were selected in a previous example. Notice the two writer operands; the WTRSTART operand starts the writer and the WTR operand connects the writer to the component.

   ```
   trace ct,wtrstart=wtdasd2
   trace ct,on,comp=sysxcf
   * 44 ITT006A ....
   r 44,wtr=wtdasd2,options=(serial,status),end
   ```

   **Example: TRACE CT command specifying a parmlib member:** This example requests the same trace using parmlib member CTWXCF04.

   ```
   trace ct,on,comp=sysxcf,parm=ctwxcf04
   ```

2. **Verify that the Trace and the Writer Are Running:** See "Verifying component tracing" on page 395.

3. **Stop the component trace:** The operator enters a TRACE CT command on the console with MVS master authority.

   **Example 1: TRACE CT,OFF Command**

   ```
   trace ct,off,comp=sysxcf
   * 56 ITT006A ....
   r 56,end
   ```

   **Example 2: TRACE CT Command to Disconnect the Writer:** To stop sending trace records to the trace data set, but keep the trace running, the operator can enter the following when the trace is currently running.

   ```
   trace ct,on,comp=sysxcf
   * 56 ITT006A ....
   r 56,wtr=disconnect,end
   ```

   The operator should stop the external writer.

4. **Stop the External Writer:** The operator enters a TRACE CT command on the console with MVS master authority.

   **Example: TRACE CT,WTRSTOP Command:**

   ```
   trace ct,wtrstop=wtdasd2
   ```
Change trace data sets
If you are running a component trace to a trace data set or sets, you can determine if you have the needed records without stopping the trace. Ask the operator to perform the following actions:
1. Enter a TRACE CT,WTRSTART command for a different set of source JCL for each external writer to trace data sets.
2. Enter a TRACE CT command that starts the trace with the different source JCL for the writer.

The new source JCL sends the trace records to the new data set or sets.

Note: You might lose one or more trace records.

You can view the previous data set or sets to verify the collected trace records, then continue or stop the trace, as needed.

Example: Changing the trace data sets
1. Enter the following command to start the new external writer NewWtr:
   TRACE CT,WTRSTART=NewWtr
2. Enter the following commands to disconnect SYSxxx CTRACE from the current writer CurrWtr:
   TRACE CT,ON,COMP=SYSxxx
   REPLY xx,WTR=DISCONNECT,END
3. Enter the following commands to connect the new external writer NewWtr to SYSxxx CTRACE:
   TRACE CT,ON,COMP=SYSxxx
   REPLY xx, WTR=NewWtr, END
4. Enter the following command to stop the current external writer CurrWtr:
   TRACE CT,WTRSTOP=CurrWtr

Request component tracing for systems in a sysplex
The following topics describe one way to obtain traces for a component on more than one system in a sysplex. The approach is to obtain a trace in the dump of each system and merge the traces from the dumps, using an IPCS MERGE subcommand. To be useful for diagnosis, the traces must cover the same time period and end at the same time.

You can also trace to data sets, if each system uses a unique source JCL for each external writer, so that the trace for each system goes to its own data set. If your installation has a shared SYS1.PROCLIB system library, use a unique parmlib member for each system; each unique parmlib member must specify a unique set of source JCL. If the source JCL is shared, all systems will write trace records on one data set, possibly causing contention problems.

Prepare for specific component traces on systems in a sysplex
The system programmer performs the tasks.

1. Create a Parmlib Member to Start the Traces: Create a parmlib member to start the traces of the component. Place the member in the shared SYS1.PARMLIB for the sysplex or in the parmlib for each system to be traced. If a parmlib member is used for each system, give it the same name so that one TRACE CT command can start all the component traces on the systems. See “Create CTnccxx parmlib members for some components” on page 377.
Component Trace

Example: CTWXCF33 to Start XCF Trace: For XCF, create CTWXCF33 to start the trace.

```
TRACEOPTS
ON
   OPTIONS('SERIAL','STATUS')
```

The directions for the task assume that a parmlib member can be used. If the component to be traced does not have a parmlib member, the operator can start it with a TRACE CT command in a ROUTE command. The operator has to enter a reply for each system. (The ROUTE command can be used only on MVS systems with JES2.)

2. Make Sure the component trace Buffers Will Be Dumped: The location of the address-space and data-space trace buffers depends on the component being traced. For XCF, the extended local system queue area (ELSQA) of the XCF address space contains the XCF component trace buffers.

Example: Obtaining XCF and XES Trace Buffers:
- For XCF, the operator should specify SQA and LSQA on the REPLY for the DUMP command.
- For XES, the operator should specify SDATA=(XESDATA).

Perform component tracing on the systems in the sysplex
The operator performs the tasks. Note that these tasks are for a specific component trace, rather than for a trace started by the system at initialization.

1. Start the component traces: On a console with MVS master authority on one system in the sysplex, the operator enters a ROUTE command containing a TRACE CT command. (The ROUTE command can be used only on MVS systems with JES2.)

The command specifies a parmlib member with the same name in each system being traced. Note that, if parmlib members are not specified, all systems issue message ITT006A to prompt for options. If the component to be traced does not have a parmlib member, specify the IBM-supplied CTIITT00 member to avoid the prompts.

Example 1: Command to start traces in all systems: The command starts the trace in all systems in the sysplex.

```
route *all,trace ct,on,comp=sysxcf,parm=ctwxcf33
```

Example 2: Command to start traces in some systems: The command starts the trace in a subset of systems. Both commands specify the CTWXCF33 parmlib member on each system being traced.

```
route subs2,trace ct,on,comp=sysxcf,parm=ctwxcf33
```

Example 3: Command for a component without a parmlib member: The following command turns on tracing for a SYSVLF trace in the systems of a sysplex, without prompts for replies to the TRACE command. The SYSVLF component trace has no parmlib member.

```
route *all,trace ct,on,comp=sysvlf,parm=ctiitt00
```

2. Verify that the traces Are running: See "Verifying component tracing" on page 395.
3. **Obtain the dumps containing the component trace records**: The operator requests an SVC dump for each system being traced.

**Example: DUMP command for systems in a sysplex**: The example shows the DUMP operator command entered on a console with MVS master authority on one system in the sysplex. The reply requests dumps on all of the systems named in the pattern of S*. The example assumes that the systems being traced have the following names: S1, S2, S3, and S4; any other systems in the sysplex have names that do not fit the pattern, such as B1 or T2.

```plaintext
dump comm=(dump for xcf component trace)
* 32 IEE094D ...
  r 32,remote=(syslist=(s*)),end
    ...
IEA911E ...
```

The system identifies the data set containing the dump in message IEA911E. If an exit moves a dump, the operator should look for a message identifying the data set containing the moved dump and tell you the name of the dump and the data set containing it.

4. **Stop the component traces**: On a console with MVS master authority on one system in the sysplex, the operator enters a ROUTE command containing a TRACE CT,OFF command to stop the traces. (The ROUTE command can be used only on MVS systems with JES2.)

**Example 1: Command to stop traces in all systems**: The command stops the traces in all systems in the sysplex.

```plaintext
route *all,trace ct,off,comp=sysxcf
```

**Example 2: Command to Stop Traces in Some Systems**: The command stops the traces in a subset of systems in the sysplex.

```plaintext
route subs2,trace ct,off,comp=sysxcf
```

### Verifying component tracing

The operator should do the following tasks after starting a component trace to make sure that it started successfully. How the task is done depends on whether the component trace has sublevels and whether an external writer is used.

#### Verify tracing for component traces without sublevels

The operator should do one of the following:

- Identify all current tracing by entering the following DISPLAY TRACE command on the console with MVS master authority. The response, in message IEE843I, gives the status in short form of all current component traces.

  ```plaintext
display trace
IEE843I ...
```

- Identify current tracing and the options for the traces by entering one of the following DISPLAY TRACE commands on the console with MVS master authority. The first command requests the status for all current component traces; the second command requests it for one component trace, such as XCF. The response, in message IEE843I, gives full information about the status.
Component Trace

display trace,comp=all
IEE843I ...

display trace,comp=sysxcf
IEE843I ...

Verify tracing for component traces with sublevels
The commands for verification depend on the component trace.

To verify a SYSJES component trace, the operator enters the following command to verify a SYSJES trace; the system will show the 4 sublevel traces.
display trace,comp=sysjes,sublevel,n=4

When a SYSXES component trace has multiple sublevel traces, a DISPLAY command shows only one sublevel. To verify a SYSXES component trace, the operator needs to enter multiple DISPLAY commands to see the multiple sublevels.

A SYSXES component trace has structures, address spaces, and connections. The following examples show the DISPLAY (D) command entered by the operator and the type of information that the system returns.

1. To see how the SYSXES component trace is set up.
   D TRACE,COMP=SYSXES
   IEE843I 15.24.40 TRACE DISPLAY 213
   SYSTEM STATUS INFORMATION
   ST=(ON,0064K,00128K) AS=ON BR=OFF EX=ON MT=(ON,024K)
   COMPONENT MODE BUFFER HEAD SUBS
   `-------------------------------
   SYSXES ON 0168K HEAD 2
   ASIDS *NOT SUPPORTED*
   JOBNAMES *NOT SUPPORTED*
   OPTIONS LOCKMGR
   WRITER *NONE*

2. To display the structure level trace for each structure and the number of subtraces available.
   D TRACE,COMP=SYSXES,SUB=(LT01),N=99
   IEE843I 15.25.00 TRACE DISPLAY 216
   SYSTEM STATUS INFORMATION
   ST=(ON,0064K,00128K) AS=ON BR=OFF EX=ON MT=(ON,024K)
   TRACENAME
   ========
   SYSXES
   MODE BUFFER HEAD SUBS
   =====================
   ON 0168K HEAD 2
   ASIDS *NOT SUPPORTED*
   JOBNAMES *NOT SUPPORTED*
   OPTIONS LOCKMGR
   WRITER *NONE*
   SUBTRACE MODE BUFFER HEAD SUBS
   `----------------------------------------------------------
   LT01 HEAD 1
   LIKEHEAD
   GLOBAL LIKEHEAD

3. To display the address space level trace for each structure. (The ASID specified is the asid in hex of the address space of the connector.)
Component Trace

4. To display the external writer and the buffer size and options associated with the connection.

D TRACE,COMP=SYSXES,SUB=(LT01.ASID(19).A1),N=99

IEE843I 15.25.56 TRACE DISPLAY 224
SYSTEM STATUS INFORMATION
ST=(ON,0064K,00128K) AS=ON BR=OFF EX=ON MT=(ON,024K)
TRACENAME
==========
SYSXES.LT01.ASID(0019)
MODE BUFFER HEAD SUBS
===================== 
ON 0168K HEAD 8
LIKEHEAD
ASIDS *NOT SUPPORTED*
JOBNAMES *NOT SUPPORTED*
OPTIONS LOCKMGR
WRITER *NONE*

Verify that the writer is active

If an external writer is being used, the operator should verify that the writer is active for the trace by entering one of the following DISPLAY TRACE commands on the console with MVS master authority. The first command requests writer status for all current component traces; the second command requests it for one writer by specifying the membername of the source JCL for the writer, such as WTDASD2. The response is in message IEE843I.

display trace,wtr=all
IEE843I ...
display trace,wtr=wtdasd2
IEE843I ...

The operator should verify that the source JCL for the writer in this display is the same as the source JCL for the writer that was started for the trace. If the membernames do not match, the component trace data is lost. The operator should
stop the writer job identified in the display and the component trace; then start the correct writer source JCL and start the trace again.

## Viewing the component trace data

During diagnosis, the system programmer performs the tasks, using IPCS. See **z/OS MVS IPCS Commands** for the COPYDUMP, COPYTRC, CTRACE, GTFTRACE, and MERGE subcommands.

1. **If your trace is in a dump in a SYS1.DUMPxx data set**, enter a COPYDUMP subcommand to move the dump to another data set. Use option 5.3 of the IPCS dialog to select the COPYDUMP subcommand.

2. **For all traces on trace data sets**, use a **COPYTRC subcommand** to reorder component trace records that are out of chronological sequence. Use option 5.3 of the IPCS dialog to select the COPYTRC subcommand.

3. **If your trace is on multiple data sets**, do one of the following to view the trace records in one chronological sequence, which is needed to understand what was happening when the problem occurred. The input data sets can be component trace data sets, SVC dumps, and stand-alone dumps.
   - Use the COPYTRC subcommand to combine the records on several data sets into a chronological sequence on one data set. Use this data set as input to the CTRACE subcommand, which formats the trace records. Use option 5.3 of the IPCS dialog to select the COPYTRC subcommand.
   - Use a MERGE subcommand to format trace records from one or more input data sets. MERGE lets you combine and format the following:
     - Component traces
     - GTF traces
     - Sublevel traces from one component on one trace data set
     - Sublevel traces from one component on separate trace data sets
   For sublevel traces, MERGE groups together the trace records for each sublevel. Use option 2.7 of the IPCS dialog to select the MERGE subcommand. MERGE allows you to issue individual CTRACE or GTFTRACE subcommands for each input data set.

4. **Use the subcommands in Table 61 when formatting the component trace records.** See **z/OS MVS IPCS Commands** for the SHORT, SUMMARY, FULL, and TALLY report type keywords and other keywords for the CTRACE subcommand.

<table>
<thead>
<tr>
<th>Trace</th>
<th>IPCS subcommand</th>
<th>CTRACE OPTIONS parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSAPPC</td>
<td>CTRACE COMP(SYSAPPC)</td>
<td>See “Formatting a SYSAPPC trace” on page 403</td>
</tr>
<tr>
<td>SYSAXR</td>
<td>CTRACE COMP(SYSAXR)</td>
<td>See “Formatting a SYSAXR trace” on page 415</td>
</tr>
<tr>
<td>SYSB CPII</td>
<td>CTRACE COMP(SYSB CPII)</td>
<td>See “SYSB CPII component trace” on page 417</td>
</tr>
<tr>
<td>SYSBHI</td>
<td>CTRACE COMP(SYSBHI)</td>
<td>See “SYSBHI component trace” on page 420</td>
</tr>
<tr>
<td>SYSCEA</td>
<td>CTRACE COMP(SYSCEA)</td>
<td>See “Formatting a SYSCEA trace” on page 427</td>
</tr>
<tr>
<td>SYSDL F</td>
<td>CTRACE COMP(SYSDL F)</td>
<td>None</td>
</tr>
</tbody>
</table>
### Component Trace

#### Table 61. Subcommands that format component trace records (continued)

<table>
<thead>
<tr>
<th>Trace</th>
<th>IPCS subcommand</th>
<th>CTRACE OPTIONS parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDOM</td>
<td>CTRACE COMP(SYSDOM)</td>
<td>See “SYSDOM component trace” on page 429</td>
</tr>
<tr>
<td>SYSDUMP</td>
<td>CTRACE COMP(SYSDUMP)</td>
<td>See “SYSDUMP component trace” on page 432</td>
</tr>
<tr>
<td>SYSGRS</td>
<td>CTRACE COMP(SYSGRS)</td>
<td>None</td>
</tr>
<tr>
<td>SYSHZS</td>
<td>CTRACE COMP(SYSHZS)</td>
<td>See “SYSHZS component trace” on page 441</td>
</tr>
<tr>
<td>SYSIEFAL</td>
<td>CTRACE COMP(SYSIEFAL)</td>
<td>None</td>
</tr>
<tr>
<td>SYSIOS</td>
<td>CTRACE COMP(SYSIOS)</td>
<td>See “Formatting a SYSIOS trace” on page 455</td>
</tr>
<tr>
<td>SYSJES</td>
<td>CTRACE COMP(SYSJES)</td>
<td>See “Formatting a SYSJES trace” on page 462</td>
</tr>
<tr>
<td>SYSjes2</td>
<td>CTRACE COMP(SYSjes2)</td>
<td>None</td>
</tr>
<tr>
<td>SYSLLA</td>
<td>CTRACE COMP(SYSLLA)</td>
<td>None</td>
</tr>
<tr>
<td>SYSLOGR</td>
<td>CTRACE COMP(SYSLOGR)</td>
<td>See “Formatting a SYSLOGR trace” on page 476</td>
</tr>
<tr>
<td>SYSOMVS</td>
<td>CTRACE COMP(SYSOMVS)</td>
<td>See “Formatting a SYSOMVS trace” on page 481</td>
</tr>
<tr>
<td>SYSOPS</td>
<td>CTRACE COMP(SYSOPS)</td>
<td>See “Formatting a SYSOPS trace” on page 492</td>
</tr>
<tr>
<td>SYSRRS</td>
<td>CTRACE COMP(SYSRRS)</td>
<td>See “Formatting a SYSRRS trace” on page 498</td>
</tr>
<tr>
<td>SYSSPI</td>
<td>CTRACE COMP(SYSSPI)</td>
<td>None</td>
</tr>
<tr>
<td>SYSTTRC</td>
<td>CTRACE COMP(SYSTTRC)</td>
<td>None</td>
</tr>
<tr>
<td>SYSVLF</td>
<td>CTRACE COMP(SYSVLF)</td>
<td>None</td>
</tr>
<tr>
<td>SYSWLM</td>
<td>CTRACE COMP(SYSWLM)</td>
<td>None</td>
</tr>
<tr>
<td>SYSXCF</td>
<td>CTRACE COMP(SYSXCF)</td>
<td>See “Formatting a SYSXCF trace” on page 529</td>
</tr>
<tr>
<td>SYSXES</td>
<td>CTRACE COMP(SYXSXES)</td>
<td>See “Formatting a SYSXES trace” on page 535</td>
</tr>
</tbody>
</table>

If some of the output in a combined or merged trace data set is for a GTF trace, use a GTFTRACE subcommand to format the GTF records and a CTRACE subcommand to format the component trace records. See Chapter 10, “The Generalized Trace Facility (GTF),” on page 229 for GTF tracing.

This example shows the CTRACE subcommand for a SYSXCF component trace, when the SERIAL and STATUS options are requested in the OPTIONS parameter.

```plaintext
ctrace comp(sysxcf) options((serial, status))
```

### SYSAPPC component trace

Table 62 summarizes information for requesting a SYSAPPC component trace for APPC/MVS.

#### Table 62. Requesting SYSAPPC component trace for APPC/MVS

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSAPPC:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTnAPPxx; there is no default member</td>
</tr>
<tr>
<td>Default tracing</td>
<td>No; cannot turn trace ON or OFF in CTnAPPxx</td>
</tr>
</tbody>
</table>


Component Trace

Table 62. Requesting SYSAPPC component trace for APPC/MVS (continued)

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSAPPC:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTnAPPxx or REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td></td>
</tr>
<tr>
<td>• Default: 32 MB</td>
<td></td>
</tr>
<tr>
<td>• Range: 64 KB - 256 MB</td>
<td></td>
</tr>
<tr>
<td>• Size set by: CTnAPPxx member or REPLY for TRACE command</td>
<td></td>
</tr>
<tr>
<td>• Change size after IPL: Yes, while a trace is running</td>
<td></td>
</tr>
<tr>
<td>• Location: In data space. A TRACE CT,OFF command requests a dump, which includes the trace buffers.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trace records location</th>
<th>Dataspace buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request of SVC dump</td>
<td>By the component when the operator stops SYSAPPC tracing with a TRACE CT,OFF command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSAPPC)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>NO</td>
</tr>
</tbody>
</table>


Requesting a SYSAPPC trace

Specify options for requesting a SYSAPPC component trace on a CTnAPPxx parmlib member or on the reply for a TRACE CT command.

CTnAPPxx parmlib member

You can specify the parameters listed in Table 63 on a CTnAPPxx parmlib member.

Table 63. CTnAPPxx parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnAPPxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>No</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>No</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>No</td>
</tr>
</tbody>
</table>

TRACE and REPLY commands

Table 64 and Table 65 on page 401 summarize the parameters you can specify on TRACE CT commands and a REPLY.

Table 64. Parameters allowed on TRACE CT

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, nnnnK, nnM, or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
</tbody>
</table>
Component Trace

Table 64. Parameters allowed on TRACE CT (continued)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 65. Parameters allowed on REPLY

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>No</td>
</tr>
</tbody>
</table>

**Automatic Dump:** The component requests an SVC dump when the operator stops the trace.

**OPTIONS parameter**

APPC trace request options are **hierarchical**. Figure 106 shows the hierarchy of SYSAPPC trace options. Each option traces its own events, plus all the events of the options below it. For example, if you specify the SCHEDULE trace option, the system also traces ENQWORK, DEQWORK, and ASMANAGE events.

---

**Figure 106. Hierarchy of SYSAPPC Component Trace Options.**

SYSAPPC tracing always includes all exception (error) events. If no trace options are specified, the trace output includes only the exception events.

If you do not know where the errors are occurring, use the GLOBAL trace option to catch the full range of APPC/MVS events. GLOBAL can slow performance, but you will catch the error in one re-create.
The values for the OPTIONS parameter for the CTnAPPxx parmlib member and reply for a TRACE command, in alphabetical order, are:

**ASMANAGE**
Traces events related to the creation and deletion of the APPC/MVS transaction scheduler's subordinate address space. ASMANAGE is a subset of SCHEDULE events.

**BADFMH5**
Traces events related to incorrect FMH-5s. BADFMH5 is a subset of FMH-5 events.

**CONVCLUP**
Traces events related to conversation cleanup. CONVCLUP is a subset of GLOBAL events.

**DEQWORK**
Traces the process of removing work requests from an APPC/MVS scheduler queue. DEQWORK is a subset of SCHEDULE events.

**ENQWORK**
Traces the process of adding work requests to an APPC/MVS scheduler queue. ENQWORK is a subset of SCHEDULE events.

**FMH5**
Traces FMH-5 events. FMH5 is a subset of INBOUND events.

**FMH7**
Traces FMH-7 events. FMH7 is a subset of INBOUND events.

**GLOBAL**
Traces the full range of APPC/MVS events.

**GLOBALQK**
Traces a subset of important GLOBAL trace events.

**INBOUND**
Traces inbound transaction processor (TP) requests. INBOUND is a subset of GLOBAL events.

**NODUMP**
Specifies no dumping when the operator stops the SYSAPPC component trace. Otherwise, component trace requests an SVC dump with the trace data when the operator stops tracing with a TRACE CT,OFF command.

IBM does not recommend the NODUMP option because the option makes obtaining the trace buffers difficult. The operator would have to identify the data space containing the buffers and specify it in a SLIP command or the reply for a DUMP command.

**PBI**
Traces events related to protocol boundary. PBI is a subset of VERBS events.

**RR**
Traces events related to the participation of APPC/MVS in resource recovery for protected conversations. RR is a subset of VERBS events.

**SERVER**
Traces events related to the APPC/MVS servers. SERVER is a subset of GLOBAL events.

**SCHEDULE**
Traces events related to the APPC/MVS transaction scheduler. SCHEDULE is a subset of GLOBAL events.
Component Trace

TRANSCH
Traces events related to APPC/MVS transaction scheduler interface support. TRANSCH is a subset of GLOBAL events.

USERID=(userid,userid)
Traces events for only the specified userid or userids. Specify the TSO/E userid of the person reporting a problem with an APPC/MVS application. Specify from 1 through 9 userids.

VERBS
Traces events related to outbound TPs or LU services. VERBS is a subset of GLOBAL events.

Examples of requesting SYSAPPC traces
This section contains examples of how to request SYSAPPC traces.

• Example 1: CTnAPPxx Member
The member requests SERVER and VERBS options for the address space or spaces for the TSO/E userid JOHNDOE.

  TRACEOPTS
  OPTIONS('SERVER','VERBS','USERID=(JOHNDOE)')

• Example 2: TRACE command specifying a parmlib member
The example specifies that options are to be obtained from the parmlib member CTWAPP03.

  trace ct,on,comp=sysappc,parm=ctwapp03

• Example 3: TRACE Command with Options Specified in a REPLY
The example requests the same trace as Example 2, but specifies all options in the REPLY.

  trace ct,on,comp=sysappc
  * 15 ITT006A ...
  reply 15,options=(server,verbs,userid=(johndoe)),end

• Example 4: TRACE Command Requesting GLOBAL Options
The example requests GLOBAL options for all address spaces using APPC/MVS.

  trace ct,on,comp=sysappc
  * 14 ITT006A ...
  reply 14,options=(global),end

Formatting a SYSAPPC trace
Format the trace with an IPCS CTRACE COMP(SYSAPPC) subcommand. Its OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records.

The options follow. The first option is either FILTER or CORRELATE, which are mutually exclusive; the first option controls how the other options select the records.

FILTER
The FILTER option selects the trace records that match only one of the specified options. The options that are valid with the FILTER option are:

• AQTOKEN
• CONVCOR
• CONVID
Component Trace

- FUNCID
- INSTNUM
- LUNAME
- LUWID
- NETNAME
- SEQNUM
- SESSID
- TPIDPRI
- TPIDSEC
- URID
- USERID

The formats of the OPTION parameter with FILTER are:

```
OPTION((FILTER,option))
OPTION((FILTER,option,option,...,option))
```

**CORRELATE**

The CORRELATE option selects the trace records that match a specified option and, for an unspecified option, uses the option's default values. The DEFAULTS keyword defines how default values are found for the unspecified options. The options that are valid with the CORRELATE option are:

- AQTOKEN
- CONVCOR
- CONVID
- DEFAULTS
- INSTNUM
- LUNAME
- LUWID
- NETNAME
- SEQNUM
- SESSID
- TPIDPRI
- TPIDSEC
- URID

The formats of the OPTION parameter with CORRELATE are:

```
OPTION((CORRELATE,option))
OPTION((CORRELATE,option,option,...,option))
```

**AQTOKEN**(allocate-queue-token)

Use with either the FILTER or CORRELATE option to specify an allocate queue token. The allocate-queue-token is an 8-byte hexadecimal string.

**CONVCOR**(conversation-correlator)

Use with either the FILTER or CORRELATE option to specify a conversation correlation. The conversation-correlator is an 8-byte hexadecimal string.
Component Trace

**CONVID(Conversation-id)**
Use with either the FILTER or CORRELATE option to specify a conversation identifier. The **conversation-id** is a 4-byte hexadecimal string.

**|DEFAULTS(NONEANYEXACT)**
Use only with the CORRELATE option to specify the values to be used for matching unspecified options.

- **NONE**
  Tells component trace to format only the trace records that match one or more of the specified options. **NONE** is the default.

- **ANY**
  Tells component trace to format:
  - Trace records matching one or more of the specified options.
  - Related trace records that match default values established for the unspecified options. Component trace derives the defaults from the values for unspecified options found in the first records that match **any** of the specified options.

- **EXACT**
  Tells component trace to format:
  - Trace records matching one or more of the specified options.
  - Related trace records matching default values established for the unspecified options. Component trace derives the defaults from the values for unspecified options found in the first records that match **all** of the specified options.

**FUNCID(Function-id)**
Use only with the FILTER option to specify the APPC/MVS subcomponent trace records to format. Specify one **function-id**:

- **01** Recovery
- **02** Verb services
- **03** FMH-5 manager
- **04** Conversation manager
- **05** System data file manager (SDFM)
- **06** VTAM exits
- **07** LU manager
- **08** State machine
- **09** Test enablement
- **10** APPC/MVS scheduler (ASCH)
- **11** Transaction scheduler interface
- **12** Allocate queue services

**INSTNUM(Instance-number)**
Use with either the FILTER or CORRELATE option to specify the instance number for a logical unit of work. The **instance-number** is a 6-byte hexadecimal string.
**Component Trace**

**LUNAME**(local-luname)
Use with either the FILTER or CORRELATE option to specify the LU name for the local logical unit of work. The local-luname is an 8-byte EBCDIC character string.

**LUWID**(logical-unit-of-work-id)
Use either the FILTER or CORRELATE option to specify a logical unit of work identifier, which represents the processing a program performs from one sync point to the next. To specify the logical-unit-of-work-id, enter the hexadecimal string as it appears in the CTRACE report, without including blank spaces.

**NETNAME**(network-name)
Use with either the FILTER or CORRELATE option to specify the network name for a logical unit of work. The network-name is an 8-byte EBCDIC character string, which is the same as the network-ID portion of a network-qualified LU name.

**SEQNUM**(sequence-number)
Use with either the FILTER or CORRELATE option to specify the sequence number for a logical unit of work. The sequence-number is a 2-byte hexadecimal string.

**SESSID**(session-id)
Use with either the FILTER or CORRELATE option to specify the session identifier. The session-id is an 8-byte hexadecimal string.

**TPIDPRI**(tp-id)
Use with either the FILTER or CORRELATE option to specify the primary TP identifier. The tp-id is an 8-byte hexadecimal string.

**TPIDSEC**(tp-id)
Use with either the FILTER or CORRELATE option to specify the secondary TP identifier, which is used for multi-trans TPs. The tp-id is an 8-byte hexadecimal string.

**URID**(unit-of-recovery-id)
Use with either the FILTER or CORRELATE option to specify a unit of recovery identifier, which represents part of a TP's processing for a protected conversation. The unit-of-recovery-id is a 32-byte hexadecimal string.

**USERID**(userid)
Use only with the FILTER option to specify a userid as a filter. The userid is an 8-byte EBCDIC character string.

**Examples of subcommands to format a SYSAPPC trace**

- **Example 1:** CTRACE subcommand to view all trace entries
  To view all the SYSAPPC trace records, enter:
  
  CTRACE COMP(SYSAPPC)

- **Example 2:** CTRACE Subcommand to view exception entries
  To format abnormal SYSAPPC events, such as abends or VTAM return codes, enter:
  
  CTRACE COMP(SYSAPPC) EXCEPTION

- **Example 3:** CTRACE subcommand for subcomponent
  To format all the records for one APPC/MVS subcomponent, enter the following subcommand. Use this subcommand to locate an error if you have narrowed the problem down to one subcomponent.
  
  CTRACE COMP(SYSAPPC) OPTIONS((FILTER,Funcid(nn)))

- **Example 4:** CTRACE subcommand to view a userid’s entries
To format all the records for userid JOHNDOE, who is experiencing problems, enter the following subcommand. If you specified the USERID option when requesting the trace, this formatting option is redundant.

```
CTRACE COMP(SYSAPPC) OPTIONS((FILTER,USERID(JOHNDOE)))
```

### Output from a SYSAPPC Trace

The following topics contain examples of different types of output produced by a SYSAPPC trace.

**CTRACE COMP(SYSAPPC) SHORT subcommand output**

The SHORT parameter shows one line of output for each trace record. Figure 107 shows an example of SYSAPPC component trace output formatted with the SHORT parameter.

```
VEFMH5XT 00004101 155705.182844 VEFMH-5 RECEIVED
VEFMH5ER 00000101 155705.367233 VEFMH-5 IN TPEND
```

*Figure 107. CTRACE COMP(SYSAPPC) SHORT subcommand output*

The fields in each SHORT report line are:

**Mnemonic**
- For example, VEFMH5XT.

**Entry ID**
- The identifier for the trace record. For example, 00004101.

**Time**
- The time in hh:mm:ss.tttttt format. For example, 15:57:05.182844.

**Title**
- The title of the record. For example, VEFMH-5 RECEIVED. Each title begins with a prefix that indicates the APPC/MVS subcomponent that wrote the trace record. For example, VE, which represents the VTAM exits subcomponent. Table 66 relates the title prefixes to their APPC/MVS subcomponents.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Subcomponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMI</td>
<td>Verb services</td>
</tr>
<tr>
<td>ASCH</td>
<td>APPC/MVS scheduler (ASCH)</td>
</tr>
<tr>
<td>CM</td>
<td>Conversation manager</td>
</tr>
<tr>
<td>ERROR</td>
<td>Recovery</td>
</tr>
<tr>
<td>FMH5</td>
<td>FMH-5 manager</td>
</tr>
<tr>
<td>LUM</td>
<td>LU manager</td>
</tr>
<tr>
<td>PC</td>
<td>Protected conversations</td>
</tr>
<tr>
<td>SDFM</td>
<td>MVS system data file manager (SDFM)</td>
</tr>
<tr>
<td>SF</td>
<td>Allocate queue services</td>
</tr>
<tr>
<td>SM</td>
<td>State machine</td>
</tr>
<tr>
<td>TE</td>
<td>Test enablement</td>
</tr>
<tr>
<td>TSI</td>
<td>Transaction scheduler interface</td>
</tr>
<tr>
<td>VC</td>
<td>Verb services</td>
</tr>
<tr>
<td>VE</td>
<td>VTAM exits</td>
</tr>
<tr>
<td>VS</td>
<td>Verb services</td>
</tr>
</tbody>
</table>

Table 66. Summary of the title prefixes and APPC/MVS subcomponents
Component Trace

CTRACE COMP(SYSAPPC) SUMMARY subcommand output

The SUMMARY parameter gives the line in the SHORT report and most fields in each trace record. An example of SYSAPPC component trace output formatted with the SUMMARY parameter follows in Figure 108.

```
SY1  PCESC  00007802  13:07:29.491950  PC:ENTRY STATE CHECK EXIT
  FUNCID...  02
  USERID... IBMUSER  JOBNAME.. APPC
  ASIDHOME.. 001C  ASIDPRI.. 001C
  TPIDPRI.. 00000000  TPIDSEC.. 00000000
  SESSID.. E723E663  AAB04BF0  CONVID.. 01000814
  CONVCOR.. 063313F8  00000000  AQTOKEN.. 00000000
  LUWID... 1064E2C9  C2DE4F90  4BE9F9C3  F0C1D7F0  F36FB02A  C0220700  01
  NETNAME.. USIBMZ0  LUNAME... Z0C0AP03
  INSTNUM.. 6FD82AC0  2207  SEQNUM.. 0001
  URID..... AD3FDB7  7EEF8000  00000007  01010000
```

Figure 108. CTRACE COMP(SYSAPPC) SUMMARY subcommand output

The fields in the SUMMARY report, after the first line, follow. See the SHORT report for the first line.

**FUNCID**

An identifier of the APPC/MVS subcomponent that wrote the trace record. See the FUNCID option for the identifiers.

**USERID**

The system was processing work for this userid when the trace record event occurred.

**JOBNAME**

The name of the job that the system was processing when the trace record event occurred.

**ASIDHOME**

The address space identifier (ASID) of the primary address space the system was processing when the trace record event occurred.

**TPIDPRI**

The TP identifier of a primary TP. (Multitrans TPs have a primary and a secondary TP)

**TPIDSEC**

The TP identifier for a secondary TP. (Multitrans TPs have a primary and a secondary TP)

**SESSID**

The identifier for a session.

**CONVID**

The identifier for a conversation.

**AQTOKEN**

The identifier for an allocate queue.

**LUWID**

The identifier for a logical unit of work. The following fields refer to the logical unit of work: If the LUWID is either all zeros or not valid,* the fields contain asterisks (•).

**NETNAME**

The network name for the logical unit of work.
Component Trace

**LUNAME**
The name of the local LU.

**INSTNUM**
The instance number for the logical unit of work.

**SEQNUM**
The sequence number for the logical unit of work.

**URID**
The identifier for a unit of recovery.

**CTRACE COMP(SYSAPPC) FULL subcommand output**
The FULL parameter gives all the data in the trace records. It contains the line in the SHORT report, the fields in the SUMMARY report, and KEY and ADDR fields. An example of SYSAPPC component trace output formatted with the FULL parameter follows in Figure 109.

See the SHORT report and the SUMMARY report for the fields in the report. IBM might need the KEY and ADDR fields for diagnosis.

**FMH-5 trace data**
FMH-5 trace records contain information useful for tracking TP flow and diagnosing the following types of problems:
- Persistent verification problems
- Password maintenance problems
- APPC/MVS security problems

To obtain FMH-5 data, request the SYSAPPC component trace with an FMH5, INBOUND, or GLOBAL option. To isolate the FMH-5 records in the trace output, enter the following IPCS subcommand:
Table 67 gives the mnemonic and title for each FMH-5 trace record and explains the record. Most of the trace records have FMH-5 itself formatted in KEY field X'0012'.

For the format of the FMH-5, see:
- [z/OS Communications Server: SNA Programmer’s LU 6.2 Guide](#)
- [z/OS Communications Server: SNA Programmer’s LU 6.2 Reference](#)

### Table 67. FMH-5 trace entries in the SYSAPPC component trace

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Title</th>
<th>Description/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMH5BDSC</td>
<td>FMH5:BAD SECURITY COMBINATION</td>
<td>APPC/MVS found an incorrect security option or security subfields or both. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5ERCV</td>
<td>FMH5:FMH-5 RECEIVE FAILURE</td>
<td>An FMH-5 was not successfully received by the local MVS LU. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5INCD</td>
<td>FMH5:FMH-5 COMMAND IS NOT VALID</td>
<td>APPC/MVS detected an incorrect FMH-5 command. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5LUNA</td>
<td>FMH5:LU IS NOT ACTIVE</td>
<td>An LU is not active. See the LUNAME field in the trace output. Enter a DISPLAY APPC command to find the status of this LU.</td>
</tr>
<tr>
<td>FMH5NOTP</td>
<td>FMH5:TP NAME IS NOT RECOGNIZED</td>
<td>The TP name was not specified correctly in the FMH-5.</td>
</tr>
<tr>
<td>FMH5NSCH</td>
<td>FMH5:NOT SERVED AND NO SCHEDULER</td>
<td>The TP cannot be scheduled because no scheduler is associated with the LU.</td>
</tr>
<tr>
<td>FMH5PFST</td>
<td>FMH5:FMH-5 PROFILE IS NOT VALID</td>
<td>The FMH-5 profile is incorrect; it is greater than 8 characters.</td>
</tr>
<tr>
<td>FMH5PIP</td>
<td>FMH5:PIP DATA PRESENT IN FMH-5</td>
<td>APPC/MVS found profile initialization parameters (PIP) data in the FMH-5; PIP data is not valid in FMH-5 for APPC/MVS.</td>
</tr>
<tr>
<td>FMH5PWCC</td>
<td>FMH5:PW CONV CLEANUP FAILED</td>
<td>Internal error. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5PWDE</td>
<td>FMH5:PW DEALLOCATE FAILED</td>
<td>Internal error. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5WDF</td>
<td>FMH5:PW DEQUE REQUEST FAILED</td>
<td>An attempt to attach the SIGNON/Change password TP failed. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5PWQF</td>
<td>FMH5:PW QUEUE REQUEST FAILED</td>
<td>Internal error. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5PWRF</td>
<td>FMH5:QW RACF REQUEST REJECTED</td>
<td>Internal error. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5PWR1</td>
<td>FMH5:PW RECEIVE DATA FAILED 1</td>
<td>The __SIGNON/Change password TP attempted to perform a ReceiveandWait call for a GDS variable. See the following KEY fields:</td>
</tr>
<tr>
<td>FMH5PWR2</td>
<td>FMH5:PW RECEIVE DATA FAILED 2</td>
<td><strong>KEY X'007E'</strong> contains the status received__ value returned to the SIGNON/Change password TP by ReceiveandWait.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>KEY X'007F'</strong> contains the data received value__ returned to the SIGNON/Change password TP by ReceiveandWait.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>KEY X'003F'</strong> contains the return code from__ ReceiveandWait.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Make sure that your GDS variable was sent correctly. If you cannot resolve the problem, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>
Table 67. FMH-5 trace entries in the SYSAPPc component trace (continued)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Title</th>
<th>Description/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMH5PWSD</td>
<td>FMH5:PW SEND DATA FAILED</td>
<td>A SIGNON/Change password TP SendData call failed. Verify that your TP has a valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>conversation established with the SIGNON/Change password TP. If you cannot resolve</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the problem, contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5PWSF</td>
<td>FMH5:PW SEND MESSAGE FAILED</td>
<td>A persistent verification signoff flow to the partner LU failed. The __SIGNEDONTO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>list in the partner LU may not be in sync with the local __SIGNEDONFROM list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the following KEY fields:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KEY X'0026' contains the TCB address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KEY X'001A' contains the name of the partner LU.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Key X'002F' contains the userid of the user whose SIGNOFF failed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you cannot resolve the problem, contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5PWSM</td>
<td>FMH5:PW SEND MESSAGE</td>
<td>APPC/MVS could not attach the X'30F0F5F2' expired password notification program to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>notify a partner system user that the user's password expired. See the following KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fields:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KEY X'0026' contains the TCB address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KEY X'001A' contains the name of the partner LU.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• KEY X'002F' contains the USERID of the user whose attach request failed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you cannot resolve the problem, contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5PWSR</td>
<td>FMH5:PW SIF RESERVE FAILURE</td>
<td>Internal error. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5PWST</td>
<td>FMH5:FMH-5 PASSWORD IS NOT VALID</td>
<td>The FMH-5 password is incorrect; it is greater than 8 characters.</td>
</tr>
<tr>
<td>FMH5QMFL</td>
<td>FMH5:FMFP QUEUE MANAGER FAILURE</td>
<td>Internal error. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>FMH5RECV</td>
<td>FMH5:FMH-5 SUCCESSFULLY RECEIVED</td>
<td>An FMH-5 was successfully received by the local MVS LU.</td>
</tr>
</tbody>
</table>
### Component Trace

**Table 67. FMH-5 trace entries in the SYSAPEPC component trace (continued)**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Title</th>
<th>Description/Action</th>
</tr>
</thead>
</table>
| FMH5RFRJ   | FMH5:RACF REQUEST REJECTED     | The system received a bad return code from one of the RACF services. See KEY X'0053' for a code identifying the RACF service that failed. The code can be one of the following:  
  1. RACROUTE REQUEST=VERIFY  
  2. RACROUTE REQUEST=SIGNON TYPE=SIGNIN  
  3. RACROUTE REQUEST=SIGNON TYPE=QSIGNON  
  4. RACROUTE REQUEST=SIGNON TYPE=SIGNOFF  
See the following KEY fields:  
• KEY X'0054' contains the return code for the RACF service request.  
• KEY X'0055' contains the reason code for the RACF service request.  
• KEY X'0021' contains the security authorization facility (SAF) return code for the service. |
| FMH5SERF   | FMH5:APPC/MVS SERVICE FAILURE   | APPC/MVS internal failure. Contact the IBM Support Center.                        |
| FMH5SFAL   | FMH5:SEND MESSAGE FAILED        | Persistent verification signoff flow to the partner LU failed. Make sure you have valid sessions established. See the following KEY fields:  
• KEY X'0026' contains the TCB address.  
• KEY X'001A' contains the name of the partner LU. |
| FMH5SOFF   | FMH5:SIGNOFF FLOW               | Persistent verification signoff flow to the partner LU completed. See the following KEY fields:  
• KEY X'0026' contains the TCB address.  
• KEY X'001A' contains the name of the partner LU. |
| FMH5SVC    | FMH5:ACCEPTED BY SRVR FACILITIES| APPC/MVS placed the inbound request on an allocate queue to await later processing by an APPC/MVS server. |
| FMH5TEST   | FMH5:FMH5 ACCEPTED FOR TESTING  | An FMH-5 is accepted for testing.                                                 |
| FMH5TPAD   | FMH5:TP PROFILE ACCESS DENIED   | TP profile access denied. Request=AUTH failed.                                    |
| FMH5TPNA   | FMH5:TP PROFILE IS NOT ACTIVE   | The TP profile is not active. Get the TP name from the FMH-5 formatted at KEY X'0012' in this trace record. Then use the SDFM utility to look at the TP profile. |
| FMH5TPRQ   | FMH5:TP PROFILE IS REQUIRED     | The system found no TP profile for the requested TP. The scheduler associated with the TP requires a TP profile. The error is probably due to an SDFM problem. Look for trace records with a prefix of SDFM. |
| FMH5U1ST   | FMH5:FMH-5 USERID IS NOT VALID  | The FMH-5 user ID is incorrect; it is greater than 8 characters.                 |
| FMH5VALID  | FMH5:FMH5 SUCCESSFULLY VALIDATED| An FMH-5 has been successfully validated.                                        |
| FMH5XLNFE  | FMH5:EXCHANGE LOG NAME FAILED   | APPC/MVS rejected the protected conversation because required log-name exchange processing did not occur. |
### Component Trace

Table 67. FMH-5 trace entries in the SYSAPPC component trace  (continued)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Title</th>
<th>Description/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMANFAIL</td>
<td>FMH5:FMAX QUEUE MANAGER FAILURE</td>
<td>Internal error. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>RESVFAIL</td>
<td>FMH5:SIF RESERVE FAILURE</td>
<td>Internal error. Contact the IBM Support Center.</td>
</tr>
</tbody>
</table>

**SYSAXR component trace**

Before using this component trace, ensure that you have read:
- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSAXR component trace for System REXX component.

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSAXR:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTIAXRnn. Default member: CTIAXR00</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; error; error events</td>
</tr>
<tr>
<td>Trace request OPTIONS</td>
<td>In CTIAXRxx and REPLY for TRACE command</td>
</tr>
<tr>
<td>parameter</td>
<td></td>
</tr>
<tr>
<td>Buffer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 2MB</td>
</tr>
<tr>
<td></td>
<td>Range: 1MB - 2GB</td>
</tr>
<tr>
<td></td>
<td>Size set by: CTIAXRnn parmlib member or REPLY to TRACE CT command</td>
</tr>
<tr>
<td></td>
<td>Change size after IPL: Yes, when restarting a trace after stopping it</td>
</tr>
<tr>
<td></td>
<td>Location: System REXX trace data space.</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer; System REXX trace data space; trace data set</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSAXR)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Requesting a SYSAXR trace

Specify options for requesting a SYSAXR component trace in a CTIAXRxx parmlib member or on the reply for a TRACE CT command. Changing SYSAXR trace options after AXR has started requires stopping and restarting the trace.

**CTIAXRnn parmlib member**

The following table indicates the parameters you can specify in a CTIAXRnn parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTIAXRnn?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>No</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Component Trace

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTIAXRnn?</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>MOD</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note:** The buffer size can be changed after IPL. To become effective the System REXX address space (AXR) must be restarted. Specify the new buffer size in the BUFSIZE parameter in the CTIAXRnn member being used.

The IBM supplied CTIAXR00 parmlib member initializes error tracing as soon as the System REXX address space starts. The contents of CTIAXR00 are:

```
TRACEOPTS
ON
OPTIONS('ERROR')
BUFSIZE(2M)
```

**TRACE and REPLY commands**
The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, OFF or nnnnM</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**OPTIONS parameter**
The values for the OPTIONS parameter in the CTIAXRxx parmlib member and reply for a TRACE command are:

**ALL**
- Trace everything.
Component Trace

**AXRCMD**
Trace Command function package events.

**AXRMLWTO**
Trace multiline WTO function package events.

**AXRWTO**
Trace WTO function package events.

**AXRWAIT**
Trace wait function package events.

**AXRINFO**
Trace information function package events.

**GETRXLIB**
Trace AXREXX REQUEST=GETREXXLIB events.

**CANCEL**
Trace AXREXX REQUEST=CANCEL events.

**COMMAND**
Trace System REXX command events.

**ERROR**
Trace error events.

**EXEC**
Traces only events that occur under the specified exec name.

**REXXARGS**
Trace all events associated with REXX arguments.

**REXXVARS**
Trace all events associated with REXX variables.

**RXCLIENT**
Trace events that occur under the invoker of AXREXX.

**RXSERVER**
Trace all server events.

**Formatting a SYSAXR trace**
Format the trace with an IPCS CTRACE COMP(SYSAXR) subcommand.

**Output from a SYSAXR Variables Trace**
Each trace record has a header associated with it, as Figure 110 on page 416 shows. This header is consistent among ALL trace records, although not all fields are filled in (for example, there is no TracePrefixReqToken for command processing).

**Note:** This is NOT an interface and should only be used for diagnostic purposes.
The following shows the formatted IPCS output produced from the CTRACE COMP(SYSAXR) subcommand after running a REXX exec invoked using the AXREXX programming interface with the CTRACE REXXVARS option enabled.

SY1 REXXVARS 04130D06 20:40:59.969289 REXX VAR NAME

00330032 C1E7D904 C1C9D540 C1E7D904 | ....AXRMAIN AXRM |
C1C9D540 006EAEB8 C8C1D909 C9E2E540 | AIN .>.hHARRISV |
00320000 00050000 00000001 BFBC7970 | .......... |
95690DB8 00000001 D4E8E5C1 D94BF1 | n.......MYVAR.1 |

The 1st 4 bytes following the header contains the index of the variable in the variable list (AXRArgLst). The remainder of the trace entry contains the value of the variable name.

SY1 REXXVARS 04130D02 20:40:59.969292 REXX VAR BEFORE EXEC

00330032 C1E7D904 C1C9D540 C1E7D904 | ....AXRMAIN AXRM |
C1C9D540 006EAEB8 C8C1D909 C9E2E540 | AIN .>.hHARRISV |
00320000 00050000 00000001 BFBC7970 | .......... |
95690DB8 00000001 0000000B 40404040 | n........... |
40404040 4040F1 | 1 |

The 1st 4 bytes following the header contains the index of the variable in the variable list (AXRArgLst). The next 4 bytes contain the length of the value. The remainder contains the value of the variable on input to the exec.

SY1 REXXVARS 04140D04 22:01:15.525516 REXX VAR AFTER EXEC

00150032 C1E7D904 C1C9D540 C1E7D904 | ....AXRMAIN AXRM |
C1C9D540 006EAEB8 C8C1D909 C9E2E540 | AIN .>.hHARRISV |
00320000 00050000 00000001 BFBC7970 | .......... |
09AB884A 00000001 00000003 F1F0F1 | .yd!........101 |

The 1st 4 bytes following the header contains the index of the variable in the variable list (AXRArgLst). The next 4 bytes should contain the length of the output. The next set of bytes should contain the output variable. If the result was truncated it will contain the truncated result.

The input/output variable contained 1 on entry to the exec and its final value when the exec completed was 101.
SYSBCPII component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following table summarizes information for requesting a SYSBCPII component trace for base control program internal interface (BCPii).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSBCPII:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTIHWI00</td>
</tr>
<tr>
<td>Default and only member:</td>
<td>CTIHWI00. If no valid CTIHWI00 member exists,</td>
</tr>
<tr>
<td></td>
<td>minimal tracing is activated at BCPii address</td>
</tr>
<tr>
<td></td>
<td>space initialization.</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Minimal tracing is always in effect for SYSBCPII.</td>
</tr>
<tr>
<td></td>
<td>If no valid CTIHWI00 member exists, if CTrace is</td>
</tr>
<tr>
<td></td>
<td>turned OFF, or if CTrace is ON with no format</td>
</tr>
<tr>
<td></td>
<td>OPTIONS specified, minimal tracing occurs.</td>
</tr>
<tr>
<td>Trace request OPTIONS</td>
<td>In CTIHWI00 and REPLY for TRACE command</td>
</tr>
<tr>
<td>parameter</td>
<td></td>
</tr>
<tr>
<td>Buffer</td>
<td>• Size: 4M</td>
</tr>
<tr>
<td></td>
<td>• Size set by: BCPii address space</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: No</td>
</tr>
<tr>
<td></td>
<td>• Location: In the component data space</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Data space buffer</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command, or by the component if</td>
</tr>
<tr>
<td></td>
<td>trace is ON for SYSBCPII and a TRACE CT,OFF</td>
</tr>
<tr>
<td></td>
<td>command is issued.</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSBCPII)</td>
</tr>
<tr>
<td>Trace format OPTIONS</td>
<td>MIN, ALL</td>
</tr>
<tr>
<td>parameter</td>
<td></td>
</tr>
</tbody>
</table>

### Requesting a SYSBCPII trace

Specify options for requesting a SYSBCPII component trace in a CTIHWI00 parmlib member or on a reply to a TRACE CT,ON command.

You can change options for SYSBCPII tracing while the trace is running.

**CTIHWI00 parmlib member**

The following table indicates the parameters you can specify on a CTIHWI00 parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTIHWI00?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You cannot change the SYSBCPII component trace buffer size of 4M.
The IBM-supplied CTIHWI00 parmlib member initializes minimal error tracing as soon as the HWIBCPII address space starts.

The contents of CTIHWI00 are:

```
TRACEOPTS ON OPTIONS('MIN')
```

It is suggested that you use these default settings in the CTIHWI00 parmlib member, unless the IBM Support Center requests different tracing options for BCPii. If the CTIHWI00 parmlib member cannot be found during BCPii initialization, or if it is in error, minimal tracing will be activated.

**TRACE and REPLY commands**

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You can change options while a SYSBCPII trace is running.

**OPTIONS parameter**

The values for the OPTIONS parameter for the CTIHWI00 parmlib member and reply for a TRACE command, in an alphabetical order, are:

- **ALL**
  Traces events listed for all the options, including module flow and tracing for every request in both success and failure paths.

- **MIN**
  Traces events related to BCPii component recovery, abnormal conditions, and other non-mainline paths.

**Examples of requesting SYSBCPII traces**

- The CTIHWI00 member requests ALL options.
  ```
  TRACEOPTS
  ON
  OPTIONS('ALL')
  ```

- The TRACE command specifying a Parmlib Member
  ```
  trace ct,on,comp=sysbcpii,parm=ctihwi00
  ```

- The TRACE command with Options Specified in a REPLY
  ```
  trace ct,on,comp=sysbcpii
  * 8 ITT006A ...
  reply 8,options=(all),end
  ```

- The TRACE command to Stop Tracing
  ```
  trace ct,off,comp=sysbcpii
  ```
Formatting a SYSBCPII trace

Format the trace with an IPCS CTRACE COMP(SYSBCPII) subcommand.

Output from a SYSBCPII trace

Figure 111 is an example of SYSBCPII component trace records formatted with the CTRACE COMP(SYSBCPII) SHORT subcommand:

<table>
<thead>
<tr>
<th>COMPONENT TRACE SHORT FORMAT</th>
<th>COMP(SYSBCPII)</th>
<th>**** 07/25/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSNAME</td>
<td>MNEMONIC</td>
<td>ENTRY ID</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>SY1</td>
<td>Message</td>
<td>10012130</td>
</tr>
<tr>
<td>SY1</td>
<td>Request</td>
<td>100321C0</td>
</tr>
<tr>
<td>SY1</td>
<td>State</td>
<td>10002007</td>
</tr>
<tr>
<td>SY1</td>
<td>Request</td>
<td>1000200E</td>
</tr>
<tr>
<td>SY1</td>
<td>Request</td>
<td>10002011</td>
</tr>
<tr>
<td>SY1</td>
<td>Request</td>
<td>10002005</td>
</tr>
<tr>
<td>SY1</td>
<td>Request</td>
<td>10052220</td>
</tr>
<tr>
<td>SY1</td>
<td>State</td>
<td>10072351</td>
</tr>
<tr>
<td>SY1</td>
<td>Request</td>
<td>10072352</td>
</tr>
</tbody>
</table>

Figure 111. Example: SYSBCPII component trace records formatted with CTRACE COMP(SYSBCPII) SHORT subcommand

CTRACE COMP(SYSBCPII) FULL subcommand output

Figure 112 on page 420 is an example of SYSBCPII component trace records formatted with the CTRACE COMP(SYSBCPII) FULL subcommand.
### Component Trace

COMPONENT TRACE FULL FORMAT
COMP(SYSBCPII)
**** 07/25/2008

<table>
<thead>
<tr>
<th>SY1</th>
<th>Message 10012130 13:50:09.359769</th>
<th>HWIPHCPI About to call HWIPHRES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASIDHOME. 0018 ASIDPRI.. 0018</td>
<td>JOBNAME.. HWIBCPII TCBADDR.. 005DFA48</td>
</tr>
<tr>
<td></td>
<td>KEY...... 0004 LEN...... 0026 COUNT.... 0001</td>
<td></td>
</tr>
</tbody>
</table>

BCPii About to pass HSDB to HWIPHRES.

<table>
<thead>
<tr>
<th>SY1</th>
<th>Request 100321C0 13:50:09.359963</th>
<th>HWIPHRES HSDB created</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASIDHOME. 0018 ASIDPRI.. 0018</td>
<td>JOBNAME.. HWIBCPII TCBADDR.. 005DFA48</td>
</tr>
<tr>
<td></td>
<td>KEY...... 0005 LEN...... 003C COUNT.... 0001</td>
<td></td>
</tr>
</tbody>
</table>

SY1 Request 10002007 13:50:09.361439 HWIPHARI Appl Extension located

<table>
<thead>
<tr>
<th>SY1</th>
<th>State 10002007 13:50:09.368871</th>
<th>HWIPHARI Session Elem defined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASIDHOME. 0018 ASIDPRI.. 0018</td>
<td>JOBNAME.. HWIBCPII TCBADDR.. 005DFA48</td>
</tr>
<tr>
<td></td>
<td>KEY...... 0005 LEN...... 0080 COUNT.... 0001</td>
<td></td>
</tr>
</tbody>
</table>

SY1 Request 1000200E 13:50:09.368871 HWIPHARI Session Elem defined

<table>
<thead>
<tr>
<th>SY1</th>
<th>Request 1000200E 13:50:09.368871</th>
<th>HWIPHARI Session Elem defined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASIDHOME. 0018 ASIDPRI.. 0018</td>
<td>JOBNAME.. HWIBCPII TCBADDR.. 005DFA48</td>
</tr>
<tr>
<td></td>
<td>KEY...... 0005 LEN...... 0000 COUNT.... 0001</td>
<td></td>
</tr>
</tbody>
</table>

---

Figure 112. Example: SYSBCPII component trace records formatted with CTRACE COMP(SYSBCPII) FULL subcommand

### SYSBHI component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398
Basic HyperSwap socket support component trace is described by the following attributes:

- Trace buffers reside in 64-bit common ECSA. Size is controlled by the TRACE CT operator command.
- Minimal and unexpected event tracing is activated during Basic HyperSwap management address space initialization.
- Component trace buffers externalized through:
  - DUMP or SLIP operator command when the Basic HyperSwap management address space or a BHIHSRV address space is requested to be dumped.
  - SVC dumps issued by Basic HyperSwap management address space or the BHIHSRV address space.
  - MVS component trace (CTRACE) external writer.
- Trace options revert to minimal event and exception tracing when the operator turns the trace off.

The following summarizes information for requesting a SYSBHI component trace for Basic HyperSwap.

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSBHI:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTIBHIXx; Default member: CTIBHI00 (IBM provides a sample in SYS1.SAMPLIB)</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; error; error events</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTIBHIXx and REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: 4MB</td>
</tr>
<tr>
<td></td>
<td>• Range: 4MB - 64MB</td>
</tr>
<tr>
<td></td>
<td>• Size set by: CTIBHIXx parmlib member or REPLY to TRACE CT command</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: Yes</td>
</tr>
<tr>
<td></td>
<td>• Location: 64-bit Common Storage Area (ECSA)</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer; trace data sets</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command when dumping the HyperSwap Management address space or one of the BHIHSRV address spaces</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSBHI)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Requesting a SYSBHI trace**

Specify options for requesting a SYSBHI component trace in a CTIBHIXx parmlib member or on the reply for a TRACE CT command.

**CTIBHIXx parmlib member**

The following table indicates the parameters you can specify in a CTIBHIXx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTICEAnn?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Component Trace

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTICEAnn?</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The IBM supplied CTIBHI00 parmlib member in SYS1.SAMPLIB can be used as a starting point for defining BHI CTRACE options. The contents of CTIBHI00 are:

```
TRACEOPTS
    ON
    BUFSIZE(4M)
    OPTIONS('MINIMUM')
```

**TRACE and REPLY commands**
The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, OFF or nnnnM</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**OPTIONS parameter**
The values for the OPTIONS parameter in the CTIBHIxx parmlib member and reply for a TRACE command are:

**ALL**
Trace everything.

**FLOW**
Trace the flow of all requests through their processing.

**INITTERM**
Trace the initialization and termination process of the BHIHSRV address spaces and tasks.
**MINIMUM**

Trace errors and unusual events.

**Formatting a SYSBHI trace**

Format the trace with an IPCS CTRACE COMP(SYSBHI) subcommand.

**Output from a SYSBHI trace**

Figure 113 shows an example of the formatted IPCS output produced from the CTRACE COMP(SYSBHI) SHORT subcommand.

```plaintext
Component Trace

### COMPONENT TRACE SHORT FORMAT
COMP(SYSBHI)
**** 12/20/2012

<table>
<thead>
<tr>
<th>SYNAME</th>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SY1</td>
<td>InitMsg</td>
<td>00000001</td>
<td>19:20:28.105817</td>
<td>CTrace Initialized</td>
</tr>
<tr>
<td>SY1</td>
<td>TskStrtd</td>
<td>00000002</td>
<td>19:20:28.105858</td>
<td>Task has been started</td>
</tr>
<tr>
<td>SY1</td>
<td>BPXRslt</td>
<td>0000002B</td>
<td>19:20:28.106110</td>
<td>BPX Service Results</td>
</tr>
<tr>
<td>SY1</td>
<td>BPXRslt</td>
<td>0000002B</td>
<td>19:20:28.107162</td>
<td>BPX Service Results</td>
</tr>
<tr>
<td>SY1</td>
<td>RACFRslt</td>
<td>00000035</td>
<td>19:20:56.880588</td>
<td>Racf Results</td>
</tr>
<tr>
<td>SY1</td>
<td>LognRejt</td>
<td>00000036</td>
<td>19:20:56.880589</td>
<td>Logon Rejection</td>
</tr>
<tr>
<td>SY1</td>
<td>BPXRslt</td>
<td>0000002B</td>
<td>19:20:56.880626</td>
<td>BPX Service Results</td>
</tr>
</tbody>
</table>
```

Figure 113. Example: formatted IPCS output produced from CTRACE COMP(SYSBHI) SHORT subcommand

Figure 114 on page 424 shows an example of the formatted IPCS output produced from the CTRACE COMP(SYSBHI) FULL subcommand.
### Component Trace

**Component Trace Full Format**

**COMP(SYSBHI)**

**** 12/20/2012

<table>
<thead>
<tr>
<th>SY1</th>
<th>InitMsg 00000001 19:20:28.105817</th>
<th>CTrace Initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID....</td>
<td>0025 IssueMod. IOSHMSTR TCB......</td>
<td>00506048</td>
</tr>
<tr>
<td>HSAILidx.</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td>MiscEbcd.</td>
<td>CTrace Initialization Complete</td>
<td></td>
</tr>
</tbody>
</table>

**SY1**  TaskStrtd 00000002 19:20:28.105858  Task has been started

| ASID....| 0025 IssueMod. IOSHMSTR TCB......| 00506048          |
| HSAILdx.| 00000000                         |                   |

**SY1**  BPXRslt 0000002B 19:20:28.106110 BPX Service Results

| ASID....| 0025 IssueMod. IOSHMSTR TCB......| 00506048          |
| HSAILdx.| 00000000                         |                   |
| BPXServ.| BPX1QDB - Querying DUB Status    |                   |
| RetValue.| 00000000                         |                   |
| RetCode.| 00000000                         |                   |
| RsnCode.| 00000000                         |                   |

**SY1**  BPXRslt 0000002B 19:20:28.107162 BPX Service Results

| ASID....| 0025 IssueMod. IOSHMSTR TCB......| 00506048          |
| HSAILdx.| 00000000                         |                   |
| BPXServ.| BPX1ENV - Reg USS Shutdown Exit   |                   |
| RetValue.| 00000000                         |                   |
| RetCode.| 00000000                         |                   |
| RsnCode.| 00000000                         |                   |

**SY1**  RACFRslt 00000035 19:20:56.880588 Racf Results

| ASID....| 0025 IssueMod. IOSHMSTR TCB......| 00506048          |
| HSAILdx.| 00000000                         |                   |
| MiscEbcd.| Reg=Verify Envir=Create          |                   |
| RetCode.| 00000008                         |                   |
| RsnCode.| 00000000                         |                   |

**SY1**  LognRejt 00000036 19:20:56.880589 Logon Rejection

| ASID....| 0025 IssueMod. IOSHMSTR TCB......| 00506048          |
| HSAILdx.| 00000000                         |                   |

--- TOH Start ---

Acronym.. TOH  Version.. 01  Size..... 00000038  
Function Code: 000000C8 (Logon Request)  
User Token  
DA80E926 92859940 40404040 40404040 | MyToken |  
BuffSize. 00000000 Buff_Off. 0000  
Return Code: 05080800  
RACROUTE Function : Reg=Verify Envir=Create  
RACROUTE Return Code: 08  
RACF Return Code: 08  
RACF Reason Code: 00

--- TOH End ---

**SY1**  BPXRslt 0000002B 19:20:56.880626 BPX Service Results

| ASID....| 0025 IssueMod. IOSHMSTR TCB......| 00506048          |
| HSAILdx.| 00000000                         |                   |
| BPXServ.| BPX1SND - Reject Logon           |                   |
| RetValue.| 00000038                         |                   |
| RetCode.| 00000000                         |                   |
| RsnCode.| 00000000                         |                   |
| SockTokn.| 23B00000                         |                   |

--- TOH Start ---

Acronym.. TOH  Version.. 01  Size..... 00000038  
Function Code: 000000C8 (Logon Request)  
User Token  
DA80E926 92859940 40404040 40404040 | MyToken |  
BuffSize. 00000000 Buff_Off. 0000  
Return Code: 05080800  
RACROUTE Function : Reg=Verify Envir=Create  
RACROUTE Return Code: 08  
RACF Return Code: 08  
RACF Reason Code: 00

--- TOH End ---

**Figure 114. Example: formatted IPCS output produced from CTRACE COMP(SYSBHI) FULL subcommand**
SYSCEA component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSCEA component trace for common event adapter component.

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSCEA:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTICEAnn; default member: CTICEA00</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; error; error events</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTICEAxx and REPLY for TRACE command</td>
</tr>
</tbody>
</table>
| Buffer | • Default: 2MB  
• Range: 1MB - 2GB  
• Size set by: CTICEAnn parmlib member or REPLY to TRACE CT command  
• Change size after IPL: Yes, when restarting a trace after stopping it  
• Location: common event adapter trace data space. |
| Trace records location | Address-space buffer; common event adapter trace data space; trace data set |
| Request of SVC dump | By DUMP or SLIP command |
| Trace formatting by IPCS | CTRACE COMP(SYSCEA) |
| Trace format OPTIONS parameter | Yes |

Requesting a SYSCEA trace

Specify options for requesting a SYSCEA component trace in a CTICEAxx parmlib member or on the reply for a TRACE CT command. Changing SYSCEA trace options after CEA has started requires stopping and restarting the trace.

CTICEAnn parmlib member

The following table indicates the parameters you can specify in a CTICEAnn parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTICEAnn?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>No</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>MOD</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Component Trace

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTICEAnn?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note:** The buffer size can be changed after IPL. To become effective, the common event adapter address space (CEA) must be restarted. Specify the new buffer size in the BUFSIZE parameter in the CTICEAnn member being used.

The IBM supplied CTICEA00 parmlib member initializes error tracing as soon as the common event adapter address space starts. The contents of CTICEA00 are:

```
TRACEOPTS
  ON
  BUFSIZE(2M)
  OPTIONS('ERROR')
```

**TRACE and REPLY commands**
The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, OFF or nnnnM</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**OPTIONS parameter**
The values for the OPTIONS parameter in the CTICEAx parmlib member and reply for a TRACE command are:

**ALL**
  Trace everything.

**CNTLFLOW**
  Trace CNTLFLOW events.

**EVNTFLOW**
  Trace EVNTFLOW events.

**JOBSFLOW**
  Trace JOBSFLOW events.
Component Trace

PDWBFLOW
Trace PDWBFLOW events.

ERROR
Trace error events.

Formatting a SYSCEA trace
Format the trace with an IPCS CTRACE COMP(SYSCEA) subcommand.

Output from a SYSCEA trace
Figure 115 shows an example of the formatted IPCS output produced from the CTRACE COMP(SYSCEA) FULL subcommand.

SYSDLF component trace
Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSDLF component trace for the data lookaside facility (DLF).
**Component Trace**

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSDLF:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>None</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; always on when DLF is running</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>None</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: N/A</td>
</tr>
<tr>
<td></td>
<td>• Range: N/A</td>
</tr>
<tr>
<td></td>
<td>• Size set by: MVS system</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: No</td>
</tr>
<tr>
<td></td>
<td>• Location: Data space. In the REPLY for the DUMP command, specify DSPNAME=(DLF.CCOFGSDO)</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer, data-space buffer</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSDLF)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>None</td>
</tr>
</tbody>
</table>

**Requesting a SYSDLF trace**

The trace runs whenever DLF is in control. No actions are needed to request it.

**Formatting a SYSDLF trace**

Format the trace with an IPCS CTRACE COMP(SYSDLF) subcommand. The subcommand has no OPTIONS values.

**Output from a SYSDLF trace**

Figure 116 is an example of DLF component trace records formatted with a CTRACE COMP(SYSDLF) FULL subcommand. It shows formatted exception records from the trace buffers.

---

**DLF COMPONENT TRACE FULL FORMAT**

```
COFRCVRY 00000000 15:47:40.397545 DLF RECOVERY ENTRY
HASID... 000E SASID... 000E CPUID... FF170067 30900000
MODNAME. COFMCON2 ABEND... 840C1000 REASON.. 00000001
EPTABLE. CON2 EST2 .... .... .... .... .... .... .... ....
COFRCVRY 00000001 15:47:40.397625 DLF RECOVERY EXIT
HASID... 000E SASID... 000E CPUID... FF170067 30900000
MODNAME. COFMCON2 ABEND... 840C1000 REASON.. 00000001
RETCODE. 0000002C RSNCODE.. 0000C200 FTPRTS.. C0000000 DATA.... 00000000
```

*Figure 116. CTRACE COMP(SYSDLF) FULL subcommand output*

The fields in the report are:

**COFRCVRY**

The name or identifier of the trace record.

**00000000**

The identifier in hexadecimal.

**15:47:40.397545**

The time stamp indicating when the record was placed in the trace table.

**HASID... 000E**

The home address space identifier.
SASID... 000E  
The secondary address space identifier.

CPUID... FF170067 30900000  
The identifier of the processor that placed the record in the trace table.

CALLER  
The address of the routine that issued a DLF service request.

MODNAME COFMCON2  
The name of the module that was running.

ABEND... 840C1000  
The abend that occurred and caused DLF to enter recovery.

REASON.. 00000001  
The reason code associated with the abend.

EPTABLE. CON2 EST2  
Information used for diagnosis by IBM.

RETCODE. 0000002C  
The return code that was issued by the module that is exiting.

RSNCODE. 0000C200  
The reason code that was issued by the module that is exiting.

FTPRTS.. C0000000  
Information used for diagnosis by IBM.

DATA.... 00000000  
Information used for diagnosis by IBM.

**SYSDSOM component trace**

Before using this component trace, ensure that you have read:
- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSDSOM component trace for distributed SOMobjects (DSOM).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSDSOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>None</td>
</tr>
<tr>
<td>Default tracing</td>
<td>No</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: N/A</td>
</tr>
<tr>
<td></td>
<td>• Range: N/A</td>
</tr>
<tr>
<td></td>
<td>• Size set by: MVS system</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: No</td>
</tr>
<tr>
<td></td>
<td>• Location: Address space</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSDSOM)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Component Trace

**Requesting a SYSDSOM trace**
Request the trace by specifying any non-zero value on the SOMDTRACELEVEL DSOM environment variable.

**Formatting a SYSDSOM trace**
Format the trace with an IPCS CTRACE COMP(SYSOM) subcommand. The subcommand has the following OPTIONS values:

- **SKIPID**
  Omits the jobname, ASID, and thread identifier from the output.

- **LONGFORM**
  Tells the system to display detailed output. In the output, each trace function might have multiple trace elements, each on a separate line. If you do not specify LONGFORM, the default is SHORTFORM. Do not specify both LONGFORM and SHORTFORM on the OPTIONS parameter.

- **SHORTFORM**
  Tells the system to display abbreviated output. In the output, each trace function is on one line. SHORTFORM is the default value. Do not specify both LONGFORM and SHORTFORM on the OPTIONS parameter.

**Output from a SYSDSOM trace**
The output shown in Figure 117 is an example of DSOM component trace records formatted with a CTRACE COMP(SYSOM) FULL OPTIONS((SKIPID,SHORTFORM)) subcommand. It shows formatted exception records from the trace buffers.

```
DSOM COMPONENT TRACE FULL FORMAT

KESYS522 METRETRN 00000004 21:31:34.864277 Return from method
Entry to method: ImplRepository::somInit
```

Figure 117. Example: DSOM component trace records formatted with CTRACE COMP(SYSOM) FULL OPTIONS((SKIPID,SHORTFORM))

**Figure 118 on page 431** is an example of DSOM component trace records formatted with a CTRACE COMP(SYSOM) FULL subcommand. It shows formatted exception records from the trace buffers.
The output shown in Figure 119 is an example of DSOM component trace records formatted with CTRACE COMP(SYSDSOM) FULL OPTIONS((SKIPID)) subcommand. It shows formatted exception records from the trace buffers.

The fields in the report are:

**KESYS522**
- The name of the system.

**METDEBUG**
- The name of the trace event.

**00000004**
- The decimal identifier of the trace event.

**21:31:39.410681**
- The time stamp indicating when the record was placed in the trace table.

**ASID**
- The ASID of the job listed in the JOBNAME field.

**JOBNAME. KREPROC**
- The job name.

**THREADID 04233100 00000000**
- The POSIX thread identifier.
Component Trace

Entry to method
The entry to the somInit method in class SOMOA.

Exiting Method
The exit from the somInit method in class SOMOA.

SYSDUMP component trace
Before using this component trace, ensure that you have read:
- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

SDUMP component trace is described by the following attributes:
- Trace buffers reside in 64-bit common storage area. Size is controlled by the TRACE CT operator command.
- Tracing is activated during DUMPSRV address space initialization.
- SDUMP CTRACE is captured at the end of the SDUMP capture phase and dumped at the end of the dump. If the trace data is needed at other times, you must request a new dump to collect the SDUMP CTRACE.
- The MVS component trace (CTRACE) external writer can also capture SDUMP CTRACE.
- Trace options revert to minimal event tracing when the operator turns the trace off.

The following table summarizes the information required to request a SYSDUMP component trace for SDUMP:

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSDUMP:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTIDMPxx; Default member: CTIDMP00</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; full tracing</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTIDMPxx and REPLY for TRACE CT command</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: 4MB</td>
</tr>
<tr>
<td></td>
<td>• Range: 4MB – 32MB</td>
</tr>
<tr>
<td></td>
<td>• Size set by: CTIDMPxx parmlib member or TRACE CT command</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: Yes</td>
</tr>
<tr>
<td></td>
<td>• Location: 64-bit Common Storage Area</td>
</tr>
<tr>
<td>Trace records location</td>
<td>64-bit Common Storage Area; trace data sets</td>
</tr>
<tr>
<td>Externalizing trace data</td>
<td>Included in every SDUMP; can also use external writer to write data to a trace data set</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSDUMP)</td>
</tr>
<tr>
<td>IPCS trace format OPTIONS parameter</td>
<td>None</td>
</tr>
<tr>
<td>Maximum trace entry size</td>
<td>4050 bytes</td>
</tr>
</tbody>
</table>

Requesting a SYSDUMP trace
Specify options for requesting a SYSDUMP component trace in a CTIDMPxx parmlib member or on the reply for a TRACE CT command.
CTIDMPxx parmlib member

The following table indicates the parameters you can specify in a CTIDMPxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTIDMPxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>MOD</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You can use the IBM supplied CTIDMP00 parmlib member in SYS1.PARMLIB as a starting point for defining SDUMP CTRACE options. The contents of CTIDMP00 are as follows:

```
TRACEOPTS
ON
BUFSIZE(4M)
OPTIONS('ALL')
```

TRACE and REPLY commands

The following tables indicate the parameters that you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, OFF or nnnnM</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>
OPTIONS parameter
The following values are available for the OPTIONS parameter in the CTIDMPxx
parmlib member and reply for a TRACE command:

  ALL
   Trace everything.

  MINIMUM
   Trace errors.

Formatting a SYSDUMP trace
Format the trace with an IPCS CTRACE COMP(SYSDUMP) subcommand.

Output from a SYSDUMP trace
Figure 120 on page 435 is an example of the formatted IPCS output that is
produced from the CTRACE COPM(SYSDUMP) FULL subcommand.
## Component Trace

<table>
<thead>
<tr>
<th>SYSTYPE</th>
<th>MNEMONIC</th>
<th>ENTRY_ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>S790</td>
<td>WtDSVSST</td>
<td>0000007F</td>
<td>17:59:02.744031</td>
<td>Wait for DSVSSTECB to be posted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..... 0005 IssueMod. IEAVTSSST TCB....... 004DFD90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RetnAddr. 8980258A</td>
</tr>
<tr>
<td>S790</td>
<td>LockSmp</td>
<td>00000024</td>
<td>18:03:35.343314</td>
<td>Sdump is locked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..... 001B IssueMod. IEAVAD00 TCB....... 004FF200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RetnAddr. 862028EC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CVTSDBF.. 80950FFB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RTCTSDPL. 00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JobName.. DCMTESTC</td>
</tr>
<tr>
<td>S790</td>
<td>DmpStrt</td>
<td>00000002</td>
<td>18:03:35.343317</td>
<td>Sdump started</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..... 001B IssueMod. IEAVAD00 TCB....... 004FF200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RetnAddr. 86202904</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RTCTSDPL. 894CF620</td>
</tr>
<tr>
<td>S790</td>
<td>ClrStor</td>
<td>00000029</td>
<td>18:03:35.343782</td>
<td>Storage cleared for new dump</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..... 001B IssueMod. IEAVAD00 TCB....... 004FF200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RetnAddr. 8620247B</td>
</tr>
<tr>
<td>S790</td>
<td>EntyTSPR</td>
<td>00000013</td>
<td>18:03:35.343788</td>
<td>Entry at IEAVTSPR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..... 001B IssueMod. ........ TCB....... 004FF200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RetnAddr. 812D6184</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RTDSFNCd. 0001</td>
</tr>
<tr>
<td>S790</td>
<td>EntyTSPR</td>
<td>00000013</td>
<td>18:03:35.343795</td>
<td>Entry at IEAVTSPR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..... 001B IssueMod. ........ TCB....... 004FF200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RetnAddr. 812D6184</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RTDSFNCd. 0005</td>
</tr>
<tr>
<td>S790</td>
<td>SdumpPm</td>
<td>00000004</td>
<td>18:03:35.343813</td>
<td>Sdump PList after calling TSPR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..... 001B IssueMod. IEAVAD00 TCB....... 004FF200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RetnAddr. 86203120</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RTCTSDPL. 023CF40</td>
</tr>
<tr>
<td></td>
<td>Sdump ParmList:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0000 01A19010 00000000 00000000 00000000 023C838</td>
<td>~ H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0010 00000000 00000000 00000000 00000000</td>
<td>~&amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0020 00000000 00000000 00000000 25E04000</td>
<td>~ &amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0030 00000000 00000000 00000000 00000000</td>
<td>~&amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0040 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0050 00000000 00000000 090803F0 00000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0060 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0070 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0080 00000000 00000000 025B4190 00000000</td>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0090 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+00A0 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+00B0 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+00C0 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+00D0 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+00E0 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+00F0 00000000 00000000 00000000 00000000</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Title.... DMPTESTC: Test case dump for &gt;2G local capture optimization.........</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S790</td>
<td>DumpSupp</td>
<td>00000006</td>
<td>18:03:35.343820</td>
<td>Dump is not being suppressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..... 001B IssueMod. IEAVAD00 TCB....... 004FF200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RetnAddr. 862033CC</td>
</tr>
</tbody>
</table>

---

**Figure 120. Example: SDUMP component trace records formatted with CTRACE COMP(SYSDUMP) FULL**

Chapter 12. Component trace 435
Viewing SDUMP CTRACE in IPCS Active

You can view SDUMP CTRACE data in IPCS from active system storage. Ensure that the dump source is set to 'ACTIVE' and that the user has authorized READ access to the FACILITY class resource BLSACTV.ADDRSPAC. See “Active Storage Processing” in z/OS MVS IPCS User’s Guide for more information.

SYSGRS component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSGRS component trace for global resource serialization.

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSGRS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTnGRSxx; default member: CTIGRS00 specified in GRSCNF00 member</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes, if global resource serialization ring is active; CONTROL and MONITOR options</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes, if global resource serialization star is active; CONTROL1, CONTROL2, SIGNAL0 and MONITOR options</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTnGRSxx and REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td>Buffer: Default: 16MB, Range: 128KB - 2047MB (System rounds size up to nearest 64KB boundary.), Size set by: CTnGRSxx member, Change size after IPL: Yes, when restarting a trace after stopping it, Location: In the GRS address space above the bar.</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer, trace data set</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>“SYSGRS component trace”</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>FLOW, CONTROL, MONITOR, REQUEST, SIGNAL, and RSA. See “OPTIONS parameter” on page 438 for details on sub-options.</td>
</tr>
</tbody>
</table>

Requesting a SYSGRS trace

Specify options for requesting a SYSGRS component trace on a CTnGRSxx parmlib member or on the reply for a TRACE CT command.

You can change options for SYSGRS tracing while the trace is running.

CTnGRSxx parmlib member

The following table indicates the parameters you can specify on a CTnGRSxx parmlib member.
### Parameters Allowed on CTnGRSxx?

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Allowed on CTnGRSxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

IBM-supplied GRSCNF00 parmlib member specifies CTIGRS00 as the default.

IBM-supplied CTIGRS00 parmlib member:
- Specifies that GRS tracing is begun at IPL
- The parmlib member contains:

  **TRACEOPTS OFF**
  
  This parameter turns off all SYSGRS tracing options except for the minimum options (MINOPS).
  
- See the "Specify buffers" section for information about the buffer size default and possible sizes.

IBM recommends that you use the CTIGRS00 parmlib member, unless the IBM Support Center requests different tracing for global resource serialization.

### TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

#### Parameters Allowed on TRACE CT for trace?

| Parameters                     | Allowed on TRACE CT for trace?
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, nnnnK, nnnnM, or OFF</td>
<td>One is required. The buffer size can be changed only when the trace is OFF or the trace is ON.</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Parameters Allowed on TRACE CT for writer?

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

#### Parameters Allowed on REPLY for trace?

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>
You can change options while a SYSGRS trace is running. However, to change the buffer size, you have to stop the trace and restart it with the new buffer size.

**OPTIONS parameter**

The values for the OPTIONS parameter for the CTnGRSxx parmlib member and reply for a TRACE command are listed below. The sub-options on the CONTROL, REQUEST, MONITOR, SIGNAL and FLOW, allow you to refine the set of events traced for the major option. When you select the major option, all events pertaining to that option are traced. However, you can select one or more of the sub-options instead of the major option and thus limit the trace to only those events included in the sub-options specified. A major option, such as MONITOR, and all of its sub-options (in this case MONITOR0, MONITOR1, and MONITOR2 through MONITORF) is referred to as an option group. In alphabetical order the values for the OPTIONS parameter are:

**CONTROL**

Traces unusual events and events related to the establishment, modification, or termination of the control structure needed for processing such as:

- Dynamic RNL changes
- Error events
- XCF services used when setting up for processing

When you specify CONTROL, all of the following sub-options are traced.

**CONTROL0**

Traces dynamic RNL changes only.

**CONTROL1**

Traces events related to the establishment of or termination of membership in the global resource serialization group connection to the global resource serialization coupling facility structures.

**CONTROL2**

Traces global resource serialization recovery processing only.

**CONTROL3**

Traces global resource serialization resource manager events for abnormal task and ASID termination only.

**CONTROL4-CONTROL9**

Reserved for IBM use.

**CONTROLF**

Traces all other unusual events not included in sub-options CONTROL0 through CONTROL3.

**FLOW**

Traces the flow of control from one entry point to another.

**FLOW0**

Traces GRS Star system server processing only.

**FLOW1**

Traces GQSCAN processing only.

**FLOW2**

Traces cross-system communications processing only.

**FLOW3**

Traces command processing only.
Component Trace

FLOW4
Traces storage manager services only.

FLOW5
Traces coupling facility processing only.

FLOW6
Traces initialization processing only.

FLOW7
Traces contention monitor processing only.

FLOW8
Traces general ENQ/DEQ processing only.

FLOW9
Traces entry to GQSCAN/ISQUERY only.

FLOWA
Traces GRS Latch Manager processing only.

FLOWB–FLOWD
Reserved for IBM use.

FLOWE
Activates extended tracing for the GRS Storage Manager. Do not turn on this option without direction from IBM Service.

FLOWF
Reserved for IBM use.

Monitor
Traces events for selected global resource serialization invocations of monitoring and communication services provided by other components.

MONITOR0
Traces use of XES services.

MONITOR1
Traces use of XCF services.

MONITOR2–MONITORF
Reserved for IBM use.

REQUEST
Traces events for global ENQ, DEQ, GQSCAN, and RESERVE macro requests, and GRS command processing.

REQUEST0
Traces ENQ/RESERVE requests only.

REQUEST1
Traces DEQ requests only.

REQUEST2
Traces GQSCAN only.

REQUEST3
Traces IXLLOCK only.

REQUEST4
Traces command processing only.

REQUEST5
Traces lock structure (ISGLOCK) rebuild processing only.
REQUEST6-REQUESTF
Reserved for IBM use.

RSA
Traces events for RSA control information.

SIGNAL
Traces events for selected global resource serialization invocations of cross-system coupling facility (XCF) signalling service processing.

SIGNAL0
Traces migration signals only.

SIGNAL1
Traces GQSCAN signals only.

SIGNAL2
Traces ENQ/DEQ signals, including RNL change signals only.

SIGNAL3
Traces contention monitor signals only.

SIGNAL4-SIGNALF
Reserved for IBM use.

Examples of requesting SYSGRS traces

• Example 1: CTnGRSxx member
  The member requests CONTROL, MONITOR, and RSA options and doubles the default buffer size.
  TRACEOPTS
  ON
  OPTIONS('CONTROL','MONITOR','RSA')
  BUFSIZE(32M)

• Example 2: TRACE command
  The example requests a trace of CONTROL, MONITOR, and REQUEST trace events.
  trace ct,on,comp=sysgrs
  * 17 ITT006A ...
  reply 17,options=(control,monitor,request),end

Formatting a SYSGRS trace

Format the trace with an IPCS CTRACE COMP(SYSGRS) subcommand. It is possible to use the OPTIONS subcommand for COMP (SYSGRS) with values of FLOW, CONTROL, REQUEST, MONITOR, SIGNAL, and RSA for filtering.

Output from a SYSGRS trace

Figure 121 on page 441 is an example of SYSGRS component trace records formatted with the CTRACE COMP(SYSGRS) SHORT subcommand.
Figure 121. Example: SYSGRS component trace records formatted with CTRACE COMP(SYSGRS) SHORT

Figure 122 is an example of SYSGRS component trace records formatted with the CTRACE COMP(SYSGRS) TALLY subcommand.

Component Trace

SYSGRS COMPONENT TRACE SHORT FORMAT

<table>
<thead>
<tr>
<th>FMTID</th>
<th>COUNT</th>
<th>INTERVAL</th>
<th>MNEMONIC</th>
<th>DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>0</td>
<td></td>
<td>RSAIN1</td>
<td>RSA has no CMD area no QWB data</td>
</tr>
<tr>
<td>00000002</td>
<td>0</td>
<td></td>
<td>RSAIN2</td>
<td>RSA has QWB data but no CMD area</td>
</tr>
<tr>
<td>00000003</td>
<td>0</td>
<td></td>
<td>RSAIN3</td>
<td>RSA has CMD area but no QWB data</td>
</tr>
<tr>
<td>00000004</td>
<td>0</td>
<td></td>
<td>RSAIN4</td>
<td>RSA has CMD area and QWB data</td>
</tr>
<tr>
<td>00000005</td>
<td>0</td>
<td></td>
<td>RSAOUT1</td>
<td>RSA has no CMD area no QWB data</td>
</tr>
<tr>
<td>00000006</td>
<td>898</td>
<td>2,037,854</td>
<td>RSAOUT2</td>
<td>RSA has QWB data but no CMD area</td>
</tr>
<tr>
<td>00000007</td>
<td>0</td>
<td></td>
<td>RSAOUT3</td>
<td>RSA has CMD area but no QWB data</td>
</tr>
<tr>
<td>00000008</td>
<td>8</td>
<td>149,924,356</td>
<td>RSAOUT4</td>
<td>RSA has CMD area and QWB data</td>
</tr>
<tr>
<td>00000009</td>
<td>5</td>
<td>328,792,726</td>
<td>INVB8E1</td>
<td>ISG8BE - QMERGE</td>
</tr>
<tr>
<td>0000000A</td>
<td>0</td>
<td></td>
<td>INVB8E2</td>
<td>ISG8BE - not QMERGE</td>
</tr>
<tr>
<td>0000000B</td>
<td>4</td>
<td>490,847,959</td>
<td>MAIRNF1</td>
<td>Main Ring Failure</td>
</tr>
<tr>
<td>0000000C</td>
<td>13</td>
<td>149,346,135</td>
<td>CEXBCI1</td>
<td>Control Exited from ISGBCI</td>
</tr>
<tr>
<td>0000000D</td>
<td>0</td>
<td></td>
<td>CINQSCD</td>
<td>Cleanup after Quiesced from ring</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>00000058</td>
<td>0</td>
<td></td>
<td>UEVENT8</td>
<td>Recovery during write</td>
</tr>
<tr>
<td>00000059</td>
<td>0</td>
<td></td>
<td>UEVENT9</td>
<td>Recovery during read</td>
</tr>
<tr>
<td>0000005A</td>
<td>0</td>
<td></td>
<td>UEVENTA</td>
<td>Remote discarded message</td>
</tr>
</tbody>
</table>

Total trace entries: 1,120

Figure 122. Example: SYSGRS component trace records formatted with the CTRACE COMP(SYSGRS) TALLY

SYSHZS component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSHZS component trace for IBM Health Checker for z/OS.
Component Trace

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSHZS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTIHZS00</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTIHZS00 and REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Default: 4MB</td>
</tr>
<tr>
<td></td>
<td>• Range: 16KB - 4MB (System rounds size up to nearest 64KB boundary.)</td>
</tr>
<tr>
<td></td>
<td>• Size set by: CTIHZS00 member</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: Yes, when restarting a trace</td>
</tr>
<tr>
<td></td>
<td>• Location: In the IBM Health Checker for z/OS address space</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSHZS) - see &quot;Output from a SYSHZS trace&quot; on page 444</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>None</td>
</tr>
</tbody>
</table>

**Requesting a SYSHZS trace**

Specify options for requesting a SYSHZS component trace on a CTIHZS00 parmlib member or on the reply for a TRACE CT command.

You can change options for SYSHZS tracing while the trace is running.

**CTIHZS00 parmlib member**

The following table indicates the parameters you can specify on a CTIHZS00 parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTIHZS00</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

IBM supplies the CTIHZS00 parmlib member, which specifies the IBM Health Checker for z/OS tracing begun at IPL. The contents of CTIHZS00 are:

This parameter turns off all SYSHZS tracing options.

```
TRACEOPTS
  0FF
```
If additional SYSHZS tracing options are turned on, additional buffer space might be required.

The default trace buffer size is 4MB.

IBM recommends that you use the CTIHZS00 parmlib member, unless the IBM Support Center requests different tracing for IBM Health Checker for z/OS.

**TRACE and REPLY commands**

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, nnnnK, nnnnM, or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You can change options while a SYSHZS trace is running. However, to change the buffer size, you have to stop the trace and restart it with the new buffer size.

**OPTIONS parameter**

The values for the OPTIONS parameter for the CTnHZSxx parmlib member and reply for a TRACE command are listed below:

**CHECKS**
- Traces unusual events and events related to IBM Health Checker for z/OS checks.

**COMMANDS**
- Traces information about the f hzsproc command, the HZSPRMxx parmlib member and the HZSCHECK macro.

**STORAGE**
- Traces information about storage used by the IBM Health Checker for z/OS address space.

**LOGGER**
- Traces information about the IBM Health Checker for z/OS log stream.

**MISC**
- Traces miscellaneous information.

**ALL**
- Traces all events for IBM Health Checker for z/OS. ALL is the default.
Examples of requesting SYSHZS traces

- Example 1: CTIHZS00 member
  The member requests ALL IBM Health Checker for z/OS component tracing:
  ```
  TRACEOPTS
  ON
  OPTIONS('ALL')
  BUFSIZE(4M)
  ```

- Example 2: TRACE command
  The example requests a trace of ALL trace events.
  ```
  trace ct,on,comp=syshzs
  * 17 ITT006A ... 
  reply 17,options=(all),end
  ```

Formatting a SYSHZS trace
Format the trace with an IPCS CTRACE COMP(SYSHZS) FULL subcommand.

Output from a SYSHZS trace
Figure 123 on page 445 is an example of SYSHZS component trace records formatted with the CTRACE COMP(SYSHZS) FULL subcommand.
Component Trace

Figure 123. Example: SYSHZS component trace records formatted with the CTRACE COMP(SYSHZS) FULL

SYSIEFAL component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSIEFAL component trace for Allocation.

Chapter 12. Component trace 445
### Component Trace

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSIEFAL:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTIIEFx; default member: CTIIEFAL</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; FLOW0, FLOW1, FLOW6, FLOWF, DATA, CONTROL0, CONTROL1, CONTROL6, CONTROLF, SERIAL1, and SERIALF options</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTIIEFx or REPLY for TRACE command</td>
</tr>
</tbody>
</table>
| Buffer                                          | • Default: 8M  
• Range: 256KB - 64MB  
• Size set by: CTIIEFx member  
• Change size after IPL: Yes  
• Location: In the component address space |
| Trace records location                          | Address-space buffer |
| Request of SVC dump                             | By DUMP or SLIP command |
| Trace formatting by IPCS                        | CTRACE COMP(SYSIEFAL) |
| Trace format OPTIONS parameter                  | Yes; FLOW, CONTROL, SERIAL, DATA, MISC, and ERROR |

### Requesting a SYSIEFAL trace

Specify options for requesting a SYSIEFAL component trace on a CTIIEFx parmlib member or on the reply for a TRACE CT command.

You can change options for SYSIEFAL tracing while the trace is running.

#### CTIIEFx parmlib member

The following table indicates the parameters you can specify on a CTIIEFx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTIIEFx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

IBM supplies the CTIIEFAL parmlib member, which specifies the Allocation tracing begun at IPL. The contents of CTIIEFAL are:

```
TRACEOPTS
ON
OPTIONS( 'FLOW0', 'FLOW1', 'FLOW6', 'FLOWF'
```
If additional SYSIEFAL tracing options are turned on, additional buffer space may be required.

The default trace buffer size is 8M.

**TRACE and REPLY commands**

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, nnnnK, nnnnM, or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You can change options while a SYSIEFAL trace is running. However, to change the buffer size immediately, you have to stop the trace and restart it with the new buffer size. If the trace is not stopped and restarted, the buffer size will be changed when the current set of buffers has filled up and a new set is acquired.

**OPTIONS parameter**

The values for the OPTIONS parameter for the CTIIEFxx parmlib member and reply for a TRACE command are listed. The suboptions on the CONTROL, DATA, FLOW, and SERIAL, allow you to refine the set of events traced for the major option. When you select the major option, all events pertaining to that option are traced. However, you can select one or more of the suboptions instead of the major option and thus limit the trace to only those events included in the suboptions specified. A major option, such as DATA, and all of its suboptions (in this case DATA0, DATA1, and so forth) is referred to as an option group. In alphabetical order the values for the OPTIONS parameter are:

**CONTROL**

- Traces the control within a module.
Component Trace

CONTROL0
   Traces common allocation processing only.

CONTROL1
   Traces allocation services 1 processing only.

CONTROL2
   Traces unallocation processing only.

CONTROL3
   Traces volume mount and verify processing only.

CONTROL4
   Traces assign/unassign processing only.

CONTROL5
   Traces allocation services 2 processing only.

CONTROL6
   Traces allocation device management processing only.

CONTROL7
   Traces Dynamic Allocation processing only.

CONTROL8
   Reserved for IBM use.

CONTROL9
   Traces JFCB Houskeeping processing only.

CONTROLA
   Traces TCTIOT management processing only.

CONTROLB-CONTOLE
   Reserved for IBM use.

CONTROLF
   Traces unexpected or unusual control within a module.

DATA
   Traces when data is processed or changed.

DATA0
   Traces when an ATSP device type array is being processed only.

DATA1
   Traces when an ATSP device array is being processed only.

DATA2
   Traces when an IGDE is going through XCF messaging only.

DATA3
   Traces when an IGDE goes through a state change only.

DATA4
   Traces UCB changes only.

DATA5
   Traces ENQ changes for DDR SWAP processing only.

DATA6
   Traces device management data only.

DATA7
   Traces Dynamic Allocation processing only.
Component Trace

DATA8
Traces when an allocation occurs only.

DATA9
Traces JFCB Housekeeping data only.

DATAA
Traces TCTIOT management data only.

DBA-DATAE
Reserved for IBM use.

DATAF
Traces when data is unexpected or unusual.

FLOW
Traces the flow of control from one entry point to another.

FLOW0
Traces common allocation processing only.

FLOW1
Traces allocation services 1 processing only.

FLOW2
Traces unallocation processing only.

FLOW3
Traces volume mount and verify processing only.

FLOW4
Traces assign/unassign processing only.

FLOW5
Traces allocation services 2 processing only.

FLOW6
Traces device management data only.

FLOW7
Traces Dynamic Allocation processing only.

FLOW8
Reserved for IBM use.

FLOW9
Traces JFCB Housekeeping processing only.

FLOWA
Traces TCTIOT management processing only.

FLOWB-FLOWE
Reserved for IBM use.

FLOWF
Traces unexpected or unusual flow from one entry point to another.

SERIAL
Traces serialization events.

SERIAL0
Traces locking (SETLOCK) serialization events only.

SERIAL1
Traces ENQ/DEQ serialization events only.
Component Trace

SERIAL2
Traces latch manager serialization events only.

SERIAL3
Traces compare and swap serialization events only.

SERIAL4
Traces SYSDSN ENQ/DEQ serialization events only.

SERIAL5–SERIALE
Reserved for IBM use.

SERIALF
Traces unexpected or unusual serialization events.

Examples of requesting SYSIEFAL traces
- Example 1: CTIIIEFxx member
  The member requests FLOW and DATA options and requests a buffer size of 8 megabytes.

```plaintext
TRACEOPTS
ON
  OPTIONS('FLOW','DATA')
  BUFSIZE(8M)
```
- Example 2: TRACE command
  The example requests a trace of DATA1 and CONTROL1 trace events.

```plaintext
trace ct,on,comp=sysiefal
  * 17 IT006A ...
reply 17,options=(data1,control1),end
```

Formatting a SYSIEFAL trace
Format the trace with an IPCS CTRACE COMP(SYSIEFAL) subcommand. You can filter trace records for COMP (SYSIEFAL) by using the OPTIONS subcommand. Specify one or more of the following values on the OPTIONS subcommand:
- FLOW
- DATA
- CONTROL
- SERIAL
- ERROR
- MISC

Events in the ERROR and MISC trace groups are always recorded, and are always displayed regardless of what filters are requested.

Output from a SYSIEFAL trace
Figure 124 on page 451 is an example of SYSIEFAL component trace records formatted with the CTRACE COMP(SYSIEFAL) SHORT subcommand.
**Component Trace**

**COMPONENT TRACE SHORT FORMAT**

```
COMP(SYSIEFAL)
**** 09/13/2001
```

<table>
<thead>
<tr>
<th>SYSNAME</th>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>N67</td>
<td>FLOW0</td>
<td>000000100</td>
<td>15:40:47.306228</td>
<td>Common Allocation Flow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..007A TCB..008E1980 MODNAME..IEFAB492 JOBNAME..T015067</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EBCDIC Data...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ENTR</td>
</tr>
</tbody>
</table>

**COMPONENT TRACE FULL FORMAT**

```
COMP(SYSIEFAL)
**** 09/13/2001
```

<table>
<thead>
<tr>
<th>SYSNAME</th>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>N67</td>
<td>FLOW0</td>
<td>000000100</td>
<td>15:40:47.306228</td>
<td>Common Allocation Flow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ASID..007A TCB..008E1980 MODNAME..IEFAB492 JOBNAME..T015067</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EBCDIC Data...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ENTR</td>
</tr>
</tbody>
</table>

**SYSTEMS component trace**

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 377
- "Obtaining a component trace" on page 385
- "Viewing the component trace data" on page 398

IOS component trace is described by the following attributes:
**Component Trace**

- Trace buffers reside in common ESQA Subpool 248. Size is controlled by the TRACE CT operator command. As the buffers become full they are copied to a private IOS data space. For information on specifying an IOS data space size, see "OPTIONS parameter" on page 454.
- Minimal and unexpected event tracing is activated during IOS NIP processing.
- Component trace buffers externalized through:
  - DUMP or SLIP operator command when the IOS address space is requested to be dumped.
  - SVC dumps issued by IOS, dynamic device reconfiguration (DDR), or execute channel program (EXCP) component recovery.
  - MVS component trace (CTRACE) external writer
- Trace options revert to minimal event and exception tracing when the operator turns the trace off.

The following summarizes information for requesting a SYSIOS component trace for IOS:

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSIOS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTnIOSxx specified in the IEIOSxx member through the CTRACE(CTnIOSxx) statement. Default member: None</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes, activated during IOS NIP processing.</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTnIOSxx or REPLY for TRACE command</td>
</tr>
</tbody>
</table>
| Buffer                                  | • Default: 324KB  
  • Range: 324KB - 1.5M  
  • Size set by: CTnIOSxx member or REPLY for TRACE command  
  • Change size after IPL: Yes, when component trace (CTRACE) is active  
  • Location: Common ESQA subpool 248 and SYSIOS private IOS data space. |
| Trace records location                  | Common ESQA subpool 248 and SYSIOS private IOS data space and trace data set.  
                                         | By DUMP or SLIP command when the IOS address space is requested to be dumped. In the REPLY for the DUMP command, specify the IOS address space to be dumped.  
                                         | By SLIP command.  
                                         | By the component during SVC dumps issued by IOS, DDR, or EXCP component recovery. |
| Trace formatting by IPCS                | CTRACE COMP(SYSIOS)                             |
| Trace format OPTIONS parameter          | No                                              |

The areas of IOS traced as part of minimal and unexpected event tracing include:
- Dynamic Configuration Changes
- Parallel Access Volume (PAV) Processing
- Dynamic Channel Path Management (DCM) Processing
- Unconditional Reserve (U/R) Recovery Processing
Component Trace

- Channel Subsystem Call (CHSC) Processing
- Channel Report Word (CRW) Processing
- Missing Interrupt Handler (MIH) Recovery Processing
- Control Unit Initiated Reconfiguration (C.U.I.R.) Request Processing
- Dynamic Pathing Support (DPS) Validation
- Dynamic Device Reconfiguration (DDR) Processing
- Self-Description Processing
- PCIE Initialization and Exceptional Conditions Processing

**Note:** Additional areas are traced when OPTIONS are set for IOS component trace. See "OPTIONS parameter" on page 454.

**Requesting a SYSIOS trace**

No actions are required to request a SYSIOS trace. Minimal and unexpected event tracing is activated during IOS NIP processing and is always active. In addition to this tracing, the user can request a SYSIOS trace using specific options by doing the following:

- Using a CTnIOSxx SYS1.PARMLIB member during NIP processing by specifying the CTRACE(CTnIOSxx) statement in the IECIOSxx SYS1.PARMLIB member.
- Using a CTnIOSxx SYS1.PARMLIB member after NIP by issuing the TRACE system command.
- Using the TRACE system command and specifying the options in response to the prompts that CTRACE provides.

**CTnIOSxx parmlib member**

The following table indicates the parameters you can specify in a CTnIOSxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnIOSxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>RESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

If additional SYSIOS tracing options are turned on, additional buffer space might be required.

**TRACE and REPLY commands**

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
</tbody>
</table>
Component Trace

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>nnnnK, nnnnM</td>
<td>Yes</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The WTR and WTRSTART parameters can be used in a parmlib member specified on the TRACE CT command. The parameters cannot be specified in a parmlib member that is read at IPL because the external writer is not available when the IOS component is defined.

**OPTIONS parameter**

The values for the OPTIONS parameter for the CTnIOSxx parmlib member and reply for a TRACE command are listed below.

**DCM**

Traces events relating to Dynamic Channel Path Management.

**Note:** When DCM is active and this option is set, large amounts of trace data will be recorded. Users may wish to consider using an external writer when this option is set.

Traces the results of the cancel subchannel (XSCH) instruction.

**EXTEND**

Traces all functions that are attached in the IOS address space.

**Note:** Some functions attached in the IOS address space are traced during minimum or unexpected event tracing.

Traces the results of the cancel subchannel (XSCH) instruction.

**STORAGE**

Traces events related to IOS or EXCP storage management. When the STORAGE option is specified, the NOFILTER option or ASID/JOBNAME keywords must also be set.

**CAPTURE**

Traces the capturing and uncapturing of UCBs. When the CAPTURE option is specified, the NOFILTER option or ASID/JOBNAME keywords must also be set.

**NOFILTER**

Allows the STORAGE option to be set without requiring ASID or JOBNAME filtering.
Component Trace

**DS=nnnn**
Allows the user to tailor the IOS Trace Data Space where *nnnn* is the data space size in megabytes.

**Note:**
1. *nnnn* must be a valid decimal digit within the range of 1-1024.
2. This option can only be specified once at IPL time and cannot be modified using the TRACE CT command.
3. The default size for the IOS Trace Data Space is 512M. This can require additional auxiliary storage on systems with a small amount of available auxiliary storage. Please refer to “Decide where to collect the trace records” on page 384 for information about auxiliary storage for CTRACE data space buffers. Since some options such as STORAGE and DCM will cause more CTRACE entries to be recorded, the IOS Trace Data Set may fill up more rapidly than if none of these options is specified. Users who do not have enough auxiliary storage capacity to handle a full data space may choose to use the DS=*nnnn* option to set up a smaller IOS Trace Data Space. Similarly, users who do have enough auxiliary storage capacity to handle a full data space may choose to use the DS=*nnnn* option to set up a larger IOS Trace Data Space. Doing this will prevent potentially valuable debug information from being lost due to wrapping. Note that the number of records contained in a trace data set are highly variable and dependent upon trace settings and system usage.
4. If the DS=*nnnn* option is specified more than once, the request is rejected and the following message is issued:
   
   **IOS622I IOS COMPONENT TRACE OPTION xxxxxxxx IS NOT VALID - THE TRACE DATA SPACE SIZE HAS ALREADY BEEN SET FOR THIS IPL**
5. If the size specified is not valid, then the request is rejected, the default IOS Trace Data Space size is used, and the following message is issued:
   
   **IOS622I IOS COMPONENT TRACE OPTION xxxxxxxx IS NOT VALID - THE REQUESTED SIZE FOR THE TRACE DATA SPACE IS INCORRECT**

**HPAV**
Traces events relating to HyperPAV bind and unbind activity. This option may cause large amounts of trace data to be recorded. It should be utilized as a debug tool when needed and not enabled for normal operations.

**PCIE**
Traces events related to PCIE activity.

**Examples of Requesting SYSIOS traces**
- **Example 1:** CTnIOSxx SYS1.PARMLIB member
  
  This SYS1.PARMLIB member sets the STORAGE option using JOBNAME filtering for JOB001 and sets the buffer size to 600K.

  ```
  TRACEOPTS
  ON
  BUFSIZE(600K)
  OPTIONS(STORAGE)
  JOBNAME(JOB001)
  ```

**Formatting a SYSIOS trace**
IPCS CTRACE formatting services can be used to format the contents of the CTRACE trace entries. Format the trace with the following IPCS subcommand:

```
IPCS CTRACE COMP(SYSIOS) SUMMARY|FULL|SHORT|TALLY
```
Component Trace

**SUMMARY**
Shows the trace entry header and the formatted data for each trace entry.

**FULL**
Shows the trace entry header and the unformatted (hex) data for each trace entry.

**SHORT**
Shows the trace entry header for each trace entry.

**TALLY**
Shows each trace entry and how many times they were traced.

The subcommand has no options.

**CTRACE COMP(SYSIOS) subcommand output**

The following is an example of SYSIOS component trace records formatted with the CTRACE COMP(SYSIOS) SUMMARY option.

**Example: SYSIOS component trace records formatted with CTRACE COMP(SYSIOS) SUMMARY option**

```
CTRACE COMP(SYSIOS) SUMMARY
**** 03/28/1996
SYSTYPE MNEMONIC ENTRY ID TIME STAMP DESCRIPTION
------- -------- -------- --------------- -------------
S530 MIH     00080001 19:42:42.220726 MIH Recovery Halt/Clear Block

Trace Record Function: MIH
MIH condition detected: Start Pending

Record ID: IOSDMHCB.MHCBHSHB  Length: 0034
+0000 00F168E8 20980180 F00000F0 0027FF0  .1.Y......0...0...
+0010 3868B8E8 FFFFFFFF 00000000 00804400  ...Y...........
+0020 3885DA88 00000000 00000000  .e.h.............
+0030 00000000

Record ID: IOSDMHCB.MHCBUCB  Length: 0080
+0000 00000000 00F16890 00000000 00000000 | ... .........
+0010 00000000 00F16890 00000000 00000000 | ... .........
+0020 00000000 00F16890 00000000 00000000 | ... .........
+0030 00000000 00F16890 00000000 00000000 | ... .........
+0040 00000000 00F16890 00000000 00000000 | ... .........
+0050 00000000 00F16890 00000000 00000000 | ... .........
+0060 00000000 00F16890 00000000 00000000 | ... .........
+0070 00000000 00F16890 00000000 00000000 | ... .........

Record ID: IOSDMHCB.MHCBTMJB  Length: 0018
+0000 ACA29DF9 49B6E706 F0F0F0F0 F1F5F0F0 | ..s..X.00001500
+0010 5CD4C1E2 E3C5D95C | MASTER

Record ID: IOSDMHCB.MHCBADDL  Length: 0008
+0000 BCBC2010 01000040 | ....

S530 U/R 000C0001 19:42:52.885939 Unconditional Reserve Sense
Device: 0180 Channel Path: 38
U/R Sense issued by: Missing Interrupt Handler
U/R Sense completion code: 52

S530 U/R 000C0002 19:43:13.927140 Unconditional Reserve I/O
```
Component Trace

Device: 0180
Recovery I/O issued: Reset Allegiance
Return code from IOSRRRSV: 04

Record ID: IOSDRESV.RESS Length: 0044

S530  U/R  000C0002  17:40:50.336526  Unconditional Reserve I/O

Device: 0160  Channel Path: 65
Recovery I/O issued: Sense Path Group ID
Return code from IOSRRRSV: 00

Record ID: IOSDICCW.SNID Length: 000C

S530  U/R  000C0002  22:51:49.169701  Unconditional Reserve I/O

Device: 0180  Channel Path: 0B
Recovery I/O issued: Reset Allegiance
Return code from IOSRRRSV: 00

Record ID: IOSDICCW.RSTA Length: 0020

S530  U/R  000C0002  19:43:39.951626  Unconditional Reserve I/O

Device: 0180  Channel Path: 38
Recovery I/O issued: Unconditional Reserve with release
Return code from IOSRRRSV: 00

SYSJES component trace

Before using this component trace, ensure that you have read:
- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSJES component trace for the JES common coupling services component, also known as JES XCF.

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSJES:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTnJESxx. Default members: CTJES01, CTJES02, CTJES03, CTJES04</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; detailed tracing for sublevel FLOW; minimal tracing of unexpected events for sublevels MSGTRC, USRXIT, XCFEVT</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>None</td>
</tr>
</tbody>
</table>

Chapter 12. Component trace  457
**Component Trace**

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSJES:</th>
</tr>
</thead>
</table>
| Buffer      | • Default: N/A  
             | • Range: N/A    
             | • Size set by: MVS system  
             | • Change size after IPL: No  
             | • Location: In the component address space |
| Trace records location | Address-space buffer, trace data set |
| Request of SVC dump | By the component |
| Trace formatting by IPCS | CTRACE COMP(SYSJES) |
| Trace format OPTIONS parameter | Yes |

**Note:** To get a complete dump for JES XCF, request also the JES and JES XCF address spaces and data spaces, plus SDATA options RGN, SQA, and CSA.

SYSJES tracing is started during initialization. SYSJES contains 4 sublevel traces, which run concurrently. Each sublevel must be started individually. The sublevels are:

- **MSGTRC, message tracing:** MSGTRC records message data sent by the IXZXIXSM service. The default tracing for this sublevel is minimal tracing of unexpected events only. You can optionally start and stop detailed MSGTRC tracing. Use the data from this sublevel in conjunction with USRXIT trace data to get information about message data modified by installation exits IXZXIT01 or IXZXIT02.

- **USRXIT, installation exit tracing:** USRXIT records the exit parameter list (SPELL) passed to and returned from installation exits IXZXIT01, IXZXIT02, and IXZXIT03 processing. The default tracing for this sublevel is minimal tracing of unexpected events only. You can optionally start and stop detailed USRXIT tracing. Use the data from this sublevel in conjunction with MSGTRC trace data to get information about message processing through installation exits IXZXIT01, IXZXIT02, and IXZXIT03.

- **FLOW, module footprint tracing:** FLOW records messages and events as they flow through the JES common coupling services component. By default, FLOW is always active and produces detailed tracing.

**Note:** IBM recommends that this trace always remain active to record diagnostic data such as errors, system state changes, and processing events.

- **XCFEVT, system event (SYSEVENT) tracing:** XCFEVT records SYSEVENT data processed by the JES common coupling services component. By default, XCFEVT always produces minimal tracing.

Tracing for SYSJES can run all 4 sublevels concurrently. USRXIT and MSGTRC trace only error events by default; you can turn on detailed tracing for these two sublevels.

### Requesting a SYSJES trace

IBM recommends the following when requesting SYSJES TRACING:

- Start and stop the four sublevels for a system all at once in one parmlib member. Request SYSJES component tracing in a CTnJESxx parmlib member which you specify on a TRACE CT command.
IBM provides two parmlib members, IXZCTION and IXZCTIOF, in SYS1.SAMPLIB as examples of how to start and stop SYSJES sublevels. Copy the members into parmlib, and rename them CTIJESON and CTIJESOF. The CTIJESON parmlib member starts all the sublevels and connects them to the external writer. The CTIJESOF parmlib member stops tracing in all sublevels and disconnects them from the external writer.

- Use the external writer for gathering trace records, because SYSJES tracing produces a large volume of data. Create source JCL for the external writer, using the following guidelines:
  - Code all TRCOUTnn DD statements with a SPACE parameter of at least 10 cylinders to accommodate the volume of SYSJES trace data.
  - For traces larger than 10 cylinders, specify a unique volser for each TRCOUTnn statements if you need to reduce I/O contention on one volume.
  - The data set name defined in the TRCOUT01 DD statement must be unique on each system.
  - Use the IPCS COPYTRC command to merge records from multiple TRCOUTnn DD statements into one data set. See z/OS MVS IPCS Commands for information.

The following example shows an external writer procedure, IXZCTW, that sends SYSJES trace output to trace data sets.

```plaintext
//CTWDASD PROC
//IEFPROC EXEC PGM=ITTTRCWR
//SYSPRINT DD SYSOUT=A
//TRCOUT01 DD DSN=SYS1.JESXCF1,VOL=SER=TRACE6,UNIT=DASD,
//      SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
//TRCOUT02 DD DSN=SYS1.JESXCF2,VOL=SER=TRACE7,UNIT=DASD,
//      SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
```

Figure 126. Example: Cataloged procedure for SYSJES

- If you are tracing in a sysplex environment, the data set names on TRCOUTnn DD statements must be unique throughout the sysplex. An ENQUEUE error results if the data set names are not unique.

**CTnJESxx parmlib member**
The following table indicates the parameters you can specify on a CTnJESxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnJESxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>No</td>
</tr>
<tr>
<td>SUB</td>
<td>Yes</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Component Trace

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>No</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>Yes</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples of requesting SYSJES traces

- Example 1: Start SYSJES tracing with the CTIJESON member
  The following example shows the CTIJESON parmlib member supplied in SYS1.SAMPLIB to start tracing for one system:

```
TRACEOPTS
WTRSTART(IXZCTW) WRAP
SUB(MSGTRC)
ON
WTR(IXZCTW)
SUB(USRXIT)
ON
WTR(IXZCTW)
SUB(FLOW)
ON
WTR(IXZCTW)
SUB(XCFEVT)
ON
WTR(IXZCTW)
```

- Example 2: Stop SYSJES tracing with the CTIJESOF member
  The following example shows the CTIJESOF parmlib member supplied in SYS1.SAMPLIB to stop tracing for one system:

```
TRACEOPTS
SUB(MSGTRC)
OFF
SUB(USRXIT)
OFF
SUB(FLOW)
OFF
SUB(XCFEVT)
OFF
```

Stop tracing with the following command:
```
TRACE CT,OFF,COMP=SYSJES,PARM=CTIJESOF
```
Then specify the following command to stop the external writer (assuming IXZCTW is the membername of the source JCL for the external writer):

```
TRACE CT, WTRSTOP=IXZCTW
```

**Requesting a SYSJES trace for problems during initialization**

Use this procedure only when requested to by the IBM Support Center. The procedure requests SYSJES tracing for JES XCF problems occurring during JES subsystem initialization. The procedure consists of using the default parmlib members CTIJES01, CTIJES02, and CTIJES04 to request tracing of these SYSJES sublevels. Note that parmlib member CTIJES03 contains module footprint tracing that is active, by default, on your system. Therefore, you do not need to take any action to modify this trace.

Activating all four traces can negatively impact system performance because of the heavy volume of trace data produced. For that reason, you should only use this procedure when requested by IBM, and you should not leave this full tracing on. The identical parmlib members are supplied with tracing set off, since you should only run with full tracing at IBM’s request. The default contents of parmlib members CTIJES01, CTIJES02, and CTIJES04 is:

```
TRACEOPTS
OFF
```

The default contents of parmlib members CTIJES03 is:

```
TRACEOPTS
ON
```

IBM recommends that you keep this tracing sublevel on at all times.

1. **Modify the CTIJES01, CTIJES02, and CTIJES04 parmlib members to turn tracing on:** In parmlib members CTIJES01, CTIJES02, and CTIJES04, alter the parmlib members to return sublevel tracing on and connect the sublevel to the external writer. The parmlib members are supplied with tracing off and no connection to the writer.

When you initialize the JES subsystem with the modified parmlib members, full tracing for JES XCF starts automatically.

**Figure 127** is an example of the CTIJESxx parmlib member after having been modified for gathering trace data during JES subsystem initialization at the direction of the IBM Support Center. The member turns tracing on for the sublevel and connects the sublevel to the external writer.

```
TRACEOPTS
WTRSTART(IXZCTW)
ON
WTR(IXZCTW)
```

**Figure 127. Example: Turning on tracing in a CTIJESxx member**

2. **Create a CTIJESOF parmlib member to stop SYSJES tracing:** Use the CTIJESOF parmlib member to stop the full SYSJES tracing turned on during initialization and to disconnect them from the external writer.
3. **Stop SYSJES tracing after initialization tracing is complete:** Enter a TRACE CT operator command referencing the CTIJESOF parmlib member on the console with master authority as follows:

```
TRACE CT,OFF,COMP=SYSJES,PARM=CTIJESOF
```

4. **Remodify the CTIJES01, CTIJES02, and CTIJES04 parmlib members to return to default Tracing:** In parmlib members CTIJES01, CTIJES02, and CTIJES04, alter the parmlib member to return sublevel tracing to off.

Figure 128 shows the CTIJES01, CTIJES02, and CTIJES04 parmlib members after having been returned to their original contents, with tracing set off.

---

**Formatting a SYSJES trace**

Format the trace with an IPCS CTRACE COMP(SYSJES) subcommand. To format SYSJES tracing, you must:

- Enter the CTRACE command for SYSJES once for each of the four sublevels you wish to format. See "Format SYSJES sublevel information."
- Specify SYSJES options on the OPTIONS parameter. See "OPTIONS parameter for formatting a SYSJES trace."
- Merge the output from the different sublevels requested. See "Merging SYSJES information from sublevels" on page 463.

For SYSJES traces, use the IPCS MERGE subcommand to display traces that are not likehead in timestamp order.

**Format SYSJES sublevel information**

You must enter the CTRACE command separately for each SYSJES sublevel you wish to format. For example, to request formatting of SYSJES trace data for sublevels MSGTRC and USRXIT, you would enter the following two commands:

```
CTRACE COMP(SYSJES) SUB((USRXIT)) FULL
CTRACE COMP(SYSJES) SUB((MSGTRC)) FULL
```

These examples would yield tracing without any options requested.

**OPTIONS parameter for formatting a SYSJES trace**

IBM might request that you enter options for SYSJES tracing. You can specify options for SYSJES tracing on the OPTIONS parameter of the CTRACE command. The options include:

- Options valid for all sublevels:
  - MSGTOKEN=msgid
  - REQTOKEN=reqtoken
  - MSGBUF=msgbuf
  - CTCR=ctcr
- Options valid for the FLOW sublevel only:
  - MODID=id
  - MODFLOW
MSGFLOW

Options valid for the MSGTRC sublevel only:
- SEND
- RECEIVE

Options valid for the USRXIT sublevel only:
- EXIT1
- EXIT3
- EXIT2

Merging SYSJES information from sublevels
Because SYSJES can run four sublevel traces simultaneously, you will need to merge the data for a complete chronological picture of SYSJES trace data. For example, to merge JESXCF trace data, from all four sublevels, enter the following command:

```
MERGE
CTRACE COMP(SYSJES) SUB((USRXIT)) FULL
CTRACE COMP(SYSJES) SUB((MSGTRC)) FULL
CTRACE COMP(SYSJES) SUB((XCFEVT)) FULL
CTRACE COMP(SYSJES) SUB((FLOW)) FULL
MERGEEND
```

You can write an IPCS CLIST to issue the CTRACE command for the four sublevels and merge the output automatically. See "z/OS MVS IPCS Customization" for information on writing a CLIST.

Output from a SYSJES trace

"Example: merged output from four SYSJES sublevel traces with FULL parameter" is an example of merged output from the four SYSJES sublevel traces with the FULL parameter specified.

Example: merged output from four SYSJES sublevel traces with FULL parameter

```
********** MERGED TRACES **********
01. CTRACE comp(sysjes) sub((flow)) full
02. CTRACE comp(sysjes) sub((usrxit)) full
03. CTRACE comp(sysjes) sub((msgtrc)) full
04. CTRACE comp(sysjes) sub((xcfevt)) full
```

```
COMPONENT TRACE FULL FORMAT
SYSNAME(SY1)
COMP(SYSJES) SUBNAME((FLOW))
```

```
COMPONENT TRACE FULL FORMAT
SYSNAME(SY1)
COMP(SYSJES) SUBNAME((USRXIT))
```

```
COMPONENT TRACE FULL FORMAT
SYSNAME(SY1)
COMP(SYSJES) SUBNAME((MSGTRC))
```

```
COMPONENT TRACE FULL FORMAT
SYSNAME(SY1)
COMP(SYSJES) SUBNAME((XCFEVT))
```
### Component Trace

#### 10/25/93

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>04. XCFEV</td>
<td>00000030</td>
<td>16:53:54.876999</td>
<td>XCF event message buffer</td>
</tr>
<tr>
<td>HOMEASID</td>
<td>0013</td>
<td>JOBNAMEx JESx CF</td>
<td>HOMETCB@ 007E0A6F8</td>
</tr>
<tr>
<td>CPUID...</td>
<td>FF710945</td>
<td>CTCR@.. 03062460</td>
<td>MEMBER.. SY1</td>
</tr>
<tr>
<td>SYSNAME.</td>
<td>SY1</td>
<td>MSGBUF@. 20000014</td>
<td>ALET.... 0101001B</td>
</tr>
<tr>
<td>REQTOKEN</td>
<td>00000001</td>
<td>20000001</td>
<td>MSGTOKEN 00000001</td>
</tr>
<tr>
<td>+0000</td>
<td>E8C9E7C5 D5400100</td>
<td>9999999999</td>
<td>YIXEN ........</td>
</tr>
<tr>
<td>+0010</td>
<td>E2E8E2E9 D1C5E2F3</td>
<td>9999999999</td>
<td>SYSZJES3SYSEVENT</td>
</tr>
<tr>
<td>+0020</td>
<td>40404040 40404040</td>
<td>9999999999</td>
<td>SYSEVENT</td>
</tr>
<tr>
<td>+0030</td>
<td>40404040 40404040</td>
<td>9999999999</td>
<td>SYSEVENT</td>
</tr>
<tr>
<td>+0040</td>
<td>40404040 40404040</td>
<td>9999999999</td>
<td>SYSEVENT</td>
</tr>
<tr>
<td>+0050</td>
<td>40404040 40404040</td>
<td>9999999999</td>
<td>SYSEVENT</td>
</tr>
</tbody>
</table>

#### 10/25/93

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>02. PREXIT3</td>
<td>00000007</td>
<td>16:53:55.303610</td>
<td>SPELL prior to Exit 3 processing</td>
</tr>
<tr>
<td>HOMEASID</td>
<td>0017</td>
<td>JOBNAMEx JES3</td>
<td>HOMETCB@ 007FD2B8</td>
</tr>
<tr>
<td>CPUID...</td>
<td>FF710945</td>
<td>CTCR@.. 03062460</td>
<td>MEMBER.. SY1</td>
</tr>
<tr>
<td>SYSNAME.</td>
<td>SY1</td>
<td>MSGBUF@. 03368000</td>
<td>ALET.... 0102001B</td>
</tr>
<tr>
<td>REQTOKEN</td>
<td>00000001</td>
<td>1FFFF814</td>
<td>MSGTOKEN 1FFFF814</td>
</tr>
<tr>
<td>+0000</td>
<td>E8C9E7C5 D5400100</td>
<td>9999999999</td>
<td>YIXEN ........</td>
</tr>
<tr>
<td>+0010</td>
<td>E2E8E2E9 D1C5E2F3</td>
<td>9999999999</td>
<td>SYSZJES3SYSEVENT</td>
</tr>
<tr>
<td>+0020</td>
<td>40404040 40404040</td>
<td>9999999999</td>
<td>SYSEVENT</td>
</tr>
<tr>
<td>+0030</td>
<td>40404040 40404040</td>
<td>9999999999</td>
<td>SYSEVENT</td>
</tr>
<tr>
<td>+0040</td>
<td>40404040 40404040</td>
<td>9999999999</td>
<td>SYSEVENT</td>
</tr>
<tr>
<td>+0050</td>
<td>40404040 40404040</td>
<td>9999999999</td>
<td>SYSEVENT</td>
</tr>
</tbody>
</table>

#### 10/25/93

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>03. INDATA</td>
<td>00000011</td>
<td>16:55:17.318441</td>
<td>input message data</td>
</tr>
<tr>
<td>HOMEASID</td>
<td>0017</td>
<td>JOBNAMEx JES3</td>
<td>HOMETCB@ 007FD2B8</td>
</tr>
<tr>
<td>CPUID...</td>
<td>FF710945</td>
<td>CTCR@.. 03062460</td>
<td>MEMBER.. SY1</td>
</tr>
</tbody>
</table>

---

464  z/OS V2R2 MVS Diagnosis: Tools and Service Aids
Figure 129 on page 466 is an example of merged output from the four SYSJES sublevel traces with the TALLY parameter specified. Use the TALLY report to look at event identifiers in the trace output.
### Component Trace

<table>
<thead>
<tr>
<th>FMTID</th>
<th>COUNT</th>
<th>INTERVAL</th>
<th>MNEMONIC</th>
<th>DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000020</td>
<td>29</td>
<td>6,902,004</td>
<td>MSGFLOW</td>
<td>module flow for this message</td>
</tr>
<tr>
<td>00000021</td>
<td>114</td>
<td>1,755,686</td>
<td>MODFLOW</td>
<td>module flow</td>
</tr>
<tr>
<td>0000002E</td>
<td>3</td>
<td>2,631,973</td>
<td>MSGERR</td>
<td>module error flow</td>
</tr>
<tr>
<td>0000002F</td>
<td>7</td>
<td>27,207,632</td>
<td>MODERR</td>
<td>module error flow</td>
</tr>
</tbody>
</table>

Total trace entries: 153

### Component Trace

<table>
<thead>
<tr>
<th>FMTID</th>
<th>COUNT</th>
<th>INTERVAL</th>
<th>MNEMONIC</th>
<th>DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>13</td>
<td>17,076,718</td>
<td>SEND</td>
<td>message queued to be sent</td>
</tr>
<tr>
<td>00000011</td>
<td>17</td>
<td>12,821,783</td>
<td>INDATA</td>
<td>input message data</td>
</tr>
<tr>
<td>00000012</td>
<td>0</td>
<td>0</td>
<td>SENDBUF</td>
<td>buffer sent by XCF</td>
</tr>
<tr>
<td>00000013</td>
<td>0</td>
<td>0</td>
<td>SENDQERR</td>
<td>Sendq queueing error</td>
</tr>
<tr>
<td>00000014</td>
<td>0</td>
<td>0</td>
<td>SBUFERR</td>
<td>Send error from XCF</td>
</tr>
<tr>
<td>00000018</td>
<td>13</td>
<td>17,061,037</td>
<td>RECEIVE</td>
<td>msg dequeued from receive</td>
</tr>
<tr>
<td>00000019</td>
<td>10</td>
<td>30,944,882</td>
<td>OUTDATA</td>
<td>output message data</td>
</tr>
<tr>
<td>0000001A</td>
<td>26</td>
<td>8,192,076</td>
<td>RECVBUF</td>
<td>buffer received from XCF</td>
</tr>
<tr>
<td>0000001B</td>
<td>0</td>
<td>0</td>
<td>RBUFERR</td>
<td>Receive error from XCF</td>
</tr>
<tr>
<td>0000001C</td>
<td>0</td>
<td>0</td>
<td>RECVMERR</td>
<td>Receiveq queueing error</td>
</tr>
</tbody>
</table>

Total trace entries: 79

### Component Trace

<table>
<thead>
<tr>
<th>FMTID</th>
<th>COUNT</th>
<th>INTERVAL</th>
<th>MNEMONIC</th>
<th>DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000030</td>
<td>27</td>
<td>35,794,857</td>
<td>XCFEVT</td>
<td>XCF event message buffer</td>
</tr>
<tr>
<td>00000031</td>
<td>0</td>
<td>0</td>
<td>XCFERR</td>
<td>XCF event message error</td>
</tr>
</tbody>
</table>

Total trace entries: 27

### Component Trace

<table>
<thead>
<tr>
<th>FMTID</th>
<th>COUNT</th>
<th>INTERVAL</th>
<th>MNEMONIC</th>
<th>DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>1</td>
<td>PREXIT1</td>
<td>SPELL prior to Exit 1 processing</td>
<td></td>
</tr>
<tr>
<td>00000002</td>
<td>0</td>
<td>POSTXIT1</td>
<td>SPELL after Exit 1 processing</td>
<td></td>
</tr>
<tr>
<td>00000003</td>
<td>1</td>
<td>EXITIERR</td>
<td>SPELL error Exit 1 processing</td>
<td></td>
</tr>
<tr>
<td>00000004</td>
<td>1</td>
<td>PREXIT2</td>
<td>SPELL prior to Exit 2 processing</td>
<td></td>
</tr>
<tr>
<td>00000005</td>
<td>0</td>
<td>POSTXIT2</td>
<td>SPELL after Exit 2 processing</td>
<td></td>
</tr>
<tr>
<td>00000006</td>
<td>1</td>
<td>EXIT2ERR</td>
<td>SPELL error Exit 2 processing</td>
<td></td>
</tr>
<tr>
<td>00000007</td>
<td>14</td>
<td>60,329,405</td>
<td>PREXIT3</td>
<td>SPELL prior to Exit 3 processing</td>
</tr>
<tr>
<td>00000008</td>
<td>14</td>
<td>60,374,425</td>
<td>POSTXIT3</td>
<td>SPELL after Exit 3 processing</td>
</tr>
<tr>
<td>00000009</td>
<td>0</td>
<td>EXIT3ERR</td>
<td>SPELL error Exit 3 processing</td>
<td></td>
</tr>
</tbody>
</table>

Total trace entries: 32

Figure 129. Example: merged output from the four SYSJES sublevel traces with the TALLY parameter
SYSjes2 component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSjes2 component trace for the JES2 subsystem. For ease of explanation here, this component trace is referred to as **SYSjes2** although you might need to replace *jes2* with the name you assigned to your JES2 subsystem (primary or secondary). For example, to obtain trace information for JESA, a name you might have used to name your secondary JES2 in a poly-JES environment, use **SYSJESA** as the component name.

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSjes2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>n/a</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; full tracing for sublevels JQE and JOE</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>None</td>
</tr>
<tr>
<td>Buffer</td>
<td></td>
</tr>
<tr>
<td>Default: N/A</td>
<td></td>
</tr>
<tr>
<td>Range: N/A</td>
<td></td>
</tr>
<tr>
<td>Size set by: JES2</td>
<td></td>
</tr>
<tr>
<td>Change size after IPL: No</td>
<td></td>
</tr>
<tr>
<td>Location: In the component address space</td>
<td></td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By the component, or DUMP or SLIP</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSjes2)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>

SYSjes2 tracing is started automatically during initialization. SYSjes2 contains 2 sublevel traces, which run continuously and concurrently. The sublevels are:

- **JQE service tracing**: JQE records all job queue service calls (to include: $QADD, $QPUT, $QREM, $QMOD, $QIX, $GETJLOK, $QRBDCHK, $QBUSY, $GETLOKW, $FREJLOK, and $FRELOKW).
- **JOE service tracing**: JOE records all job output element service calls (to include: $#ADD, $#PUT, $#REM, $#MOD, $#RBDCHK, $#BUSY, $#GET, and $#CAN).

**Requesting a SYSjes2 trace**

You need not take any action to request a SYSjes2 trace. The trace is active whenever your JES2 subsystem is in control.

**Formatting SYSjes2 sublevel trace Information**

You must enter the CTRACE command separately for each SYSJES sublevel you wish to format. For example, to request formatting of SYSJES trace data for sublevels JQE and JOE, you would enter the following two commands:

CTRACE COMP(SYSjes2) SUB((JQE)) FULL
CTRACE COMP(SYSjes2) SUB((JOE)) FULL
Merging SYSjes2 information from sublevels

Because SYSjes2 runs two sublevel traces simultaneously, you will need to merge the data for a complete chronological picture of SYSjes2 trace data. To merge SYSjes2 trace data, enter the following command string:

```
MERGE
CTRACE COMP(SYSjes2) SUB((JQE)) FULL
CTRACE COMP(SYSjes2) SUB((JQE)) FULL
MERGEND
```

You can write an IPCS CLIST to issue the CTRACE command for both sublevels and merge the output automatically. See [z/OS MVS IPCS Customization](#) for information on writing a CLIST.

**Output from a SYSjes2 trace**

The output from the SYSjes2 trace contains a maximum of 500 trace entries presented in a wrapping (or rolling) trace format. That is, once the trace table is filled with 500 entries, the next entries (501, 502, 503,...) overwrite entries 1, 2, 3... in a continuous wrapping manner.

[Figure 130](#) is an example of merged output from both SYSjes2 sublevel traces with the FULL parameter specified.

```
********** MERGED TRACES **********
01. CTRACE comp(sysjes2) sub((jqe)) full
02. CTRACE comp(sysjes2) sub((joe)) full
```

[Figure 130. Example: merged output from both SYSjes2 sublevel traces with FULL parameter](#)

[Figure 131](#) is an example of merged output from both SYSjes2 sublevel traces with the SHORT parameter specified:

```
***********************short format screen **********************
COMPONENT TRACE SHORT FORMAT
COMP(SYSjes2) SUBNAME((JQE))
**** 07/11/94
SYSTYPE MNEMONIC ENTRY ID TIME STAMP DESCRIPTION
------- -------- -------- --------------- -------------
MVS1 JQE 00000200 14:33:35.162400 $QADD
MVS1 JQE 00000204 14:33:35.162404 $QJIX (ALLOC new number)
MVS1 JQE 00000200 14:33:36.174242 $QADD
```

[Figure 131. Example: merged output from both SYSjes2 sublevel traces with SHORT parameter](#)

The following is an example of merged output from both SYSjes2 sublevel traces with the FULL parameter specified. Use mapping macro, $ROTT (rolling trace table), to map the fields presented in this trace.
**Component Trace**

---------------------

**********full format screen ***********

IPCS OUTPUT STREAM ---------------------------- Line 0 Co
********** TOP OF DATA **********

COMPONENT TRACE FULL FORMAT
COMP(SYSjes2) SUBNAME((JQE))
**** 07/11/94

<table>
<thead>
<tr>
<th>SYSNAME</th>
<th>MNEONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVS1</td>
<td>JQE</td>
<td>00000200</td>
<td>14:33:35.162400</td>
<td>$QADD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>03B06C20</td>
<td>00000000 0000064</td>
<td>FF200100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00000000 0000</td>
<td>......</td>
</tr>
<tr>
<td>MVS1</td>
<td>JQE</td>
<td>00000204</td>
<td>14:33:35.162404</td>
<td>$QJIX (ALLOC new number)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>03B06C20</td>
<td>00000001 0000064</td>
<td>20200100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>04000000 0000</td>
<td>......</td>
</tr>
<tr>
<td>MVS1</td>
<td>JQE</td>
<td>00000200</td>
<td>14:33:36.174242</td>
<td>$QADD</td>
</tr>
</tbody>
</table>

------------------------------------------------------------------------

The following is an example of merged output from both SYSjes2 sublevel traces with the TALLY parameter specified:

---------------------

**********tally format screen ***********

COMPONENT TRACE TALLY REPORT
COMP(SYSjes2) SUBNAME((JQE))
TRACE ENTRY COUNTS AND AVERAGE INTERVALS (IN MICROSECONDS)

<table>
<thead>
<tr>
<th>FMTID</th>
<th>COUNT</th>
<th>INTERVAL</th>
<th>MNEONIC</th>
<th>DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000200</td>
<td>12</td>
<td>96,278,898</td>
<td>JQE</td>
<td>$QADD</td>
</tr>
<tr>
<td>00000201</td>
<td>0</td>
<td>JQE</td>
<td>$QPUT</td>
<td></td>
</tr>
<tr>
<td>00000202</td>
<td>0</td>
<td>JQE</td>
<td>$QREM</td>
<td></td>
</tr>
<tr>
<td>00000203</td>
<td>30</td>
<td>36,473,953</td>
<td>JQE</td>
<td>$QMOD</td>
</tr>
<tr>
<td>00000204</td>
<td>12</td>
<td>96,278,898</td>
<td>JQE</td>
<td>$QJIX (ALLOC new number)</td>
</tr>
<tr>
<td>00000205</td>
<td>0</td>
<td>JQE</td>
<td>$QJIX (SWAP job numbers)</td>
<td></td>
</tr>
<tr>
<td>00000206</td>
<td>5</td>
<td>273,117,249</td>
<td>JQE</td>
<td>$GETJLOK</td>
</tr>
<tr>
<td>00000207</td>
<td>5</td>
<td>274,068,170</td>
<td>JQE</td>
<td>$FREJLOK</td>
</tr>
</tbody>
</table>

Total trace entries: 89

******** END OF DATA ********

**SYSLLA component trace**

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSLLA component trace for library lookaside (LLA) of contents supervision.
Component Trace

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSLLA:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>None</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; always on when LLA is running</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>None</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: N/A</td>
</tr>
<tr>
<td></td>
<td>• Range: N/A</td>
</tr>
<tr>
<td></td>
<td>• Size set by: MVS system</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: No</td>
</tr>
<tr>
<td></td>
<td>• Location: In the component address space</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By the component</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSLLA)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>None</td>
</tr>
</tbody>
</table>

Requesting a SYSLLA trace

The trace runs whenever LLA is in control. No actions are needed to request it.

Formatting a SYSLLA trace

Format the trace with an IPCS CTRACE COMP(SYSLLA) subcommand. The subcommand has no OPTIONS values.

SYSLOGR component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

Table 68 summarizes information for requesting a SYSLOGR component trace for the system logger component. SYSLOGR tracing is started during initialization.

Table 68. Summary of SYSLOGR component trace request

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSLOGR:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CtnLOGxx</td>
</tr>
<tr>
<td></td>
<td>Default member: CTILOG00, is required and provided as a component trace option in GRSCNFxx parmib member.</td>
</tr>
<tr>
<td></td>
<td>1. IXGCNFxx member CTRACE field specified at IPL or SET IXGCNF command</td>
</tr>
<tr>
<td></td>
<td>2. SETLOGR CTRACE command</td>
</tr>
<tr>
<td></td>
<td>3. TRACE command.</td>
</tr>
<tr>
<td>Parmlib default tracing options</td>
<td>CONNECT, LOGSTRM, DATASET, SERIAL, MISC, LOCBUFF, RECOVERY</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; activated during System Logger (IXGLOGR) address space initialization</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CtnLOGxx and REPLY for TRACE command</td>
</tr>
</tbody>
</table>

SYSLOGR component trace
Table 68. Summary of SYSLOGR component trace request (continued)

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSLOGR:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>• Parmlib default: 16MB</td>
</tr>
<tr>
<td></td>
<td>• Default: 2MB</td>
</tr>
<tr>
<td></td>
<td>• Range: 2MB - 2047MB</td>
</tr>
<tr>
<td></td>
<td>• Size set by: CTnLOGxx parmlib member or REPLY for TRACE CT command.</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: Yes</td>
</tr>
<tr>
<td></td>
<td>• Location: system logger trace data space</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer; system logger trace data space, trace data set</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSLOGR)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Obtaining a dump of system logger information

Use the following examples to obtain the appropriate diagnostic information for system logger. The amount of information requested in the dumps may be very large. You may need to set your MAXSPACE on the CHNGDUMP setting to 999 mb before obtaining the logger dumps.

CD SET,SDUMP,MAXSPACE=999M

For structure dumps, verify that the coupling facility has dump space defined by issuing the following command:

D CF,CFNAME=xxxx

Note: There are several sample Logger dump parmlib members that can be used as models for automating the procedures listed below. The samples parmlib members are shipped in 'SYS1.SAMPLIB(IEADMCLx)'. Refer to the IEADMCLx members for more information, or see Table 7 on page 20.

1. For all types of logstreams, always include the following:
   a. The IXGLOGR (Logger) asid and the data spaces associated with the IXGLOGR asid through the DSPNAME parm. These will be dumped using the JOBNAME= parm.
   b. The trace data space SYSLOGR0, will be included in the dump if DSNNAME=('IXGLOGR.*) is specified in the reply for the dump command.

   Note: If you are running OS/390® Release 2 or lower, Logger will not have a SYSLOGR* data space for tracing, in which case the DSPNAME=('IXGLOGR.*) option can be omitted.

   DUMP COMM=(dump system logger with component trace)
   * rr IEEE94D SPECIFY OPERAND(S) FOR DUMP COMMAND
   rr, JOBNAME=(IXGLOGR), CONT
   ss, DSPNAME=('IXGLOGR.*SYSLOGR*), CONT
   tt, SDATA=(COUPLE, ALLNUC, LPA, LSQA, PSA, RGN, SQA, SUM, TRT, ESA, GRSQ, XESDATA), END

2. When using CF list structure based log streams, include the following
   a. The XCF asid and trace data spaces. These will be dumped using the JOBNAME= parm and DSPNAME parm.
b. The XES STRUCTURE data. This is dumped using the STRLIST= parameter and by specifying the structure name. structure_name is the affected STRUCTURE name.

**Note:** Be sure to allocate adequate DUMPSPACE() as defined in your CF definition in the CFRI policy. If you do not allocate adequate space, all or part of the STRUCTURE will NOT be dumped.

c. In the case of “loss of data” or “block not found”, dumping the OFFLOAD data sets using IDCAMS is a good idea.

DUMP COMM=(dump system logger with xes and structure data)
* rr IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
rr, JOBNAME=(IXGLOGR,XCFAS), CONT
ss, DSPNAME=('XCFAS'.*, 'IXGLOGR'.SYSLOGR+), CONT
tt, SDATA=(COUPLE, ALLNUC, LPA, LSQA, PSA, RGN, SQA, SUM, TRT, CSA, GRSQ, XESDATA), CONT
uu, STRLIST=(STRNAME=structure_name, LOCKENTRIES, ACC=NOLIM,
(LISTNUM=ALL, ADJUNCT=DIRECTIO, ENTRYDATA=UNSERIALIZE)), END

3. When system logger dumps are needed on multiple systems in the sysplex, include the REMOTE parameter.

DUMP COMM=(dumps for system logger around the sysplex)
* rr IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
rr, JOBNAME=(IXGLOGR, XCFAS), CONT
ss, DSPNAME=('XCFAS'.*, 'IXGLOGR'.SYSLOGR+), CONT
tt, SDATA=(COUPLE, ALLNUC, LPA, LSQA, PSA, RGN, SQA, SUM, TRT, CSA, GRSQ, XESDATA), CONT
uu, STRLIST=(STRNAME=structure_name, LOCKENTRIES, ACC=NOLIM,
(LISTNUM=ALL, ADJUNCT=DIRECTIO, ENTRYDATA=UNSERIALIZE)), CONT
vv, REMOTE=(SYSLIST=*'XCFAS', 'IXGLOGR'), DSPNAME, SDATA), END

4. Other diagnostic considerations
The CTILOG00 buffer size is 16MB. It is strongly recommended that this value not be lowered. See the CTnLOGxx member, described in Table 68 on page 470

Remember that much of the data that system logger uses is persistent across an IPL. That means that if this data is corrupted and adversely affects system logger, an IPL will not correct the problem. In the case of a persistent system logger failure, you can FORCE the IXGLOGR address space. Prior to doing this you should bring down all of the applications connected. Then issue the FORCE command (FORCE IXGLOGR, ARM) and restart system logger using the supplied procedure in SYS1.PROCLIB (IXGLOGRS). (S IXGLOGRS)

If FORCE IXGLOGR, ARM does not resolve the situation, an IPL is not likely to either. This is the time to take a dump if one was not already taken by system logger.

**Note:**

a. A CICS dump will not dump the IXGLOGR address space. Connect to a new (non-corrupted) LOGSTREAM. This will result in a LOSS OF DATA for some applications such as CICS, forcing them to INITIAL START. However, this may be the only way to get the application restarted. Connecting to a new logstream (of a different name) will allow the corrupted data to remain in the structure for diagnostic review later.

b. In preparation for connecting to this new LOGSTREAM, you may want to define an unused LOGSTREAM to each STRUCTURE during setup.

If running CICS, always run with the following SLIP:

SLIP SET, COMP=1CS, REASON=804,
STRLIST=(STRNAME=stringname1, LOCKE, ACC=NOLIM,
(LNUM=ALL, EDATA=SER, ADJ=CAP)),

---

**Component Trace**

b. The XES STRUCTURE data. This is dumped using the STRLIST= parameter and by specifying the structure name. structure_name is the affected STRUCTURE name.

**Note:** Be sure to allocate adequate DUMPSPACE() as defined in your CF definition in the CFRI policy. If you do not allocate adequate space, all or part of the STRUCTURE will NOT be dumped.

c. In the case of “loss of data” or “block not found”, dumping the OFFLOAD data sets using IDCAMS is a good idea.

DUMP COMM=(dump system logger with xes and structure data)
* rr IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
rr, JOBNAME=(IXGLOGR, XCFAS), CONT
ss, DSPNAME=('XCFAS'.*, 'IXGLOGR'.SYSLOGR+), CONT
tt, SDATA=(COUPLE, ALLNUC, LPA, LSQA, PSA, RGN, SQA, SUM, TRT, CSA, GRSQ, XESDATA), CONT
uu, STRLIST=(STRNAME=structure_name, LOCKENTRIES, ACC=NOLIM,
(LISTNUM=ALL, ADJUNCT=DIRECTIO, ENTRYDATA=UNSERIALIZE)), END

3. When system logger dumps are needed on multiple systems in the sysplex, include the REMOTE parameter.

DUMP COMM=(dumps for system logger around the sysplex)
* rr IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
rr, JOBNAME=(IXGLOGR, XCFAS), CONT
ss, DSPNAME=('XCFAS'.*, 'IXGLOGR'.SYSLOGR+), CONT
tt, SDATA=(COUPLE, ALLNUC, LPA, LSQA, PSA, RGN, SQA, SUM, TRT, CSA, GRSQ, XESDATA), CONT
uu, STRLIST=(STRNAME=structure_name, LOCKENTRIES, ACC=NOLIM,
(LISTNUM=ALL, ADJUNCT=DIRECTIO, ENTRYDATA=UNSERIALIZE)), CONT
vv, REMOTE=(SYSLIST=*'XCFAS', 'IXGLOGR'), DSPNAME, SDATA), END

4. Other diagnostic considerations
The CTILOG00 buffer size is 16MB. It is strongly recommended that this value not be lowered. See the CTnLOGxx member, described in Table 68 on page 470

Remember that much of the data that system logger uses is persistent across an IPL. That means that if this data is corrupted and adversely affects system logger, an IPL will not correct the problem. In the case of a persistent system logger failure, you can FORCE the IXGLOGR address space. Prior to doing this you should bring down all of the applications connected. Then issue the FORCE command (FORCE IXGLOGR, ARM) and restart system logger using the supplied procedure in SYS1.PROCLIB (IXGLOGRS). (S IXGLOGRS)

If FORCE IXGLOGR, ARM does not resolve the situation, an IPL is not likely to either. This is the time to take a dump if one was not already taken by system logger.

**Note:**

a. A CICS dump will not dump the IXGLOGR address space. Connect to a new (non-corrupted) LOGSTREAM. This will result in a LOSS OF DATA for some applications such as CICS, forcing them to INITIAL START. However, this may be the only way to get the application restarted. Connecting to a new logstream (of a different name) will allow the corrupted data to remain in the structure for diagnostic review later.

b. In preparation for connecting to this new LOGSTREAM, you may want to define an unused LOGSTREAM to each STRUCTURE during setup.

If running CICS, always run with the following SLIP:

SLIP SET, COMP=1CS, REASON=804,
STRLIST=(STRNAME=stringname1, LOCKE, ACC=NOLIM,
(LNUM=ALL, EDATA=SER, ADJ=CAP)),
Component Trace

\( JL=(XCFAS,IXGLOGR), DN=\{"XCFAS"*,"IXGLOGR"*\}, \)
\( SD=\{COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,TRT,CSA,GRSQ, \)
\( XESDATA,SUM\}, \)
\( SUMLIST=\{13R?-7FFF,13R?+7FFF\}, END \)

**Note**: You might add a **JOBNAME=DFH** to limit SLIP to CICS. A RSN804 is a “block not found”, which is always bad for CICS but not necessarily so for other applications. Setting this SLIP will cause system logger to dump on all RSN804s in CICS.

5. Frequent stumbling blocks
   a. OFFLOAD data sets must be VSAM SHAREOPTIONS(3,3) unless you are in a MONOPLEX.
   b. After OW33261, system logger recommends for performance reasons using 24K CI size for OFFLOAD data sets. Staging data sets must remain at 4K CI size. Code your ACS routines appropriately.
   c. Size of XES structures. “Bigger is not always better.” Follow exploiting application recommendations.
   d. Allow for OFFLOAD directory extents. Reference “Format Utility for Couple Data Sets” in [IBM z/OS MVS Setting Up a Sysplex](#).
   e. System logger uses HSM services to recall (ARCHRCAL) and to delete (ARCHDEL) offload data sets. HSM contention or a wait for a WTOR such as ARC0055A can hang all of the log streams that require the system logger allocation task.

### Requesting a SYSLOGR trace

Specify options for requesting a SYSLOGR component trace in a CTnLOGxx parmlib member or on the reply for a TRACE CT command.

Also, refer to the following as alternatives for specifying the CTnLOGxx parmlib member to be used:
- IXGCNFxx SYS1.PARMLIB member (see IXGCNFXX SYS1.PARMLIB member).
- SET IXGCNF=xx command.
- SETLOGR CTRACE command.

Specify options for requesting a SYSLOGR component trace in a CTnLOGxx parmlib member or on the reply for a TRACE CT command.

You can change the options and the trace data space buffer size for SYSLOG trace while the trace is running. However, if the SYSLOG trace has not been connected to an external writer and you are reducing the size of the trace data space buffer, you **must** dump the contents of the buffer (see “Obtaining a dump of system logger information” on page 471) **before** reducing the buffer size if this data is important for debugging. Trace data in the trace data space is discarded when the buffer size is reduced.

Note that if the trace is being turned off (either through a TRACE CT,OFF command or a CTnLOGxx parmlib member) and if the SYSLOGR trace is not connected to an external writer, the trace data **must** be dumped **before** turning the trace off to avoid loss of data.

**CTnLOGxx parmlib member**

The following table indicates the parameters you can specify in a CTnLOGxx parmlib member.
Parameters Allowed on CTnLOGxx?

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

IBM supplies the CTILOG00 parmlib member, which specifies the System Logger tracing activated at initialization. The contents of CTILOG00 as of V1.4 with OA07611 applied are:

```
TRACEOPTS ON
   BUFSIZE(16M)
   OPTIONS('CONNECT,LOGSTRM,DATASET,SERIAL,MISC,LOCBUFF,RECOVERY')
```

These parameters turn on the default system logger tracing to ensure that specific trace options are included and to establish a default trace buffer of 16MB. These trace options are activated at System Logger initialization.

If the PARMLIB trace options specify OFF or when the operator turns the trace off, logger uses minimal tracing which consists only of unexpected events (i.e. COMPERR trace entries).

**Example of CTnLOGxx parmlib member:** The statements in the following CTnLOGxx parmlib member example specify a 24MB trace buffer. All system logger trace records will be included in the trace output:

```
TRACEOPTS
   ON
   BUFSIZE(24M)
   OPTIONS('ALL')
```

The statements in the following CTxLOGxx example specify a 32MB trace buffer, with tracing of logstream functional request processing for logstreams SYSPLEX.OPERLOG in ASID 09. In addition, an external writer, EXTWTR, will be started, and SYSLOGR will be connected to the external writer:

```
TRACEOPTS
   WTRSTART(EXTWTR)
   ON
   BUFSIZE(32M)
   ASID(09)
   WTR(EXTWTR)
   OPTIONS('LOGSTRM', 'STRMNAME=(SYSPLEX.OPERLOG)')
```

**TRACE and REPLY commands**
The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.
If you reduce the size of the trace data space buffer and the SYSLOG trace has not been connected to an external writer, you must dump the contents of the buffer (see "Obtaining a dump of system logger information" on page 471) before reducing the buffer size if this data is important for debugging. Trace data in the trace data space buffer is discarded when the buffer size is reduced.

If the trace is being turned off (either through a TRACE CT,OFF command or a CTnLOGxx parmlib member) and the trace is not connected to an external writer, the trace data must be dumped before turning the trace off to avoid loss of data.

### OPTIONS parameter

The values for the OPTIONS parameter for the CTnLOGxx parmlib member and reply for a TRACE command, in alphabetical order, are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Subparameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DATASET</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>INVENTORY</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LOCBUFF</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LOGSTRM</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MISC</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RECOVERY</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SERIAL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>STRMNAME</td>
<td>No</td>
<td>Logstream</td>
</tr>
<tr>
<td>STORAGE</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**ALL**

Traces all system logger events.

**ASID**

Traces events for only the specified address space identifiers (ASID).
**Component Trace**

**COMPERR**
Traces internal system logger errors or unexpected events. This option is not specifiable and is always traced, as it is considered the minimal tracing.

**CONNECT**
Traces list structure connections, disconnections, rebuild and event exit processing.

**DATASET**
Traces log stream data set allocation and management.

**INVENTORY**
Traces log stream and structure definition and deletion processing as well as all LOGR CDS accesses. Do not specify this option unless requested by the IBM Support Center as this generate a large amount of records and may cause the buffer to wrap frequently.

**LOCBUFF**
Traces system logger local buffer management.

**LOGSTRM**
Traces log stream functional request processing.

**MISC**
Traces system logger internal miscellaneous services.

**RECOVERY**
Traces system logger component recovery, detecting abnormal conditions during processing.

**SERIAL**
Traces system logger serialization services.

**STORAGE**
Traces system logger storage management. Do not specify this option unless requested by the IBM Support Center as this will generate a large amount of records and may cause the buffer to wrap frequently.

**STRMNAME**
Traces events for only the specified log streams. If you specify STRMNAME, the specified log streams filter the CONNECT, LOGSTRM, INVENTORY, and DATASET options. The STRMNAME parameter must be specified STRMNAME=(strmname1). If you specify more than one log stream, STRMNAME must be specified STRMNAME=(strmname1,strmname2). A maximum of eight log stream names can be specified.

Note that the system does not verify the log stream names specified.

**Formatting a SYSLOGR trace**
Format the trace with an IPCS CTRACE COMP(SYSLOGR) subcommand. IBM might request that you enter options for SYSLOGR formatting. You can specify options for SYSLOG0R tracing on the OPTIONS parameter of the CTRACE command. The options include:

**COMPERR**
Displays internal system logger component errors.

**CONNECT**
Displays list structure connections, disconnections, rebuild and event exit processing.
Component Trace

**DATASET**
Displays log stream data set allocation and management.

**INVENTORY**
Displays log stream and structure definition and deletion processing as well as LOGR policy processing.

**LOCBUFF**
Displays system logger local buffer management.

**LOGSTRM**
Displays log stream functional request processing.

**MISC**
Displays system logger internal miscellaneous services.

**RECOVERY**
Displays system logger component recovery, detecting abnormal conditions during processing.

**RQE(request_address)**
Specify an 8-byte hexadecimal RQE address. Displays the specified RQE control block.

**SERIAL**
Displays system logger serialization services.

**STACK(request_address)**
Specify an 8-byte hexadecimal stack address. Displays the stack at the specified address.

**STORAGE**
Displays system logger storage management.

Output from a SYSLOGR trace

Figure 132 is an example of system logger component trace records formatted with the CTRACE COMP(SYSLOGR) subcommand:

```
COMPONENT TRACE FULL FORMAT
COMP(SYSLOGR)
**** 09/12/1994

CONNECT 03190001 13:03:28.955894 System Logger Services
C9E7C3E2 C9C7F0F3 40404040 40404040 | IXC5IG3
E203C3E3 C5E2E3F1 0100 | SLCTEST1.

RECOVERY 07040001 13:03:58.055519 System Logger Services
C9E7C7C3 F4D9C6C3 840C1000 00000001 | IXGC4RFCd......
03171D80 0000 | ...... |

RECOVERY 07040001 13:09:55.906719 System Logger Services
C9E7C7C3 F4D9C6C3 840C1000 00000001 | IXGC4RFCd......
031700A0 0000 | ...... |

COMPERR 01070007 13:30:58.322696 System Logger Services
E2E8E2F0 F0F0F1F1 C9E7C7D3 D6C7D94B | SYS00001IXGLGR.
E2C306E3 E3F34BC1 F0F0F0F0 F0F0F140 | SCOTT3,A0000001
40404040 40404040 40404040 40404040 | ..
```

Figure 132. Example: system logger component trace records formatted with CTRACE COMP(SYSLOGR) subcommand

Chapter 12. Component trace 477
Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSOMVS component trace for z/OS UNIX.

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSOMVS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTnBPXxx. Default member: CTIBPX00 specified in BPXPRM00 member</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; minimal; unexpected events</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTnBPXxx and REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: 4MB.</td>
</tr>
<tr>
<td></td>
<td>• Range: 4MB - 64MB.</td>
</tr>
<tr>
<td></td>
<td>• Size set by: CTnBPXxx member.</td>
</tr>
<tr>
<td></td>
<td>• Location: Data space. In the REPLY for the DUMP command, specify DSPNAME=(asid,SYSZBPX2) where asid is the ASID for z/OS UNIX.</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Data space buffer, trace data set</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSOMVS)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Requesting a SYSOMVS trace

Specify options for requesting a SYSOMVS component trace on a CTnBPXxx parmlib member or on the reply for a TRACE CT command.

You can change options for SYSOMVS tracing while the trace is running.

### CTnBPXxx parmlib member

The following table indicates the parameters you can specify on a CTnBPXxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnBPXxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes – see note</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes – see note</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>
IBM supplies the CTIBPX00 parmlib member, which specifies the z/OS UNIX tracing begun at IPL. The contents of CTIBPX00 are:

```
TRACEOPTS
  ON
  BUFSIZE(128K)
```

The parameters turn on the minimal SYSOMVS tracing. These parameters request the unexpected or important z/OS UNIX System Services events. The trace buffer size is 128KB. This member activates the minimal trace at IPL. In the IBM-supplied BPXPRM00 parmlib member, the CTRACE parameter specifies CTIBPX00 as the default.

**TRACE and REPLY commands**

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>No</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You can change options while a SYSOMVS trace is running.

**OPTIONS parameter**

The values for the OPTIONS parameter for the CTnBPXxx parmlib member and reply for a TRACE command, in alphabetical order, are:

**ALL**

Traces all events.
CBTR\((\text{cbid}, \text{offset}, \text{length})\)

Traces a field or fields of a control block to be traced. The contents of the trace will be included in the trace record for all SYSCALL trace records.

- \text{cbid} specifies the name of any of the supported z/OS UNIX control blocks in 1–4 characters.
- \text{offset} specifies the offset of the desired field in the control block in range X'0'-X'FFFF'.
- \text{length} specifies the length of the data, in bytes, to be traced in the control block. \text{length} is an integer hexadecimal value with a range of X'1'-X'8'.

CHARS

Traces character special events.

DEVPTY

Traces pseudoterminal events.

DEVRTY

Traces outboard communication server (OCS) remote terminal events.

FILE

Traces file system events. In a shared file system configuration, selecting the FILE option also activates the XCF option.

IPC

Traces interprocess communication activity for shared memory, message queues and semaphores.

LOCK

Traces locking services events.

PIPE

Traces pipe events.

PROCESS

Traces process events.

PTRACE

Traces PTRACE events.

SIGNAL

Traces signaling events.

STORAGE

Traces storage management events.

SYSCALL

Traces callable service layer events.

XCF

Traces file sharing events when using a shared file system configuration.

Examples of requesting SYSOMVS traces

- Example 1: CTnBPXxx member
  The member requests DEVPTY, FILE, and SIGNAL options.
  \text{TRACEOPTS}
  \text{ON}
  \text{OPTIONS('DEVPTY','FILE','SIGNAL')}

- Example 2: TRACE command
  The example requests a trace of DEVPTY and FILE trace events.
Component Trace

```
TRACE CT,ON,COMP=SYSOMVS
* 18 ITT906A ...
REPLY 18,OPTIONS(=,(CBTR(PPRP,0,4))),END
```

- **Example 3: TRACE command**
  The example requests a trace of four bytes at offset zero of control block PPRP.
  ```
  TRACE CT,ON,COMP=SYSOMVS
  * 18 ITT906A ...
  REPLY 18,OPTIONS(=,(CBTR(PPRP,0,4))),END
  ```

**Formatting a SYSOMVS trace**

Format the trace with an IPCS CTRACE COMP(SYSOMVS) subcommand. The OPTIONS parameter specifies the options that select trace records to be formatted.

Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records.

**ALL**
- Formats all events.

**CHARS**
- Formats character special events.

**DEVPTY**
- Formats pseudoterminal events.

**DEVRTY**
- Formats OCS remote terminal events.

**EXCEPTION**
- Formats exceptional events, such as recovery records or error records.

**FILE**
- Formats file system events.

**IPC**
- Formats events for shared memory, message queues and semaphores.

**KERNINFO**
- Formats the output to include a header for each record that includes descriptive information regarding the system call, process ID, and the module that requests the trace.

**LOCK**
- Formats locking services events.

**PIPE**
- Formats pipe events.

**PROCESS**
- Formats process events.

**PTRACE**
- Formats PTRACE events.

**SCOUNTS**
- Counts the number of syscalls that occur in the trace. Also counts the number of function codes that occur in a trace. The outputs are displayed in tables. Formatting is suppressed. The function codes refer to the types of messages that are crossing between systems in a sysplexed environment. In a non-sysplex dump, the functions code table will be empty. You could run an
Component Trace

application while collecting CTRACE data, and then use this option to
determine the frequency of syscalls and function codes being made by the
application.

SEARCH
Starting at the specified offset, searches trace entries for a specific value and
displays the matches. A CLIST called BPXMSCER is provided to allow the
search to be performed against specific entity ids that will identify syscall exits
that have failed.

SIGNAL
Formats signaling events.

STORAGE
Formats storage management events.

SYSCALL
Formats callable service layer events.

SYSID(nnn)
Formats sysplex system events. When this is requested by the user, only those
trace records that contain a sysplex system id will be formatted and displayed.
(nnn) is the sysplex number or name of the system in the sysplex whose
records will be displayed. See "Example of CTRACE DISPLAY PARAMETERS
panel" for an example of a CTRACE DISPLAY PARAMETERS panel and the
SYSID option on that panel.

XCF
Formats XCF events.

Example of CTRACE DISPLAY PARAMETERS panel
The CTRACE DISPLAY PARAMETERS panel has the following format. When
SYSID is specified, only those trace records that contain a sysplex system ID will
be formatted and displayed. If SYSID is not specified, data from all the systems
will be displayed.

Example of subcommands to format a SYSOMVS trace
• Example 1: CTRACE subcommand requesting SEARCH option
The example requests the SEARCH option to search every CTRACE entry, starting at the offset specified by `offset`, for the value specified by `value`.

```plaintext
CTRACE COMP(SYSOMVS) FULL OPTIONS((SEARCH(x'offset',x'value')))
```

**Example 2: CTRACE subcommand requesting SCOUNTS option**

The example requests the SCOUNTS option to count the number of syscalls from within the trace.

```plaintext
CTRACE COMP(SYSOMVS) FULL OPTIONS((SCCOUNTS))
```

**Output from a SYSOMVS trace**

```
SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) FULL is an example of SYSOMVS component trace records formatted with the CTRACE COMP(SYSOMVS) FULL subcommand.
```

### SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) FULL

```
COMPONENT TRACE FULL FORMAT
COMP(SYSOMVS)
**** 05/25/1999

<table>
<thead>
<tr>
<th>SYNAME</th>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SY1</td>
<td>XCF</td>
<td>0D890407</td>
<td>18:14:15</td>
<td>XCF BUFFER I/O TRACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASID...0025 USERID....WELLIE1 STACK@....2566DF18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TC8...009E0440 EUID......00000008 SYSCALL...00000036</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0000 E2C505C4 01800101 02000001 000A0002 SEND...........</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0010 B2D8CB82 285F5AC7 7B670500 403E3000 .H. !G#x...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0020 01FF0006 0008178 C6000000 ........a.F...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SY1</td>
<td>XCF</td>
<td>0D6F0401</td>
<td>18:14:15</td>
<td>NXMSO--&gt;XCF MESSAGE OUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASID...0025 USERID....WELLIE1 STACK@....2566DF18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TC8...009E0440 EUID......00000008 SYSCALL...00000036</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0000 E2E805C3 01800101 000A0002 SEND...........</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0010 0113C000 02000001 000A0002 00004C48 ..............&lt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0020 7B670500 000080E0 00000000 00000000 #x......</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0030 066F0000 00030025 09E0440 00000008 .?...........</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0040 00000000 2538E980 01000000 ..........Z......</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SY1</td>
<td>XCF</td>
<td>0D6F0402</td>
<td>18:14:15</td>
<td>NXMSO--&gt;XCF MESSAGE SRB EXIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASID...001E USERID....OMVS STACK@....25385F28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TC8...00000000 EUID......00000000 SYSCALL...00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0000 D9C5E2D7 A6000101 009D6C68 00000000 RESP...........</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0010 00030000 0A010014 01176B94 0013402C ............M...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0020 02000001 000A0002 00000118 01FF0006 .................</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0030 0030098 24C02910 00000000 00000000 ..........{..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+0040 00000000 00000000 00000000 ...............</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SY1</td>
<td>XCF</td>
<td>0D6F0401</td>
<td>18:14:15</td>
<td>NXMSO--&gt;XCF MESSAGE OUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASID...0025 USERID....WELLIE1 STACK@....2566DF18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TC8...009E0440 EUID......00000008 SYSCALL...00000036</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|        | +0000 C6D9C5C5 10000100 00000000 00000000 FREE...........
|        | +0010 00000000 00000000 00000000 00000000 ............... |
|        | +0020 00000000 00000000 00000000 00000000 ............... |
|        | +0030 066F0000 00000000 00000000 00000000 ............... |
|        | +0040 00000000 2538E980 00000000 ..........Z...... |
| SY2    | XCF      | 0D6F0401 | 18:14:15  | NXMSO-->XCF MESSAGE SRB EXIT |
|        |          |          |            |              |
|        | ASID...001E USERID....OMVS STACK@....25389F28 |
|        | TC8...00000000 EUID......00000000 SYSCALL...00000000 |
|        | +0000 D9C53C5D D4600201 009E0440 002580E0 RECVM........... |
|        | +0010 00030000 0A010014 01176B94 0013402C .............M{. |
|        | +0020 01000001 000A0001 0008178 01FF0006 ............a.... |
```

Chapter 12. Component trace 483
SY1 trace flow

Figure 133 on page 485 and Figure 134 on page 485 contain the SY1 trace information found in "SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) FULL" on page 483.

Figure 133 on page 485 describes the CTRACE entries generated by the BPXNXMSO processing on the client side. The ASID / TCB highlighted describe the client making the request.

The most important information is the Unique Request-ID (as noted with an asterisk (*)). This is used to track a request from the client to through the server, and back again.

Two separate trace entries are provided. One states that a message has been entered into a block of messages, and the other states that the block has been written. The buffer address (as noted with an @) is used to cross reference these two trace entries.
Figure 134 describes the response arriving on the client system. First, the XCF SRB (BPXNXMSG) processes the incoming response to cause the client task to be activated. And then, the target task (no longer remapped) wakes up, and, in this example, explicitly frees the resources that were allocated to it as part of the XCF message processing.

SY2 trace flow

Figure 135 on page 486 and Figure 136 on page 487 contain the SY2 trace information found in “SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) FULL” on page 483.
Figure 135 describes the server side XCF SRB processing by first queuing the request (BPXNXMSG), and then having a worker task pick up that piece of work for subsequent processing (BPXNXWRK).

As noted with an *, the Request-ID is used to cross reference the individual trace entries.

When a SYSNAME field is included in a trace entry, the ASID / TCB information actually describes the client side requester information. The SYSNAME field describes the originating system. The highlighted field with an & is the TCB address of the worker task resident in the server system.

```
SY2 XCF 00690402 18:14:14.553698 NXMSG-->XCF MESSAGE SRB EXIT
ASID..000E USERID....OMVS STACK0....2538F2B
TCB...00000000 EUID....00000000 SYSCALL....00000000
+0000 09C5C3E5 04600201 009E04A0 002580E0 |RECV------|
+0010 00030000 10A010014 +01170804 $0013C000 ...........M..|
+0020 01000001 000A0001 00008178 01FF0006 ........a....|
+0030 40060998 80000009 00000000 00000000 ....q........|
+0040 00000000 00000000 4F4F4F4F |................|
```
Component Trace

<table>
<thead>
<tr>
<th>SYNAME</th>
<th>MNAMEIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SY2</td>
<td>XCF</td>
<td>00900407</td>
<td>18:14:14.553881</td>
<td>XCF BUFFER I/O TRACE</td>
</tr>
</tbody>
</table>

Figure 137 is an example of SYSOMVS component trace records requested with OPTIONS(CBTR(PPRP,0,4)) to trace a four byte field at offset zero in the PPRP control block. The trace data was then formatted with the CTRACE COMP(SYSOMVS) subcommand:

Figure 138 on page 488 is an example of SYSOMVS component trace records formatted with the CTRACE COMP(SYSOMVS) SHORT subcommand.
The output from a SYSOMVS trace using the SCCOUNTS option has 2 formats, shown in Figure 139 and Figure 140.

Figure 139 is sorted by frequency, with the highest values appearing at the top of the list. SYSCALL# is the hexadecimal number of the syscall. FREQUENCY/SEC is the number of times the syscall has been invoked within the interval.

<table>
<thead>
<tr>
<th>SYSCALL#</th>
<th>SYSCALL NAME</th>
<th>COUNT</th>
<th>FREQUENCY/SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F</td>
<td>BPX1CHO</td>
<td>5000</td>
<td>nnn</td>
</tr>
<tr>
<td>2F</td>
<td>BPXISTA</td>
<td>150</td>
<td>nnn</td>
</tr>
</tbody>
</table>

Figure 139. SCCOUNT Function Displaying SYSCALL Frequency

The output from a SYSOMVS trace using the KERNINFO option is shown in Figure 141 on page 489. The syscall function name (FCN) and the process ID (PID) are shown on the first line of the trace entry.
SYSOPS component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSOPS component trace for the operations services component (OPS).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSOPS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CtnOPSxx. Default member: CTIOPS00 specified in CONSOLxx member</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; minimal; unexpected events</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CtnOPSxx or REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: 2MB</td>
</tr>
<tr>
<td></td>
<td>• Range: 64KB - 16MB</td>
</tr>
<tr>
<td></td>
<td>• Size set by: CtnOPSxx member or REPLY for TRACE CT command</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: Yes, when restarting a trace after stopping it</td>
</tr>
<tr>
<td></td>
<td>• Location: Console address space (private)</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer, trace data set</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSOPS)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Note: To get a complete dump for OPS, request also the NUC, CSA, and SQA.
**Component Trace**

**Requesting a SYSOPS trace**

Specify options for requesting a SYSOPS component trace in a CTnOPSxx parmlib member or in the reply for a TRACE CT command.

**CTnOPSxx parmlib member**

The following table indicates the parameters you can specify on a CTnOPSxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnOPSxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

IBM supplies the CTIOPS00 parmlib member, which defines the component trace to the system and establishes a trace buffer of 2M. The contents of CTIOPS00 are:

```
TRACEOPTS
ON
BUFSIZE(2M)
```

IBM does not supply a sample CONSOL00 parmlib member. Create a CONSOLxx parmlib member and specify CON=xx in the IEASYSxx parmlib member. Specify CTIOPS00 as the default on the CTRACE parameter of the INIT statement of CONSOLxx.

**TRACE and REPLY commands**

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, nnnnK, nnnnM, or OFF</td>
<td>One is required. The buffer size can be changed only when the trace is OFF or the trace is ON.</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Component Trace

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**OPTIONS parameter**
The values for the OPTIONS parameter for the CTnOPSxx parmlib member and reply for a TRACE command, in alphabetical order, are:

**COMMAND**
Traces events related to command processing.

**CONSDATA**
Traces events related to console state changes.

**LOGGING**
Traces events related to Operlog and Syslog processing.

**MCACHE**
Traces events related to the message cache.

**MESSAGES**
This option includes the WTO, MSGDLVRY, MCACHE and LOGGING options.

**MSG=msgid**
Traces processing of a specific message. It REQUIRES either one of the following OPTIONS: MESSAGES, WTO, MSGDLVRY, MCACHE or LOGGING. 
msgid is 1-10 alphanumeric characters in length indicating the message id that will be traced.

**MSGDLVRY**
Traces events for WQE processing, MCS console message queueing, and extended MCS console message processing.

**RECOVERY**
Traces recovery events.

**SERIALIZ**
Traces latch serialization events.

**SYSPLEX**
Traces events for XCF signalling, sysplex serialization services, sysplex clean-up processing, and the manipulation of various queues.

**WTO**
Traces the effects of MPFSTxx, the user exits, and the SSI on message content and attributes.

These additional options increase the number of trace records the system collects and can slow system performance. Each time you change the trace options, you must respecify any options you want to keep in effect from the last trace.

**Note:** Before you use the MESSAGES, WTO, MSGDLVRY, MCACHE or LOGGING options, you should do the following:

- Increase the buffer size
- Start and connect the external writer.
Component Trace

This is especially important if you are starting tracing at IPL. This might not be necessary if you are tracing a message ID since you would only be cutting records for the particular message.

**Examples of requesting SYSOPS traces**

- **Example 1: Activating trace options while system is running**
  
  Create parmlib member CTIOPS01, specifying the following parameters. Assume that procedure OPSWTR is in SYS1.PROCLIB.

  ```
  TRACEOPTS
  WTRSTART(.OPSWTR)
  ON
  WTR(OPSWTR)
  OPTIONS('MESSAGES','MSG=IEE136I')
  ASID(1,2,3)
  JOBNAME(PAYROLL)
  BUFSIZE(2M)
  ```

- **Example 2: Specifying trace options on a REPLY command**
  
  The example requests the same trace as Example 1, but specifies all options on the REPLY.

  ```
  trace ct,wtrstart=opswtr
  trace ct,2m,comp=sysops
  ```

  When the system prompts you for the trace options, enter the following command, replacing `id` with the reply identifier:

  ```
  reply 27,wtr=opswtr,options=(messages,msg=IEE136I),asid=(1,2,3),
  jobname=(payroll),end
  ```

- **Example 3: CTnOPSxx member used at IPL**
  
  The member requests the SYSPLEX option, doubles the default buffer size, and limits the tracing to ASID 1 and JOBNAME JOB1.

  ```
  TRACEOPTS
  ON
  OPTIONS('SYSPLEX')
  BUFSIZE(4M)
  ASID(1)
  JOBNAME(JOB1)
  ```

**Formatting a SYSOPS trace**

Format the trace with an IPCS CTRACE COMP(SYSOPS) subcommand. The `OPTIONS` parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records. The options for formatting a SYSOPS trace are:

- **COMMAND**
  
  Traces events related to command processing.

- **CONSDATA**
  
  Traces events related to console state changes.

- **LOGGING**
  
  Traces events related to Operlog and Syslog processing.

- **MCACHE**
  
  Traces events related to the message cache.

- **MESSAGES**
  
  This option includes the WTO, MSGDLVRY, MCACHE and LOGGING options.
MSG=msgid
Traces processing of a specific message. It REQUIRES either one of the following OPTIONS: MESSAGES, WTO, MSGDLVRY, MCACHE or LOGGING. msgid is 1-10 alphanumeric characters in length indicating the message id that will be traced.

MSGDLVRY
Traces events for WQE processing, MCS console message queueing, and extended MCS console message processing.

RECOVERY
Traces recovery events.

SERIALIZ
Traces latch serialization events.

SYSPLEX
Traces events for XCF signalling, sysplex serialization services, sysplex clean-up processing, and the manipulation of various queues.

WTO
Traces the effects of MPFLSTxx, the user exits, and the SSI on message content and attributes.

Output from a SYSOPS trace

Figure 142 is an example of OPS component trace records formatted with the CTRACE COMP(SYSOPS) SHORT subcommand.

<table>
<thead>
<tr>
<th>COMPONENT TRACE SHORT FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP(SYSOPS)</td>
</tr>
<tr>
<td>**** 05/20/93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITMSG</td>
<td>00000001</td>
<td>14:41:06.918883</td>
<td>Message prior to MPF processing</td>
</tr>
<tr>
<td>POSTMPF</td>
<td>00000002</td>
<td>14:41:06.919198</td>
<td>Message after MPF processing</td>
</tr>
<tr>
<td>POSTEXIT</td>
<td>00000003</td>
<td>14:41:06.919595</td>
<td>Post Message Exit</td>
</tr>
<tr>
<td>POSTSSI</td>
<td>00000004</td>
<td>14:41:06.920118</td>
<td>Post Subsystem Interface</td>
</tr>
<tr>
<td>INITMSG</td>
<td>00000001</td>
<td>14:41:06.930088</td>
<td>Message prior to MPF processing</td>
</tr>
<tr>
<td>POSTMPF</td>
<td>00000002</td>
<td>14:41:06.930405</td>
<td>Message after MPF processing</td>
</tr>
<tr>
<td>POSTEXIT</td>
<td>00000003</td>
<td>14:41:06.930803</td>
<td>Post Message Exit</td>
</tr>
<tr>
<td>POSTSSI</td>
<td>00000004</td>
<td>14:41:06.931267</td>
<td>Post Subsystem Interface</td>
</tr>
<tr>
<td>INITMSG</td>
<td>00000001</td>
<td>14:41:06.931637</td>
<td>Message prior to MPF processing</td>
</tr>
<tr>
<td>POSTMPF</td>
<td>00000002</td>
<td>14:41:06.931934</td>
<td>Message after MPF processing</td>
</tr>
</tbody>
</table>

Figure 142. Example: OPS component trace records formatted with CTRACE COMP(SYSOPS) SHORT subcommand

Figure 143 on page 494 is an example of OPS component trace records formatted with the CTRACE COMP(SYSOPS) FULL subcommand.
Component Trace

**COMPONENT TRACE FULL FORMAT**
**COMP(SYSOPS)**
**** 05/20/93

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITMSG</td>
<td>00000001</td>
<td>14:41:06.918883</td>
<td>Message prior to MPF processing</td>
</tr>
<tr>
<td>TR_GROUP</td>
<td>WTO</td>
<td></td>
<td>Write to Operator Services</td>
</tr>
<tr>
<td>HOMEASID</td>
<td>0001</td>
<td></td>
<td>HOMEJOBN <em>MASTER</em></td>
</tr>
<tr>
<td>REQASID</td>
<td>0001</td>
<td></td>
<td>REQJOBN UNKNOWN</td>
</tr>
<tr>
<td>CPU_ADDR</td>
<td>0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+0000</td>
<td>LKP...... 00000000</td>
<td>TXTL...... 005F</td>
<td>RTCT...... 0000</td>
</tr>
<tr>
<td>+000A</td>
<td>USE...... 0000</td>
<td>PAD...... 40</td>
<td>TS........ 10.41.06</td>
</tr>
<tr>
<td>+00DI</td>
<td>TS2....... .91</td>
<td>PAD1....... 40</td>
<td>JOBNM....</td>
</tr>
<tr>
<td>+001E</td>
<td>PAD2...... 40</td>
<td>TXT....... ITTO38I</td>
<td>ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND WERE</td>
</tr>
<tr>
<td>+0044</td>
<td>ESTED VIA THE TRACE CT COMMAND WERE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+009E</td>
<td>TXTL...... 40</td>
<td>PAD3....... 00</td>
<td>XA....... 01</td>
</tr>
<tr>
<td>+00A1</td>
<td>ASID...... 0001</td>
<td>AVAIL....... 50</td>
<td>TCB....... 008057C0</td>
</tr>
<tr>
<td>+00A8</td>
<td>SYSID...... 03</td>
<td>SEQN....... 00049D</td>
<td>MCSF1...... 00</td>
</tr>
<tr>
<td>+00AD</td>
<td>MCSF2...... 10</td>
<td>MSGT1....... 00</td>
<td>ROUT1...... 00</td>
</tr>
<tr>
<td>+00B1</td>
<td>ROUT2...... 00</td>
<td>CHAR1....... 00</td>
<td>FLG1....... 00</td>
</tr>
<tr>
<td>+00B4</td>
<td>UCMD...... 0A</td>
<td>FLG1....... 04</td>
<td>RPYID...... 0000</td>
</tr>
<tr>
<td>+00B8</td>
<td>DC1...... 08</td>
<td>DC2....... 00</td>
<td>RSV26...... 0000</td>
</tr>
<tr>
<td>+00BC</td>
<td>JSTCB...... 008057C0</td>
<td>VRSN....... 05</td>
<td>MFLG1...... 00</td>
</tr>
<tr>
<td>+00C2</td>
<td>MCE1...... 42</td>
<td>MCE2...... 00</td>
<td>SYSMM...... SYS2</td>
</tr>
<tr>
<td>+00CC</td>
<td>DATE...... 93140</td>
<td>RFB1...... 00</td>
<td>RFB2...... 00</td>
</tr>
<tr>
<td>+006D</td>
<td>RFB3...... 00</td>
<td>SUPB....... 90</td>
<td>ML1....... 00</td>
</tr>
<tr>
<td>+009D</td>
<td>ML2...... 00</td>
<td>LENG....... 0160</td>
<td>DSQN...... 00000000</td>
</tr>
<tr>
<td>+00E0</td>
<td>ERC1...... 00</td>
<td>ERC2....... 00</td>
<td>ERC3....... 00</td>
</tr>
<tr>
<td>+00E3</td>
<td>ERC4...... 00</td>
<td>ERC5....... 00</td>
<td>ERC6....... 00</td>
</tr>
<tr>
<td>+00E6</td>
<td>ERC7...... 00</td>
<td>ERC8....... 00</td>
<td>ERC9....... 00</td>
</tr>
<tr>
<td>+00E9</td>
<td>ERC10..... 00</td>
<td>ERC11...... 00</td>
<td>ERC12...... 00</td>
</tr>
<tr>
<td>+00EC</td>
<td>ERC13..... 00</td>
<td>ERC14...... 00</td>
<td>ERC15...... 00</td>
</tr>
<tr>
<td>+00EF</td>
<td>ERC16..... 00</td>
<td>KEY...... 40404040</td>
<td>40404040</td>
</tr>
<tr>
<td>+00FB</td>
<td>TKNN...... 00000000</td>
<td>CNID...... 0000000A</td>
<td>OJBID...... 40404040</td>
</tr>
</tbody>
</table>

*Figure 143. Example: OPS component trace records formatted with CTRACE COMP(SYSOPS) FULL subcommand*

**SYSRRS component trace**

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSRRS component trace for RRS.

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSRRS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTRnRRSxx. No default member</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; minimal; unexpected events; general UR services</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTRnRRSxx and REPLY for TRACE command</td>
</tr>
</tbody>
</table>
Information | For SYSRRS:
--- | ---
**Buffer** | • Default: 1MB  
• Range: 1MB - 2045MB  
• Size set by: CTnRRSxx member or REPLY for TRACE CT command  
• Change size after IPL: Yes, when restarting a trace after stopping it  
• Location: Data space and component address space. In the REPLY for the DUMP command, specify DSPNAME=('RRS'.ATRTRACE) and SDATA=RGN.

**Trace records location** | Data space buffer, trace data set, trace buffers in the RRS address space

**Request of SVC dump** | By DUMP or SLIP command

**Trace formatting by IPCS** | CTRACE COMP(SYSRRS)

**Trace format OPTIONS parameter** | Yes

### Requesting a SYSRRS trace

Specify options for requesting a SYSRRS component trace on a CTnRRSxx parmlib member or on the reply for a TRACE CT command.

To change options or the buffer size, you have to stop the trace and restart it with the new options, buffer size, or both.

### CTnRRSxx parmlib member

The following table indicates the parameters you can specify on a CTnRRSxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnRRSxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

A CTnRRSxx parmlib member is optional. If not specified, the SYSRRS component trace runs a minimal trace beginning when the RRS component is started and ending when the component is stopped.

### TRACE and REPLY commands

The following tables indicate the parameters you can specify on a TRACE CT command and a REPLY.
Component Trace

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, nnnnK, nnnnM, or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

OPTIONS parameter

The OPTIONS parameter for the CTnRRSxx parmlib member or reply for a TRACE command contains keyword subparameters. These subparameters allow you to control the information that RRS component trace collects. The first subparameter, EVENTS, specifies the events to be traced; the other subparameters act as filters to screen the events with up to three checks. An event must pass all checks for component trace to generate a trace record. The order for the checks is:

1. That the event is specified.
2. That the event matches, in an OR check, one of the following filters:
   - The address space ID (ASID) in the REPLY command or CTnRRSxx TRACEOPTS statement
   - The job name (JOBNAME) in the REPLY command or CTnRRSxx TRACEOPTS statement
   - The user ID (USERID) in the OPTIONS parameter
   - The logical work unit ID (LUWID) in the OPTIONS parameter
3. That the event matches a resource manager name (RMNAME) in the OPTIONS parameter.

   OPTIONS=([EVENTS(event[,event]...)]
            [USERID(userid[,userid]...)]
            [RMNAME(rmname[,rmname]...)]
            [LUWID([luwid],[luwid]...)]
            [EID([eid],[eid]...)]

Note: In the REPLY to the TRACE CT command, separate the options by one or more blanks.

EVENTS(event[,event]...)

Indicates the events to be traced. An EVENTS parameter is required if any other options are specified. The events, in alphabetical order, are:

ALL

Traces all events. Component trace ignores other events, if specified.
Component Trace

**CONTEXT**
Traces calls to context services.

**EXITS**
Traces events related to running the RRS exit routines provided by the resource managers.

**FLOW**
Traces entry into and exit from RRS entry points.

**LOGGING**
Traces events related to logging data by RRS.

**RESTART**
Traces events related to RRS initialization and restart.

**RRSAPI**
Traces events related to the application programming interface, which consists of calls to the Application_Commit_UR service and the Application_Backout_UR service.

**STATECHG**
Traces events involving changes in the state of units of recovery (URs).

**URSERVS**
Traces general events related to services for a UR (traced by default).

**USERID(userid[.userid]...)**
Specifies 1 to 16 user IDs as filters for specified events. The system traces only events relating to the user IDs.

**RMNAME(rmname[,rmname]...)**
Specifies 1 to 16 resource manager names as filters for specified events. For trace events sensitive to the resource manager name, the system traces only events relating to the resource managers.

**LUWID((luwid)[,(luwid)]...)**
Specifies 1 to 16 logical unit of work identifiers (LUWIDs) as filters for specified events. The system traces only events relating to the LUWIDs. Each *luwid* consists of:

- netid.luname[,instnum][,seqnum]

Component trace ignores leading and trailing blanks.

**netid.luname**
Specifies the network ID and the local logical unit name. These portions of the LUWID are required.

- You can use an asterisk (*) as a wildcard character as:
  - The last character in the *netid*, the *luname*, or both
  - The only character in the either the *netid* or the *luname*, but not as the only character in both

**instnum**
Specifies the instance number as a 1 - 12 hexadecimal integer. You can omit leading zeros.

**seqnum**
Specifies the sequence number as a 1 - 4 hexadecimal integer. You can omit leading zeros.

Examples of LUWIDs are:
Component Trace

\[(A.B,5,1)\]  
\[(A.B,5)\]  
\[(A.B,1)\]  
\[(A.B)\]  
\[(A.*,5,1)\]  
\[(A*,B*)\]

**EID\((\text{eid}), (\text{eid})\ldots\)**  
Specifies 1 to 16 Enterprise identifiers (EIDs) as filters for specified events. The system traces only events relating to the EIDs. Each \(\text{cid}\) consists of:
\[[\text{tid}],[\text{gtid}]\]

You can omit leading zeros. Component trace ignores leading and trailing blanks.

**tid**  
Specify the 4-byte hexadecimal transaction identifier (TID).

**gtid**  
Specify the 8-40 byte hexadecimal global transaction identifier (GTID).

You can obtain the EID for a UR by using the RRS ISPF panels to browse the RRS log streams. The Retrieve_Work_Identifier service can also return an EID.

Examples of EIDs are:
\[(1,C)\]  
\[(,C)\]  
\[(1)\]

**Examples of requesting SYSRRS traces**  
In Figure 144, the member requests context services events filtered by the user ID JONES and requests a 1024KB buffer.

```
TRACEOPTS
  ON
  OPTIONS('EVENTS(CONTEXT) USERID(JONES)')
  BUFSIZE(1024K)
```

**Figure 144. Example: CTnRRSxx member requests context services events**

**Figure 145** is an example of using the TRACE command to request the same trace shown in **Figure 144**

```
trace ct, off, comp=sysrrs
trace ct,1024K,comp=sysrrs
+ 17 ITT006A ...
reply 17, options='\{events\{context\} userid\{jones\}\}', end
```

**Figure 145. Example: TRACE command requests context services events**

**Formatting a SYSRRS trace**

Format the trace with an IPCS CTRACE COMP(SYSRRS) subcommand. Its OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the formatting options to narrow down the records displayed so that you can more easily locate any errors. If you specify no options on the CTRACE subcommand, IPCS displays all the trace records.
You can specify one or more OPTIONS subparameters. If you specify no OPTIONS subparameters, all trace records are formatted. A trace record must match all specified OPTIONS subparameters to be formatted.

```plaintext
OPTIONS=((option[option]...))
option is one of the following:
  LUWID(luvid)
  EID(eid)
  RMNAME(rmname)
  URID(urid)
  USERID(userid)
```

**LUWID(luvid)**
- Specifies one of the logical unit of work identifiers (LUWIDs) specified when the trace was generated.

**EID(eid)**
- Specifies one of the Enterprise identifiers (EIDs) specified when the trace was generated. Specify `eid` as:
  - `tid`
  - `tid,gtid`

Or, if you omitted `tid` when you specified the identifier: `*,gtid`

**RMNAME(rmname)**
- Specifies one of the resource manager names specified when the trace was generated.

**URID(urid)**
- Specifies a UR identifier. The URID is a 16-byte character string returned to the resource manager by one of the following callable services:
  - `Change_Interest_Type`, `Express_UR_Interest`, `Retain_Interest`,
  - `Retrieve_UR_Interest`, or `Retrieve_UR_Data`. The URID is saved in the resource manager log; you can obtain it through an RRS panel.

**USERID(userid)**
- Specifies one of the user IDs specified when the trace was generated. Note that USERID does not filter out trace records in which the user ID is blank.

The following is an example of how to specify an IPCS CTRACE OPTIONS parameter:

```plaintext
OPTIONS=((RMNAME(datamgr),USERID(jjones)))
```

### Output from a SYSRRS trace

Fields that do not contain printable characters are filled with asterisks (*). The value is shown in hexadecimal on a separate line.

**Note:** RRS provides the same report for the SUMMARY and FULL parameters on the CTRACE subcommand.

"CTRACE COMP(SYSRRS) SHORT subcommand output" on page 500 is an example of SYSRRS component trace records formatted with the CTRACE COMP(SYSRRS) SHORT subcommand.
Component Trace

CTRACE COMP(SYSRRS) SHORT subcommand output

COMPONENT TRACE SHORT FORMAT
COMP(SYSRRS)
**** 01/20/1997

<table>
<thead>
<tr>
<th>SY</th>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SY1</td>
<td>COMPERR</td>
<td>00000000</td>
<td>07:53:43.615114</td>
<td>Resource Recovery Services</td>
</tr>
<tr>
<td>SY1</td>
<td>FLOW</td>
<td>0801FFFE</td>
<td>07:53:43.615114</td>
<td>ATRB1PCT EXIT</td>
</tr>
<tr>
<td>SY1</td>
<td>FLOW</td>
<td>0801FFFF</td>
<td>07:53:43.619823</td>
<td>ATRB1PCT ENTRY</td>
</tr>
<tr>
<td>SY1</td>
<td>FLOW</td>
<td>0201FFFF</td>
<td>07:53:43.619867</td>
<td>ATRU1EIN ENTRY</td>
</tr>
<tr>
<td>SY1</td>
<td>CONTEXT</td>
<td>02010002</td>
<td>07:53:43.620027</td>
<td>CTXMEINT call</td>
</tr>
<tr>
<td>SY1</td>
<td>URSERV5</td>
<td>02010001</td>
<td>07:53:43.620699</td>
<td>ATREINT invoked</td>
</tr>
<tr>
<td>SY1</td>
<td>FLOW</td>
<td>0201FFFE</td>
<td>07:53:43.620887</td>
<td>ATRU1EIN EXIT</td>
</tr>
</tbody>
</table>

“CTRACE COMP(SYSRRS) SUMMARY or FULL output” is an example of SYSRRS component trace records formatted with the CTRACE COMP(SYSRRS) SUMMARY or FULL subcommands.

CTRACE COMP(SYSRRS) SUMMARY or FULL output

COMPONENT TRACE FULL FORMAT
COMP(SYSRRS)
**** 01/20/1997

<table>
<thead>
<tr>
<th>SY</th>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
<th>HASID..</th>
<th>HJOBNAME.</th>
<th>APPLAS</th>
<th>SASID..</th>
<th>SJOBNAME.</th>
<th>RMAS1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SY1</td>
<td>COMPERR</td>
<td>00000000</td>
<td>07:53:43.615114</td>
<td>Resource Recovery Services</td>
<td>00000000</td>
<td>00000000</td>
<td>0001FE6E</td>
<td>00000000</td>
<td>7E6D0000</td>
<td>00000000</td>
</tr>
<tr>
<td>SY1</td>
<td>FLOW</td>
<td>0801FFFE</td>
<td>07:53:43.615114</td>
<td>ATRB1PCT EXIT</td>
<td>00AA</td>
<td>HJOBNAME.</td>
<td>APPLAS</td>
<td>00A3</td>
<td>SJOBNAME.</td>
<td>RMAS1</td>
</tr>
<tr>
<td>SY1</td>
<td>FLOW</td>
<td>0801FFFF</td>
<td>07:53:43.619823</td>
<td>ATRB1PCT ENTRY</td>
<td>00AA</td>
<td>HJOBNAME.</td>
<td>APPLAS</td>
<td>00A3</td>
<td>SJOBNAME.</td>
<td>RMAS1</td>
</tr>
<tr>
<td>SY1</td>
<td>FLOW</td>
<td>0201FFFF</td>
<td>07:53:43.619867</td>
<td>ATRU1EIN ENTRY</td>
<td>00AA</td>
<td>HJOBNAME.</td>
<td>APPLAS</td>
<td>00A3</td>
<td>SJOBNAME.</td>
<td>RMAS1</td>
</tr>
<tr>
<td>SY1</td>
<td>CONTEXT</td>
<td>02010002</td>
<td>07:53:43.620027</td>
<td>CTXMEINT call</td>
<td>00A3</td>
<td>SJOBNAME.</td>
<td>RMAS1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SY1</td>
<td>URSERV5</td>
<td>02010001</td>
<td>07:53:43.620699</td>
<td>ATREINT invoked</td>
<td>00A3</td>
<td>SJOBNAME.</td>
<td>RMAS1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SY1</td>
<td>FLOW</td>
<td>0201FFFE</td>
<td>07:53:43.620887</td>
<td>ATRU1EIN EXIT</td>
<td>00A3</td>
<td>SJOBNAME.</td>
<td>RMAS1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

z/OS V2R2 MVS Diagnosis: Tools and Service Aids
“CTRACE COMP(SYSRRS) TALLY output” is an example of SYSRRS component trace records formatted with the CTRACE COMP(SYSRRS) TALLY subcommands.

### CTRACE COMP(SYSRRS) TALLY output

**COMPONENT TRACE TALLY REPORT**

**COMP(SYSRRS)**

**TRACE ENTRY COUNTS AND AVERAGE INTERVALS (IN MICROSECONDS)**

<table>
<thead>
<tr>
<th>FMTID</th>
<th>COUNT</th>
<th>INTERVAL</th>
<th>MNEMONIC</th>
<th>DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>7</td>
<td>23,774,056</td>
<td>COMPERR</td>
<td>Resource Recovery Services</td>
</tr>
<tr>
<td>02010001</td>
<td>46</td>
<td>2,398,672</td>
<td>URSERVS</td>
<td>ATREINT invoked</td>
</tr>
<tr>
<td>02010002</td>
<td>46</td>
<td>2,398,670</td>
<td>CONTEXT</td>
<td>CTXMEINT call</td>
</tr>
<tr>
<td>02018001</td>
<td>6</td>
<td>18,803,430</td>
<td>STATECHG</td>
<td>UR State Change</td>
</tr>
<tr>
<td>02018008</td>
<td>3</td>
<td>47,008,505</td>
<td>STATECHG</td>
<td>UR being created</td>
</tr>
<tr>
<td>0201FFFF</td>
<td>46</td>
<td>2,398,673</td>
<td>FLOW</td>
<td>ATRUIEIN EXIT</td>
</tr>
<tr>
<td>0201FFFF</td>
<td>46</td>
<td>2,398,670</td>
<td>FLOW</td>
<td>ATRUIEIN ENTRY</td>
</tr>
<tr>
<td>02030001</td>
<td>0</td>
<td></td>
<td>URSERVS</td>
<td>ATRDINT invoked</td>
</tr>
<tr>
<td>02038009</td>
<td>0</td>
<td></td>
<td>STATECHG</td>
<td>UR being destroyed</td>
</tr>
<tr>
<td>0203FFFF</td>
<td>0</td>
<td></td>
<td>FLOW</td>
<td>ATRUIDIN EXIT</td>
</tr>
<tr>
<td>0203FFFF</td>
<td>0</td>
<td></td>
<td>FLOW</td>
<td>ATRUIDIN ENTRY</td>
</tr>
<tr>
<td>02050001</td>
<td>0</td>
<td></td>
<td>URSERVS</td>
<td>ATRPDUE invoked</td>
</tr>
<tr>
<td>0205FFFF</td>
<td>0</td>
<td></td>
<td>FLOW</td>
<td>ATRUIDPU EXIT</td>
</tr>
<tr>
<td>0205FFFF</td>
<td>0</td>
<td></td>
<td>FLOW</td>
<td>ATRUIDPU ENTRY</td>
</tr>
</tbody>
</table>

---

### SYSRSM component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSRSM component trace for the real storage manager (RSM).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSRSM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTnRSMxx. No default member</td>
</tr>
<tr>
<td>Default tracing</td>
<td>No.</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTnRSMxx or REPLY for TRACE command</td>
</tr>
</tbody>
</table>
Component Trace

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSRSM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>• Default: 3 buffers of 32 pages</td>
</tr>
<tr>
<td></td>
<td>• 2 -7 page-fixed primary buffers, 4 - 262,144 pages per buffer</td>
</tr>
<tr>
<td></td>
<td>• 1 - 2047 MB for secondary buffers</td>
</tr>
<tr>
<td></td>
<td>• Size set by: CTnRSMxx member or REPLY for TRACE CT command</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: Yes, when starting a trace</td>
</tr>
<tr>
<td></td>
<td>• Common service area (LIKECSA) and, if specified in CTnRSMxx, high virtual private storage of the RASP address space.</td>
</tr>
<tr>
<td>Trace records location</td>
<td>In the trace data set, trace buffers in high common memory, and the RASP address space.</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>• By DUMP or SLIP command</td>
</tr>
<tr>
<td></td>
<td>• Through the DMPREC option on the CTnRSMxx parmlib member or on the REPLY for the TRACE CT command when RSM enters recovery processing (default)</td>
</tr>
<tr>
<td></td>
<td>• Through the DMPOFF option of CTnRSMxx or the TRACE CT reply when SYSRSM tracing is turned off</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSRSM)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>None</td>
</tr>
</tbody>
</table>

Requesting a SYSRSM trace

Specify options for requesting a SYSRSM component trace in a CTnRSMxx parmlib member or in the reply for a TRACE CT command.

CTnRSMxx parmlib member

The following table indicates the parameters you can specify on a CTnRSMxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnRSMxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes. To trace all batch jobs, specify ‘INIT’ in the list of job names.</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

IBM provides two sample CTIRSMxx parmlib members in SYS1.PARMLIB. These are not default members.
Component Trace

- **CTIRSM01**: Shows how to request tracing of all RSM functions and events using the options DMPOFF, NOCOMASID, and NODMPREC.
- **CTIRSMSP**: Shows how to request address space, job name filtering, and trace request options to limit the tracing.

**RSM trace data in high virtual private storage**: RSM supports collecting trace data into high private memory. This storage is used as a secondary buffer data, which is first collected into buffers in fixed high common storage, and later moves out into high private memory. Having a secondary buffer is another way, besides trace data sets, to handle a large number of RSM trace records. Note that the paging activity for the secondary buffer can appear in the RSM trace records.

If you suspect that your system has a paging problem, collect the RSM trace records in page-fixed primary buffers to keep from losing records while paging. A record can be lost as the system reuses a full secondary buffer

For RSM, use BUFF in the CTnRSMxx OPTIONS parameter to specify the number of page-fixed buffers and their page sizes. In Figure 146, the statements in CTWRSM05 specify four page-fixed buffers that are 64 pages and a secondary buffer in high private memory of 640 KB.

```
TRACEOPTS
   ON
   BUFSIZE(640k)
   OPTIONS('BUFF(4,64)')
```

*Figure 146. Example: Using the CTWRSM05 parmlib member*

For RSM, the BUFSIZE parameter in the CTnRSMxx parmlib member specifies the total size of the high virtual private buffers located in the RASP address space. The space in the buffer is divided evenly between the amount of installed processors.

**TRACE and REPLY commands**

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, nnnnK, nnnnM, or OFF</td>
<td>One is required. nnnnK and nnnnM specify the size of the buffer in the RASP address space (high virtual private).</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>Yes. To trace all batch jobs, specify 'INIT' in the list of job names.</td>
</tr>
</tbody>
</table>

Chapter 12. Component trace 503
Component Trace

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Automatic Dump:** The component requests an SVC dump when the operator stops the trace or when RSM enters recovery processing. To prevent these automatic dumps when the trace is written to a trace data set or when the operator is to request the dump, specify NODMPREC and NODMPOFF in the OPTIONS parameter in the TRACE CT command or the CTnRSMxx parmlib member.

**OPTIONS parameter**
The values for the OPTIONS parameter for the CTnRSMxx parmlib member and reply for a TRACE command follow.

If you turn on the SYSRSM trace without specifying any filters or options, the component trace records every RSM function and event in all address spaces and jobs. This trace collects an enormous amount of data and degrades system performance. Use the SYSRSM filters and options to limit the amount of data recorded by the component trace. Specify tracing of specific address spaces, jobs, RSM events, and RSM functions.

The RSM trace options are divided into three groups:
- Special trace options
- RSM function trace options
- RSM event trace options

**Special trace options:** These options set the size of the fixed RSM trace buffers, tell the trace to record common area activity, and tell the system when to dump the trace data. The options are:

**BUFF=(x,y)**
Specifies the number and size of the SYSRSM trace buffers, which reside in fixed high virtual common service area (LIKECSA) storage:

- **x** The number of buffers, from 2 to 7. The default is 3.
- **y** The number of pages per buffer, from 4 to 262,144. The default is 32.

The storage for each buffer is distributed between the installed processors in the system. For that reason, it is recommended to increase the buffer size on systems with a large amount of processors. The more buffers that are specified, the more often the SRB, which empties out the fixed buffers, is called (if there is a high virtual private buffer or an external writer). The larger the buffer, the greater the burst of events it can contain without losing any trace entries.

For example, if you specify BUFF=(5,10), the component trace uses five fixed trace buffers. Each buffer contains 10 pages. The total amount of fixed storage used is 200 kilobytes.

**Note:** When choosing the amount of fixed storage to use for trace buffers, consider the amount of central storage available.

**COMASID**
Traces activity in the common area page. This is the default.

**NOCOMASID**
Prevents tracing of activity in the common area page.
Component Trace

DMPREC
Includes trace data in the SVC dump requested when RSM enters recovery processing. The SYSRSM trace is suspended while the dump is in progress. The dump contains the most recent trace data recorded prior to the problem. With this dump option, which is a default:
- The trace tables are not dumped when RSM enters recovery processing.
- Tracing continues on other processors during recovery processing.

NODMPREC
Prevents trace data from being dumped if RSM enters recovery processing.

DMPOFF
Causes trace data to be dumped when tracing for RSM is turned off with a TRACE CT,OFF,COMP=SYSRSM command or with an OFF parameter in a CTnRSMxx parmlib member.

NODMPOFF
Prevents writing of a dump when the TRACE operator command is entered to stop the trace. This is the default.

Function Trace Options: Function trace options identify the RSM functions and services to be traced. The options are:

ALLOC2G
Traces events for allocating 2G pages.

ASPCREAT
Traces events for the address space create function.

BLOCKMG3R
Traces RSM SCM block manager events.

COPYSRVG
Traces RSM copy services group. The group options, which can be specified separately, are:
- COPYSERV
  Traces copy services.
- COPYSRVC
  Traces high virtual copy service.

DFSTEAL
Traces events for the double frame steal function.

DIV
Traces all events in the data-in-virtual services group. The group options, which can be specified separately, are:
- DIVACCUN
  Trace the DIV ACCESS and DIV UNACCESS services.
- DIVMAP
  Traces the data-in-virtual MAP service.
- DIVMAPLV
  Traces the data-in-virtual MAP service (with LOCVIEW=MAP on previous ACCESS).
- DIVRES
  Traces the data-in-virtual RESET service.
Component Trace

DIVRESLV
Traces the data-in-virtual RESET service (with LOCVIEW=MAP on previous ACCESS).

DIVRTR
Traces the data-in-virtual services router.

DIVSAVE
Traces the data-in-virtual SAVE service.

DIVSLIST
Traces the data-in-virtual SAVELIST service.

DIVUNMAP
Traces the data-in-virtual UNMAP service.

DSPCONV
Traces events in the data space convert interface function.

DSPLIMIT
Traces events in the data space limit interface function.

DATASPACE
Traces all events in the data space and hiperspace group. The group options, which can be specified separately, are:

DSPSERV
Traces all events in the data space services group. The group options, which can be specified separately, are:

DSPCREAT
Traces events in the DSPSERV CREATE service.

DSPDELETE
Traces events in the DSPSERV DELETE service.

DSPDREFOFF
Traces events in DSPSERV define DREF off.

DSPDREFOFF
Traces events in DSPSERV define DREF on.

DSPEXTEN
Traces events in the DSPSERV EXTEND service.

DSPLOAD
Traces events in the DSPSERV LOAD service.

DSPIOFF
Traces events in the DSPSERV IOFF service.

DSPIOON
Traces events in the DSPSERV IOON service.

DSPOUT
Traces events in the DSPSERV OUT service.

DSPREL
Traces events in the DSPSERV RELEASE service.

DSPSRTR
Traces events in the DSPSERV router service.

DSPSRTRD
Traces events in the DSPSERV disabled RTR service.
Component Trace

**HSPSERV**
Traces all events in the hiperspace services group. The group options, which can be specified separately, are:

**HSPCACHE**
Traces events in the HSPSERV cache services.

**HSPSCROL**
Traces events in the HSPSERV scroll services.

**DUMPSERV**
Traces the dumping function.

**FAULTS**
Traces all events in the fault services group. The group options, which can be specified separately, are:

**FLTASP**
Traces all events in the address space faults group. The group options, which can be specified separately, are:

**FLTADPAG**
Traces disabled address space page faults.

**FLTAEPAG**
Traces enabled address space page faults.

**FLTAESEG**
Traces enabled address space segment faults.

**FLTAHPAG**
Traces address space page faults for address above the 2 gigabytes bar.

**FLTAHSEG**
Traces address space segment faults for address above the 2 gigabytes bar.

**FLTAREGN**
Traces address space region faults.

**FLTATYPE**
Traces address space type faults.

**FLTDSP**
Traces all events on the data space faults group. The group options, which can be specified separately, are:

**FLTDEN**
Traces enabled data space faults.

**FLTDDIS**
Traces disabled data space faults.

**FLTEPROT**
Traces protection faults.

**FREEFRAM**
Traces the free frame function.

**GEN**
Traces all events in the general function group. The group options, which can be specified separately, are:

**GENDEFER**
Traces general defers.
Component Trace

**GENIOCMP**
Trace general I/O completion.

**GENTERM**
Traces general abends.

**GLRUSTL**
Traces the global LRU Steal function.

**IARVSERV**
Traces all IARVSERV requests. The Virtual Services group options, which can be specified separately, are:

- **VSCHGACC**
  Traces IARVSERV CHANGEACCESS requests.

- **VSROUTR**
  Traces IARVSERV service router.

- **VSSSHARE**
  Traces IARVSERV SHARE requests.

- **VSSHSEG**
  Traces IARVSERV SHARESEG requests.

- **VSUNSHAR**
  Traces IARVSERV UNSHARE requests.

**IARV64**
Traces all IARV64 requests. The High Virtual services group options, which can be specified separately are:

- **V6CHACC**
  Traces IARV64 CHANGEACCESS requests.

- **V6CHGUARD**
  Traces IARV64 CHANGEGUARD requests.

- **V6COUNT**
  IARV64 COUNTPAGES requests.

- **V6DETACH**
  Traces IARV64 DETACH requests.

- **V6DISCAR**
  Traces IARV64 DISCARDDATA requests.

- **V6GETCOM**
  Traces IARV64 GETCOMMON requests.

- **V6GETSHR**
  Traces IARV64 GETSHARED requests.

- **V6GETSTR**
  Traces IARV64 GETSTOR requests.

- **V6LIST**
  Traces IARV64 LIST requests.

- **V6PAGFIX**
  Traces IARV64 PAGEFIX requests.

- **V6PAGIN**
  Traces IARV64 PAGEIN requests.
Component Trace

V6PAGOUT
Traces IARV64 PAGEOUT requests.

V6PAGUNF
Traces IARV64 PAGEUNFIX requests.

V6PROTEC
Traces IARV64 PROTECT/UNPROTECT requests.

V6ROUTR
Traces IARV64 service router.

V6SHMOMB
Traces IARV64 SHAREMEMOBJ requests.

LPGALLOC
Traces the Large Page Frame Allocation function.

MACHCHK
Traces the machine check function.

PGSER
Traces all events in the paging services group. The group options, which can be specified separately, are:

PGANY
Traces events in the page anywhere service.

PGFIX
Traces events in the page fix service.

PGFREE
Traces events in the page free service.

PGLOAD
Traces events in the page load service.

PGOUT
Traces events in the page out service.

PGPROT
Traces events in the page protect service.

PGREL
Traces events in the page release service.

PGSRTR
Traces events in the paging service routers.

PGUNPROT
Traces events in the page unprotect service.

QFSTEAL
Traces events for the quad frame steal function.

RECONFIG
Traces the reconfiguration function.

RPBPMGMT
Traces the RSM cell pool management function.

RSMPIN
Traces the RSMPIN services.
Component Trace

**SUBSPACE**
Traces all events in the subspace group. The group options, which can be specified separately, are:

- **SSPCONV**
  Traces the subspace conversion services.

- **IARSUBSP**
  Traces the subspace services group. The group options, which can be specified separately, are:
    - **SSPIDENT**
      Traces the IARSUBSP IDENTIFY service.
    - **SSPCREATE**
      Traces the IARSUBSP CREATE service.
    - **SSPASSIGN**
      Traces the IARSUBSP ASSIGN service.
    - **SSPUNASSIGN**
      Traces the IARSUBSP UNASSIGN service.
    - **SSPDELETE**
      Traces the IARSUBSP DELETE service.
    - **SSSHARE**
      Traces the IARSUBSP SHARE service.
    - **SSPUNID**
      Traces the IARSUBSP UNIDENTIFY service.
    - **SSPSRTR**
      Traces the IARSUBSP router.

**SWAP**
Traces all events in the swap services group. The group options, which can be specified separately, are:

- **REALSWAP**
  Traces events during in-real swap processing.
- **SWAPIN**
  Traces events in the swap-in service.
- **SWAPOUT**
  Traces events in the swap-out service.

- **TRACE**
  Traces the trace function. This function is always traced.

- **UIC**
  Traces the unreferenced interval count function.

- **UMCPU**
  Traces the free CPU related frames function.

- **VIO**
  Traces the virtual I/O function.

- **VR**
  Traces the V=R allocation function.

- **VSM**
  Traces all events in the VSM services group. The group options, which can be specified separately are:
Component Trace

VSMFRMN
Traces events in the FREEMAIN service.

VSMGMTN
Traces events in the GETMAIN service.

WAITSER
Traces RSM Wait function.

XCHUP
Traces the exchange up function.

XMPOST
Traces the cross memory posting function.

Event Trace Options: Event trace options identify the events for RSM to collect trace data. The options are:

ESTOR
Traces all events in the expanded storage management group. The group options, which can be specified separately, are:

ESGET
Traces get expanded storage.

ESENQ
Traces enqueue expanded storage.

ESDEQ
Traces dequeue expanded storage.

ESFREE
Traces free expanded storage.

FUNCREQ
Traces the function request event.

PAGEREQ
Traces all events in the page request group. The group options, which can be specified separately, are:

PAGEA2R
Traces requests to move a page from auxiliary to central storage.

PAGEDEF
Traces requests to move a page to central storage was deferred for lack of a frame

PAGEIPT
Traces requests to invalidate a page table entry.

PAGEP2R
Traces requests to move a page from permanent to central storage.

PAGEREL
Traces requests related to I/O in-progress or related to a defer event.

PAGER2A
Traces requests to move a page from central storage to auxiliary storage.

PAGER2P
Traces requests to move a page from central storage to permanent storage.

PAGER2R
Traces requests to move a page from central storage to central storage.
Component Trace

PGEVENTS
Traces all events in the page fix/free group. The group options, which can be specified separately, are:

FIX
Traces a page being fixed.

FREE
Traces a page being freed.

REGIONGR
Traces all events in the region table group. The group options, which can be specified separately, are:

CREG1ST
Traces the creation of a region 1st table.

CREG2ND
Traces the creation of a region 2nd table.

CREG3RD
Traces the creation of a region 3rd table.

RSTOR
Traces all events in the frame management group. The group options, which can be specified separately, are:

HVFGRP
Traces events for frame management of high virtual frames. The group options, which can be specified separately, are:

HVFRDEQ
Traces when a frame is dequeued from the high virtual frame queue.

HVFRENQ
Traces when a frame is enqueued onto the high virtual frame queue.

HVPGTDEQ
Traces when a frame is dequeued from the high virtual page table frame queue.

HVPGTEQ
Traces when a frame is enqueued onto the high virtual page table frame queue.

RSDEQ
Traces all events in the dequeue frame group. The group options, which can be specified separately, are:

RSDDER
Traces when a frame is dequeued from the deferred FREEMAIN frame queue or the orphan frame queue.

RSDFIX
Traces when a frame is dequeued from the fixed frame queue or the local quad frame queue.

RSGDFER
Traces when a frame is dequeued from the general defer frame queue.

RDPAG
Traces when a frame is dequeued from the pageable frame queue.
Component Trace

**RSDSBUF**
Traces when a frame is dequeued from the central storage buffer frame queue.

**RSDSQA**
Traces when a frame is dequeued from the SQA frame queue.

**RSDVRW**
Traces when a frame is dequeued from the V=R waiting frame queue.

**RSSENQ**
Traces all events in the enqueue frame group. The group options, which can be specified separately, are:

**RSEDEFER**
Traces when a frame is enqueued onto the deferred or orphan frame queue.

**RSEFIX**
Traces when a frame is enqueued onto the fixed frame queue or the local quad frame queue.

**RSEPAG**
Traces when a frame is enqueued onto the pageable frame queue.

**RSESBUF**
Traces when a frame is enqueued onto the central storage buffer frame queue.

**RSEGDFER**
Traces when a frame is enqueued on the general defer frame queue.

**RSESQA**
Traces when a frame is enqueued onto the SQA frame queue.

**RSEVRW**
Traces when a frame is enqueued onto the V=R waiting frame queue.

**RSFREE**
Traces all events in the free frame group. The group options, which can be specified separately, are:

**FREE2G**
Traces when a 2G frame group is freed.

**PLFREE**
Traces when a pageable large (1 MB) frame is freed.

**PSFREE**
Traces when a single pageable large (1 MB) frame is freed.

**QFFREE**
Traces when a quad group is freed.

**QHFREE**
Traces when a quad holding frame is freed.

**QSFREE**
Traces when a single quad frame is freed.

**RSFDBL**
Traces when a double frame is freed.
Component Trace

RSFSNG
Traces when a single frame is freed.

RSGET
Traces all events in the get frame group. The group options, which can be specified separately, are:

GET2G
Traces when a 2G frame group is gotten.

LSGET
Traces when a single large frame is gotten.

PLGET
Traces when a pageable large frame (1 MB) group is obtained.

PSGET
Traces when a single pageable large (1 MB) frame group is obtained.

QFGET
Traces when a quad group is gotten.

QHGET
Traces when a quad holding frame is gotten.

QSGET
Traces when a single quad frame is gotten.

RSGDBL
Traces when a double frame is gotten.

RSGSNG
Traces when a single frame is gotten.

SCMBLKMNG
Traces SCM Block Manager events. The group options, which can be specified separately, are:

SCMEVACA
Traces SCM Evacuation Add to Table.

SCMFREE
Traces the SCM Block Manager Free Block.

SCMICHCNG
Traces SCM Block Manager Increment Change.

SCMOBT
Traces the SCM Block Manager Obtain Block.

SCMOFLIN
Traces SCM Block Manager Offline Event.

SCMTRANS
Traces SCM Transfer Block ID Event

SCMSTART
Start of SCM Evacuation

SCMEND
End of SCM Evacuation

SCMSTASP
Start SCM Evacuation Address Space
Component Trace

SCMENDAS
   End SCM Evacuation Address Space

SCMSTDSP
   Start SCM Evacuation Data Space

SCMENDSP
   End SCM Evacuation Data Space

SCMPOOLG
   Get a blockid from a pool

SCMPOOLF
   Free a blockid to a pool

SCMEVACR
   Evacuate SCM from stg range

SCMDFGST
   Start SCM Defragmentation

SCMADDIN
   Add SCM increment

SCMADDEX
   Add SCM extent

SHRDATA
   Traces all events in the IARVSERV services group. The group options, which can be specified separately, are:

   GRPCREAT
      Traces the creation of new sharing groups.

   GRPDEL
      Traces the deletion of existing sharing groups.

   GRPPART
      Traces the partitioning of existing sharing groups.

   VIEWADD
      Traces the addition of views to sharing groups.

   VIEWCHG
      Traces the changing of storage attributes of the view.

   VIEWDEL
      Traces the deletion of views from sharing groups.

   VIEWMOVE
      Traces the move of existing views from one sharing group to another.

SHRINT
   Traces High Virtual Shared Interest events. The group options, which can be specified separately, are:

   SHRADD
      Traces adding shared interest.

   SHRDEL
      Traces removing shared interest.

STORMOD
   Traces all events in the storage state modification group. The group options, which can be specified separately, are:
Component Trace

**CLONEPAG**
Traces the page table entry copied to a subspace.

**CLONESEG**
Traces the segment table entry copied to a subspace.

**TRACEB**
Traces the trace buffer event. This event is always traced.

**WORKUNIT**
Traces all events in the net event trace group. The group options, which can be specified separately, are:

**ENABLE**
Traces requests to enable a unit of work.

**SUSPEND**
Traces requests to suspend a unit of work.

**RESUME**
Traces requests to resume a unit of work.

**XEPLINK**
Traces all events in the external entry point linkage group. The group options, which can be specified separately, are:

**XEPENTRY**
Traces entry to the entry point.

**XEPEXIT**
Traces exit from the entry point.

**Examples of requesting SYSRSM traces**

- **Example 1: CTRnRSMxx member**
  The member requests tracing of the FAULTS services group, the PGANY service, and the VIO function, but only for address spaces X’11’ and X’41’ and for job PGM1.
  TRACEOPTS
  ON
  ASID(11,41)
  JOBNAME(PGM1)
  OPTIONS(’FAULTS’,’PGANY’,’VIO’)

- **Example 2: TRACE command**
  The example specifies that options are to be obtained from the parmlib member CTWRSM17.
  trace ct,on,comp=sysrsm,parm=ctwrsm17

- **Example 3: TRACE command**
  The example requests the same trace as Example 2, but specifies all options in the REPLY.
  trace ct,on,comp=sysrsm
  * 78 ITT006A ...
  reply 78,options=(faults,pgany,vio),asid=(11,41),jobname=(pgm1),end

**Formatting a SYSRSM trace**

Format the trace with an IPCS CTRACE COMP(SYSRSM) subcommand. The subcommand has no OPTIONS values.
Component Trace

Output from a SYSRSM trace

Example: RSM component trace records formatted with CTRACE
COMP(SYSRSM) FULL subcommand is an example of RSM component trace records formatted with the CTRACE COMP(SYSRSM) FULL subcommand.

Example: RSM component trace records formatted with CTRACE
COMP(SYSRSM) FULL subcommand

XEPENTRY 00000001 18:18:06.441411 External Entry Point Entry
  FUNC1... PGFIX  Page Fix
  JOBN1... ASNB  ASID1... 000D  PLOCKS.. 80000001 CPU...... 0000
  JOBN2... ASNB  ASID2... 000D  RLOCKS.. 80000000
  KEY..... 0036  ADDR.... 015FF008 ALET.... 00000000
  7D00
  KEY..... 002C  ADDR.... 005FEF80 ALET.... 00000000
  00000000 005EF000 005EFFFF 00F00200 00F8D000 00F8D000 00F8D900
  005EFFC3 005EF02C 00F8D050 005EF02C 00F8E000 00F8E000 005FEF78
  00FF91DA 81082798

FREE 00000003 18:18:06.441921 Page Being Freed
  FUNC1... PGFREE Page Free
  JOBN1... ASNB  ASID1... 000D  PLOCKS.. 88004001 CPU...... 0000
  JOBN2... ASNB  ASID2... 000D  RLOCKS.. 88004000
  KEY..... 0036  ADDR.... 015FF008 ALET.... 00000000
  8100
  KEY..... 005D  ADDR.... 005EF000 ALET.... 00000000
  0151182C 012D2E60 81C00000 03000000 0000000D 005EF000 00000000 00000000

FREE 00000004 18:18:06.461766 Page Being Freed
  FUNC1... PGFREE Page Free
  JOBN1... ASNB  ASID1... 000D  PLOCKS.. 88004001 CPU...... 0000
  JOBN2... ASNB  ASID2... 000D  RLOCKS.. 88004000
  KEY..... 0036  ADDR.... 015FF008 ALET.... 00000000
  8100
  KEY..... 005D  ADDR.... 011F8220 ALET.... 00000000
  0151182C 012D2E60 81C00000 03000000 0000000D 005EF000 00000000 00000000

The fields that you may need in the report are:

**FUNC1**
- The function in control at the time the trace event was recorded.

**JOBN1**
- The job name identifying the address space that contains the unit of work requesting the RSM service.
Component Trace

**JOBN2**
The job name that matched a name in the job name list provided with the TRACE operator command.

**ASID1**
The ASID identifying the address space that contains the unit of work requesting the RSM service.

**ASID2**
The ASID that matched an identifier in the ASID list provided with the TRACE operator command.

**CPU**
The central processor identifier for the processor the trace is running on.

---

**SYSSPI component trace**

Before using this component trace, ensure that you have read:
- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSSPI component trace for the service processor interface (SPI).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSSPI:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTISPI00</td>
</tr>
</tbody>
</table>
|                              | If there is an CTISPI00 member in the parmlib concatenation, it will be used during IPL to control the initial state of the trace.  
|                              | If there is no CTISPI00 member in parmlib, the initial state will be OFF. |
| Default tracing              | No          |
| Trace request OPTIONS parameter | In CTISPI00 and REPLY for TRACE command |
| Buffer                       | • Default: 64KB  
|                              | • Range: 32KB to 1216KB  
|                              | • Size set by: TRACE CT,nnnK,COMP=SYSSPI command or the BUFSIZE specified in parmlib member CTISPIxx.  
|                              | • Change size after IPL: Yes, by turning the trace off, changing the size and then turning the trace back on.  
|                              | • Location: In the component area |
| Trace records location       | ECSA        |
| Request of SVC dump          | By the component |
| Trace formatting by IPCS     | CTRACE COMP(SYSSPI) |
| Trace format OPTIONS parameter | None        |

---

**Requesting a SYSSPI trace**

Request a SYSSPI trace at the direction of the IBM Support Center. Do the following:
Component Trace

1. Start the trace with the command:
   TRACE CT,ON,COMP=SYSSPI

2. After the interval specified by IBM, stop the trace with the command:
   TRACE CT,OFF,COMP=SYSSPI

When the buffer fills up, the component requests an SVC dump, which includes the contents of the buffer. Optionally, the operator could enter a DUMP command.

CTISPI00 parmlib member

The following table indicates the parameters you can specify on a CTISPI00 parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTISPI00?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>No</td>
</tr>
</tbody>
</table>

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>No</td>
</tr>
</tbody>
</table>

OPTIONS parameter

The values for the OPTIONS parameter for the CTISPI00 parmlib member and reply for a TRACE command, in an alphabetical order, are:

DMPOFF
   Take a SVC dump when the trace is turned off. This is the default.

NODMPOFF
   Do not take an SVC dump when the trace is turned off.

NOWRAP
   When the buffer becomes full, take an SVC dump and discard all trace data.
   This is the default.

WRAP
   When the buffer becomes full, do not take an SVC dump, and discard some of the oldest trace data to make room for new trace data.

Formatting a SYSSPI trace

1. Use SPZAP to change a module. IBM supplies the change.
2. Format the trace with an IPCS CTRACE COMP(SYSSPI) subcommand. The subcommand has no options values.
SYSTTRC transaction trace

Transaction trace does not participate in component trace-controlled processing. It is a standalone tracing facility. Do not use trace CT commands for transaction trace. Do not attempt to add a component trace parmlib member for transaction trace. For information on transaction trace, see Chapter 13, “Transaction trace,” on page 539.

SYSVLF component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSVLF component trace for the virtual lookaside facility (VLF).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSVLF:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>None</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; minimal; unexpected events</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>None</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: N/A</td>
</tr>
<tr>
<td></td>
<td>• Range: N/A</td>
</tr>
<tr>
<td></td>
<td>• Size set by: MVS system</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: No</td>
</tr>
<tr>
<td></td>
<td>• Location: Data space. Enter DISPLAY J,VLF to identify the VLF data spaces. In the REPLY for the DUMP command, specify DSPNAME=('VLF'.Dclsname,'VLF'.Cclsname), where clsname is a VLF class name.</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer, data-space buffer</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command or when SYSVLF full tracing is turned off</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSVLF)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>None</td>
</tr>
</tbody>
</table>

Requesting a SYSVLF trace

A minimal trace runs whenever VLF is in control. No actions are needed to request the minimal trace.

To record more than the minimal trace, request full tracing with the TRACE operator command. Note that full tracing can slow system performance. The following table indicates the parameters you can specify on a TRACE CT command. In response to the command, the system does not prompt the operator for a reply.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>No</td>
</tr>
</tbody>
</table>
When you turn the full tracing off, the system writes a dump containing the trace records, then resumes minimal tracing.

**Examples of requesting and stopping SYSVLF full traces**

- **Example 1:** Requesting a SYSVLF full trace
  
  The command requests a full trace.
  
  \[ \text{TRACE CT,ON,COMP=SYSVLF} \]

- **Example 2:** Stopping a SYSVLF full trace
  
  The command turns off full tracing. In response, the system writes a dump and resumes minimal SYSVLF tracing.
  
  \[ \text{TRACE CT,OFF,COMP=SYSVLF} \]

- **Example 3:** Command for SYSVLF tracing in a sysplex
  
  The following command turns on tracing for a SYSVLF trace in the systems of a sysplex. Because SYSVLF has no parmlib member, the CTIITT00 member is used to prevent prompts.
  
  \[ \text{route *all,trace ct,on,comp=sysvlf,parm=ctiitt00} \]

**Formatting a SYSVLF trace**

Format the trace with an IPCS CTRACE COMP(SYSVLF) subcommand. The subcommand has no OPTIONS values.

**Output from a SYSVLF trace**

Figure 147 is an example of VLF component trace records formatted with the CTRACE COMP(SYSVLF) FULL subcommand. It shows formatted exception records from the trace buffers.

```
VLF COMPONENT TRACE FULL FORMAT
**** 01/27/90
COFRCVRY 00000000 16:03:02.181262 VLF RECOVERY ENTRY
HASID... 000E SASID... 000E CPUID... FF170284 30900000
MODNAME. COFMPURG ABEND... 840C4000 REASON.. 00000011
EPTABLE. PURG ESTA .... .... .... .... .... .... .... .... .... ....
COFRCVRY 00000001 16:03:02.181324 VLF RECOVERY EXIT
HASID... 000E SASID... 000E CPUID... FF170284 30900000
MODNAME. COFMPURG ABEND... 840C4000 REASON.. 00000011
RETCODE. 00000000 RSNCODE. 00000000 FTPRTS.. 80300000 DATA.... 00000000
```

Figure 147. Example: VLF component trace records formatted with CTRACE COMP(SYSVLF) FULL subcommand

The following explains fields in the report. Additional fields that are not shown in the example can be in a report. These additional fields are explained below in the Other Fields section.
Component Trace

**COFRCVRY**
- The name or identifier of the trace record.

**00000000**
- The identifier in hexadecimal

**16:03:02.181262**
- The time stamp indicating when the record was placed in the trace table

**HASID... 000E**
- The home address space identifier

**SASID... 000E**
- The secondary address space identifier

**CPUID... FF170284 30900000**
- The identifier of the processor that placed the record in the trace table

**CALLER**
- The address of the routine that issued a VLF service request, such as DEFINE, CREATE, NOTIFY, PURGE, etc..

**MODNAME. COFMPURG**
- The name of the module that was running

**ABEND... 840C4000**
- The abend that occurred and caused VLF to enter recovery is 0C4

**REASON... 00000011**
- The reason code associated with the abend

**EPTABLE. PURG ESTA**
- Information used for diagnosis by IBM

**RETCODE. 00000000**
- The return code that was issued by the module that is exiting

**RSNCODE. 00000000**
- The reason code that was issued by the module that is exiting

**FTPRTS.. 80300000**
- Information used for diagnosis by IBM

**DATA.... 00000000**
- Information used for diagnosis by IBM

*Other Fields:* Fields that are not shown in the example CTRACE output but that may appear in a report are:

**CINDX**
- The concatenation index of the major name for which an object has been created or retrieved

**CLASS... NPDS3**
- The name of a VLF class

**DDNAME**
- The DDNAME of the concatenated data set list

**FUNC=xxxx**
- Indication of the function for which a NOTIFY occurred

**FUNCCODE**
- The hexadecimal value of the NOTIFY function code when it cannot be interpreted
**Component Trace**

**MAJOR**
The major name

**MINADDR**
Address of a field containing a minor name

**MINALET**
Access list entry token (ALET) associated with the address used to locate the minor name

**MINOR**
The minor name

**OBJSIZE**
The total size, in bytes, of the object returned by a COFRETRI macro

**PARMS**
Hexadecimal dump of the COFNOTIF macro parameter list

**TLSTADDR**
Address of a target list for a COFRETRI macro

**TLSTALET**
Access list entry token (ALET) of a target list for a COFRETRI macro

**TLSTSIZE**
The length, in bytes, of the target list

**UTOKEN**
User token returned by a COFIDENT macro and required as input for COFREMOV, COFCREAT, and COFRETRI macros

**VOLSER**
The volume serial

---

**SYSWLM component trace**

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSWLM component trace for the workload manager (WLM).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSWLM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>None</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; minimal; unexpected events</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>None</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: 64KB</td>
</tr>
<tr>
<td></td>
<td>• Range: 64KB - 16M</td>
</tr>
<tr>
<td></td>
<td>• Size set by: MVS system</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: Yes, when starting a trace</td>
</tr>
<tr>
<td></td>
<td>• Location: Extended common service area (ECSA)</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer, trace data set</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSWLM)</td>
</tr>
</tbody>
</table>
Information for SYSWLM:

Trace format OPTIONS parameter: None

Requesting a SYSWLM trace

Request a SYSWLM component trace by a TRACE CT command.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, nnnnK, OFF</td>
<td>One is required. nnnnK specifies the size of the buffer.</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Allowed on TRACE CT for Writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples of requesting SYSWLM traces

Figure 148 shows an example of how to request a SYSWLM component trace.

```
trace ct,on,comp=syswlm
* 17 ITT006A ...
reply 17,end
```

Figure 148. Example: requesting a SYSWLM component trace

Formatting a SYSWLM trace

Format the trace with an IPCS CTRACE COMP(SYSWLM) subcommand. The subcommand has no OPTIONS values.

Output from a SYSWLM trace

Figure 149 on page 525 is an example of SYSWLM component trace records formatted with the CTRACE COMP(SYSWLM) SHORT subcommand.
Component Trace

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLMEPEXT</td>
<td>00005004</td>
<td>14:07:42.807618</td>
<td>Entry Point Exited</td>
</tr>
<tr>
<td>WLMEPENT</td>
<td>00005003</td>
<td>14:07:46.335563</td>
<td>Entry Point Entered</td>
</tr>
<tr>
<td>WLMEPEXT</td>
<td>00005004</td>
<td>14:07:46.336376</td>
<td>Entry Point Exited</td>
</tr>
<tr>
<td>WLMEPENX</td>
<td>00005005</td>
<td>14:07:46.336540</td>
<td>Entry Point Entered Exception</td>
</tr>
<tr>
<td>WLMEPENT</td>
<td>00005003</td>
<td>14:07:46.336557</td>
<td>Entry Point Entered</td>
</tr>
<tr>
<td>WLMEPENT</td>
<td>00005003</td>
<td>14:07:46.337909</td>
<td>Entry Point Entered</td>
</tr>
<tr>
<td>SMSYNMEM</td>
<td>00000921</td>
<td>14:07:46.512018</td>
<td>SM Synch XCF Member</td>
</tr>
<tr>
<td>WLMEPEXT</td>
<td>00005004</td>
<td>14:07:46.512360</td>
<td>Entry Point Exited</td>
</tr>
<tr>
<td>WLMEPENT</td>
<td>00005003</td>
<td>14:07:46.512374</td>
<td>Entry Point Entered</td>
</tr>
<tr>
<td>SMSYNMEM</td>
<td>00000921</td>
<td>14:07:46.594486</td>
<td>SM Synch XCF Member</td>
</tr>
</tbody>
</table>

Figure 149. Example: SYSWLM component trace records formatted with CTRACE COMP(SYSWLM) SHORT subcommand

Figure 150 shows an example of SYSWLM component trace records formatted with the CTRACE COMP(SYSWLM) FULL subcommand.

<table>
<thead>
<tr>
<th>SYSNAME</th>
<th>MNEMONIC</th>
<th>ENTRY ID</th>
<th>TIME STAMP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SY1</td>
<td>WLMEPENT</td>
<td>00005003</td>
<td>14:52:23.449339</td>
<td>Entry Point Entered</td>
</tr>
<tr>
<td></td>
<td>FUNCID...</td>
<td>0409</td>
<td>CPU...</td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td>HOMEASID.</td>
<td>000B</td>
<td>HJOBNAME.</td>
<td>WLM</td>
</tr>
<tr>
<td></td>
<td>REQASID..</td>
<td>0000</td>
<td>RJOBNAME.</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td></td>
<td>KEY.....</td>
<td>5018</td>
<td>RUCA_EPIDS</td>
<td>IWMDMPRP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>04098000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEY.....</td>
<td>501E</td>
<td>PARM1</td>
<td>00000084</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEY.....</td>
<td>501F</td>
<td>PARM2</td>
<td>00000040</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEY.....</td>
<td>5020</td>
<td>PARM3</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Figure 150. Example: SYSWLM component trace records formatted with CTRACE COMP(SYSWLM) FULL subcommand

The following explains fields in the report.

**FUNCID**

The module table entry for the module that wrote the trace record.

**CPU**

The CPU that the module was running on.

**HOMEASID**

ASID from PSAAOLD.

**REQASID**

ASID that was explicitly coded on trace invocation.

**HJOBNAME**

JOBNAME of home address space.

**RJOBNAME**

JOBNAME that was explicitly coded on trace invocation.

**KEY**

Identifies the type of data that follows. The data is formatted in both HEX and EBCDIC.
The following summarizes information for requesting a SYSXCF component trace for the cross-system coupling facility (XCF).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSXCF:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTnXCFxx. Default member: CTIXCF00 specified in COUPLE00 member</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; minimal; unexpected events</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTnXCFxx or REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default: 4MB</td>
</tr>
<tr>
<td></td>
<td>• Range: 16KB - 16MB (System rounds size up to a multiple of 72 bytes.)</td>
</tr>
<tr>
<td></td>
<td>• Size set by: CTnXCFxx member</td>
</tr>
<tr>
<td></td>
<td>• Change size after IPL: No</td>
</tr>
<tr>
<td></td>
<td>• Location: Extended local system queue area (ELSQA) of XCFAS</td>
</tr>
<tr>
<td>Trace records location</td>
<td>Address-space buffer, trace data set</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSXCF)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Requesting a SYSXCF trace**

Specify options for requesting a SYSXCF component trace on a CTnXCFxx parmlib member or on the reply for a TRACE CT command.

If you specify additional tracing options while the system is running, place the trace records in a trace data set or sets, because the trace buffer size specified at initialization cannot be changed while the system is running. Specify NOWRAP to keep from losing trace records.

**Note**: NOWRAP prevents trace records written to the data set or sets from being overwritten. Once the data set or sets are filled, no more records are written to them. The system still writes trace records to the address-space buffers. The system wraps the address-space buffers, so that trace records may be lost. Be sure to allocate enough space on the data set or sets to hold all the records needed for diagnosis.

**CTnXCFxx parmlib member**

The following table indicates the parameters you can specify on a CTnXCFxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnXCFxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Component Trace

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnXCFxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes, at IPL or when reinitializing XCF</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PRESET</td>
<td>No</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note:** You can change the buffer size only at IPL or when reinitializing XCF. Specify the new buffer size in the BUFSIZE parameter on the CTnXCFxx member being used.

IBM supplies a CTIXCF00 parmlib member, which specifies that component tracing for XCF be initialized with the component default buffer size and minimal component tracing active. The content of CTIXCF00 is:

```
TRACEOPTS ON
```

**TRACE and REPLY commands**
The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>No</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>No</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**OPTIONS parameter**
The values for the OPTIONS parameter for the CTnXCFxx parmlib member and reply for a TRACE command, in alphabetical order, are:

**ARM**
Traces events for automatic restart management services.
Component Trace

CFRM
   Traces events for coupling facility resource management services.

GROUP
   Traces events for group services, such as XCF groups joining or disassociating from XCF services.

GRPNAME=(groupname[,groupname]...)
   Reduces tracing to events for only the specified XCF groups. If GRPNAME is specified, the GROUP, SERIAL, SIGNAL, and STATUS options are filtered by the specified XCF group or groups; the STORAGE option is not filtered by GRPNAME. You can specify up to 8 XCF groups.

MODID=(modid[,modid]...)
   Reduces tracing of all events to only the specified XCF module IDs. You can specify up to eight XCF Module IDs. Use XCF Module ID tracing only when requested by IBM Support.

NOTEPAD
   Traces events for XCF note pad services processing.

NPNAME=(npname[,npname]...)
   Reduces tracing of NOTEPAD events related to XCF note pad services processing to only the specified note pad names or note pad names that match the specified note pad name patterns. If NPNAME is specified, the NOTEPAD option trace events are filtered by the specified note pad names. You can specify up to four note pad names or note pad name patterns. To be valid, each npname must meet the following conditions:
   - A note pad name can consist of two to four sections separated by periods.
   - If a section is not specified, it is defaulted to all blanks.
   - The first and the second sections must not be blank.
   - Each section, if specified, must be left-justified with no trailing blanks.
   - Each section can contain up to eight upper case alphanumeric (A-Z,0-9), national (@,#,$), or underscore (_) characters.
   - Any section can contain the asterisk (*) wild card character, which is used to match zero (0) or more characters (for example, OWN*.*).

SERIAL
   Traces events for serialization services.

SERVER
   Traces events for client/server services processing.

SFM
   Traces events for sysplex failure management services.

SIGNAL
   Traces events for signalling services processing.

STATUS
   Traces events for XCF monitoring services and sysplex partitioning services.

STORAGE
   Traces events for storage management services.

SRVNAME=(servername[,servername]...)
   Reduces tracing of SERVER events related to XCF client/server processing to only the specified server names or server names that match the specified server name patterns. If SRVNAME is specified, the SERVER option trace
Component Trace

Events are filtered by the specified server names. You can specify up to 4 server names/server name patterns. To be valid, the server name must meet the following conditions:

- Server names consist of one to four 8 byte sections separated by a period.
- If a section is not specified, it is defaulted to all blanks.
- Each section can contain any alphanumeric (A-Z,a-z,0-9), national (@,#,$), or underscore (_) character.
- Any section but the first can be entirely blank.
- Any section can contain the asterisk (*) wild card character which is used to match zero (0) or more characters (for example, SYS*.*).
- The server name can be enclosed within single quotes to preserve case sensitivity (for example, 'sysxcf.*')

**VECTOR**

Traces all events related to list notification or local cache vector processing.

One of the other trace options might also trace these events.

**Examples of requesting SYSXCF traces**

In Figure 151, the CTnXCFxx member requests STORAGE and SIGNAL options. To minimize lost trace data, the member also starts external writer WTRDASD1 with the NOWRAP option specified and connects the trace to the writer.

```
TRACEOPTS
WTRSTART(WTDASD1) NOWRAP
ON
WTR(WTDASD1)
OPTIONS('STORAGE','SIGNAL')
```

Figure 151. Example: CTnXESxx member requesting a SYSXCF trace

The example shown in Figure 152 uses the TRACE command to request the same type of trace that is shown in Figure 151.

```
trace ct,wrtstart=wtdasd1
trace ct,comp=sysxcf
* 62 ITT006A ...
 r 62,trace=start,options=(storage,signal),end
```

Figure 152. Example: TRACE command for SYSXCF trace

**Formatting a SYSXCF trace**

Format the trace with an IPCS CTRACE COMP(SYSXCF) subcommand. The OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records.

The options are:

**ARM**

Formats trace records for automatic restart management services.
Component Trace

CFRM
  Formats trace records for coupling facility resource management services.

CLUSTER
  Formats trace records for XCF z/OS cluster manageable resource services.

GROUP
  Formats trace records for XCF group services, such as groups joining or
disassociating from XCF services.

NOTEPAD
  Formats trace records for XCF Note Pad Services.

SERIAL
  Formats trace records for serialization services.

SERVER -
  Formats trace records for XCF client/server services.

SFM
  Formats trace records for sysplex failure management services.

SIGNAL
  Formats trace records for signalling services processing.

STATUS
  Formats trace records for XCF monitoring services and sysplex partitioning
  services.

STORAGE
  Formats trace records for storage management services.

THREAD=(thread, [ ,thread]...)
  Formats trace records filtered by one or more specific trace threads. The XCF
  component trace thread filter option provides the capability to limit the
  presentation of trace entries to specified trace threads. XCF uses trace threads
to correlate processing of units of work between the subcomponents of the
XCF component. You can enter trace threads entered in hexadecimal notation
using x'tttttttt' or tttttttt notation (for example, x'01001768' or 01001768) with
or without the leading x and surrounding quotes (').

Output from a SYSXCF trace

Figure 153 on page 531 shows an example of SYSXCF component trace records
formatted with the CTRACE COMP(SYSXCF) FULL subcommand.
SYSXES component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 377
- “Obtaining a component trace” on page 385
- “Viewing the component trace data” on page 398

The following summarizes information for requesting a SYSXES component trace for cross-system extended services (XES).

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSXES:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parmlib member</td>
<td>CTnXESxx; default member: CTIXES00 specified in COUPLE00 member</td>
</tr>
<tr>
<td>Default tracing</td>
<td>Yes; minimal; unexpected events</td>
</tr>
<tr>
<td>Trace request OPTIONS parameter</td>
<td>In CTnXESxx or REPLY for TRACE command</td>
</tr>
<tr>
<td>Buffer</td>
<td>• Default:</td>
</tr>
<tr>
<td></td>
<td>  • 336KB for connector-related SUB trace buffers. There are multiple</td>
</tr>
<tr>
<td></td>
<td>instances of these trace buffers in use on the system</td>
</tr>
<tr>
<td></td>
<td>  • 32MB for the GLOBAL SUB trace buffer</td>
</tr>
<tr>
<td></td>
<td>  • Range:</td>
</tr>
<tr>
<td></td>
<td>     • 16KB - 16MB for connector-related SUB trace buffers</td>
</tr>
<tr>
<td></td>
<td>     • The size for the GLOBAL SUB trace buffer cannot be changed.</td>
</tr>
<tr>
<td></td>
<td>  • Size set by: CTnXESxx member or TRACE CT command</td>
</tr>
<tr>
<td></td>
<td>  • Change size after IPL: Yes</td>
</tr>
<tr>
<td></td>
<td>  • Location: 64-bit common storage above the 2GB bar. In the REPLY for</td>
</tr>
<tr>
<td></td>
<td>the DUMP command, specify SDATA=XESDATA.</td>
</tr>
</tbody>
</table>
Component Trace

<table>
<thead>
<tr>
<th>Information</th>
<th>For SYSXES:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace records location</td>
<td>64-bit common storage above the 2GB bar, trace data set</td>
</tr>
<tr>
<td>Request of SVC dump</td>
<td>By DUMP or SLIP command</td>
</tr>
<tr>
<td>Trace formatting by IPCS</td>
<td>CTRACE COMP(SYSXES)</td>
</tr>
<tr>
<td>Trace format OPTIONS parameter</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Ensure that the HVCOMMON system parameter which specifies the size of the 64-bit common area reflects a size adequate to allow for the allocation of 4GB of 64-bit common storage by XES for SYSXES trace buffers plus additional system requirements for 64-bit common storage. The system default for HVCOMMON is 64GB.

SYSXES supports sublevel tracing. Tracing options are inherited through a hierarchy of trace levels. If you set trace options without specifying a sublevel, the options apply at the highest level, or head, of the hierarchy. A sublevel inherits its trace options from the next higher level, unless options are specified explicitly for the sublevel. If you set trace options for a sublevel, the options are inherited by any sublevels lower in the hierarchy. Figure 154 shows the hierarchical structure of SYSXES traces.

Figure 154. SYSXES SUB Trace Structure

Two classes of sublevel traces inherit the head trace options:

- The global sublevel trace has its own trace buffer and controls tracing that is not related to any particular connection. Request GLOBAL tracing by specifying SUB=(GLOBAL).
• Connection sublevels control tracing for a particular connection. Each connection has its own trace buffer. Connection sublevels are filtered hierarchically based on:
  1. Structure name (STRNAME) of the coupling facility structure to which the system is connected
  2. Address space identifier (ASID) of the address space from which the connection was made
  3. Connection name (CONNAME) of the particular connector for which tracing is requested

Therefore, options specified for a particular structure name are inherited only by address spaces connected to that structure. Options specified for a particular address space are inherited only by connections that are connected through that address space.

Specify SUB=(strname), SUB=(strname.asid), or SUB=(strname.asid.conname), depending on the degree of specificity you need. Do not specify conname without specifying asid. Also, do not specify asid without specifying strname.

When specifying a structure name or connector name on the SUB option, if the name contains special characters, it must be in quotation marks. If it is in quotation marks, upper and lower case characters are not the same. Therefore the case information is important and must identically match the name used by the system. Once the name is enclosed in quotation marks it becomes case sensitive.

Requests a SYSXES trace

Specify options for requesting a SYSXES component trace on a CTnXESxx parmlib member or on the reply for a TRACE CT command.

CTnXESxx parmlib member

The following table indicates the parameters you can specify on a CTnXESxx parmlib member.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on CTnXESxx?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>Yes</td>
</tr>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>SUB</td>
<td>Yes, but only for a sublevel trace</td>
</tr>
<tr>
<td>PRESET</td>
<td>Yes, but only for a sublevel trace</td>
</tr>
<tr>
<td>LIKEHEAD</td>
<td>No</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Setting buffer size: To select a size for your trace buffers, consider the following:

• The trace buffers can be smaller if you are using an external writer, because buffer wrapping is not a concern.
• When re-creating a problem, you might first want to make the buffer size larger.
• SYSXES has one trace buffer of the specified size per connector, plus one for the global trace. The amount of storage used can be significant if the system is going
to have many connectors. The SYSXES trace buffers are allocated in 64-bit
common storage. If the range of 64-bit common storage allocated by XES is used
up (4GB), subsequent connections will not be traced because buffer space is not
available.

- When the BUFSIZE parameter is not specified, SYSXES will allocate a buffer at
  the default size per connector.
- The SYSXES trace buffers are in disabled reference (DREF) 64-bit common
  storage, so storage constraints may limit buffer size.

Changing buffer size: To change the size of your trace buffers while a trace is
running, either issue a TRACE CT command or activate a different CTnXESxx
parmlib member. You can use these methods to change SUB levels in the hierarchy
so that different SUB traces can have different sized buffers. The SYSXES GLOBAL
sub buffer size is set by the system to 32MB. You cannot override this default
value.

TRACExE and REPLY commands
The following tables indicate the parameters you can specify on TRACE CT
commands and a REPLY.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON or OFF</td>
<td>One is required</td>
</tr>
<tr>
<td>nnnnK or nnnnM</td>
<td>Yes</td>
</tr>
<tr>
<td>COMP</td>
<td>Required</td>
</tr>
<tr>
<td>SUB</td>
<td>Yes</td>
</tr>
<tr>
<td>PARM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on TRACE CT for Writer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTRSTART or WTRSTOP</td>
<td>One is required, if a writer is being used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Allowed on REPLY for trace?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
</tr>
<tr>
<td>JOBNAME</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Yes</td>
</tr>
<tr>
<td>WTR</td>
<td>Yes</td>
</tr>
</tbody>
</table>

OPTIONS parameter
The values for the OPTIONS parameter for the CTnXESxx parmlib member and
reply for a TRACE command, in alphabetical order, are:

ALL
  Traces events listed for all of the options.

CONFIG
  Traces changes in the state of connectivity to the coupling facility, such as
  addition or removal of paths.

CONNECT
  Traces events for system and subsystem components that connect to or
  disconnect from XES resources and for exit processing.
Component Trace

HWLAYER
Traces events for the XES services that handle communications with the coupling facility.

LOCKMGR
Traces events related to global management of resources and to global management-related exits.

RECOVERY
Traces events within the modules that handle XES resource access failures, for both resource allocation and mainline command processing. This option provides more details than is provided by default.

REQUEST
Traces events related to requests to access data through XES mainline services.

SIGNAL
Traces events related to XES internal signalling.

STORAGE
Traces events related to management of XES control blocks.

VECTOR
Traces all events related to list notification or local cache vector processing. One of the other trace options might also trace these events. The VECTOR option also traces usage of the IXLVECTR API. This trace is distinct from component trace and is formatted by the IPCS XESDATA CONNECT report rather than through the IPCS CTRACE command. Activating the VECTOR option might impact performance, so enable it at the connector or structure subtrace level rather than for all SYSXES traces.

Examples of requesting SYSXES traces
In Figure 155 the CTnXESxx member requests a trace of HWLAYER, LOCKMGR, CONNECT, and REQUEST trace events and a buffer size of 100KB.

```
TRACEOPTS
ON
  OPTIONS('HWLAYER','LOCKMGR','CONNECT','REQUEST')
  BUFSIZE(100K)
```

Figure 155. Example: CTnXESxx member requesting a SYSXES trace

The example shown in Figure 156 requests a trace of CONNECT, CONFIG, and STORAGE trace events for connection CON3 in ASID 5 for structure STR3.

```
trace ct,on,comp=sysxes,sub=(str3.asid(5).con3)
+ 17 ITT006A ...
reply 17,options=(connect,config,storage),end
```

Figure 156. Example: TRACE command for SYSXES trace

Formatting a SYSXES trace
Format the trace with an IPCS CTRACE COMP(SYSXES) subcommand. The OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you
Component Trace

requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records.

**ALL**
- Formats all trace records.

**CONFIG**
- Formats changes in the state of connectivity to the coupling facility.

**CONNECT**
- Formats events for system and subsystem components that connect to or disconnect from XES resources and for exit processing.

**HWLAYER**
- Formats events for the XES services that handle communications with the coupling facility.

**LOCKMGR**
- Formats events related to global management of resources and to global management-related exits.

**RECOVERY**
- Formats events within the modules that handle XES resource access failures.

**REQUEST**
- Formats events related to requests to access data through XES mainline services.

**SIGNAL**
- Formats events related to XES internal signalling.

**STORAGE**
- Formats events related to management of XES control blocks.

In the CTRACE subcommand, the SUB((subname.subname.subname)) parameter specifies the sublevel traces. A subname is:

- **GLOBAL** for an event not related to a particular connection.
- **strname.asid.conname** for an event related to the specified connector. The subname for a connection-related sublevel can contain up to three parts:
  - **strname** for the structure
  - **asid** for the address space identifier (ASID)
  - **conname** for the connection name, if **asid** is also specified

Any sublevel specification is valid for the QUERY option; for example:

- **SUB(STR3)**
- **SUB(STR3.ASID(5))**

Only the GLOBAL and fully qualified connection sublevel specifiers are valid with a COMP parameter; for example:

- **SUB(GLOBAL)**
- **SUB(STR3.ASID(5).CON3)**

### Output from a SYSXES trace

Figure 157 on page 537 is an example of SYSXES component trace records formatted with the following subcommand:

```bash
CTRACE COMP(SYSXES) SUB((GLOBAL)) SHORT OPTIONS((CONNECT,HWLAYER))
```
<table>
<thead>
<tr>
<th>COMPONENT TRACE SHORT FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP(SYSXES) SUBNAME((GLOBAL))</td>
</tr>
<tr>
<td>**** 10/20/93</td>
</tr>
<tr>
<td>MNEMONIC</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>HWLAYER</td>
</tr>
<tr>
<td>HWLAYER</td>
</tr>
<tr>
<td>HWLAYER</td>
</tr>
<tr>
<td>CONNECT</td>
</tr>
<tr>
<td>HWLAYER</td>
</tr>
<tr>
<td>HWLAYER</td>
</tr>
<tr>
<td>HWLAYER</td>
</tr>
<tr>
<td>HWLAYER</td>
</tr>
<tr>
<td>CONNECT</td>
</tr>
<tr>
<td>CONNECT</td>
</tr>
<tr>
<td>HWLAYER</td>
</tr>
</tbody>
</table>

Figure 157. Example: formatted SYSXES component trace records
Chapter 13. Transaction trace

Transaction trace provides a consolidated trace of key events for the execution path of application or transaction type work units running in a multi-system application environment. By tracing the path of a work unit running in a single system, or (more importantly) across systems in a sysplex environment, that is being processed by multi-system transaction servers, subsystem interfaces, and resource managers, transaction trace enables a system programmer to debug problems in those environments.

The essential task of transaction trace is to aggregate data showing the flow of work between components in the sysplex that combine to service a transaction. Transaction trace traces events such as component entry, exit, exceptions and major events such as COMMIT, and ROLLBACK. Do not use transaction trace as a component tracing facility.

How transaction trace works

Transaction trace (TTrace) is attached as a daughter task in the system trace address space, after master scheduler initialization completes. Once initialization has completed, and the first transaction trace command is entered with a filter that specifies the attributes of the work unit(s) to be traced, transaction trace is activated. Additional information, such as the use of an external writer, is also allowable for transaction trace processing.

Once transaction trace is activated, WLM Classify invokes a filter exit to determine whether the current work unit is traced. The work unit's attributes are compared with the command filter attributes to determine if tracing should occur. If tracing is required, a non-zero token is built and returned to the Classify caller. If no tracing is performed for that work unit, set the transaction trace token to zero. The caller (CICS or IMS, for example) propagates the token in a manner similar to the propagation of the service class token.

Next, transaction trace macros:

- determine if tracing can be performed (ITZQUERY)
- initiate the writing of a transaction trace record (ITZEVENT).

Transaction trace writes trace data in a transaction trace data space in the trace address space. When an external writer is defined, the record is also written to the external writer. Use interactive problem control system (IPCS) to view the transaction trace records.

Transaction trace commands

Use the following commands with transaction trace. For information about using the TRACE or DISPLAY TRACE commands with transaction trace, see z/OS MVS System Commands.

- TRACE TT
- DISPLAY TRACE,TT
The TRACE TT command

Transaction trace uses the MVS TRACE command with the TT keyword to:

- Start transaction trace.
- Add additional trace filter sets.
- Remove an active trace filter set.
- Stop transaction trace.
- Start a CTRACE external writer.
- Stop a CTRACE external writer.
- Change the transaction trace buffer size.
- Specify a level indicator.
- Specify whether or not latent transactions is traced.

Starting transaction trace

Transaction trace is started when a TRACE TT command is issued with filter information. Following is an example of defining a transaction trace filter set with a user ID of TESTERP1 and transaction name of TRAN1.

```
trace tt,user=testerp1,tran=tran1
ITZ002I 'BUFSIZ' IS SET TO 0001M
ITZ001I TRANSACTION TRACE IS NOW ACTIVE WITH FILTER SET 01
```

When multiple filter keywords are specified, as in the preceding example, a 'logical AND' is used to determine if the transaction should be traced or not traced.

Adding additional trace filter sets

Up to five transaction trace filter sets can be concurrently active. They are activated when the TRACE TT command is issued with filter information. The command in the following example defines an additional transaction trace filter set with a user ID of DONNA*. The use of an asterisk (*) in the last character position indicates a wildcard is being defined. When determining if a transaction trace token is to be created, any user ID with a prefix of DONNA will result in a match.

```
trace tt,user=donna*
ITZ001I TRANSACTION TRACE IS NOW ACTIVE WITH FILTER SET 02
```

If multiple filter sets are specified a 'logical OR' is used among the filter sets to determine if the transaction should be traced or not traced.

Removing an active trace filter set

A transaction trace filter set is removed when the OFF=x keyword is used. For example, the following command indicates that the transaction trace filter set 02 has been turned off.

```
trace tt,off=2
ITZ016I TRANSACTION TRACE FILTER SET TURNED OFF
```

Stopping transaction trace

Use the OFF=ALL keyword to stop transaction trace. For example:

```
trace tt,off=all
ITZ007I TRANSACTION TRACE IS NO LONGER ACTIVE.
A DUMP COMMAND MAY BE ISSUED TO DUMP THE TRANSACTION TRACE DATA SPACE.
```

Use the DUMP command to dump the transaction trace data space. For example:

```
DUMP COMM=(TTrace for TRAN=ATM1)
R x,DSPNAME='TRACE'.SYSTTRC
```
Starting a CTRACE external writer
Transaction trace supports the use of an external writer for processing transaction trace records. An external writer is specified on the initial command that activates transaction trace or is specified standalone while transaction trace is active. For example:

```
trace tt,wtr=abcdefg
```

Component trace messages are issued in response to this command.

Stopping a CTRACE external writer
Transaction trace external writer processing can be stopped with the use of the WTR=OFF keyword. For example:

```
trace tt,wtr=off
```

Component trace messages are issued in response to this command.

Changing the data space size
The transaction trace TTRACE TT command allows a change in the transaction trace data space size. The data space is from 16K to 999K or 1M to 32M. For example:

```
trace tt,bufsiz=2m
ITZ002I 'BUFSIZ' IS SET TO 0002M
```

Specifying a level indicator
The transaction trace TTRACE TT command allows definition of a level indicator for each filter set.
- 1 pertains to component entry, exit, exceptions, and major events.
- 2 pertains to detail, controlled by component external; the default is 2

```
trace tt,bufsiz=2m, user=testerp1, tran=tran1, lvl=01
ITZ002I 'BUFSIZ' IS SET TO 0002M
ITZ001I TRANSACTION TRACE IS NOW ACTIVE WITH FILTER SET 01
```

Tracing latent transactions
Use the transaction trace TTRACE TT command to specify whether or not latent transactions are traced. The default is to trace latent processing. Consider the following when deciding what to specify:
- The transaction is currently active in the system.
- The transaction is marked for tracing.
- The filter set used to mark the transaction eligible for tracing is no longer active.

```
trace tt,latent=no
ITZ002I 'LATENT' IS SET TO NO
```

DISPLAY TRACE,TT
Use the TT keyword on the DISPLAY TRACE command to determine the status of transaction trace. Do not use the component trace display command to inquire on the status of transaction trace. In addition to displaying information specified on the TRACE TT command, the DISPLAY TRACE,TT response also displays a list of the systems participating in transaction trace sysplex processing. Figure 158 on page 542 is an example of a DISPLAY TRACE,TT command response.
Using IPCS to view transaction trace output

Use the IPCS subcommand CTRACE COMP(SYSTTRC) to view transaction trace records. To obtain a sysplex TTrace stream, use the IPCS MERGE subcommand to format TTrace records from multiple input data sets. Any generalized trace facility (GTF) records imbedded in the TTrace records are processed without having to specify additional keywords to the above command. Figure 159 is an example of a short IPCS CTRACE COMP(SYSTTRC) SHORT command response.

```
ctrace comp(systtrc) short
COMPONENT TRACE SHORT FORMAT
COMP(SYSTTRC)
*** 09/23/1999
SYSTYPE NAME     MNEMONIC    ENTRY ID     TIME STAMP       DESCRIPTION
----- --------- ------------ -------------- ----------------
SY1    TTOMD    00000002  14:17:20.833847  TRACE TT Command
SY1    TTCMD    00000002  14:18:11.611755  TRACE TT Command
SY1    EVENT    00000003  14:31:55.813125  TRACE EVENT
SY1    EVENT    00000003  14:31:55.899216  TRACE EVENT
SY1    EVENTU   00000005  14:31:56.378480  TRACE EVENT with User Data
SY1    EVENTG   00000004  14:31:56.818367  TRACE EVENT with GTF Data
```

Figure 159. Example: IPCS CTRACE COMP(SYSTTRC) SHORT response
**CTRACE COMP(SYSTTRC) FULL**

**COMPONENT TRACE FULL FORMAT**

**COMP(SYSTTRC)**

**** 09/23/1999

<table>
<thead>
<tr>
<th>SY1</th>
<th>TT CMD</th>
<th>00000002</th>
<th>14:17:20.83847</th>
<th>TRACE TT Command CMDID...0501 COMMAND...TRACE TT,BUFSIZE=2M,USER=TESTERP1,TRAN=TRAN1, LVL=01</th>
</tr>
</thead>
<tbody>
<tr>
<td>SY1</td>
<td>TT CMD</td>
<td>00000002</td>
<td>14:18:11.611755</td>
<td>TRACE TT Command CMDID...0402 COMMAND...TRACE TT,USER=DONNA+</td>
</tr>
<tr>
<td>SY1</td>
<td>EVENT</td>
<td>00000003</td>
<td>14:31:55.813125</td>
<td>TRACE EVENT COMPONENT...COMP EVENTDESC..TTVAPIEA008 CMDID....0501 FUNCTION...TEST_ITZEVENT_WITH_FUNCTIONNAME. TCB...007ED9C8 ASID..0022 TRACETOKEN..SY1 B2E447A3 F32E4048 05010100 00000000</td>
</tr>
<tr>
<td>SY1</td>
<td>EVENT</td>
<td>00000003</td>
<td>14:31:55.899216</td>
<td>TRACE EVENT COMPONENT...COMP EVENTDESC..TTVAPIEA009 CMDID....0000 FUNCTION................................... TCB...007ED7A8 ASID..0022 TRACETOKEN.. 40404040 40404040 40404040 40404040 LATENT workunit traced.</td>
</tr>
<tr>
<td>SY1</td>
<td>EVENTU</td>
<td>00000005</td>
<td>14:31:56.378480</td>
<td>TRACE EVENT with User Data COMPONENT...COMP EVENTDESC..TTVAPIEA003 CMDID....0501 FUNCTION................................... TCB...007ED148 ASID..0022 TRACETOKEN..SY1 B2E447A3 F30056C1 0105FF00 00000000 +0000 E3C8C9E2 40C9E240 E3E340C4 C1E3C140 THIS IS TT DATA +0010 C6D60940 C1E3C140 C0D5E2C1 C3E3C9D6 FOR A TRANSACTION +0020 D5A0E309 C1C3C540 D9C5C09D D9C44BE3 N TRACE RECORD. +0030 C9C9E240 C9E240E3 E340C4C1 E3C140C6 HIS IS TT DATA F +0040 D6D940C1 40E3D9C1 DE2E1C1C3 E39D0605 OR A TRANSACTION +0050 40E3D9C1 40E3D9C1 DE2E1C1C3 E39D0605 TRACE RECORD.</td>
</tr>
<tr>
<td>SY1</td>
<td>EVENTG</td>
<td>00000004</td>
<td>14:31:56.818367</td>
<td>TRACE EVENT with GTF Data COMPONENT...COMP EVENTDESC..TTVAPIEA004 CMDID....0402 FUNCTION................................... TCB...007ED368 ASID..0022 TRACETOKEN..SY1 B2E447A3 F566F584 03030300 00000000 HEXFORMAT AID FF FID 00 E10 E000 +0000 E3C8C9E2 40C9E240 C7E3C640 C4C1E3C1 THIS IS GTF DATA +0010 40C6D09D 40C140E3 D9C1D5E2 C1E3C9D9 FOR A TRANSACTION +0020 D6D540E3 D9C1C3C5 40D9C5C3 D6D9C44B ON TRACE RECORD. +0030 E3C8C9E2 40C9E240 C7E3C640 C4C1E3C1 THIS IS GTF DATA +0040 40C6D09D 40C140E3 D9C1D5E2 C1E3C9D9 FOR A TRANSACTION +0050 D6D540E3 D9C1C3C5 40D9C5C3 D6D9C44B ON TRACE RECORD.</td>
</tr>
</tbody>
</table>

*Figure 160. Example: IPCS CTRACE COMP(SYSTTRC) LONG response*
Chapter 14. GETMAIN, FREEMAIN, STORAGE (GFS) trace

GFS trace is a diagnostic tool that collects information about the use of the GETMAIN, FREEMAIN, or STORAGE macro. You can use GFS trace to analyze the allocation of virtual storage and identify users of large amounts of virtual storage. You must use the generalized trace facility (GTF) to get the GFS trace data output.

The following topics describe GFS trace:
- "Starting and stopping GFS trace"
- "Receiving GFS trace data" on page 546
- "Formatted GFS trace output" on page 546
- "Unformatted GFS trace output" on page 548

Starting and stopping GFS trace

The following procedure explains how to request a GFS trace.

1. In the DIAGxx parmlib member, set the VSM TRACE GETFREE parameter to ON and define the GFS trace control data.

   **Example: DIAGxx parmlib member for starting GFS tracing:** The following DIAGxx parmlib member starts GFS trace and limits the trace output to requests to obtain or release virtual storage that is 24 bytes long and resides in address spaces 3, 5, 6, 7, 8 and 9:

   ```
   VSM TRACE GETFREE (ON)
   ASID (3, 5-9)
   LENGTH (24)
   DATA (ALL)
   ```

   You will need another DIAGxx parmlib member defined to stop GFS tracing. See 5 on page 546.

2. Ask the operator to enter the SET DIAG=xx command to activate GFS trace using the definitions in the DIAGxx parmlib member.

3. Start a GTF trace (ask the operator to enter a START `membername` command on the operator console). `membername` is the name of the member that contains the source JCL (either a cataloged procedure or a job). Tell the operator to specify a user event identifier X'F65' to trace GTF user trace records.

   **Example: Starting a GTF trace for GFS data:** In the following example, the operator starts GTF tracing with cataloged procedure GTFPROC to get GFS data in the GTF trace output. The contents of cataloged procedure GTFPROC are as follows:

   ```
   //GTF PROC MEMBER=GTFPROC
   /* Starts GTF
   //IEFPROC EXEC PGM=AHLGTF,REGION=32M,
   // PARM='MODE=EXT,DEBUG=NO,TIME=YES,BLOK=40K,SD=0K,SA=40K'
   //IEFRRDER DD DSN=D31POOL.PJREDGTF_TRACE,
   // DISP=SHR,UNIT=3380,VOL=SER=CTDS1
   ```

   The operator then replies to messages AHL100A with the USRP option. When message AHL101A prompts the operator for the keywords for option USRP, the operator replies with USR=(F65) to get the GFS user trace records in the GTF trace output.

   START GTFPROC
   00 AHL100A SPECIFY TRACE OPTIONS
GETMAIN, FREEMAIN, STORAGE trace

4. To stop the GTF trace, ask the operator to enter a STOP procname command on the operator console.

5. To stop GFS trace, create a DIAGxx parmlib member with VSM TRACE GETFREE(OFF) and have the operator enter a SET DIAG=xx command.

Example: DIAGxx parmlib member for stopping GFS tracing: The following DIAGxx parmlib member stops GFS trace:

VSM TRACE GETFREE (OFF)

For additional information, see the following references:

- See z/OS MVS Initialization and Tuning Reference for the syntax of the DIAGxx parmlib member.
- See z/OS MVS System Commands for the syntax of the SET and START commands.
- See Chapter 10, “The Generalized Trace Facility (GTF),” on page 229 for information about how to specify GTF EIDs.

Receiving GFS trace data

GTF places the GFS trace data in a user trace record with event identifier XF65. To obtain GFS trace data, do one of the following:

- When GTF writes trace data in a data set, format and print the trace data with the IPCS GTFTRACE subcommand.
- When GTF writes trace data only in the GTF address space, use a dump to see the data. Request the GTF trace data in the dump through the SDATA=TRT dump option.
- Issue the IPCS GTFTRACE subcommand to format and see the trace in an unformatted dump.

See z/OS MVS IPCS Commands for the GTFTRACE subcommand.

Formatted GFS trace output

Figure 161 on page 547 shows an example of formatted GFS trace output.
The GETMAIN / FREEMAIN / STORAGE trace produces a second type of record with a slightly different format. Following is an example of this record type:

USRDA F65 ASCB 00FA0800 JOBN GTFJM2
Getmain SVC(120) Cond=Yes
Loc=(Below,Below) Bndry=Dbld
Return address=849CA064 Asid=001A Jobname=GTFJM2
Subpool=229 Key=0 Asid=001A Jobname=GTFJM2 TCB=008DCA70 Retcode=0
Storage address=008D6768 Length=10392 X'2898'
GPR Values
0-3 00002898 00000000 7FFFC918 0B601E88
4-7 01FE3240 008FF830 849CA000 00FA0800
8-11 00000000 00000DE8 049CBFFE 849CA000
12-15 049CAFFF 0B601A9C 00FE9500 000E510


USRDA F65 ASCB 00FA0800 JOBN GTFJM2
Freemain SVC(120) Cond=No
Return address=882D608A Asid=001A Jobname=GTFJM2
Subpool=230 Key=0 Asid=001A Jobname=GTFJM2 TCB=008DCA70 Retcode=0
Storage address=7F73DFF8 Length=8 X'8'
GPR Values
0-3 00000000 7F73DFF8 008D82D8 008D7BC0
4-7 008D958 008D6808 008D85C8 08335000
8-11 00000002 00000000 7F73DFF8 008D862C
12-15 882D6044 008D8C98 849D242A 0000E603


Figure 161. Example of formatted GFS trace output
THEME, FREEMAIN, STORAGE trace

This type of record is unique because it does not trace a return address. It writes whenever an individual area of storage is FREEMAINed as part of a subpool FREEMAIN request. There may be many of these records in a row. The last record of the sequence is followed by a record that indicates a subpool FREEMAIN request. This record includes the return address of the issuer of the subpool FREEMAIN.

Unformatted GFS trace output

"Layout of the GFS trace output" shows unformatted GFS trace output as it would appear in the trace data set where GTF puts the output. You can use this information to write your own formatting or analysis routines.

Layout of the GFS trace output

Unformatted GFS Trace Output

**Part 1 - This part is in every GFS trace entry.**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Flags</td>
</tr>
<tr>
<td>X'80'</td>
<td>-</td>
<td>Common storage</td>
</tr>
<tr>
<td>X'40'</td>
<td>-</td>
<td>Caller's registers are traced</td>
</tr>
<tr>
<td>X'20'</td>
<td>-</td>
<td>This is a subpool release range entry</td>
</tr>
<tr>
<td>X'10'</td>
<td>-</td>
<td>Copy of VSWKOWNINFO</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Actual subpool after translation</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>ASID which owns the storage</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Address of storage area</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Actual length of storage area</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>Address of TCB</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>Copy of VSWKSKEY</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>Copy of VSWKRC</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>Modification level number</td>
</tr>
<tr>
<td>X'01'</td>
<td>-</td>
<td>HBB6606</td>
</tr>
<tr>
<td>X'02'</td>
<td>-</td>
<td>HBB7703</td>
</tr>
<tr>
<td>X'03'</td>
<td>-</td>
<td>HBB7730</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>Offset of Part 2</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>Offset of Part 3</td>
</tr>
</tbody>
</table>

**Part 2 - This part is in every GRS trace entry except for subpool release range entries.**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Caller's return address</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Minimum length for a variable request</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Maximum length for a variable request</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>Name of job which owns the storage</td>
</tr>
</tbody>
</table>
| 14     | 8      | Name of job which contained the program which
GETMAIN, FREEMAIN, STORAGE trace

requested the storage

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1C</td>
<td>2</td>
<td>ASID which contained the program which requested the storage</td>
</tr>
<tr>
<td>1E</td>
<td>1</td>
<td>Copy of VSNKEPSL</td>
</tr>
<tr>
<td>1F</td>
<td>1</td>
<td>Copy of VSNK SVC</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>Copy of VSNKRFLG</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>Copy of VSNKPFLG</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>Copy of VSNKFGLG</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>Copy of VSNKRFLG2</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>Copy of VswkRtAddrHigh</td>
</tr>
<tr>
<td>28</td>
<td>4</td>
<td>Copy of VswkAR15Value</td>
</tr>
<tr>
<td>2C</td>
<td>4</td>
<td>Copy of VswkAR1Value</td>
</tr>
</tbody>
</table>

Part 3 - This part is in the GFS trace record if the caller's registers are traced.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'40'</td>
<td>Caller's registers 0-15</td>
</tr>
</tbody>
</table>

Note: The IGVVSMWK macro contains field names beginning with VSWK. For more information, see z/OS MVS Data Areas in z/OS Internet library [http://www.ibm.com/systems/z/os/zos/bkserv/].
GETMAIN, FREEMAIN, STORAGE trace
Chapter 15. Recording logrec error records

When an error occurs, the system records information about the error in the logrec data set or the logrec log stream. The information provides you with a history of all hardware failures, selected software errors, and selected system conditions. Use the Environmental Record, Editing, and Printing program (EREP):

- To print reports about the system records
- To determine the history of the system
- To learn about a particular error

Collection of software and hardware information

Use the records in the logrec data set or the logrec log stream as additional information when a dump is produced. The information in the records will point you in the right direction while supplying you with symptom data about the failure. Figure 162 shows the error processing for a logrec data set, named SYS1.LOGREC, which is the default name for the logrec data set.

You can set your system up to record errors on either a logrec data set or in a logrec log stream. This topic tells you what you need to know about each medium before deciding how to collect error records. To use the logrec data set or a logrec log stream, you need to know how to initialize each of them, how to record system events on each of them, how to collect the data when it is available, and how to interpret the output through EREP.

This topic describes each of these tasks:
- "Choosing the correct logrec recording medium" on page 552
- "Initializing and reinitializing the logrec data set" on page 552
- "Defining a logrec log stream" on page 554
- "Changing the logrec recording medium" on page 556
Choosing the correct logrec recording medium

You can choose where the system will record logrec error records. When a system is not in a sysplex, an installation can use a logrec data set, associated with an individual system, to record error records. An installation can choose to continue this type of recording by initializing the logrec data set before IPLing the system that will use it.

In a sysplex, however, because each system requires its own logrec data set, you might need to look at each logrec data set when an error occurs.

To eliminate the problem of having to manage up to 32 logrec data sets, an installation can choose to define one coupling facility logrec log stream. Using a coupling facility logrec log stream eliminates the following:

- Running IFCDIP00 to initialize multiple logrec data sets
- Handling full or emergency data set conditions
- Scheduling the daily offload of logrec data sets
- Concatenating multiple history data sets
- Archiving logrec records

For more information, see the following references:

- See "Initializing and reinitializing the logrec data set" if you want to initialize a logrec data set for your system.
- See "Defining a logrec log stream" on page 554 if you want to define a logrec log stream for your installation.
- See "Changing the logrec recording medium" on page 556 for more information about changing the logrec recording medium after you start the system.

Initializing and reinitializing the logrec data set

You must initialize the logrec data set before starting the system that will use it. You reinitialize the logrec data set when an uncorrectable error occurs. You clear the logrec data set when it is full or near full. You must also initialize a logrec data set before using the SETLOGRC command to switch to it.

To initialize or reinitialize the logrec data set, use the service aid program IFCDIP00. To clear a full logrec data set, use EREP IFCDIP00 creates a header record and a time stamp record for the logrec data set.

**Attention:** The logrec data set is an unmovable data set. If you attempt to move it using a program, such as a defragmentation program, your system will experience difficulty both reading from and writing to the data set. Logrec records physical addresses into the logrec data set rather than to relative address, and this prevents the ability to move the data set. You can use the SETLOGRC command to switch to IGNORE, LOGSTREAM, or a different data set to free the data set and allow use of maintenance programs for the volume that contains the logrec data set. However, the data set itself remains unmovable.
Recording logrec error records

Initializing the logrec data set

If the logrec data set does not exist, you must first allocate it and then initialize it.

Note: Whenever you allocate or reallocate the logrec data set, the newly allocated data set is not used until you either initialize it and restart the system on which it is to be used or use the SETLOGRC command to change to it.

Figure 163 is an example of a job that scratches and uncatalogs an existing logrec data set and allocates, catalogs, and initializes a new one. (If you do not currently have a logrec data set, start with the second step of the job.) Use the JCL statements below to do the following:

- Rename the logrec data set. For example, rename SYS1.LOGREC to SYS1.LOGREC.OLD.
- Allocate and initialize a new logrec data set with new space specifications using IFCDIP00.

```plaintext
//KATHYLJ JOB (9999), 'CREATE NEW LOGREC DS', CLASS=A, MSGCLASS=X, // MSGLEVEL=(1,1), NOTIFY=KATHY
//------------------------------------------------------------------
// RENAME THE CURRENT LOGREC DATASET
// UNCATLG SYS1.LOGREC SO THE NEW LOGREC CAN BE ALLOCATED ON // ANOTHER VOLUME, IF DESIRED
//------------------------------------------------------------------
/RENAME EXEC PGM=IEHPROGM /M43RES DD VOL=SER=M43RES,UNIT=3390,DISP=SHR /SYSPRINT DD SYSOUT=* /SYSIN DD *
RENAME DSNAME=SYS1.LOGREC, VOL=3390, M43RES, X NEWNAME=SYS1.LOGREC.OLD UNCATLG DSNAME=SYS1.LOGREC
/*
//------------------------------------------------------------------
// CREATE THE NEW LOGREC DATASET AND INITIALIZE IT
//------------------------------------------------------------------
//IFCDIP00 EXEC PGM=IFCDIP00,COND=(0,LT) //SERERDS DD DSN=SYS1.LOGREC,DISP=(,CATLG), // VOLUME=SER=M43RES,,UNIT=SYS0A, SPACE=(CYL,3,,CONTIG)
/*
//Figure 163. Example: Changing the space allocation

Note: If you run the JCL in Figure 163 and an error occurs after the logrec data set has been scratched but before it has been reallocated, you will be unable to use this logrec data set dynamically or at the next system restart. To solve this problem, do one of the following:

- Use the DFSMSdss stand-alone restore program to restore your old logrec data set.
- Run the reallocate job on the data set while running under another system.

See z/OS DFSMSdss Storage Administration for information about the DFSMSdss stand-alone restore program.

Reinitializing the logrec data set

You need to reinitialize the logrec data set either when the data set is full or when an uncorrectable error occurs.

If the data set is full, use EREP to record the data in a history data set and reinitialize logrec.
Recording logrec error records

In the case of an error, invoke IFCDIP00 with JCL statements to reinitialize your existing logrec data set. IFCDIP00 resets the logrec data set header record field to indicate that the entire data set can be used and clears the time stamp record to hexadecimal zeros.

For information on using EREP, see the [EREP User’s Guide](#). Figure 164 is an example of using the IFCDIP00 service aid to reinitialize the logrec data set. It uses the following JCL statements:

- The JOB statement initiates the job; the job name INSERLOG has no significance.
- The EXEC statement specifies the program name (PGM=IFCDIP00).
- The SERERDS DD statement specifies the reinitialized logrec data set (in this case SYS1.LOGREC), which must be on a permanently mounted volume (VOL=SER=111111 in this example); the DDNAME must be SERERDS.

```
//INSERLOG JOB
//STEP1 EXEC PGM=IFCDIP00
//SERERDS DD DSNAME=SYS1.LOGREC,UNIT=3380,
// VOL=SER=111111,DISP=(OLD,KEEP)
```

Figure 164. Example: Reinitializing the logrec data set

Defining a logrec log stream

To use a logrec log stream, you must first prepare your installation to use system logger functions. IBM recommends that you use a coupling facility log stream for LOGREC so that you can merge data from multiple systems in a sysplex.

To obtain logrec records for a single system sysplex, you can also use a DASD-only log stream, which is single system in scope. Note that this is not recommended for a multi-system sysplex, because you can only have one system connect to a DASD-only log stream per sysplex. This means that if you make your logrec log stream DASD-only, only one system will be able to write and read from it. See the system logger chapter of [z/OS MVS Setting Up a Sysplex](#) for information on DASD-only log streams.

If you use the SETLOGRC command and specify a log stream name, you can additionally set up other DASD-only and coupling facility log streams to record logrec data.

See [z/OS MVS Setting Up a Sysplex](#) for more information about system logger setup.

The following steps describe how to use a coupling facility logrec log stream in place of a logrec data set:

1. Define a log stream using the system logger log stream definition utility, IXCMIAPU.

   IFBLSJCL (see Figure 165 on page 555) is available in SYS1.SAMPLIB as an example of using the administrative data utility, IXCMIAPU, to define the coupling facility logrec log stream to a sysplex.

   There are no restrictions on naming the log stream, however, when naming your log stream, consider making 'LOGREC' a part of the log stream name so as to differentiate it from other log streams.
Recording logrec error records

Note: MAXBUFSIZE must be at least 4068 because logrec writes records in one page blocks. Specify SMS storage group, storage, data and management classes such that when one data set is full, another is allocated. Allocate as much space as is allocated for all the logrec data sets on the systems in the sysplex before migrating to a logrec log stream.

The most effective way to manage all logrec records is to specify the automatic migration of log data sets to HSM. This automatic migration eliminates the need to create and maintain archival history data sets, with one exception. If the log stream data set directory is full, you can, using SUBSYS-options2 of the LOGR subsystem, copy data from a log stream to a history data set and then delete the copied data from the log stream.

2. Either specify LOGREC=LOGSTREAM in the IEASYSxx parmlib member, or use the SETLOGRC command to change the logrec recording medium to a logrec log stream. In general, any records written to any logrec data sets before changing to a logrec log stream must be read by a separate EREP job. However, the MERGE option can be used to combine logrec output from the log stream with a logrec data set in a single EREP job.

3. Change the EREP job stream as follows:
   - Change the SERLOG DD DSN= statement pointing to a logrec data set to an ACCIN DD DSN= statement pointing to a logrec log stream name, with corresponding SUBSYS parameters, to associate EREP with the logrec log stream. The SUBSYS parameters are described in “Obtaining records from the logrec log stream” on page 562.
Identify the input as a history data set. Leave the output to a history data set as currently recommended, because all subsequent steps should already use the history data set as input.

**Note:** Using a logrec log stream as input for multiple steps is not recommended because each subsequent step processes more records than the prior, causing numbers and data in successive reports not to match.

- Subsequent EREP report steps that normally process history data sets no longer need to concatenate one history data set per system.

For more information, see the following documentation:
- See [z/OS MVS Setting Up a Sysplex](#) for information about preparing an installation to use system logger functions.
- See [EREP User’s Guide](#) for more information about running an EREP job to obtain a history data set.
- See [z/OS MVS System Commands](#) for more information about the SETLOGRC command.
- See [z/OS MVS Initialization and Tuning Reference](#) for more information about the IEASYSxx parmlib member.

### Changing the logrec recording medium

You can use the SETLOGRC command to dynamically change your logrec recording medium regardless of the setting with which you started the system. Additionally, you can change to different logrec data sets or log streams. For more information, see [z/OS MVS System Commands](#).

### Error recording contents

The system creates records for every hardware or software failure and system condition that occurs and stores these records in the logrec data set or the logrec log stream. The records can contain two types of data that document failures and system conditions:

- **Error statistics**, which include the number of times that channels, machine models, and I/O devices have failed
- **Environmental data**, which include time and circumstances for each failure or system condition

**Note:** A programmer can also build symptom records using the SYMRBLD macro and have those records written into the logrec data set or the logrec log stream using the SYMREC macro.

See [z/OS MVS Programming: Assembler Services Reference IAR-XCT](#) for information about the macros.

Each record is recorded in hexadecimal format as an undefined length record. Each record provides:

- Relevant system information at the time of the failure
- Device hardware status at the time of the failure
- Results of any device/control unit recovery attempt
- Results of any software system recovery attempt
- Statistical data
When taken as a whole, these records create a history of the system, which begins early in system initialization and ends when the system stops. These records contain:

- **Full Abend History**: The system writes a logrec record for every abend, regardless of whether the dump is requested or suppressed. The logrec data set or the logrec log stream contains a full record of abnormal ends.
- **System Initialization Errors**: The system writes errors during system initialization, before other diagnostic services are completely functioning.
- **Lost Record Counts**: The system writes a logrec record to summarize lost error records. Sometimes hardware-detected or software-detected errors occur close together. When errors are too close together, the system cannot write an individual record for each error; instead, the system counts the errors and writes a summary record.

These sections describe what is in the logrec data set:

- “Logrec data set header record”
- “Logrec data set time stamp record”

This section describes what is in the logrec data set or the logrec log stream:

- “Types of logrec error records” on page 558

See [z/OS MVS Diagnosis: Reference](#) for the format of the header record, time stamp record, and logrec error records.

### Logrec data set header record

IFCDIP00 creates a header record on the logrec data set. The logrec data set header record includes:

- Information that the system recording routines can use to determine where to write new record entries onto the logrec data set
- Information that EREP can use to find existing record entries on the logrec data set. This information is valuable when you run an EREP report to find a particular error.
- Information that the system recording routines can use to issue a warning message when the logrec data set is 90% full.

**Note**: The logrec log stream does not have a header record generated.

### Logrec data set time stamp record

IFCDIP00 creates a time stamp record on the logrec data set in the first record space following the header record. The time stamp record provides current date and time information for the IPL record. This allows you to measure the approximate time interval, recorded in the IPL records, between the ending and reinitialization of the operating system.

At preset time intervals, the system obtains the current date and time and writes this information on the time stamp record, overlaying the previous date and time.

During a subsequent initialization of the system, the system obtains the date and time from the time stamp record and adds it to the IPL record.

If IFCDIP00 is used to reinitialize the logrec data set, the information in the time stamp record is overlaid with hexadecimal zeros until the system writes the current date and time.
Recording logrec error records

Note: The logrec log stream does not have a time stamp record generated.

Types of logrec error records

When the logrec data set or the logrec log stream is initialized, the system begins recording events. The system records the following types of error records, containing device-dependent or incident-dependent information:

- **Asynchronous notification records (ANR):**
  - External timer reference (ETR) records for information related to Sysplex Timer incidents.
  - Direct access storage device (DASD)-service information message (SIM) records for information concerning servicing needs.
  - Link maintenance information (LMI) records for information for a particular link incident.

- **Channel report word (CRW) records** for:
  - Channel path error
  - Subchannel error
  - Configuration alert error
  - Monitoring facility error

- **Dynamic device reconfiguration (DDR) records** for:
  - Operator and system swaps between direct access and magnetic tape devices
  - Operator swaps on unit record devices

- **End-of-day (EOD) records** for information related to end-of-day and system ending conditions whenever the RDE option has been included in the system.

- **Input/output supervisor (IOS) records** for information related to IOS recovery actions.
  - Dynamic pathing services validation (DPSV) records for recovery actions.

- **Initial program load (IPL) records** for information related to system initializations whenever the RDE option has been included in the system.

- **Machine check handler (MCH) records** for:
  - Central processor failure
  - Storage failure
  - Storage key failure
  - Timer failure

- **Miscellaneous data (MDR) records** for:
  - Buffer overflow and device failures on buffered log devices
  - Demounts on DASD with buffered logs
  - Demounts by the DFSMSdss program between DASD having buffered logs and removable disk packs
  - Device failures on teleprocessing devices connected to an IBM communication controller
  - Statistical recording by EREP on DASD with buffered logs

- **Missing interruption handler (MIH) records** for:
  - Missing I/O interruptions
  - Specified time intervals
  - Recovery actions required
  - Recovery actions performed

- **Outboard (OBR) records** for:
Recording logrec error records

- Counter overflow statistics and device failures on devices supported by the teleprocessing access methods
- End-of-day (EOD) requests
- Paging I/O errors
- Permanent channel and I/O device failures
- Statistic counter overflow
- Temporary or intermittent I/O device failures
- Demounts on IBM magnetic tape drives
- Devices that have their own diagnostic buffers
- Statistical recording by EREP on DASD with buffered logs

- Subchannel logout handler (SLH) records for channel errors.
- Software records including:
  - Machine checks (hardware-detected hardware errors, such as software recovery attempts for hard machine failures)
  - Program checks (hardware-detected software errors)
  - Restart errors (operator-detected errors)
  - Lost record errors (count of the records that did not fit in the buffer to be written to the logrec data set)
  - Software-detected errors, such as:
    - Abnormal ends, which are also called abends; reported in software records or erroneous supervisor call (SVC) instructions. These are known as SDWA-type software records.
    - Errors that are not abnormal ends; reported in symptom records.
    - Errors generated by application programs or system components; reported in symptom records.

As you can see, the system records a comprehensive list of error records that can help you when you need to diagnose a system failure.

Obtaining information from the logrec data set

You can obtain the information recorded in the logrec data set using EREP, which formats error records. EREP can perform the following functions:

- Create an accumulation data set from the logrec data set
- Clear the logrec data set
- Copy an input accumulation data set to an output accumulation data set
- Merge data from an accumulation data set and the logrec data set
- Print a detailed description of selected hardware and software error records
- Summarize and print statistics for device failures

EREP places the information from the logrec data set into reports. Using JCL, you determine the type of report you want EREP to produce.

Using EREP

EREP presents information from the logrec software error records in five reports.

Detail Edit Report for an Abend

The system obtains most of the information for an abend logrec error record from the system diagnostic work area (SDWA). The report contents are:
Recording logrec error records

- Record header: report type (SOFTWARE RECORD), system, job name, error identifier (ERRORID), date, and time
- Search argument abstract
- Serviceability information
- Time of error information
- Status information from the request block
- Recovery environment
- Recovery routine action
- Hexadecimal dump of the SDWA, including the variable recording area (VRA)

Figure 166 shows how to generate detail edits and summaries of all software and operational records:

```
//STEP7 EXEC PGM=IFCEREP1,PARM='CARD'
//ACCIN DD DSN=EHISTORY,DISP=SHR
//DIRECTWK DD UNIT=SYSDA,
// SPACE=(CYL,5,,CONTIG)
//EREPPPT DD SYSOUT=A,DCB=BLKSIZE=133
//TOURIST DD SYSOUT=A,DCB=BLKSIZE=133
//SYSIN DD *
PRINT=PS
TYPE=SIE
HIST
ACC=N
ENDPARM
```

Figure 166. Example: Printing a detail edit report

Detail edit report for a symptom record

The system obtains most of the information for a non-abend logrec error record from the symptom record identified in the SYMREC macro. A programmer can build the symptom record using the SYMRBLD macro. The report contents are:

- Record reader: report type (SYMPTOM RECORD), system, date, and time
- Search argument abstract
- System environment
- Component information
- Primary and secondary symptom strings
- Free-format component information
- Hexadecimal dump of the symptom record

System summary report

The report summarizes errors for each of your installation’s principle parts, or subsystems: processors, channels, subchannels, storage, operating system control programs, and I/O subsystems. The report contents are:

- Record header: report type (SYSTEM SUMMARY), system, date, time
- Total errors and errors for each processor for the following types of errors:
  - IPL
  - Machine check
  - Program error
  - End of day
- Identifications for processors in the report
**Recording logrec error records**

**Event history report**

The report shows the error history: the frequency, order, and pattern of errors. The report contents are:

- **Record header:** report type (EVENT HISTORY)
- **Abstracts for abend and non-abend logrec error records in chronological order**
- **Totals of the types of logrec error records for the system and for each processor**

The JCL shown in Figure 167 defines a two-step job. The first step prints an event history report for all logrec data set records. The second step formats each software, IPL, and EOD record individually. The event history report is printed as a result of the EVENT=Y parameter on the EXEC statement of the first step. It can be a very useful tool to the problem solver because it prints the records in the same sequence they were recorded and therefore shows an interaction between hardware error records and software error records.

```plaintext
//EREQ JOB MSGLEVEL=1
//EREQA EXEC PGM=IFCEREQP1,PARM='EVENT=Y,ACC=N',
// REGION=256K
//SERLOG DD DSN=SYS1.LOGREC,DISP=SHR
//TOURIST DD SYSOUT=A
//EREQPT DD SYSOUT=A,DCB=BLKSIZE=133
//SYSIN DD DUMMY
//EREQPP EXEC PGM=IFCEREQP1,PARM='TYPE=SIE,PRINT=P5,ACC=N',
// REGION=256K
//SERLOG DD DSN=SYS1.LOGREC,DISP=SHR
//TOURIST DD SYSOUT=A
//EREQPT DD SYSOUT=A,DCB=BLKSIZE=133
//SYSIN DD DUMMY
/*
```

*Figure 167. Example: Printing an event history report*

**Detail summary report**

The report summarizes information about data in logrec error records. The report contents are:

- **Record header:** report type being summarized
- **Summary information and counts**

The example in Figure 168 on page 562 shows how to generate detail summaries of all I/O errors.
Obtaining records from the logrec log stream

You can access records in the logrec log stream by either:

- Writing a program using IXGCONN and IXGBRWSE services, see “Using System Logger services to obtain records from the logrec log stream.”
- Using EREP, see “Using EREP to obtain records from the logrec log stream.”

Using System Logger services to obtain records from the logrec log stream

You can obtain records from the logrec log stream by writing a program that uses the IXGCONN and IXGBRWSE system logger services to return log data. The data returned by the IXGBRWSE service for the logrec log stream is mapped by the IFBLOGLB data area. (See z/OS MVS Programming: Assembler Services Guide for information on using system logger services.)

Note that the logrec log stream output from the IXGBRWSE service contains an individual log stream record. However, the log stream record actually contains a group of records. The logrec log stream record is mapped by the IFBLOGLB mapping macro. For information on the IFBLOGLB mapping macro, see z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

Using EREP to obtain records from the logrec log stream

You can use EREP to access the records in the logrec log stream for each system. The log stream subsystem allows existing programs to access error records from a log stream in the same way records were accessed from a logrec data set. See z/OS MVS Programming: Assembler Services Guide for information about using and starting the log stream subsystem.

JCL for the LOGR Subsystem

Use the SUBSYS parameter to call the log stream subsystem (LOGR) to access log stream data.
//ddname DD DSN=stream.name,
// SUBSYS=(LOGR[,exit_routine_name][,'SUBSYS-options1'][,'SUBSYS-options2'])

where: SUBSYS-options1:
[FROM={({yyyy/ddd}[hh:mm[ss]]) | OLDEST})
[TO={({yyyy/ddd}[hh:mm[ss]]) | YOUNGEST})
[,DURATION=(nnnn,HOURS)]
[,VIEW={ACTIVE|ALL|INACTIVE}]
[,GMT|LOCAL] SUBSYS-options2:
defined by the log stream owner

Note: Quotation marks around keywords are required when parentheses, commas, equal signs, or blank characters are used within the SUBSYS keyword.

Other DD keywords are validated, if specified, but are ignored in the LOGR subsystem processing.

DSNAME=stream.name
Specifies the name of the log stream to read. The name can be 1 to 26 characters in a data-set-name format.

SUBSYS=(LOGR[,exit_routine_name][,'SUBSYS-options1'][,'SUBSYS-options2'])
Specifies that processing of this DD is to be handled by the LOGR subsystem. The exit_routine_name is the second positional parameter and specifies the name of the exit routine to receive control from the LOGR subsystem.

- Specify or use the default value to IXGSEXIT to use the log stream subsystem exit routine.
- Specify IFBSEXIT to access records from the logrec log stream. See SUBSYS-options2 for logrec-specific parameters.
- Specify IFASEXIT to access records from SMF log streams. See SUBSYS-options2 for SMF-specific parameters.

SUBSYS-options1
Specifies options that are meaningful to all exit routines. See the documentation for a specific log stream exit for exceptions to these common options. The keywords are:

FROM=starting_time
Indicates the starting time of the first log stream block to be processed based on the log stream view that the VIEW keyword specifies. The first block is the one with a time stamp later than or equal to the specified time.

OLDEST
Indicates the first block read is the oldest block on the log stream. OLDEST is the default.

yyyy/ddd
Specifies the start date. If the date is omitted, the current date is assumed. yyyy is a 4-digit year number and ddd is a 3-digit day number from 001 through 366 (366 is valid only on leap years). For example, code February 20, 2000 as 2000/051, and code December 31, 1996 as 1996/366.

hh:mm:ss
Specifies the start time. If the time is omitted, the first block written after midnight is used. hh is a 2-digit hour number from 00 to 23, mm is a two digit minute number from 00 to 59, and ss is a 2-digit second number from 00 to 59. The seconds field and associated : delimiter can be omitted if it is not required by the log stream owner.

The FROM keyword is mutually exclusive with the DURATION keyword.
Recording logrec error records

TO=ending_time
Indicates the ending time of the last log stream block to be processed based on the log stream view that the VIEW keyword specifies. The last block is the one with a time stamp earlier than or equal to the specified time.

YOUNGEST
Indicates the last block read will be the youngest block on the log stream at the time the allocation for the DD occurs. YOUNGEST is the default.

yyyy/ddd
Specifies the end date. If the date is omitted, the current date is assumed. yyyy is a 4-digit year number and ddd is a 3-digit day number from 001 through 366 (366 is valid only on leap years). For example, code March 7, 2001 as 2001/066, and code November 12, 2000 as 2000/317.

hh:mm[:ss]
Specifies the end time. If the time is omitted, the last block written before midnight will be used. If the end date is the same as the current day, then the youngest block on the log stream at the time the allocation for the DD occurs will be used. hh is a 2-digit hour number from 00 to 23, mm is a two digit minute number from 00 to 59, and ss is a 2-digit second number from 00 to 59. The seconds field and associated: delimiter can be omitted if it is not required by the log stream owner.

The TO keyword is mutually exclusive with the DURATION keyword.

Note: If the value specified for the FROM keyword is greater than the value specified for the TO keyword, the system ends the jobstep with a JCL error.

DURATION= (nnnn,HOURS)
Specifies which blocks are to be processed. Each n is a numeric from 0 to 9. Specifying (nnnn,HOURS) requests the blocks for the last nnnn hours up to the youngest block that is to be processed based on the log stream view that the VIEW keyword specifies. The last nnnn hours are calculated from the current time of the allocation for the DD.

The first block is the one with a time stamp greater than or equal to the calculated start time. The last block read is the youngest block on the log stream at the time the allocation for the DD occurs.

The DURATION keyword is mutually exclusive with the TO and the FROM keywords.

VIEW=ACTIVE|ALL|INACTIVE
Specifies the view or portion of log data to be used to obtain records from the log stream. System logger maintains two kinds of log stream data in a log stream: an active portion and an inactive portion. The active portion of the log stream is the log data that the log stream owner has not logically deleted through an IXGDELETE request. The inactive portion of the log stream is the log data that the log stream owner has logically deleted but that has not yet been physically deleted from the log stream because the retention period (RETPD) specified for the log stream has not yet expired.
The VIEW option designates the portion(s) of the log stream to be used to obtain log data from the log stream, in addition to applying the other parameters.

Because the other parameters also apply, the combination of the FROM, TO, or DURATION parameters and the VIEW parameter might mean that the log stream subsystem exit returns no log data or only a portion of the intended log data. For example, if FROM=starting_time and VIEW=INACTIVE are both specified, and the starting_time is later (younger) than the log data in the inactive portion of the log stream, then there is no log data to meet the access criteria. In the same way, if TO=ending_time and VIEW=ACTIVE are both specified, and the ending_time is earlier (older) than the log data in the active portion of the log stream, then there is no log data to meet the access criteria.

**ACTIVE**

The view of the log stream is to include only active log data, in addition to applying the other log stream access parameters. ACTIVE is the default.

**ALL**

The view of the log stream is to include both active and inactive log data, in addition to applying the other log stream access parameters.

**INACTIVE**

The view of the log stream is to include only the inactive log data, in addition to applying the other log stream access parameters.

**GMT|LOCAL**

Specifies whether the time is local time (based on the time zone offset at the time the log was written) or GMT time. GMT is the default.

### Additional parameters for system logger

Along with the general parameters that can be specified for a log stream subsystem data set, system logger provides additional parameters in the SUBSYS-options2 specifications. The following values can be coded for a logrec log stream:

**SUBSYS-options2**

Specifies unique exit routine options. Refer to information provided by the specific log stream owner concerning these parameters.

**LASTRUN**

Indicates that the starting point of the records to be read from the logrec log stream will be from the last record read by a previous use of an application that used LASTRUN. The end point of the records will be to the youngest block in the logrec log stream.

LASTRUN is mutually exclusive with the FROM, TO and DURATION keywords in SUBSYS-options1 and with DELETE from SUBSYS-options2.

**DELETE**

Indicates that log stream records are to be deleted from the logrec log stream. The log stream itself is not deleted and remains available for use.

If the logrec log stream has been opened in the job step, all records up to but not including the last complete block read by the program will be deleted from the logrec log stream.
Recording logrec error records

If the logrec log stream has not been opened in the job step, all records prior to the time indicated on the TO keyword will not be deleted from the logrec log stream.

DELETE is mutually exclusive with the FROM and DURATION keywords in SUBSYS-options1 and the LASTRUN and SYSTEM keywords from SUBSYS-options2.

DEVICESTATS
Requests that the device statistics kept on the system where this job is running are to be recorded in the logrec log stream before any records are read.

SYSTEM=system name
Indicates that only records originating from the specified system name are to be returned to the application reading the logrec log stream. The system name value should match the name specified in the SYSNAME parameter of the IEASYSys parmlib member. SYSTEM= is mutually exclusive with the DELETE keyword from SUBSYS-options2.

Time of day considerations for logrec

When using the SUBSYS DD statement for LOGR, handle the time of day filtering carefully. The SUBSYS parameter does not accept a stop time of 24:00, but the EREP parameters do accept 24:00 as a stop time. If it is necessary to write JCL and EREP control statements, you might have to request filtering through both the SUBSYS DD statement and the EREP parameters:

- SUBSYS parameters use blocks of records, and filtering of these blocks is done using time stamps assigned after each logical record enclosed in a block has been assigned its own time stamp.

Figure 169 shows how to select logrec log stream records that were produced between 05:00 on June 1st, 1997, and the end of that day.

<table>
<thead>
<tr>
<th>Correct coding example</th>
<th>Incorrect coding example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE=(97152-97152),TIME=(0500-2400)</td>
<td>DATE=(97152-97153),TIME=(0500-0000)</td>
</tr>
</tbody>
</table>

Figure 169. Example: Using SUBSYS parameters

- EREP parameters use logrec logical records. When you use the TIME parameter with EREP, you are specifying a range of hours and minutes of interest on each day selected.

Table 69 shows how to correctly select logrec records that were produced between 05:00 on June 1st, 1997, and the end of that day.

Table 69. Example: Using EREP parameters

Creating a history data set for log data

Use the JCL in Example: Creating a history data set on page 567 to create a history data set from log data recorded on the logrec log stream. In this example, DEVICESTATS requests device statistics and the records are to be recorded in the log stream. Records are read from the last block that was processed on the
Recording logrec error records

previous submission of a "LASTRUN" EREP job up to the youngest block in the log stream. The first time a job with the "LASTRUN" option is run, the records are read from the oldest block in the log stream.

Example: Creating a history data set

```
//EREPDALY EXEC PGM=IFCEREP1,PARM=('HIST,ACC=Y,SYSUM')
//ACCIN DD DSN=SYSPLEX.LOGREC.ALLRECS,
  SUBSYS=(LOGR,IFBSEXIT,,DEVICESTATS,LASTRUN'),
  DCB=(RECFM=VB,BLKSIZE=4000)
//ACCDEV DD DSN=EREPM.EVENT,DISP=(NEW,CATLG),
  DCB=(RECFM=VB,BLKSIZE=4000),
  UNIT=SYSDA,SPACE=(CYL,(25,5))
//SERLOG DD DUMMY
//DIRECTWK DD UNIT=SYSDA,SPACE=(CYL,15,,CONTIG)
//TOURIST DD SYSOUT=A,DCB=BLKSIZE=133
//EREPT DD SYSOUT=A,DCB=BLKSIZE=133
//SYSABEND DD SYSOUT=A
//SYSIN DD DUMMY
/*
```

Producing an event history from logrec

Use the JCL shown in "Example: Producing an event history" to produce an event history report from records on the logrec log stream. By not specifying the FROM or TO keywords, the default is FROM=OLDEST and TO=YOUNGEST, indicating processing should include records from the beginning of the log stream to the end of the log stream. By specifying a print data set, EREPPT, the report can be browsed online for an overview of significant activity. When reading records by date and time, you can provide both EREP and SUBSYS parameters. EREP selects records from those passed to it from the SUBSYS parameters.

Example: Producing an event history

```
//EREPNOW EXEC PGM=IFCEREP1,REGION=4M,
//                  PARM='CARD'
//ACCIN DD DSN=SYSPLEX.LOGREC.ALLRECS,
  DISP=SHR,
  SUBSYS=(LOGR,IFBSEXIT,,)
//DIRECTWK DD UNIT=SYSDA,SPACE=(CYL,5,,CONTIG)
//EREPT DD DSN=EREPM.EVENT,DISP=(NEW,CATLG),
  DCB=BLKSIZE=133,
  UNIT=SYSDA,SPACE=(CYL,(25,5))
//TOURIST DD SYSOUT=A,DCB=BLKSIZE=133
//SYSABEND DD SYSOUT=A
//SYSIN DD DUMMY
/*
```

Obtaining information from the logrec recording control buffer

When the system writes a dump, the dump includes the records in the logrec buffer in storage; the buffer records have been either written to the logrec data set or are queued to be written to the logrec data set.

When you begin to diagnose a dump for a system problem, you can use IPCS to view the system records in the logrec recording control buffer.
Recording logrec error records

The logrec recording control buffer is one of the most important areas to be used when analyzing problems in MVS. This buffer serves as the interim storage location for hardware and software error records that are queued to be written to the logrec data set. The buffer is significant because of the error history it contains. Also, any records in the buffer that have not reached the logrec data set are almost certainly related to the problem you are trying to solve.

Formatting the logrec buffer

To format the logrec buffer, use the IPCS subcommand VERBEXIT LOGDATA. The entries that are still in the buffer will be formatted in the same way as entries that are printed in the EREP detail edit report.

Finding the logrec and WTO recording control buffers

There are two recording control buffers (RCB) in the SQA. The system uses one buffer for logrec messages, and the other for WTO messages. The CVT+X’16C’ (CVTRBCB) points to the recording buffers control block (RBCB). The RBCB contains the following information about the two recording control blocks (which are also referred to as RCBs or buffers):

For the logrec RCB:
- RBCB+X’10’ (RBCBLRCB) points to the logrec buffer.
- RBCB+X’14’ (RBCBLLEN) contains the length of the logrec buffer.

For the WTO RCB:
- RBCB+X’18’ (RBCBWRCB) points to the WTO buffer.
- RBCB+X’1C’ (RBCBWLEN) contains the length of the WTO buffer.

The logrec and WTO recording control buffers reside in fetch-protected SQA. Entries in these buffers have time stamps (8-byte TOD clock values) that allow you to look at a dump and create a chronological list of the logrec events and WTO messages.

Reading the logrec recording control buffer

The logrec recording control buffer is a “wrap-table” similar to the system trace table. The entries are variable in size. The latest entries are the most significant especially if they have not yet been written to the logrec data set. Knowing the areas of the system that have encountered errors and the actions of their associated recovery routines, information obtained from the logrec data set and from the logrec recording control buffer helps provide an overall understanding of the environment you are about to investigate.

Note: The SDWA in the logrec buffer is a compressed SDWA in which the recordable extensions start directly after the used portion of the SDWAVRA. The SDWAURAL field contains the length of the SDWAVRA.

You can find the oldest entry in the buffer by locating the end of the unused or free area, obtained from RCBFREE+RCBFLEN. (If this sum brings you to a point beyond the end of the buffer, subtract RCBTLNG from the sum.) You can also read the buffer backwards by using the entry length at the end of each entry. The latest entry appears directly before the free or unused area of the buffer.

For details about the format of the RCB, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).
Interpreting software records

There are two types of software records that are recorded in the logrec data set or the logrec log stream:

- **Software record**
  The system generates these records, providing information from the system diagnostic work area (SDWA) that describes problems detected because of an abend or a program check. See “Detail edit report for a software record” for more information.

- **Symptom record**
  Either a user's application program or the system can issue the SYMREC macro to request the creation of a symptom record. Generally, the symptom record describes problems not accompanied by an abend, but there are exceptions. See “Detail edit report for a symptom record” on page 575 for more information.

Use the search argument you obtain from the detail edit reports for either a software record or a symptom record to search for a known problem. If you do not conduct the search yourself, contact the IBM Support Center. The result of the search will be one of the following:

- The PTF that corrects the problem.
  Apply the PTF that corrects the error.

- The APAR, and possibly the related APAR, that describes the problem. In some cases, a temporary fix (either ZAP or update) or a procedure might circumvent the problem.
  Apply the temporary fix if it is available; otherwise, follow the circumvention procedure.

- A description of why the problem might have occurred, which often describes a frequent misuse of a product that causes the error record. This type of problem is referred to as a user error.
  When an error occurs because of the misuse of a product other than MVS, use the procedures documented for that product to determine how best to debug the problem.

For any case other than the three listed above, including the case where the service link database does not contain a record matching the search criteria, contact the IBM Support Center to report the problem.

**Detail edit report for a software record**

The detail edit report for a software record shows the complete contents of an error record for an abnormal end, including the system diagnostic work area (SDWA). The report is produced by EREP and, through the VERBEXIT LOGDATA subcommand, under IPCS.

Use the detail edit report for a software record to determine the cause of an abend, and the recovery action that the system or application has either taken or not taken. This report enables you to locate where an error occurred, similar to the analysis of an SVC dump. Once you locate the error, you can develop a search argument to obtain a fix for the problem.

For more information, refer to the following documentation:

- See [EREP User's Guide](#) for information about producing a detail edit report for an SDWA-type record.
Recording logrec error records

- See [z/OS MVS IPCS Commands](#) for information about the VERBEXIT LOGDATA subcommand.

The example output shown in "Example: One record with SDWARC4 and 64-bit information" is from one record with SDWARC4 and 64-bit information. This record also has information in the VRA, which is formatted.

**Example: One record with SDWARC4 and 64-bit information**

<table>
<thead>
<tr>
<th>TYPE: SOFTWARE RECORD</th>
<th>REPORT: SOFTWARE EDIT REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SVC 13)</td>
<td>DAY.YEAR</td>
</tr>
<tr>
<td>FORMATTED BY: IEAVTFDE</td>
<td>REPORT DATE: 343.04</td>
</tr>
<tr>
<td>HB87703</td>
<td>ERROR DATE: 336.04</td>
</tr>
<tr>
<td>MODEL: 2084</td>
<td>TIME: 17:43:44.72</td>
</tr>
<tr>
<td>SERIAL: 11778D</td>
<td></td>
</tr>
</tbody>
</table>

| JOBNAME: PIDA1028 | SYSTEM NAME: J50         |
| ERRORID: SEQ=00757 | CPU=0056 | ASID=0097 | TIME=17:43:44.7 |

SEARCH ARGUMENT ABSTRACT

<table>
<thead>
<tr>
<th>PIDS/5752SCLDR</th>
<th>RIDS/IEWLR00#L</th>
<th>RIDS/IEWLUNF0 AB/S0378 PRCS/00000014</th>
<th>RIDS/IEWLR0CV#R</th>
</tr>
</thead>
</table>

SYMPTOM DESCRIPTION

<table>
<thead>
<tr>
<th>PIDS/5752SCLDR</th>
<th>PROGRAM ID: 5752SCLDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIDS/IEWLR00#L</td>
<td>LOAD MODULE NAME: IEWLDR00</td>
</tr>
<tr>
<td>RIDS/IEWLUNF0</td>
<td>CSECT NAME: IEWLUNF0</td>
</tr>
<tr>
<td>AB/S0378</td>
<td>SYSTEM ABEND CODE: 0378</td>
</tr>
<tr>
<td>PRCS/00000014</td>
<td>ABEND REASON CODE: 00000014</td>
</tr>
<tr>
<td>RIDS/IEWLR0CV#R</td>
<td>RECOVERY ROUTINE CSECT NAME: IEWLRECV</td>
</tr>
</tbody>
</table>

OTHER SERVICEABILITY INFORMATION

<table>
<thead>
<tr>
<th>SUBFUNCTION: LSLOADER</th>
</tr>
</thead>
</table>

SERVICEABILITY INFORMATION NOT PROVIDED BY THE RECOVERY ROUTINE

RECOVERY ROUTINE LABEL

DATE ASSEMBLED

MODULE LEVEL

TIME OF ERROR INFORMATION

| PSW: 07041000 80000000 00000000 012F902E |
| INSTRUCTION LENGTH: 02 | INTERRUPT CODE: 000D |
| FAILING INSTRUCTION TEXT: 00181610 0A0D18CE 18FB180C |

BREAKING EVENT ADDRESS: 00000000 00000000

| AR/GR 0-1 | 00000000/00000000_84378000 |
| AR/GR 2-3 | 00000000/00000000_00000020 |
| AR/GR 4-5 | 00000000/00000000_00000000_00FFD098 |
| AR/GR 6-7 | 01FF0000/00000000_00000003 |
| AR/GR 8-9 | 00000000/00000000_00FFD4F8 |
| AR/GR 10-11 | 00000000/00000000_00FFD4F8 |
| AR/GR 12-13 | 00000000/00000000_8651F240 |
| AR/GR 14-15 | 00000000/00000000_8651F554 |

HOME ASID: 0097 PRIMARY ASID: 0097 SECONDARY ASID: 0097

PKM: 00C0 AX: 0000 EAX: 0000

RTM WAS ENTERED BECAUSE AN SVC WAS ISSUED IN AN IMPROPER MODE.

THE ERROR OCCURRED WHILE: A TYPE 1 SVC WAS IN CONTROL

A LOCKED OR DISABLED ROUTINE WAS IN CONTROL

LOCKS HELD: LOCAL/CML

NO SUPER BITS WERE SET.
RECOVERY ENVIRONMENT

RECOVERY ROUTINE TYPE: FUNCTIONAL RECOVERY ROUTINE (FRR)
PSW AT ENTRY TO FRR: 070C0000 86502368
FRR PARAMETER AREA ON ENTRY TO FRR:
+00 7F33F4A8 00000000 00000000 00000000 00000000 00000000 00000000

RECOVERY ROUTINE ACTION

THE RECOVERY ROUTINE RETRIED TO ADDRESS 0651FFA2.
THE REQUESTED SVC DUMP WAS SUCCESSFULLY STARTED.
NO LOCKS WERE REQUESTED TO BE FREED.
THE SDWA WAS REQUESTED TO BE FREED BEFORE RETRY.
THE REGISTER VALUES TO BE USED FOR RETRY:
REGISTERS 0-7
GR: 008FD0D0 00000000 7FFFC100 00000000 008FD098 7FFFC150 7FFFBF48 00F52C00
AR: 00000000 00000000 00000000 00000000 00000000 00000000 01FF000C 01FF000C
REGISTERS 8-15
GR: 7F33F4A8 06520100 00FD5750 00000001 8651F240 7F33F0E8 7F33F638 00000000
AR: 00000000 00000000 00000000 01FF000C 00000000 00000000 00000000 00000000

HEXADECIMAL DUMP

HEADER
+000 40831820 00000000 004336F 17434472  C.........?.... |
+010 0011778D 20840800  ......D..    |

JOBNAME
+000 D7C9C4C1 F1F0F2F8  PIDA1028 |

SDWA BASE
+000 00000000 04378000 00000000 00000000 00000000 00000000 00000000 00000000 |
+010 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |
+020 00000020 0000FC03 00FD5750 00000000 00000000 00000000 00000000 00000000 |
+030 00000003 00F52C00 7F33F4A8 00000000 00000000 00000000 00000000 00000000 |
+040 2C417000 7F35EC88 8651F240 7F33F0E8 5752....  |..HF.2 " :0Y |

VARIABLE RECORDING AREA (SDWAVRA)
+000 KEY: 37 LENGTH: 06
+002 C4C405C1 D4C5  DNAME   |
+008 KEY: 39 LENGTH: 06
+00A 6060C8C6 E2606060  |--HFS-- |

+0AA 0003 . . . . . . . . . . |
+0AC KEY: 53 LENGTH: 06
+0AE KEY: FF LENGTH: 06

SDWA FIRST RECORDABLE EXTENSION (SDWARC1)
+000 E2C303C4 D90E203 D6C1C4C5 D9404040  SCLDRLSLOADER   |
+010 40404040 40404040 40404040 00000000 . . . . . . . .  |
+020 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |
+030 00000000 00000000 F5F7F5F2 01000001 8752....  |
Recording logrec error records

+1c0     d1f5f040   40404040   |                        |
SDWA SECOND RECORDABLE EXTENSION (SDWARC2)
  +000 00000000 00000000 00000000 00000000 | .............. |

SDWA THIRD RECORDABLE EXTENSION (SDWARC3)
  +000 00000000 00000000 00000000 00000000 | .............. |
  +010 00000000 00000000 00000000 00000000 | .............. |

SDWA FOURTH RECORDABLE EXTENSION (SDWARC4)
  +000 00000000 84000000 00000000 84378000 | ....D.......D... |
  +010 00000000 00000020 00000000 0000FC03 | .............. & |
  +020 00000001 008F009B 00000000 00F57500 | ............jQ... |
  +030 00000000 00000001 00000001 00F52C00 | ..............S |

  +150 00000000 00000000 07041000 80000000 | .............. |
  +160 00000000 012F902E | ........ |

SDWA FIFTH RECORDABLE EXTENSION (SDWARC5)
  +000 00000000 00000000 00000000 00000000 | .............. |
  +010 00000000 00000000 00000000 00000000 | .............. |
  +020 00000000 00000000 00000000 00000000 | .............. |
  +030 00000000 00000000 00000000 00000000 | .............. |

  +090 00000000 00000000 00000000 00000000 | .............. |

ERRORID
  +000 02f50056 00970009 BD27 | .5...P.... |

TYPE: SOFTWARE RECORD
  Indicates that the detail edit report is for an SDWA-type record.

REPORT DATE
  Indicates the date on which the EREP report was created.

ERROR DATE
  Indicates the date on which the error occurred.

TIME
  Indicates the time, as local, at which the error occurred.

JOBNAME
  If the jobname is NONE-FRR, the error being recorded occurred in system or
  subsystem code covered by a functional recovery routine (FRR).

SYSTEM NAME
  Indicates the name of the system where the SDWA-type record was created.

ERRORID
  Allows you to coordinate diagnostic information from logrec, the console log
  (SYSLOG), and system dumps. The ERRORID is a concatenation of the
  following:

SEQ  A unique number assigned to each error. The sequence number
    indicates the order of the errors, but the records might not be listed in
    order. It is important to scan all entries and examine the sequence
    numbers to understand which error occurred first.

You might find the same sequence number used in more than one
entry when several recovery routines, as a result of percolation, get
control and request recording for the same error; however, the error
time stamp will be different.
Recording logrec error records

CPU  The internal identification number of the central processor that the failing process was running on at the time the error occurred. Use information from the system trace table about this CPU to learn more about the error.

ASID  The address space identifier (ASID) of the current, or home, address space at the time the error occurred.

TIME  Indicates the time of the error.

Use this symptom string to do a structured search of any IBM database.

PROGRAM ID
The program ID (PID) indicates the product and the component where the error occurred. For IBM products, see the tables in z/OS MVS Diagnosis that list the products and components. For non-IBM products, see the appropriate vendor-supplied documentation.

LOAD MODULE NAME
Indicates the load module in control at the time of the error.

CSECT NAME
Supplyd by the recovery routine that obtained control for the error. See the PSW for more information.

SYSTEM ABEND CODE
Indicates what system or user completion code was issued by the system, application, or component. See z/OS MVS System Codes for information about system abend codes. See the appropriate product documentation for user abend codes.

ABEND REASON CODE
Indicates the reason code, when available, associated with a system or user abend code.

RECOVERY ROUTINE CSECT NAME
Indicates the recovery routine that was given control to handle the error condition.

PSW
Indicates the program status word (PSW) at the time of the error.

If the software record is an SVC 13, the address in the second half of the PSW indicates the address of the module that detected the error. You need to find the caller of that module. The caller’s address will reside either in register 14, or, if register 14 points to module IEAVEEXP, use the STATUS section of the software record to determine the caller. In the STATUS section, the interrupt code will indicate the last SVC that was issued.

If the software record is a program interrupt, the address in the second half of the PSW usually points to the failing module.

FAILING INSTRUCTION TEXT
Contains 12 bytes of the instruction stream at the time of the error, including the actual instruction that caused the abend. Starting at the end of the sixth byte, subtract the instruction length to indicate the failing instruction. In the preceding example, the failing instruction is X’0A0D’.

THE ERROR OCCURRED WHILE . . .
Provides information about the system environment at the time of error, indicating what type of routine was in control, whether locks were held, and whether supervisor FRRs were set at the time of the error.
Recording logrec error records

STATUS
The PSW and registers that follow come from the request block (RB) associated with the ESTAE recovery routine that obtained control for the error. Using the information indicated will enable you to determine the program that was running at the time of the error. This information included in the STATUS section does not appear when an FRR handles recovery.

RECOVERY ROUTINE ACTION
Describes the recovery action performed or requested to be performed by the recovery routine. In the preceding example, an SVC dump was not requested. There are times, however, when the recovery routine will request an SVC dump. If SVC DUMP SUCCESSFULLY STARTED appears in this section, the error identifier (ERROR ID) appears in the SVC dump and in message IEA911E as it appears in the logrec error record.

HEXADECIMAL DUMP
Provides an unformatted hexadecimal dump of the SDWA control block. Depending on an indicator in the SDWA, which is set by the recovery routine generating the record, the SDWA is displayed in hexadecimal; EBCDIC text; or key, length, and data format.

VARIABLE RECORDING AREA (SDWAVRA)
Provides component-specific information. Using the information in the PROGRAM ID field, determine the component. For IBM products, see [z/OS MVS Diagnosis: Reference](http://www.ibm.com/systems/z/os/zos/bkserv/) for diagnostic information related to system components.

The SDWAVRA can optionally be mapped in a key-length-data format. Recovery routines use the SDWAVRA to construct messages and provide data that often contains valuable debugging information. Some MVS recovery routines use the key-length-data format to provide standardized diagnostic information for software incidents. This formatted information allows you to screen duplicate errors.

Constants for the key field have been defined to describe data, such as: return and/or reason codes, parameter lists, registers, and control block information. For example, a key of X'10' indicates a recovery routine parameter area. The SDWAVRAM bit (in the fixed portion of the SDWA) indicates that the SDWAVRA has been mapped in the key-length-data format as described by the IHAVRA mapping macro.

For the format of the SDWA, including a description of the keys, see z/OS MVS Data Areas in z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/)

SDWA RECORDABLE EXTENSIONS
In addition to the SDWA standard area and the SDWAVRA, the SDWA recordable extensions also contain valuable debugging information, as follows:

- **SDWARC1** (recording extension 1) contains additional component service data (such as the component ID, the component name, the address of the TCB representing the task that incurred the failure, the control registers, original completion code and reason code, linkage stack pointer, and translation exception access register number).
- **SDWARC2** (recording extension 2) contains additional I/O machine check data (such as the machine check interruption code).
- **SDWARC3** (recording extension 3) contains locking information (such as the locks to be freed, and the addresses of lockwords).
Recording logrec error records

**Note:** The SDWA that is in the logrec buffer is a compressed SDWA in which the recordable extensions start directly after the used portion of the SDWAVRA. The SDWAURAL field contains the length of the SDWAVRA.

**Detail edit report for a symptom record**
The SYMREC macro updates a symptom record with system environment information and then logs the symptom record in the logrec data set or logrec log stream. The system or application, using the SYMREC macro, creates a symptom record. The ADSR mapping macro maps the symptom record, and the symptom record contains diagnostic information determined by the application.

As an application or a system component detects errors during processing, it stores diagnostic information into the symptom record and issues the SYMREC macro to log the record. The diagnostic information consists of a description of a programming failure and a description of the environment in which the failure occurred.

See [z/OS MVS Programming: Assembler Services Reference IAR-XCT](http://www.ibm.com/systems/z/os/zos/bkserv/) for information about the SYMREC macro. See [z/OS MVS Data Areas](http://www.ibm.com/systems/z/os/zos/bkserv/) for information about the ADSR data area.

**Report output:** The example in “Example: Detail edit report for a symptom record” contains output from one record created by the system. Following the example is a list of the fields that are most important for diagnosis; only the highlighted fields are discussed.

**Example: Detail edit report for a symptom record**

<table>
<thead>
<tr>
<th>TYPE</th>
<th>REPORT</th>
<th>DAY, YEAR</th>
<th>REPORT DATE</th>
<th>ERROR DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYMPTOM RECORD</td>
<td>SOFTWARE EDIT REPORT</td>
<td>176.92</td>
<td>175.92</td>
<td></td>
</tr>
<tr>
<td>SCP: VS 2 REL 3</td>
<td>MODEL: 9021</td>
<td></td>
<td>TIME: 031347</td>
<td></td>
</tr>
</tbody>
</table>

**SEARCH ARGUMENT ABSTRACT:**

PIDS/5752SC1CM AB/S080A PRCS/00000010 RIDS/IEAVTRSR
RIDS/IGC0101CFL FLDS/PRWSORIGIN VALU/IEAVTRSR PCSS/FAILING

**SYSTEM ENVIRONMENT:**

<table>
<thead>
<tr>
<th>CPU MODEL: 9021</th>
<th>DATE: 175 92</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU SERIAL: 031347</td>
<td>TIME: 10:41:14.37</td>
</tr>
<tr>
<td>SYSTEM: CPUR</td>
<td>BCP: MVS</td>
</tr>
<tr>
<td>RELEASE LEVEL OF SERVICE ROUTINE: JBB4422</td>
<td></td>
</tr>
<tr>
<td>SYSTEM DATA AT ARCHITECTURE LEVEL: 10</td>
<td></td>
</tr>
<tr>
<td>COMPONENT DATA AT ARCHITECTURE LEVEL: 10</td>
<td></td>
</tr>
<tr>
<td>SYSTEM DATA: 00000000 00000000</td>
<td>........</td>
</tr>
</tbody>
</table>

**COMPONENT INFORMATION:**

| COMPONENT ID: 5752SC1CM |
| COMPONENT RELEASE LEVEL: D10 |
| DESCRIPTION OF FUNCTION: RTM2 RECURSION ERROR RECORDING |

**PRIMARY SYMPTOM STRING:**

PIDS/5752SC1CM AB/S080A PRCS/00000010 RIDS/IEAVTRSR
RIDS/IGC0101CFL FLDS/PRWSORIGIN VALU/IEAVTRSR PCSS/FAILING
PCSS/CSECT PCSS/UNKNOWN FLDS/RTM2SCTC FLDS/FROMPRWA
VALU/H00040000

<table>
<thead>
<tr>
<th>SYMPTOM</th>
<th>SYMPTOM DATA</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIDS/5752SC1CM</td>
<td>5752SC1CM</td>
<td>COMPONENT IDENTIFIER</td>
</tr>
<tr>
<td>AB/S080A</td>
<td>080A</td>
<td>ABEND CODE - SYSTEM</td>
</tr>
</tbody>
</table>
Recording logrec error records

<table>
<thead>
<tr>
<th>FIELD</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRCS/00000010</td>
<td>00000010</td>
</tr>
<tr>
<td>RIDS/IEAVTRSR</td>
<td>IEAVTRSR</td>
</tr>
<tr>
<td>RIDS/IGCO010C</td>
<td>IGCO010C</td>
</tr>
<tr>
<td>FLD5/SR#ORIGIN</td>
<td>SR#ORIGIN</td>
</tr>
<tr>
<td>VALU/CIEAVTRSR</td>
<td>IEAVTRSR</td>
</tr>
<tr>
<td>PCSS/FAILING</td>
<td>FAILING</td>
</tr>
<tr>
<td>PCSS/CSECT</td>
<td>CSECT</td>
</tr>
<tr>
<td>PCSS/UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FLD5/RTM2SCCTC</td>
<td>RTM2SCCTC</td>
</tr>
<tr>
<td>FLD5/FROM#PRWA</td>
<td>FROM#PRWA</td>
</tr>
<tr>
<td>VALU/H00040000</td>
<td>00040000</td>
</tr>
<tr>
<td>SYMPTOM</td>
<td>SYMPTOM</td>
</tr>
<tr>
<td>SYMPTOM DATA</td>
<td>SYMPTOM DATA</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td>EXPLANATION</td>
</tr>
<tr>
<td>FLD5/RTM2SCCTR</td>
<td>RTM2SCCTR</td>
</tr>
<tr>
<td>FLD5/RTM2SCCTX</td>
<td>RTM2SCCTX</td>
</tr>
<tr>
<td>VALU/H00040000</td>
<td>00040000</td>
</tr>
</tbody>
</table>

FREE FORMAT COMPONENT INFORMATION:

- KEY: FF00
- LENGTH: 0048 (0030)

HEX DUMP OF RECORD:

HEADER
- +000 C6831800 00000000 0092175F 10411437 |<C........K. ....|
- +010 A6031347 90210000 |W....... |

SYMPTOM RECORD
- +000 E20DF9F0 F2F1FOF3 F1F3F4F7 00000000 |SR9021031347....|
- +010 A5E2A254 A5ED4104 40404040 40404040 |VSS.V... |
- +020 4040C307 E404040 4040F5F7 F5F2D1C2 | CPUR 5752JB|

TYPE: SYMPTOM RECORD
- Indicates that the detail edit report is for a symptom record.

SEARCH ARGUMENT ABSTRACT
- Provides information you can use to create a search argument. If enough information exists in this field, you can search the IBM service link problem reporting database to determine if there is a PTF to correct the error.
- The information that follows the search argument abstract in a symptom record depends on the options specified on the SYMREC macro either by a user program or by a system component. In the report output listed above, the system recorded a recursive error. The information contained in a symptom record is variable. To obtain an interpretation, contact the IBM Support Center for the product or for the component that built the record.

Customizing symptom record location: You can control the location of logrec symptom records from non-authorized programs. Use the ASREXIT installation exit just before writing the logrec record to control:
- If a program can write symptom records
- The location of the symptom record: the logrec data set, job log, both, or neither

See z/OS MVS Installation Exits for information about ASREXIT.
Chapter 16. AMBLIST: Map load modules and program objects

AMBLIST provides the following problem data:

- Formatted listing of an object module
- Map of the control sections (CSECTs) in a load module or program object
- List of modifications to the code in a CSECT
- Map of all modules in the link pack areas (LPA)
- Map of the contents of the DAT-on nucleus. The map no longer represents the IPL version and message AMB129I will be issued.

These formatted listings can help you diagnose problems related to modules as they currently exist on your system. AMBLIST is a batch job that runs in problem state.

The following topics describe AMBLIST:

- “Obtaining AMBLIST output” on page 578
- “Reading AMBLIST output” on page 590

Long name support: AMBLIST will process external names (labels and references) up to 32767 bytes long. Names exceeding 16 bytes in length will be abbreviated in the formatted part of the listings and an abbreviation-to-long name equivalence table will be printed at the end of the listing. AMBLIST functions that provide long names support are: LISTLOAD, LISTIDR, and LISTOBJ (XSD and GOFF only).

Note:

1. Any load module to be formatted and printed by AMBLIST must have the same format as those created by the linkage editor or by the program management binder.
2. Any program object to be formatted and printed by AMBLIST must have the same format as those created by the program management binder.
3. A program object format 2 or greater having the non-editable attribute cannot be listed by AMBLIST.

See “LISTLOAD OUTPUT=XREF output (comparison of load module and program object version 1)” on page 623 for a comparison of the formatted output of a load module and a program object.

AMBLIST supports all data sets allocated in the extended addressing space (EAS) of an extended address volume (EAV).

AMBLIST supports the following dynamic allocation (DYNALLOC or SVC 99) options for all data sets: S99TIOEX(XTIOT), S99ACUCB (NOCAPTURE), and S99DSABA (DSAB above the line).
Obtaining AMBLIST output

To obtain AMBLIST output, you must code JCL or use the UNIX System Services amblist command. The amblist command is described in the z/OS UNIX System Services Command Reference.

This section describes these topics:
- "Specifying the JCL statements"
- "Controlling AMBLIST processing"
- "Examples of running AMBLIST" on page 583
- "Examples for z/OS UNIX System Services file support" on page 589

Specifying the JCL statements

Generally, the minimum partition or region for running AMBLIST is 64 kilobytes for all functions except LISTLPA, which requires 100 kilobytes. However, for large load modules, IBM recommends a minimum region size of 200 kilobytes. For program objects, IBM recommends a minimum region size of 12 megabytes.

AMBLIST requires the following JCL statements:

**JOB**
- Initiates the job.

**EXEC PGM=AMBLIST**
- Calls for the processing of AMBLIST.

**SYSPRINT DD**
- Defines the message data set.

**Anynname DD**
- Defines an input dataset or a z/OS UNIX System Services pathname.
  - Use the PATH parameter to specify a z/OS UNIX System Services pathname, and also use the PATHOPTS parameter to be able to access the file for reading.
  - If pathname is a directory name, the filename must be specified on the MEMBER parameter of the AMBLIST control statement. If the pathname ends with the filename, then omit the MEMBER parameter.

**SYSIN DD**
- Defines the data set (in the input stream) that contains AMBLIST control statements.

Controlling AMBLIST processing

You control AMBLIST processing by supplying one or more control statements in the input stream. Code the control statement that applies to the data you want to obtain according to the following rules:

- Leave column 1 blank, unless you want to supply an optional symbolic name. A symbolic name is analogous to the label name in a program. The maximum length of a symbolic name is eight characters. A symbolic name must end with one or more blanks.
- If a complete control statement will not fit on a single line, end the first line with a comma or a non-blank character in column 72 and continue on the next line. Begin all continuation statements in columns 2 - 16. Do not split parameters between two lines. The only exceptions are the MEMBER parameters, which you can split at any internal comma.
LISTLOAD control statement
Use the LISTLOAD control statement to obtain a listing of load module or program objects. The listed data can help you verify why certain link-edit errors might have occurred.

```
LISTLOAD
[OUTPUT={MODLIST|XREF|BOTH|MAP}]
[,TITLE=('title',position)]
[,DDN=ddname]
[,MEMBER={member}(member1,membern...)]
[,RELOC=hhhhhhhh]
[,ADATA={YES|NO}]
[,IMPEXP={DUMP|SYMBOLS}]
[,SECTION1={YES|NO}]
```

**OUTPUT={MODLIST|XREF|BOTH|MAP}**
OUTPUT=MODLIST requests a formatted listing of the text and control information of a load module or program object.
OUTPUT=XREF requests a module map and cross-reference listing for the load module or program object.
OUTPUT=BOTH requests both a formatted listing of the load module or program object and its map and cross-references.
OUTPUT=MAP requests a numerical map for the load module or program object.
If this parameter is omitted, OUTPUT=BOTH will be assumed.

**TITLE=('title',position)**
Specifies a title, from one to 40 characters long, to be printed below the heading line on each page of output. (The heading line identifies the page number and the type of listing being printed, and is not subject to user control.) The position subparameter specifies whether or not the title is indented; if TITLE=('title',1) is specified, or if the position parameter is omitted, the title will be printed flush left, that is, starting in the first column. If you want the title indented from the margin, use the position parameter to specify the number of characters to leave blank before the title. If you specify a position greater than 80, the indentation from the margin defaults to 1.

**DDN=ddname**
Identifies the DD statement that defines the data set containing the input object module. If the DDN= parameter is omitted, AMBLIST will assume SYSLIB as the default ddname.

**MEMBER={member}(member1,membern...)**
Identifies the input load module or program object by member name or alias name. To specify more than one load module or program object, enclose the list of names in parentheses and separate the names with commas.

**Note:**
AMBLIST

1. If you specify MEMBER=IEANUCxx, where xx is the suffix of the member used during the current IPL, AMBLIST will list the DAT-ON nucleus. If you do not include the MEMBER parameter, AMBLIST will print all modules in the data set.

2. AMBLIST will accept member names up to 63 bytes in length. For aliases longer than 63 bytes, their primary member names must be entered instead.

3. If the DD name associated with this operation is allocated to a z/OS UNIX System Services directory, the pathname must end with a file, or there must also be a MEMBER parameter specifying the file or files in that directory.

RELOC=hhhhhhhh

Specifies a relocation or base address in hexadecimal of up to eight characters. When the relocation address is added to each relative map and cross-reference address, it gives the absolute central storage address for each item on the output listing. If you omit the RELOC parameter, no relocation is performed.

ADATA=\{YES|NO\}

ADATA=YES on LISTLOAD OUTPUT=MODLIST or OUTPUT=BOTH requests a formatted listing of all ADATA classes, if they exist, in the program object to be displayed in the traditional dump format, 32 bytes per line, with hexadecimal representation on the left and EBCDIC on the right, in addition to the output listing with the specified output parameter.

OUTPUT=NO on LISTLOAD OUTPUT=MODLIST or OUTPUT=BOTH requests a normal formatted listings with the specified output parameter, and ADATA suppressed.

If this parameter is omitted, ADATA=NO will be assumed.

IMPEXP=\{DUMP|SYMBOLS\}

IMPEXP=SYMBOLS indicates that section IEBWBCIE text is displayed as a symbolically formatted IMPORT/EXPORT section of the output.

IMPEXP=DUMP indicates that section IEBWBCIE text is displayed in the traditional dump format (as described under the ADATA parameter).

SECTION1=\{YES|NO\}

SECTION1=YES requests that the module level section information be displayed.

SECTION1=NO requests that the module level section information not be displayed.

LISTOBJ control statement

Use the LISTOBJ control statement to obtain listings of selected object modules.

LISTOBJ supports traditional object modules as well as object modules in XOBJ or GOFF format.

```plaintext
LISTOBJ
  \[TITLE=('title',position)\]
  [,DDN=ddname]
  [,MEMBER={member|(member1,membern...)}]
```

TITLE=('title',position)

Specifies a title, from one to 40 characters long, to be printed below the heading line on each page of output. (The heading line identifies the page number and the type of listing being printed, and is not subject to user
control.) The position parameter specifies whether or not the title is indented; if TITLE=(title,1) is specified, or if the position parameter is omitted, the title will be printed flush left, that is, starting in the first column. If you want the title indented from the margin, use the position parameter to specify the number of characters to leave blank before the title. If you specify a position greater than 80, the indentation from the margin defaults to 1.

**DDN=ddname**
Identifies the DD statement that defines the data set containing the input module. If the DDN parameter is omitted, AMBLIST will assume SYSLIB as the default ddname.

**MEMBER={member|{member1[,member2]...}}**
Identifies the input object module by member name or alias name. To specify more than one object module, enclose the list of names in parentheses and separate the names with commas.

**Note:**
1. You must include the MEMBER parameter if the input object modules exist as members in a partitioned data set (PDS or PDSE). If you do not include the MEMBER parameter, AMBLIST will assume that the input data set is organized sequentially and that it contains a single, continuous object module.
2. AMBLIST will accept member names up to 63 bytes in length. For aliases longer than 63 bytes, their primary member names must be entered instead.
3. If the DD name associated with this operation is allocated to a z/OS UNIX System Services directory, the pathname must end with a file, or there must also be a MEMBER parameter specifying the file or files in that directory.

**Example: Processing a pathname:** In this example, AMBLIST processes the pathname /path/to/dir/longmembername.
```
//SYSLIB DD PATH='/path/to/dir'
//SYSIN DD *
LISTOBJ MEMBER=longmembername
```

**LISTIDR control statement**
Use the LISTIDR control statement to obtain listings of selected CSECT identification records (IDR).

```
LISTIDR
[OUTPUT={IDENT|ALL}],TITLE=('title',position)]
[,DDN=ddname]
[,MEMBER={member|{member1,member2}...}]
[,MODLIB]

OUTPUT={IDENT|ALL}
```

Specifies whether AMBLIST must print all CSECT identification records or only those containing SPZAP data and user data. If you specify OUTPUT=ALL, all IDRs associated with the module will be printed. If you specify OUTPUT=IDENT, AMBLIST will print only those IDRs that contain SPZAP
data or user-supplied data. If you omit this parameter, AMBLIST will assume a
default of OUTPUT=ALL. Do not specify OUTPUT if you specify the MODLIB
parameter.

**TITLE=('title',position)**
Specifies a title, from one to 40 characters long, to be printed below the
heading line on each page of output. (The heading line identifies the page
number and the type of listing being printed, and is not subject to user
control.) The position parameter specifies whether or not the title is indented;
if TITLE=('title',1) is specified, or if the position parameter is omitted, the title
is printed flush left, that is, starting in the first column. If you want the title
indented from the margin, use the position parameter to specify the number of
characters that are left blank before the title. If a position greater than 80 is
specified, the indentation from the margin defaults to 1.

**DDN=ddname**
Identifies the DD statement that defines the data set containing the input
module. If you omit the DDN parameter, AMBLIST will assume SYSLIB as the
default ddname.

**Note:**
If the DD name associated with this operation is allocated to a z/OS UNIX
System Services directory, there must also be a MEMBER parameter
specifying the file or files in that directory.

**MEMBER={member|(member1,membern...)}**
Identifies the input load module or program object by member name or alias
name. To specify more than one load module or program object, enclose the
list of names in parentheses and separate the names with commas.

**Note:**
1. Do not specify MEMBER if you specify the MODLIB parameter. If you do
   not include the MEMBER parameter, AMBLIST will print all modules in the
data set.
2. AMBLIST will accept member names up to 63 bytes in length. For aliases
   longer than 63 bytes, their primary member names must be entered instead.
3. If the DD name associated with this operation is allocated to a z/OS UNIX
   System Services directory, the pathname must end with a file, or there must
   also be a MEMBER parameter specifying the file or files in that directory.

**MODLIB**
Prevents AMBLIST from printing the module summary. AMBLIST prints the
IDRs that contain SPZAP data or user-supplied data. No page ejects occur
between modules. When you specify MODLIB, the OUTPUT or MEMBER
parameters are not valid parameters.

**LISTLPA control statement**
Use the LISTLPA control statement to obtain listings of selected modules in the
fixed link pack area (LPA).

```
LISTLPA [FLPA][,MLPA][,PLPA]
```

**LISTLPA**
Lists the modules in the fixed link pack area, the modified link pack area, and
the pageable link pack area (PLPA). This listing is a map that includes modules
residing in the extended sections of each link pack area (LPA). If you do not specify any parameters on the LISTLPA control statement, then AMBLIST maps modules from all three LPAs.

The LISTLPA control statement does not support dynamic LPA. If the dynamic LPA support is used to update LPA, those changes will not be reflected in the AMBLIST LISTLPA output. The LISTLPA control statement will not be enhanced to support new operating system functions. The recommended replacement is the LPAMAP subcommand of IPCS. See z/OS MVS IPCS Commands for details about this command.

FLPA
   Requests mapping of the modules in the fixed link pack area.

MLPA
   Requests mapping of the modules in the modified link pack area.

PLPA
   Requests mapping of the modules in the pageable link pack area.

Examples of running AMBLIST

Using the control statements as input into the JCL for the job, you can invoke AMBLIST to provide output. The following examples of AMBLIST include the control statement needed to produce the output and sample JCL for each function.

List the contents of an object module

You can use AMBLIST to format three types of object module:
1. OBJ (traditional object module)
2. XOBJ (extended object module, based on OBJ)
3. GOFF (Generalized Object File Format).

You can list the following information from an object module:
• the head record (HDR) - which may contain information about the character set and expected operating environment (GOFF only)
• external symbol dictionary entries (ESD or XSD)
• relocation dictionary entries (RLD)
• the text of the program - the instructions and data, as output by the language translator (TXT)
• translator identification record (IDRL) - which contains the compiler ID and compile date
• ADATA records (GOFF only)
• LEN records (GOFF only)
• and the END record.

To list object module contents, invoke AMBLIST with the LISTOBJ control statement. For sample outputs, see “LISTOBJ outputs” on page 594.

In Figure 170 on page 584, AMBLIST is used to format and list an object module included in the input stream.
AMBLIST

OBJMOD DD Statement
Defines the input data set, which follows immediately. In this case, the input data set is an object module.

SYSIN DD Statement
Defines the data set containing AMBLIST control statements, which follows immediately.

LISTOBJ Control Statement
Instructs AMBLIST to format the data set defined by the OBJMOD DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.

In Figure 171, AMBLIST is used to list all object modules contained in the data set named OBJMOD, and three specific object modules from another data set called OBJMODS.

Note: If you are using AMBLIST to list program objects, IBM recommends that you specify REGION=12M or higher.

OBJLIB and OBJSDS DD Statements
Define input data sets that contain object modules.

SYSIN DD Statement
Defines the data set in the input stream containing AMBLIST control statements.

LISTOBJ Control Statement #1
Instructs AMBLIST to format the data set defined by the OBJSDS DD statement, treating it as a single member. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTOBJ Control Statement #2
Instructs AMBLIST to format three members of the partitioned data set (PDS or PDSF) with the specified titles.
PDSE) defined by the OBJLIB DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

Map the CSECTs in a load module or program object

You can list the organization of CSECTs within the load module or program object, the overlay structure (if any), and the cross-references for each CSECT. To map CSECTs, invoke AMBLIST with the LISTLOAD control statement.

For sample output, see “LISTLOAD OUTPUT=MODLIST output” on page 602, “Alphabetical cross-reference” on page 622, and “LISTLOAD OUTPUT=XREF output (comparison of load module and program object version 1)” on page 623.

In Figure 172, AMBLIST is used to produce formatted listings of several load modules or program objects.

Note: If you are using AMBLIST to format program objects, IBM recommends that you specify REGION=2M or higher.

Figure 172. Example: Listing several load modules or program objects

SYSLIB DD Statement

Defines an input data set, SYS1.LINKLIB, that contains load modules or program objects to be formatted.

LOADLIB DD Statement

Defines a second input data set.

SYSIN DD Statement

Defines the data set (in the input stream) containing the AMBLIST control statements.

LISTLOAD Control Statement #1

Instructs AMBLIST to format the control and text records including the external symbol dictionary and relocation dictionary records of the load module or program object TESTMOD in the data set defined by the LOADLIB DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTLOAD Control Statement #2

Instructs AMBLIST to produce a module map and cross-reference listing of the load modules or program objects MOD1, MOD2, and MOD3 in the data set defined by the LOADLIB DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTLOAD Control Statement #3

Instructs AMBLIST to produce a formatted listing of the load module or
program object and its map and cross-reference listing. Because no DDN= parameter is included, the input data set is assumed to be the one defined by the SYSLIB DD statement. Because no MEMBER parameter is specified, all load modules in the data set will be processed. This control statement also specifies a title for each page of output, to be indented 20 characters from the left margin.

Figure 173 shows how to use AMBLIST to verify three modules. Assume that an unsuccessful attempt has been made to link-edit an object module with two load modules or program objects to produce one large load module or program object.

```
//LSTLD0BJ JOB MSGLEVEL=(1,1)
// EXEC PGM=AMBLIST,REGION=64K
//SYSPRINT DD SYSOUT=A
//OBJMOD DD DSN=MYMOD,DISP=SHR
//LOADMOD1 DD DSN=YOURMOD,DISP=SHR
//LOADMOD2 DD DSN=HISMOD,DISP=SHR
//SYSIN DD *
LISTOBJ DDN=OBJMOD,
   DSN=OBJMOD,
   TITLE=('OBJECT LISTING FOR MYMOD',20)
LISTLOAD DDN=LOADMOD1,OUTPUT=BOTH,
   DSN=LOADMOD1,OUTPUT=BOTH,
   TITLE=('LISTING FOR YOURMOD',25)
LISTIDR DDN=LOADMOD1,OUTPUT=ALL,
   DSN=LOADMOD1,OUTPUT=ALL,
   TITLE=('IDRS FOR YOURMOD',25)
LISTLOAD DDN=LOADMOD2,OUTPUT=BOTH,
   DSN=LOADMOD2,OUTPUT=BOTH,
   TITLE=('LISTING FOR HISMOD',25)
LISTIDR DDN=LOADMOD2,OUTPUT=ALL,
   DSN=LOADMOD2,OUTPUT=ALL,
   TITLE=('IDRS FOR HISMOD',25)
```

Figure 173. Example: Listing several load modules or program objects

OBJMOD DD Statement
Defines an input load module or program object data set.

LOADMOD1 and LOADMOD2 DD Statements
Define input load module or program object data sets.

SYSIN DD Statement
Defines the data set containing AMBLIST control statements.

LISTOBJ Control Statement
Instructs AMBLIST to format the data set defined by the OBJMOD DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTLOAD Control Statement #1
Instructs AMBLIST to format all records associated with the data set defined by the LOADMOD1 DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.

LISTIDR Control Statement #1
Instructs AMBLIST to list all CSECT identification records associated with the data set defined by the LOADMOD1 DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.

LISTLOAD Control Statement #2
Instructs AMBLIST to format all records associated with the data set defined by the LOADMOD2 DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.
LISTIDR Control Statement #2
Instructs AMBLIST to list all CSECT identification records associated with the data set defined by the LOADMOD2 DD statement. It also specifies a title for each page of output to be indented 25 characters from the left margin.

Trace modifications to the executable code in a CSECT
You can list the information in a load module or program object's CSECT identification records (IDRs). An IDR provides the following information:
- The version and modification level of the language translator and the date that each CSECT was translated. (Translation data is available only for CSECTs that were produced by a translator that supports IDR generation.)
- The version and modification level of the linkage editor or binder that built the load module or program object and gives the date the load module or program object was created.
- Modifications to the load module or program object, by date, that might have been done using SPZAP.

An IDR might also contain optional user-supplied data.

To trace modifications, invoke AMBLIST with the LISTIDR control statement. For sample output, see "LISTIDR output" on page 628.

In Figure 174, AMBLIST is used to list the CSECT identification records in several load modules or program objects.

```
//IDRLIST JOB MSGLEVEL=(1,1)
//LISTSTEP EXEC PGM=AMBLIST,REGION=64K
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSN=SYS1.LINKLIB,DISP=SHR
//LOADLIB DD DSN=LOADMODS,DISP=SHR
//SYSIN DD *
LISTIDR TITLE=('IDR LISTINGS OF ALL MODS IN LINKLIB',20)
LISTIDR OUTPUT=IDENT,DDN=LOADLIB,MEMBER=TESTMOD
LISTIDR TITLE=('LISTING OF MODIFICATIONS TO TESTMOD',20)
LISTIDR OUTPUT=ALL,DDN=LOADLIB,MEMBER=(MOD1,MOD2,MOD3),
      TITLE=('IDR LISTINGS OF MOD1 MOD2 MOD3',20)
LISTIDR DDN=LOADLIB,MODLIB
```

Figure 174. Example: Listing IDR information for several load modules

SYSLIB DD Statement
Defines an input data set, SYS1.LINKLIB, that contains load modules or program objects to be processed.

LOADLIB DD Statement
Defines a second input data set.

SYSIN DD Statement
Defines the data set (in the input stream) containing the AMBLIST control statements.

LISTIDR Control Statement #1
Instructs AMBLIST to list all CSECT identification records for all modules in SYS1.LINKLIB (this is the default data set since no DDN parameter was included). It also specifies a title for each page of output, to be indented 20 characters from the left margin.
LISTIDR control statement #2
Instructs AMBLIST to list CSECT identification records that contain SPZAP or user-supplied data for the load module or program object named TESTMOD. TESTMOD is a member of the data set defined by the LOADLIB DD statement. This control statement also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTIDR control statement #3
Instructs AMBLIST to list all CSECT identification records for of the load modules or program objects MOD1, MOD2, and MOD3. These are members in the data set defined by the LOADLIB DD statement. This control statement also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTIDR control statement #4
Instructs AMBLIST to list CSECT identification records that contain SPZAP or user-supplied data for the LOADLIB data set. The module summary print out is suppressed.

List the modules in the link pack area and the contents of the DAT-on nucleus

You can list all modules in the fixed link pack area, the modified link pack area, and the pageable link pack area.

To map link pack area modules, invoke AMBLIST with the LISTLPA control statement. For sample output, see "LISTLPA output" on page 631.

You can also produce a map and cross-reference listing of a nucleus.

To map the contents of the DAT-on nucleus, invoke AMBLIST with the LISTLOAD MEMBER=IEANUCxx control statement.

Figure 175 shows how to use the LISTLOAD and LISTLPA control statements to list a system nucleus and map the fixed link pack area, the modified link pack area, and the pageable link pack area. Note that in this example the data set containing the nucleus is named SYS1.NUCLEUS, and the nucleus occupies the member named IEANUC01. The map no longer represents the IPL version of the nucleus and message AMB129I will be issued. Use IPCS to format the NUCMAP. For information on using IPCS, see z/OS MVS IPCS User's Guide and z/OS MVS IPCS Commands.

```plaintext
/ListNUC JOB MSGLEVEL=(1,1)
/STEP EXEC PGM=AMBLIST,REGION=100K
/SYSPRINT DD SYSOUT=A
/SYSLIB DD DSN=SYS1.NUCLEUS,DISP=SHR,UNIT=3330,
    VOL=SER=nnnnn
   // VOL=SER=nnnnn
/SYSIN DD *
   // LISTLOAD DDN=SYSLIB, MEMBER=IEANUC01,
       TITLE=('LISTING FOR NUCLEUS IEANUC01',25)
       LISTLPA
/*

Figure 175. Example: Listing a system nucleus and the link pack area

SYSLIB DD Statement
Defines the input data set, which in this case contains the nucleus.
SYSIN DD Statement
Defines the data set containing AMBLIST control statements, which follows immediately.

LISTLOAD control statement
Instructs AMBLIST to format the control and text records including the external symbol dictionary and relocation dictionary records of the load module IEANUC01 in the data set defined by the SYSLIB DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.

LISTLPA control statement
Instructs AMBLIST to map the fixed link pack area (FLPA), the modified link pack area (MLPA), and the pageable link pack area (PLPA).

Examples for z/OS UNIX System Services file support
AMBLIST will support formatted listings of program objects and object modules in z/OS UNIX System Services files.

To obtain a formatted listing of a program object in a z/OS UNIX System Services file, specify the complete pathname in a DD statement and code the control statement. Use the JCL example shown in Figure 176 as a guide.

```clike
//LIST EXEC PGM=AMBLIST
//HFS1 DD PATH='/u/USER1/outmod', PATHOPTS=(ORDONLY)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
LISTLOAD DDN=HFS1,OUTPUT=MODLIST
```

Figure 176. Example: z/OS UNIX System Services program object

To obtain a formatted listing of an object module in a z/OS UNIX System Services file, specify the complete pathname in a DD statement and code the control statement. Use the JCL example shown in Figure 177 as a guide.

```clike
//LIST EXEC PGM=AMBLIST
//HFS1 DD PATH='/u/USER1/myobject.o',PATHDISP=(KEEP,KEEP),
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
LISTOBJ DDN=HFS1
```

Figure 177. Example: z/OS UNIX System Services object module

AMBLIST will support the LISTIDR control statement for program objects in z/OS UNIX System Services files. Use the JCL example shown in Figure 178 as a guide.

```clike
//LIST EXEC PGM=AMBLIST
//HFS1 DD PATH='/u/USER1/outmod'
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
LISTIDR DDN=HFS1
```

Figure 178. Example: z/OS UNIX System Services control statement

For z/OS UNIX System Services files, MEMBER NAME is the file name and LIBRARY is the directory name. If the pathname is too long to fit in the space.
reserved on the line, it will be truncated on the left and preceded by two periods
and a space (\textquotedblleft .. \textquotedblright). In Figure 179, the library name is truncated while the member
name is not.

\begin{verbatim}
MEMBER NAME:  01234567890123456789012345678901234567890123456789012345678901234567890abc
MAIN ENTRY POINT:  00000000
LIBRARY:  .. 3456789012345678901234567890123456789012345678901234567890abc/ AMODE OF MAIN ENTRY POINT: 31
\end{verbatim}

Figure 179. Example: Differences in output

\section*{Reading AMBLIST output}

AMBLIST produces a separate listing for each control statement that you specify.

\begin{itemize}
\item The first page of each listing always shows the control statement as it was
entered.
\item The second page of the listing is a module summary, unless you requested
LISTOBJ, LISTLPA, or MODLIB with LISTIDR; in that case, no module summary
will be produced, and the second page of the listing will be the beginning of the
formatted output.
\end{itemize}

The module summary gives the member name (with aliases), the entry point
and its addressing mode, alias entry points and their addressing modes, the
attributes assigned to the module by the linkage editor or program management
binder, the system status index information (SSI), the APF code, an residence
mode for the module being formatted. For program objects, the PMAR and
PMARL are displayed in hexadecimal for diagnostic information. Figure 180 and
Figure 181 on page 591 show samples of module summary processed by the
linkage editor and the binder.

\begin{itemize}
\item The third page of the listing (or, for LISTOBJ, LISTLPA, or MODLIB with
LISTIDR the second page) is the beginning of the formatted output itself.
\end{itemize}

\section*{Module summary}

Figure 180 is sample of a module summary for a load module that was processed
by the linkage editor.

\begin{verbatim}
LISTLOAD DSN=DD1, MEMBER=TESTPR

A ***** MODULE SUMMARY *****
MEMBER NAME: TESTPR
LIBRARY: DD1
NO ALIASES **

B ***** ATTRIBUTES OF MODULE *
** BIT STATUS BIT STATUS BIT STATUS BIT STATUS **
 0 NOT-RENT  1 NOT-REGS  2 NOT-OVLY  3 NOT-TEST
 4 NOT-DL  5 BLOCK  6 EXEC  7 MULTI-RCD
 8 NOT-DC  9 ZERO-ORG  10 EP-ZERO  11 RLD
12 EDIT 13 NO-SYMS  14 F-LEVEL  15 NOT-REFR

C

MODULE SSI: NONE
APF CODE: 00000000
RMODE: 24
LONGPARM: NO

D ***** LOAD MODULE PROCESSED EITHER BY VS LINKAGE EDITOR OR BINDER
\end{verbatim}

Figure 180. Example: Module summary for a load module processed by the linkage editor

Figure 181 on page 591 is sample of a module summary for a program object
processed by the binder.
The following describes Figure 180 on page 590 and Figure 181.

**Entry Names.** For the member, the library (ddname) and member name are displayed, along with the primary entry point offset and AMODE. The MEMBER NAME field contains the primary name.

For each alias or alternate entry point, AMBLIST shows the alias name, entry point offset and AMODE. If no aliases are present, AMBLIST prints NO ALIASES. If the input name is an alias, then its name is printed in the alias section preceded by two asterisks.

The constants ALT PRIMARY will be added to the right of the amode of the alias name which was a long primary name in which the binder had converted to an alias.

**Attributes of the Module.** The attributes of the module are represented by bits. Each bit is set either ON or OFF. In the listing, AMBLIST interprets the bit settings and shows a descriptive value in the STATUS column. For example, in Figure 180 on page 590 and Figure 181, bit 0 is interpreted as NOT-RENT. This means the module is not reentrant. For a description of all the STATUS values, see Table 70 on page 592.

**Other Attributes.** The remaining module attributes are displayed following the table. This includes the system status index (SSI) field, the APF (authorized program facility) code, the RMODE (residence mode) for the

---

The following describes Figure 180 on page 590 and Figure 181.

**Entry Names.** For the member, the library (ddname) and member name are displayed, along with the primary entry point offset and AMODE. The MEMBER NAME field contains the primary name.

For each alias or alternate entry point, AMBLIST shows the alias name, entry point offset and AMODE. If no aliases are present, AMBLIST prints NO ALIASES. If the input name is an alias, then its name is printed in the alias section preceded by two asterisks.

The constants ALT PRIMARY will be added to the right of the amode of the alias name which was a long primary name in which the binder had converted to an alias.

**Attributes of the Module.** The attributes of the module are represented by bits. Each bit is set either ON or OFF. In the listing, AMBLIST interprets the bit settings and shows a descriptive value in the STATUS column. For example, in Figure 180 on page 590 and Figure 181, bit 0 is interpreted as NOT-RENT. This means the module is not reentrant. For a description of all the STATUS values, see Table 70 on page 592.

**Other Attributes.** The remaining module attributes are displayed following the table. This includes the system status index (SSI) field, the APF (authorized program facility) code, the RMODE (residence mode) for the
entire module, the PO format (loader data level), the OS compat level (binder data format), and XPLINK. If attribute bit 19 is the OFF state, NONE will be displayed in place of SSI. SSI is usually set through the SETSSI control statement in the binder or SPZAP programs.

**Note:** For an OS Compat Level less than z/OS V1R3, no Compat level will be printed.

PO format and XPLINK are applicable only for program objects. PO format is the version of the program object. XPLINK indicates whether any routines use XPLINK linkage conventions. Compat level designates the lowest OS release at which the release's binder could rebind this object. Note that the level at which the module can be executed is determined by the PO format.

Linking Program. The last line in the module summary identifies the linking program (VS linkage editor or binder) that created the module. For example:

```
*****PROGRAM OBJECT PROCESSED BY BINDER
```

This is applicable to a program object.

```
****LOAD MODULE PROCESSED EITHER BY VS LINKAGE EDITOR OR BINDER
```

The load module is either created by the linkage editor and processed by the binder, or created by the binder and processed by the linkage editor.

PMAR and PMARL. For program objects, the PMAR and PMARL are displayed in hexadecimal for diagnostic purposes, preceded by:

```
*** THE FOLLOWING ARE THE UNFORMATTED PDSE DIRECTORY ENTRY SECTIONS (PMAR AND PMARL)
```

**Table 70** summarizes the attributes of program objects and load modules. The first column shows the bit position. Columns 2 and 3 show the displayed constant and its meaning for the OFF condition. Columns 4 and 5 show the displayed constant and meaning for the ON condition.

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>OFF Value</th>
<th>Meaning</th>
<th>ON Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NOT-RENT</td>
<td>Module is not reentrant.</td>
<td>RENT</td>
<td>Module is reentrant.</td>
</tr>
<tr>
<td>01</td>
<td>NOT-REUS</td>
<td>Module is not reusable.</td>
<td>REUS</td>
<td>Module is reusable.</td>
</tr>
<tr>
<td>02</td>
<td>NOT-OVLY</td>
<td>Module is not in overlay format.</td>
<td>OVLY</td>
<td>Module is in overlay format.</td>
</tr>
<tr>
<td>03</td>
<td>NOT-TEST</td>
<td>Test option was not specified during binding.</td>
<td>TEST</td>
<td>Test option was specified during binding.</td>
</tr>
<tr>
<td>04</td>
<td>NOT-OL</td>
<td>Program can be invoked through all CSV macros.</td>
<td>ONLY-LOAD</td>
<td>Program can be loaded only through LOAD macro.</td>
</tr>
<tr>
<td>05</td>
<td>BLOCK</td>
<td>Module consists of a single, contiguous block of text. This bit is always off for program objects.</td>
<td>SCTR</td>
<td>Module can be scatter loaded (MVS nucleus only).</td>
</tr>
<tr>
<td>06</td>
<td>NON-EXEC</td>
<td>Module is marked not executable.</td>
<td>EXEC</td>
<td>Module is marked executable.</td>
</tr>
<tr>
<td>07</td>
<td>MULTI-RCD</td>
<td>Module contains multiple text records. This bit is always off for program objects.</td>
<td>1-TXT</td>
<td>Module contains no RLD items and only one block of text.</td>
</tr>
<tr>
<td>Bit Position</td>
<td>OFF Value</td>
<td>Meaning</td>
<td>ON Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>----------------------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>08</td>
<td>DC</td>
<td>Module is processable by all levels of linkage editor.</td>
<td>NOT-DC</td>
<td>Module is processable only by F-level linkage editor and above. This bit is always on for program objects.</td>
</tr>
<tr>
<td>09</td>
<td>NOT-ZERO</td>
<td>Origin of first text block greater than zero.</td>
<td>ZERO-ORG</td>
<td>Origin of first text block is zero. This bit is always on for program objects.</td>
</tr>
<tr>
<td>10</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>11</td>
<td>RLD</td>
<td>Module contains RLD items.</td>
<td>NO-RLD</td>
<td>Module contains no RLD items.</td>
</tr>
<tr>
<td>12</td>
<td>EDIT</td>
<td>Module can be reprocessed by binder.</td>
<td>NOT-EDIT</td>
<td>Module cannot be reprocessed by binder.</td>
</tr>
<tr>
<td>13</td>
<td>NO-SYMS</td>
<td>Module contains no SYM records.</td>
<td>SYMS</td>
<td>Module contains SYM records.</td>
</tr>
<tr>
<td>14</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>15</td>
<td>NOT-REFR</td>
<td>Module is not refreshable.</td>
<td>REFR</td>
<td>Module is refreshable.</td>
</tr>
<tr>
<td>16</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
</tbody>
</table>

**Note:** The following bits are shown only for program objects.

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>OFF Value</th>
<th>Meaning</th>
<th>ON Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>&lt;16M</td>
<td>Module text size is less than 16 megabytes.</td>
<td>&gt;16M</td>
<td>Module text size is greater than or equal to 16 megabytes.</td>
</tr>
<tr>
<td>18</td>
<td>NOT-PL</td>
<td>Page alignment is not required for loaded text.</td>
<td>P-ALIGN</td>
<td>Page alignment is required for loaded text.</td>
</tr>
<tr>
<td>19</td>
<td>NO-SSI</td>
<td>System status index is not present.</td>
<td>SSI</td>
<td>System status index is present.</td>
</tr>
<tr>
<td>20</td>
<td>NOT-APF</td>
<td>There is not an APF section in the directory. (APFCODE is not present.)</td>
<td>APF</td>
<td>There is an APF section in the directory. (APFCODE is present.)</td>
</tr>
<tr>
<td>21</td>
<td>NOT-PO</td>
<td>This is a load module.</td>
<td>PGM OBJ</td>
<td>This is a program object. Always on for program object.</td>
</tr>
<tr>
<td>22</td>
<td>NOT-SIGN</td>
<td>Module is not digitally signed.</td>
<td>SIGN</td>
<td>Module is digitally signed.</td>
</tr>
<tr>
<td>23</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>24</td>
<td>ALTP</td>
<td>Alternate primary name.</td>
<td>NOT-ALTP</td>
<td>Not an alternate primary name.</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>26</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>27</td>
<td>RMODE24</td>
<td>Module must be loaded below 16 megabytes.</td>
<td>RMODEANY</td>
<td>Module can be loaded anywhere below 2 gigabytes.</td>
</tr>
<tr>
<td>28</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>29</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>30</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>31</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>32</td>
<td>NON-MIGR</td>
<td>Program object cannot be converted directly to PDS load module format.</td>
<td>MIGRATE</td>
<td>Program object can be converted to PDS load module format.</td>
</tr>
<tr>
<td>33</td>
<td>PRIME</td>
<td>FETCHOPT PRIME option.</td>
<td>NO-PRIME</td>
<td>FETCHOPT NOPRIME option.</td>
</tr>
<tr>
<td>34</td>
<td>PACK</td>
<td>FETCHOPT PACK option.</td>
<td>NO-PACK</td>
<td>FETCHOPT NOPACK option.</td>
</tr>
<tr>
<td>35</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
</tbody>
</table>
AMBLIST

Table 70. Program object and load module attributes (continued)

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>OFF Value</th>
<th>Meaning</th>
<th>ON Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>37</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>38</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>39</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
<td>RESERVED</td>
<td>Reserved for IBM use.</td>
</tr>
</tbody>
</table>

LISTOBJ outputs

Figure 182 shows sample output for LISTOBJ with an object module.

Figure 182. Example: Output for LISTOBJ with an object module

The record formats for OBJ and XOBJ records are identical except that XOBJ modules contain XSD records rather than ESD records. Except for XSD records, AMBLIST formats the records in the object module one record at a time. XSD records support names up to 32767 characters. These names may be continued onto multiple records, but such a continued record will appear as a single XSD record in the AMBLIST output. If the name is longer than 16 characters, a 16-character abbreviated name is printed with the XSD record. An abbreviation table which maps abbreviated names to be true names is printed at the end of the listing.
See the description of ESD data items in z/OS MVS Program Management: Advanced Facilities for a description of the format of OBJ and XOBJ record formats.

Figure 183 shows sample output for LISTOBJ with an XSD record.

**Figure 183. Example: LISTOBJ output with XSD Record**

```plaintext
<table>
<thead>
<tr>
<th>XSD RECORD:</th>
<th>MEMBER= CALLEDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDID TYPE</td>
<td>NAME</td>
</tr>
<tr>
<td>ADDR R/R/A</td>
<td>ID/LTH</td>
</tr>
<tr>
<td>TXT:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR=000000 ESDID= 0001 TEXT: 90EC0000 00C05000 C07241E0 C06E58E0 00881B80 1B115010 C0660700 4510C048 80000070</td>
<td></td>
</tr>
<tr>
<td></td>
<td>02250000 C310D3 C5C4C140 C1206E4</td>
</tr>
<tr>
<td>TXT:</td>
<td></td>
</tr>
<tr>
<td>ADDR=000038 ESDID= 0001 TEXT: E340D306 4095E5E3 4E90D5A0 E3D465C3 C10D3C5 09</td>
<td></td>
</tr>
<tr>
<td>TXT:</td>
<td></td>
</tr>
<tr>
<td>ADDR=00004E ESDID= 0001 TEXT: 0A234100 0081411E C0660A01 1BFF8D00 00845B8D 008C980D 001A0B0E 00000000 0000E7E7</td>
<td></td>
</tr>
<tr>
<td>RLD RECORD:</td>
<td>R PTR P PTR FLAGS ADDR R PTR P PTR FLAGS ADDR R PTR P PTR FLAGS ADDR</td>
</tr>
<tr>
<td></td>
<td>0001 0001 0001 0001 0001 0001 0001 0001 0001 0001</td>
</tr>
<tr>
<td>END RECORD:</td>
<td>1566896201 020191248</td>
</tr>
</tbody>
</table>
```

**Figure 184. Example: LISTOBJ output with GOFF Records**

```plaintext
<table>
<thead>
<tr>
<th>OBJECT MODULE LISTING</th>
<th>MEMBER= CALLEDA</th>
<th>PAGE 0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSD RECORD:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESDID TYPE 1</td>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>ADDR R/R/A 1</td>
<td>ID/LTH</td>
<td></td>
</tr>
<tr>
<td>TXT:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR=000000 ESDID= 0001 TEXT: 90EC0000 00C05000 C07241E0 C06E58E0 00881B80 1B115010 C0660700 4510C048 80000070</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>02250000 C310D3 C5C4C140 C1206E4</td>
<td></td>
</tr>
<tr>
<td>TXT:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR=000038 ESDID= 0001 TEXT: E340D306 4095E5E3 4E90D5A0 E3D465C3 C10D3C5 09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXT:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR=00004E ESDID= 0001 TEXT: 0A234100 0081411E C0660A01 1BFF8D00 00845B8D 008C980D 001A0B0E 00000000 0000E7E7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLD RECORD:</td>
<td>R PTR P PTR FLAGS ADDR R PTR P PTR FLAGS ADDR R PTR P PTR FLAGS ADDR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001 0001 0001 0001 0001 0001 0001 0001 0001 0001</td>
<td></td>
</tr>
<tr>
<td>END RECORD:</td>
<td>1566896201 020191248</td>
<td></td>
</tr>
</tbody>
</table>
```

Chapter 16. AMBLIST: Map load modules and program objects 595
Description of LISTOBJ output for GOFF

The GOFF object listing is similar in function and content to the LISTOBJ format for traditional object modules. The output is formatted one logical record at a time. A logical record represents the concatenation of the first physical record (which contains the record type) and all continuation records. If a name in a record is longer than 16 characters, a 16-character abbreviated name is printed. The true name can be found from the abbreviated name to long name table, which is printed at the end of the listing. The start of a logical record is highlighted by a dingbat (">" in the first position.

A record group consists of one or more records of the same type and is preceded by a two- or three-line record header. The first line of each record header consists of the record type and the sequence number of the first record in the group. Following a page break, the record group header will be repeated, even though the record type may not have changed.

Although the GOFF format currently defines only six record types, the TXT record type is subdivided into three different text types:

- TEXT, containing the instructions and data of the program
- IDRL, containing IDR information from the compiler or assembler
- ADATA, containing additional data associated with the object module

Altogether there are eight different display formats.

Report Description: The keyed sections of this description correspond to the equivalent keys highlighting the page header and the eight record formats in "Example: LISTOBJ format for GOFF." Note that some of the flags and lengths in the GOFF format are of a structural nature and do not represent the data content of the module. To save space, those elements have been omitted from the listing. For the same reason, unsupported data elements are not shown. A list of omitted elements is provided for each record type and the reason for omission is coded in parens following the field name. Code values are S (structural or syntactic data) and U (unsupported element). PTV for all record types is not formatted.

Example: LISTOBJ format for GOFF

```
1 LISTOBJ MEMBER=HELLOW,TITLE=('MY PROGRAM IN GOFF FORMAT')
1 **** GENERALIZED OBJECT FILE FORMAT ***** PAGE 1
MY PROGRAM IN GOFF FORMAT
OBJECT MODULE LISTING MEMBER= HELLOW

0RECORD TYPE: HDR SEQUENCE: 1
  ID NAME PRODUCT VERSION PROPERTIES
  0> 00000

0RECORD TYPE: ESD SEQUENCE: 2
  ESDID TYPE PARENT OFFSET LEN/ADA SP/S BA AMD RMD REUS AL TXT ORD STR BINDER_FLAGS LNK SIGNATURE
  0>00000 1 SD 000000 N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A
  000000 NAME() 000000
>
```

Chapter 16. AMBLIST: Map load modules and program objects
Display elements in “Example: LISTOBJ format for GOFF” on page 596 are described as follows. The numbers enclosed in braces following the field heading are the location (byte.bit) in the GOFF record where the data element can be found.

1. Page Header
   - The page header is printed at the top of each page.
   - The second line contains an optional user title.

2. HDR Record
   This is the first record in each GOFF module. The only data elements printed are the character set identifier and name and the language product (compiler or assembler) which produced the module.
   Data elements formatted:
   - CCSID
   - Character Set Name
   - Language Product Identifier
   Data elements not formatted:
   - Target Hardware Environment (U)
   - Target Operating System Environment (U)

3. ESD Record
   The ESD describes each external name defined or referenced in the module. Unlike the traditional object module, which provides for up to three names per record, the GOFF format contains only one name per record.
   Data elements formatted:
   - Line 1
     - ESDID. The identifier for the name being defined or referenced.
     - ESD TYPE. Symbol Type (SD, ED, LD, PR, ER)
     - OWNER/PARENT. The ESDID of the owning or referenced record type in the ESD hierarchy.
     - ITEM OFFSET. The offset, in bytes, of the start of this named entity from the start of the higher level entity.
     - ITEM LENGTH/ADA. For ED- or PR-type ESD records, the length (in bytes) of the entity being defined. If the length field is -1, the true length will be in a LEN-type GOFF record. For LD records, the ESDID of the associated data.
     - NAME SP/S. Name space (00-99) and binding scope (S (local or section), M (module), L (library), or X (import/export)).
     - BA. Binding algorithm (C=Catenate, M=Merge)
     - AMD. AMODE (N/S, 24, 31, 64, ANY, MIN)
     - RMD. RMODE (N/S, 24, 31, 64)
     - REUS. Reusability or tasking behavior. (N/S, NONE, REUS, RENT, REFRR)
     - AL. Alignment. Print as decimal value n, where alignment boundary is 2**n. Range: 0-31.
     - TXT. Text type. Displayed in format x-y where
       - x is text record style (B (Byte oriented = 0), F (Fixed = 1), or V (Variable = 2)).
• y is Executable (U (Unspecified= 0), D (Data=1), I (Instructions=2), or digits 3-7).
- STR. Binding strength. (S (Strong=0), W (Weak=1)).
- BINDER_FLAGS. Binder attributes is a string consisting of zero or more of the following characters. The ESD types to which the attribute is applicable are listed in parenthesis.
  • L. Initial or deferred load (ED)
  • M. Movable (ED)
  • R. Read-only (ED)
  • A. Addressable. Text may contain adcons. (ED)
  • C. Common (ED)
  • I. Symbol defines or references a descriptor. (LD, ER, PR)
  • G. Mangled name (LD, PR, ER)
  • N. Name may be renamed. (LD, PR, ER)
  • D. Deferred load (ED)
  • V. Removable (ED)
- LNK. Linkage Type (S (standard, non-XPLINK), X (XPLINK))
- SIGNATURE. Any eight-byte string, printed in hexadecimal.
- Sort Key. (Priority) Optional Field. PR only.
- Extended Attributes Optional Field. Defines text location containing additional attributes for this ESD.
- Symbol name. The first line begins with NAME, followed immediately by the name (up to 16 bytes). Names longer than 16 bytes will be abbreviated and displayed here, and an abbreviation-to-long name equivalence table will be listed at the end of the listing. A closing parenthesis will immediately follow the last byte. A name consisting of a single blank character will be displayed as "NAME( )".

Data elements not formatted:
- Extended Attribute ESDID (U)
- Extended Attribute Data Offset (U)
- Alias or Alternate Symbol ID (U)
- Name Length (S)

• TEXT Record
TEXT records are a subset of the TXT record type. They contain the instructions and data of the program. TEXT is displayed in hexadecimal format.

Data elements formatted:
- Line 1
  - ESDID. Identifies the element or part to which the text belongs.
  - OFFSET. The offset within the element or part where the text is to begin.
  - TRUE LENGTH. The expanded length of the text once the encoding rules (if any) have been applied.
  - TEXT ENCODING. The technique for encoding or decoding the text. Current® values are 0 and 1.
  - ENCODED LENGTH. The unexpanded length of the text appearing in this record.
  - TEXT. The text, displayed as it appears in the record. The length of the text to be displayed appears in the ENCODED LENGTH field. Text is displayed in hexadecimal format, 32 bytes per line.
- Lines 2-n
  All text beyond byte 31 is displayed on continuation lines. All bytes beyond the last text byte must be set to blank characters.

Data elements not formatted:
- Data Length (S)

5. IDRL Record
The IDRL provides identification information for the language translator which produced the GOFF. It is a subset of the TXT record type, identified as structured record data. In format 1, the IDRL records will be displayed in 19-byte segments, four per line. In format 3, IDRL records will be displayed in 30-byte segments to support four-digit year values and time stamps, two per line.

6. RLD Record
The relocation dictionary is a directory of address constants and other data areas which must be modified during binding and loading. Multiple such data areas or adcons can be described in a single RLD record. Relocation directory items begin at [8.0] in the RLD record and vary in length according to the presence or absence of various pointers and offsets in the item.
Directory items are formatted three per line. Each item consists of up to five fields. Flags in the first byte of each directory item indicate which fields are present in the item. As a result, except for the flag bytes in positions 1-8, offsets are not fixed within the directory item as it appears in the GOFF file.

Data elements formatted:
- R-PTR. The ESDID of the target element, the value which will be used in relocating the address constant.
- P-PTR. The ESDID of the element containing the adcon or data area to be modified.
- OFFSET. The offset within the element described by the P-PTR at which the adcon or data area is located.
- TYPE. This describes the type of adcon and implies the operation to be performed on it. Bytes 1 and 2 must be printed in hexadecimal.
- LEN. Length of the adcon or data area. Range: 2-255.
- ATTRIB.
  - H - the high order bit of the target field should be set from the target AMODE.
  - S - RLD is part of a conditional sequential RLD chain and the following RLD is also part of the chain.

Data elements not formatted:
- Total data length (S)
- Flag bytes 0 (except for 0.7) and 5 (S)
- Extended Attributes ESDID and offset (U)

7. END Record
The END record is the last record in the module. It contains a count of the records in the module and an optional entry point nomination, the latter specified by name or by class and offset.

Data elements formatted:
- Line 1
  - RECORD COUNT. Count of the logical records in the module, including the HDR and END records.
- AMODE. Amode to be used for the entry point specified on this END record.
- ENTRY POINT ESDID. The identifier of the element containing the entry point.
- ENTRY POINT OFFSET. The offset of the entry point within the element identified by ESDID.

- Lines 2 contain the symbol name, if specified. The display format is identical to that on the ESD record type.

LISTLOAD OUTPUT=MODLIST output

Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object

```
LISTLOAD MEMBER=ADATA3,OUTPUT=MODLIST,ADATA=YES

***** MODULE SUMMARY *****
MEMBER NAME: ADATA3
MAIN ENTRY POINT: 00000368
LIBRARY: SYSLIB
AMODE OF MAIN ENTRY POINT: ANY

NO ALIASES **
------------------------------------------------------------------------------------------------------------------------
**** ATTRIBUTES OF MODULE ****
** BIT STATUS BIT STATUS BIT STATUS BIT STATUS **
0 NOT-RENT 1 NOT-REUS 2 NOT-ONLY 3 NOT-TEST
4 NOT-OL 5 BLOCK 6 EXEC 7 MULTI-RCED
8 NOT-DC 9 ZERO-ORG 10 RESERVED 11 RLD
12 EDIT 13 NO-SYMS 14 RESERVED 15 NOT-NEFR
16 RESERVED 17 <16M 18 NOT-PL 19 NO-SSI
20 APF 21 PGM OBJ 22 NOT-SIGN 23 RESERVED
24 NOT-ALTP 25 RESERVED 26 RESERVED 27 NO-LEA
28 RESERVED 29 RESERVED 30 RESERVED 31 RESERVED
32 NON-MIGR 33 NO-PRIME 34 NO-PACK 35 RESERVED
36 RESERVED 37 RESERVED 38 RESERVED 39 RESERVED
------------------------------------------------------------------------------------------------------------------------
MODULE SSI: NONE
APFCODE: 00000000
RMODE: 24
PO FORMAT: 3
XPLINK: NO

***** PROGRAM OBJECT PROCESSED BY BINDER
***THE FOLLOWING ARE THE UNFORMATTED PDSE DIRECTORY ENTRY SECTIONS (PMAR AND PMARL)
PMAR 001E0309 02C00C03 00000000 05A40000 03680000 03680000 00000000 0000
PMARL 00628000 00000000 000A0000 10180000 10000000 27AC0000 74980000 00E00000
01400000 00240000 011C0000 000A0000 02200002 00000000 05300000 00180000
20082011 017F0131 240FC1C4 C1E3C1F3 40403000 00010000 005C0000 30000000
0410

LISTING OF PROGRAM OBJECT ADATA3 PAGE 1

THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695PMB01 AT LEVEL 01.12 ON 01/17/2011 AT 13:12:40
------------------------------------------------------------------------------------------------------------------------
MODULE SECTION: $SUMMARY
USABILITY: UNSPECIFIED AMODE: ANY OVERLAY SEGMENT: 0 OVERLAY REGION: 0

**** ESDs ******
C_WS(ED)
CLASS: C_WS
LENGTH: 5C (HEX)
NAME SPACE: 3
ALIGNMENT: DOUBLE WORD
BIND METHOD: MERGE
RMODE: ANY
TEXT: DEFER
FORMAT: F(00000)

C_@@DLLI(ED)
CLASS: C_@@DLLI
LENGTH: 8 (HEX)
NAME SPACE: 3
ALIGNMENT: DOUBLE WORD
BIND METHOD: MERGE
RMODE: ANY
TEXT: LOAD
FORMAT: F(00000)

C_@@PPA2(ED)
CLASS: C_@@PPA2
LENGTH: 8 (HEX)
NAME SPACE: 3
ALIGNMENT: DOUBLE WORD
BIND METHOD: MERGE
RMODE: ANY
TEXT: LOAD
FORMAT: F(00000)

B_PRV(ED)
CLASS: B_PRV
LENGTH: 4 (HEX)
NAME SPACE: 2
ALIGNMENT: DOUBLE WORD
BIND METHOD: MERGE
RMODE: 24
TEXT: NOLOAD
FORMAT: F(00000)

$PRV000001(PD)
CLASS: C_@@DLLI
LENGTH: 8 (HEX)
NAME SPACE: 3
ALIGNMENT: BYTE
PRIORITY: 0 (HEX)
SCOPE: LIBRARY

$PRV0000011(PD)
CLASS: C_@@PPA2
LENGTH: 8 (HEX)
NAME SPACE: 3
ALIGNMENT: BYTE
PRIORITY: 0 (HEX)
SCOPE: LIBRARY
```
### AMBLIST

```
$PRIV000002(PD)

--- Listing of Program Object ADATA3 Page 2 ---

#### Module Section: $SUMMARY

<table>
<thead>
<tr>
<th>TYPE</th>
<th>STATUS</th>
<th>LENG</th>
<th>ATTR</th>
<th>NAME SPACE</th>
<th>XATTR NAME</th>
<th>XATTR OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLDs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Text

<table>
<thead>
<tr>
<th>TYPE</th>
<th>LENG</th>
<th>ATTR</th>
<th>NAME SPACE</th>
<th>XATTR NAME</th>
<th>XATTR OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Control Section: A

```

Chapter 16. AMBLIST: Map load modules and program objects  603
Chapter 16. AMBLIST: Map load modules and program objects

AMBLIST
AMBLIST

ATTRIBUTES: STRONG,MAPPED

C_OPTIONS(ED)

ELEM.OFF CLS.OFF TYPE STATUS LENG ATTR NSPACE TARGET NAME PARTRES XATTR NAME XATTR OFF
00000000 00000000 N-BR RES 0004 1 (+)HELDWIF $PRIV000011

ELEM.OFF CLS.OFF TYPE STATUS LENG ATTR NSPACE TARGET NAME PARTRES XATTR NAME XATTR OFF
00000000 00000000 N-BR RES 0004 1 (+)HELDWIF $PRIV000010

ELEM.OFF CLS.OFF TYPE STATUS LENG ATTR NSPACE TARGET NAME PARTRES XATTR NAME XATTR OFF
00000000 00000000 N-BR RES 0004 1 (+)HELDWIF $PRIV000010

ELEM.OFF CLS.OFF TYPE STATUS LENG ATTR NSPACE TARGET NAME PARTRES XATTR NAME XATTR OFF
00000000 00000000 N-BR RES 0004 1 (+)HELDWIF $PRIV000010

ELEM.OFF CLS.OFF TYPE STATUS LENG ATTR NSPACE TARGET NAME PARTRES XATTR NAME XATTR OFF
00000000 00000000 N-BR RES 0004 1 (+)HELDWIF $PRIV000010

NAME SPACE: 1 SCOPE: MODULE ELEMENT OFFSET: 0 (HEX) AMODE: ANY
CLASS: B_LIT TEXT TYPE: DATA CLASS OFFSET: 0 (HEX) FORMAT: F(0001)

NAME SPACE: 1 SCOPE: MODULE ELEMENT OFFSET: 0 (HEX) AMODE: ANY
CLASS: B_LIT TEXT TYPE: DATA CLASS OFFSET: 0 (HEX) FORMAT: F(0001)

NAME SPACE: 1 SCOPE: MODULE ELEMENT OFFSET: 0 (HEX) AMODE: ANY
CLASS: B_LIT TEXT TYPE: DATA CLASS OFFSET: 0 (HEX) FORMAT: F(0001)

NAME SPACE: 1 SCOPE: MODULE ELEMENT OFFSET: 0 (HEX) AMODE: ANY
CLASS: B_LIT TEXT TYPE: DATA CLASS OFFSET: 0 (HEX) FORMAT: F(0001)

CLASS: B_LIT LENGTH: 120 (HEX) CLASS OFFSET: 0 (HEX) FORMAT: F(0001)

TEXT DATA NOLOAD READ-ONLY 0 (HEX)
NAME SPACE: 1 ALIGNMENT: DOUBLE WORD BIND METHOD: CATENATE RMODE: ANY
ATTRIBUTES: STRONG,MAPPED

CONTROL SECTION: IEWBLIT

TEXT DATA NOLOAD READ-ONLY 0 (HEX)
NAME SPACE: 1 ALIGNMENT: DOUBLE WORD BIND METHOD: CATENATE RMODE: ANY
ATTRIBUTES: STRONG,MAPPED

TEXT DATA NOLOAD READ-ONLY 0 (HEX)
NAME SPACE: 1 ALIGNMENT: DOUBLE WORD BIND METHOD: CATENATE RMODE: ANY
ATTRIBUTES: STRONG,MAPPED

TEXT DATA NOLOAD READ-ONLY 0 (HEX)
NAME SPACE: 1 ALIGNMENT: DOUBLE WORD BIND METHOD: CATENATE RMODE: ANY
ATTRIBUTES: STRONG,MAPPED
Example: Output for LISTLOAD OUTPUT=MODLIST for an overlay structured load module

** Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object ** on page 602 is an example of the output produced by LISTLOAD OUTPUT=MODLIST for an overlay structured load module.

### LISTING OF LOAD MODULE PL1LOAD PAGE 0001

<table>
<thead>
<tr>
<th>RECORD#</th>
<th>TYPE</th>
<th>ESDID</th>
<th>ESD SIZE</th>
<th>CESD#</th>
<th>SYMBOL</th>
<th>TYPE</th>
<th>ADDRESS</th>
<th>SEGNUM</th>
<th>ID/LENGTH(DEC)</th>
<th>(HEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>CESD 1</td>
<td>240</td>
<td>1</td>
<td>PLITC0Z</td>
<td>00(SD)</td>
<td>000000</td>
<td>1</td>
<td>1206 486</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>CESD 2</td>
<td>240</td>
<td>2</td>
<td>PLITC0Z</td>
<td>00(SD)</td>
<td>000488</td>
<td>1</td>
<td>608 260</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>CESD 3</td>
<td>240</td>
<td>3</td>
<td>IHEQNX</td>
<td>06(PR)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>CESD 4</td>
<td>240</td>
<td>4</td>
<td>IHEQADA</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>CESD 5</td>
<td>240</td>
<td>5</td>
<td>IHEQAD0</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>CESD 6</td>
<td>240</td>
<td>6</td>
<td>IHEQGR</td>
<td>06(PR)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>CESD 7</td>
<td>240</td>
<td>7</td>
<td>IHEQCIC</td>
<td>06(PR)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>CESD 8</td>
<td>240</td>
<td>8</td>
<td>IDEMA</td>
<td>00(SD)</td>
<td>000000</td>
<td>1</td>
<td>12 C</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>CESD 9</td>
<td>240</td>
<td>9</td>
<td>IHEQAPC</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>CESD 10</td>
<td>240</td>
<td>10</td>
<td>IDELQA</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>CESD 11</td>
<td>240</td>
<td>11</td>
<td>IDELQB</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>CESD 12</td>
<td>240</td>
<td>12</td>
<td>IDEQLA</td>
<td>06(PR)</td>
<td>000010</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>CESD 13</td>
<td>240</td>
<td>13</td>
<td>IDEQMA</td>
<td>06(PR)</td>
<td>000014</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>CESD 14</td>
<td>240</td>
<td>14</td>
<td>PLITC0Z</td>
<td>06(PR)</td>
<td>000018</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>CESD 15</td>
<td>240</td>
<td>15</td>
<td>PLITC0Z</td>
<td>06(PR)</td>
<td>00001C</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
</tbody>
</table>

** Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object ** on page 602 is an example of the output produced by LISTLOAD OUTPUT=MODLIST for an overlay structured load module.

### LISTING OF LOAD MODULE PL1LOAD PAGE 0002

<table>
<thead>
<tr>
<th>RECORD#</th>
<th>TYPE</th>
<th>ESDID</th>
<th>ESD SIZE</th>
<th>CESD#</th>
<th>SYMBOL</th>
<th>TYPE</th>
<th>ADDRESS</th>
<th>SEGNUM</th>
<th>ID/LENGTH(DEC)</th>
<th>(HEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td>CESD 16</td>
<td>64</td>
<td>16</td>
<td>IHEQPA</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>CESD 17</td>
<td>64</td>
<td>17</td>
<td>IHEQPB</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>CESD 18</td>
<td>64</td>
<td>18</td>
<td>IHEQPC</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>CESD 19</td>
<td>64</td>
<td>19</td>
<td>IHEQPD</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>CESD 20</td>
<td>64</td>
<td>20</td>
<td>IHEQPE</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>CESD 21</td>
<td>64</td>
<td>21</td>
<td>IHEQPF</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>CESD 22</td>
<td>64</td>
<td>22</td>
<td>IA0A</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>CESD 23</td>
<td>64</td>
<td>23</td>
<td>I 00(SD)</td>
<td>000730</td>
<td>1</td>
<td>4 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>CESD 24</td>
<td>64</td>
<td>24</td>
<td>I 00(SD)</td>
<td>000738</td>
<td>1</td>
<td>4 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>CESD 25</td>
<td>64</td>
<td>25</td>
<td>I 00(SD)</td>
<td>000740</td>
<td>1</td>
<td>4 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>CESD 26</td>
<td>64</td>
<td>26</td>
<td>I 00(SD)</td>
<td>000748</td>
<td>1</td>
<td>4 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>CESD 27</td>
<td>64</td>
<td>27</td>
<td>IHDSR</td>
<td>06(PR)</td>
<td>000020</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>CESD 28</td>
<td>64</td>
<td>28</td>
<td>IHDSMC</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>CESD 29</td>
<td>64</td>
<td>29</td>
<td>IHDSMP</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>CESD 30</td>
<td>64</td>
<td>30</td>
<td>IHEDA</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
</tbody>
</table>

** Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object ** on page 602 is an example of the output produced by LISTLOAD OUTPUT=MODLIST for an overlay structured load module.

### LISTING OF LOAD MODULE PL1LOAD PAGE 0003

<table>
<thead>
<tr>
<th>RECORD#</th>
<th>TYPE</th>
<th>ESDID</th>
<th>ESD SIZE</th>
<th>CESD#</th>
<th>SYMBOL</th>
<th>TYPE</th>
<th>ADDRESS</th>
<th>SEGNUM</th>
<th>ID/LENGTH(DEC)</th>
<th>(HEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
<td>CESD 31</td>
<td>64</td>
<td>31</td>
<td>IHEQPB</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>CESD 32</td>
<td>64</td>
<td>32</td>
<td>IHEQPC</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>CESD 33</td>
<td>64</td>
<td>33</td>
<td>IHEQPD</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>CESD 34</td>
<td>64</td>
<td>34</td>
<td>IHEQPE</td>
<td>02(ER)</td>
<td>000000</td>
<td>3</td>
<td>4 4</td>
<td></td>
</tr>
</tbody>
</table>

** Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object ** on page 602 is an example of the output produced by LISTLOAD OUTPUT=MODLIST for an overlay structured load module.
"Example: Output for LISTLOAD OUTPUT=MODLIST for a normal (non-overlay) structured load module" is an example of the output produced by LISTLOAD OUTPUT=MODLIST for a normal (non-overlay) structured load module.

Example: Output for LISTLOAD OUTPUT=MODLIST for a normal (non-overlay) structured load module

****LOAD MODULE PROCESSED EITHER BY VS LINKAGE EDITOR OR BINDER
LISTING OF LOAD MODULE MYMOD PAGE 0001

<table>
<thead>
<tr>
<th>RECORD#</th>
<th>TYPE 20 - CESD</th>
<th>ESDID 1</th>
<th>ESD SIZE 144</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CESD# SYMBOL</td>
<td>ADDRESS</td>
<td>R/R/A ID/LENGTH(DEC)</td>
</tr>
<tr>
<td>1</td>
<td>A 00(SD)</td>
<td>000000</td>
<td>00 24 18</td>
</tr>
<tr>
<td>2</td>
<td>ENTA 03(LR)</td>
<td>000014</td>
<td>00 1 1</td>
</tr>
<tr>
<td>3</td>
<td>ENTB 03(LR)</td>
<td>000028</td>
<td>00 5 5</td>
</tr>
<tr>
<td>4</td>
<td>Q1 06(PR)</td>
<td>000014</td>
<td>03 4 4</td>
</tr>
<tr>
<td>5</td>
<td>B 00(SD)</td>
<td>000018</td>
<td>00 20 14</td>
</tr>
<tr>
<td>6</td>
<td>UNRES 02(ER)</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>DANGLE 02(ER)</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$NULL 07(NULL)000000</td>
<td></td>
<td>0 0</td>
</tr>
<tr>
<td>9</td>
<td>$NULL 07(NULL)000000</td>
<td></td>
<td>0 0</td>
</tr>
</tbody>
</table>

"Example: Output for LISTLOAD OUTPUT=MODLIST for a PDSE (program object Version 1)" is an example of the output produced by LISTLOAD OUTPUT=MODLIST for a PDSE (program object Version 1).

Example: Output for LISTLOAD OUTPUT=MODLIST for a PDSE (program object Version 1)

THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695DF108 AT LEVEL 01.00 ON 09/16/92 AT 09:42:47

---------- CONTROL SECTION: A ----------

AMODE: 24 ALIGMENT: DOUBLE WORD LENGTH: 24 (DEC) MODULE OFFSET: 0 (DEC)
RMODE: 24 USABILITY: UNSPECIFIED LENGTH: 18 (HEX) MODULE OFFSET: 0 (HEX)
STORAGE: ANY OVERLAY SEGMENT: 0 OVERLAY REGION: 0

----- IDRL -----

TRANSLATOR VER MOD DATE TIME
5668962801 02 01 09/16/92

----- ESDs -----

ALPHA(PR)  ALIGNMENT: FULL WORD LENGTH: 0000004

----- RLDS -----

SEC.OFF MOD.OFF TYPE BUY STATUS REFERENCED SYMBOL
00000004 00000004 CPR NONE RES $CUMULATIVE PSEUDO REGISTER LENGTH
00000008 00000008 PR NONE RES (+)ALPHA
00000010 00000010 CPR NONE RES $CUMULATIVE PSEUDO REGISTER LENGTH
00000014 00000014 CPR NONE RES $CUMULATIVE PSEUDO REGISTER LENGTH

----- TEXT -----

00000000 CICICICI 00000014 00000000 00000000 00000014 00000014

LISTING OF PROGRAM OBJECT TESTPR PAGE 2

PSEUDO REGISTER
VECTOR LOC LENGTH NAME
Example: Output for LISTLOAD OUTPUT=MODLIST,SECTION1=YES for a program object

Example: Output for LISTLOAD OUTPUT=MODLIST,SECTION1=YES for a program object

Example: Output for LISTLOAD OUTPUT=MODLIST,IMPEXP=YES for a program object

Example: Output for LISTLOAD OUTPUT=MODLIST,IMPEXP=YES for a program object
Description of MODLIST output for a program object

The listing produced by LISTLOAD OUTPUT=MODLIST consists of multiple parts (see "Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object" on page 602):

A page heading, displayed at the top of each page.

The page heading consists of one or two heading lines, in the following format:

    LISTING OF PROGRAM OBJECT xxxxxxxx

The heading lines are followed by the title line, entered in the TITLE parameter of the LISTLOAD control statement.

The binder-generated program object identification record (IDRB).

The IDRB record is displayed on a line by itself, in the format:

    THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695PMB01 AT LEVEL 01.12
    ON mm/dd/yyyy AT hh:mm:ss

The binder program identifier, version and level, date and time of binding are presented here. The IDRB line is followed by a line of dashes.

An individual listing for each control section in the program object, separated by a dashed line.

For each control section in the module, the ESD Section Definition (SD) record is formatted, followed by all data classes in the following sequence:

1. IDRZ - SPZAP identification data
2. IDRL - Language translator identification data
3. IDRU - User-supplied identification data
4. SYM - internal symbol dictionary
5. ESD - External Symbol Dictionary
6. RLD - ReLocation Dictionary
7. TEXT - Instructions and data for the CSECT
8. ADATA - ADATA information

The SD record occupies two print lines:

• The first begins with one of the constants CONTROL SECTION, SEGMENT TABLE, ENTRY TABLE, or MODULE SECTION, and displays either the section or common name. If there is no user-defined name, then a binder-generated name will be displayed as follows:
  – $PRIVxxxxx, where x is a number for user sections which originally had blank names or were unnamed.
  – $BLANKCOM - unnamed common
  – $MODULE - binder-generated section containing module level information, and is only output when SECTION1=YES
  – $SUMMARY - binder-generated section containing merge classes for the module
• The second line displays the USABILITY, AMODE, the overlay segment and region. USABILITY must contain one of the values UNSPECIFIED, NON-REUSABLE, REUSABLE, REENTRANT or REFRESHABLE. For non-overlay modules, the latter two fields will contain zero.

Each of the eight class subsections begin with an identifier line of the format:

===== class name =====
IDR detail is in the same format as described in “Description of LISTIDR output” on page 631, except that it is displayed only for the single section. The remainder of the classes are described below:

- SYM data is displayed 40 bytes per line
- ESD data occupies three to four lines per ESD record. The first containing the external name (abbreviated name, if name longer than 16 bytes), followed by the ESD record type in parenthesis. The rest of the formatted fields vary depending on the ESD record type:
  - ED records define an element definition. Its length, and various attributes will be used to bind and load the class contained in the section. Each ED record occupies three print lines:
    1. The first line displays:
       - CLASS name - up to 16 bytes.
       - LENGTH of defined class element in hexadecimal.
       - CLASS OFFSET - in hexadecimal.
       - FORMAT - where the first field is the class record format with either F (fixed length record), or V (variable length record), followed by the hexadecimal value of the record format in parenthesis.
    2. The second line shows:
       - NAME SPACE - in hexadecimal.
       - ALIGNMENT - any of the following:
         - DOUBLE WORD
         - QUAD WORD
         - 32 BYTE
         - 64 BYTE
         - 128 BYTE
         - 256 BYTE
         - 512 BYTE
         - 1024 BYTE
         - 2048 BYTE
         - PAGE
       DOUBLE WORD, QUAD WORD, or PAGE.
       - BIND METHOD - CATENATE or MERGE.
       - RMODE - 24, 64, ANY, or UNSPECIFIED
    3. The third line displays the binder and loader attributes:
       - Binder attributes can be DESCRIPTIVE DATA, TEXT, or REMOVABLE
       - Loader attributes can be NOLOAD, LOAD, DEFER, READ-ONLY (or FILL: UNSPEC is printed, if there is no fill character).
  - ER records define external references from the named section. For the $MODULE section these are external references with no corresponding RLDs. Each ER record occupies four print lines, where:
    1. The first line displays:
       - TEXT TYPE - can be either UNSPEC (unspecified), INSTRUC (instructions or code), DATA, or TRANS.DEF (translator defined).
    2. The second line shows:
Chapter 16. AMBLIST: Map load modules and program objects

- TARGET SECTION - Target section name (abbreviated name, if name is longer than 16 bytes).
- TARGET CLASS - Name of class containing target label.

3. The third line shows:
   - NAME SPACE - in hexadecimal.
   - SCOPE - Scope of name (SECTION, MODULE, LIBRARY, or IMP/EXP).
   - ELEMENT OFFSET - in hexadecimal.
   - AMODE - 24, 31, 64, ANY, MIN or UNSPEC

4. The fourth line displays the ER status and autocall, where:
   - ER status can either be RESOLVED or UNRESOLVED
   - AUTOCALL can either be AUTOCALL or NEVERCALL

5. The fifth line displays binder attributes. Possible attributes are XPL, STRONG or WEAK, MAPPED, INDIRECT, GENERATED, or MANGLED.

6. If extended attributes exist, a sixth line displays the location of these attributes as the class and offset within that class.

- LD records define a label or entry point in the named section. Each LD record occupies three print lines, where:

1. The first line displays:
   - CLASS NAME - up to 16 bytes.
   - TEXT TYPE - can be either UNSPEC, INSTR, DATA, or TRANS.DEF
   - CLASS OFFSET - in hexadecimal.

2. The second line shows:
   - NAME SPACE - in hexadecimal.
   - SCOPE - Scope of name (SECTION, MODULE, LIBRARY, or IMP/EXP).
   - ELEMENT OFFSET - in hexadecimal.
   - AMODE - can either be 24, 31, 64, MIN, ANY, or UNSPECIFIED

3. The third line displays the binder attributes. Possible attributes are XPL (xplink), STRONG or WEAK, MAPPED, INDIRECT, GENERATED (the LD record was generated by the binder), or MANGLED. In addition, if the record contains the name of the symbol which defines the environment or associated data (ADA), that symbol will be printed.

4. If extended attributes exist, a fourth line will contain the resident class and offset.

- PD (Part Definition) and PR (Part Reference) records define parts or pseudo registers. The PR record is a local definition of the part (within the section), whereas the PD record is a global definition for all of the associated PRs (PRs with the same name). PR and PD records contain the same formatted fields. Each record occupies three print lines, where:

1. The first line displays:
   - CLASS name - up to 16 bytes.
   - LENGTH - in hexadecimal.
   - CLASS OFFSET - in hexadecimal.

2. The second line shows:
   - NAME SPACE - in hexadecimal.
- ALIGNMENT - any of: the following:
  - BYTE
  - HALF WORD
  - FULL WORD
  - DOUBLE WORD
  - QUAD WORD
  - 32 BYTE
  - 64 BYTE
  - 128 BYTE
  - 256 BYTE
  - 512 BYTE
  - 1024 BYTE
  - 2048 BYTE
  - PAGE

- PRIORITY - Controls the order of the part within the element.
- SCOPE - Scope of part (SECTION, MODULE, LIBRARY, or IMP/EXP).

3. The third line displays binder attributes. Possible attributes are: XPL, STRONG or WEAK, MAPPED, INDIRECT, GENERATED, or MANGLED.

- RLD data is displayed one line per RLD record, by element offset of the associated address constant. Where multiple RLD records refer to the same adcon, the element offsets will be the same. RLD data shown consists of:
  - Element Offset - The offset, in hex, of the associated address constant within the element.
  - Class Offset - The offset, in hex, of the associated address constant within the class.
  - ADCON TYPE - The type of address constant associated with this RLD entry. The type may be:
    - BR - Branch or V-type
    - N_BR - Non-branch or A-type
    - SEGM - address of start of class
    - C OF - Q-type, offset from start of class
    - CPR - total length of the class
    - LTKN - loader token, for use of the system loader
    - R-IM - reference to an external symbol from a relative immediate instruction
    - DATA - data associated with the target symbol
    - LDIS - QY-type, reference to an external symbol from a long displacement instruction
    - L-PR - length of the individual PR
  - Status - This field identifies the status of the associated address constant. Valid status values are: RES (resolved), UNRES (unresolved), and N-REL (non-relocatable constant).
  - LENGTH - The adcon length in hexadecimal.
  - ATTRIBUTES, which may be:
- **HOBCHG** - High order bit of V-type adcon was changed by the binder. Possible value can be either YES or NO.
- **X** - XPLINK linkage
- **S** - part of a conditional sequential RLD string, will be followed by another RLD describing an adcon at the same location.
- **NAME SPACE** of reference in hexadecimal.
- **TARGET NAME** - Name of the referenced symbol.
- **PARTRES** - If the RLD describes an adcon on a part (PR), this will be the name of the resident part.
- **XATTR NAME** - Symbol defining the location at which the extended attributes (if any) are stored.
- **XATTR OFF** - Offset from symbol XATTR NAME at which the extended attributes are stored.

- **TEXT** data is displayed by class name. In addition to the hexadecimal representation, text data is in EBCDIC format.
  - For the **SMODULE** section, instead of **TEXT**, it is identified as **MAP** and shows the binder-generated class B_MAP information.
  - For the **IEWBCIE** section, when **IMPEXP=SYMBOLS**, instead of **TEXT**, it is identified as **IEWBCIET VERS nn** (where nn is the import/export table version number) and shows information about the imported and exported symbols. Columns with the heading +OFF/ADDR distinguish offsets from addresses by preceding offsets with a plus sign (+). They are always preceded by their corresponding class. However, when an address is shown, it is relative to the module beginning at a zero origin, not relative to the beginning of the class. To determine the address within the class (as shown in the NUMERICAL MAP), subtract off the location of the class from the address (the location of the class is available in the SEGMENT MAP TABLE).
  - The attributes shown as follows:
    - **O** or **X**
      - referred symbol is OS linkage or XPLINK linkage
    - **N** or **P**
      - reference is by name or reference is by pointer
    - **R** or **U**
      - reference is Resolved or Unresolved

- **ADATA** information, if requested through **ADATA=YES** on the **LISTLOAD OUTPUT=MODLIST** control card, like **TEXT** data is displayed by class name in both hexadecimal and EBCDIC presentation. In the EBCDIC presentation, non-printable characters are replaced with periods (.)
  - The abbreviation-to-long name equivalence table is displayed prior to end of the listing, with all the abbreviated names (external names exceeding 16 bytes in length) in the formatted part of the listing with their long names.

---

**A trailer record.**

**** END OF PROGRAM OBJECT LISTING

**Description of MODLIST Output for a Load module/PDS**

The listing produced by **LISTLOAD OUTPUT=MODLIST** consists of multiple parts (see "Example: Output for LISTLOAD OUTPUT=MODLIST for an overlay structured load module" on page 607 and "Example: Output for LISTLOAD OUTPUT=MODLIST for a normal (non-overlay) structured load module" on page 609).
A page heading, displayed at the top of each page.

The page heading consists of one or two heading lines, in the following format:

LISTING OF LOAD MODULE PL1LOAD

An individual listing for each record in the load module. Each record on DASD is identified by a sequence number (RECORD#), and type. Often there is size information for the record as well. This information is followed by its contents formatted according to the record type.

For more information on the details of the following fields, see z/OS MVS Program Management: Advanced Facilities Appendix B. Load Module formats.

TYPE 20 - CESD

This is a record containing definitions or uses of external symbols.

- CESD# - An internal number for the symbol assigned by the binder.
- SYMBOL - The name of the symbol as used in the program, or an abbreviation of it.
- TYPE - See chapter 2 of z/OS MVS Program Management: User’s Guide and Reference for a description of the various types of external symbol dictionary entries.
  - ER - External Reference
  - WX - Weak External reference
  - LR - Label Reference
  - SD - Section Definition, a control section (CSECT)
  - PC - Private Code
  - CM - Common area
  - PR - Pseudoregister
- ADDRESS - Offset within the bound program where the symbol is defined.
- SEGNUM - Overlay structured modules only: the overlay segment number.
- R/R/A - Non-overlay modules only: A bit-coded byte, displayed in hex, indicating AMODE, RMODE, and read-only attributes. The values of the three attributes are ORed together in the value displayed.
  - 08 - Read-only
  - 04 - RMODE=ANY (RMODE=24 if off)
  - 01 - AMODE=24
  - 02 - AMODE=31
  - 03 - AMODE=ANY
  - 10 - AMODE=64
- ID/LENGTH - ESD number of referenced symbol for an LR, length of the section or field for SD, PC, CM, or PR
  - (DEC) - Length or ID displayed in decimal
  - (HEX) - Length or ID displayed in hexadecimal

TYPE 0 - RLD (May be 02, 06, or 0E)

In the RLD record type, AMBLIST outputs information in the following columns:

- R-PTR - The ESDID of the target element
- P-PTR - The ESDID of the element containing the adcon or data area to be modified
- FL - flags
AMBLIST

- ADDR - address

TYPE 0 - CONTROL (May be 01, 05, or 0D)
The CONTROL record lists information on each of the control sections for the
load module:
- CESD# - CESD number of the control section
- LENGTH - length of that control section

TEXT
This record type contains the program as loaded for execution. Text data is
displayed by class name. In addition to the hexadecimal representation, text
data is in EBCDIC format.

A trailer record.

******END OF LOAD MODULE LISTING

LISTLOAD OUTPUT=XREF output
This section includes examples of LISTLOAD OUTPUT=XREF output as well as
cross-reference listings, such as:
- "Example: Output for LISTLOAD OUTPUT=XREF for a program object with
class names: B_PRV and B_TEXT" on page 618 shows the output from a
program object version 2 that contains a single initial load text class. See the
descriptions following this figure for explanations.
- Figure 185 on page 623 shows a sample segment map table for a multiple-text
class module.
- "Example: Output for LISTLOAD OUTPUT=XREF for a load module" on page
623 and "Example: Output for LISTLOAD OUTPUT=XREF for a program object"
on page 624 allow you to compare the output for a load module with the output
for a program object version 1.

The listing has the following parts:
- Segment map, which only appears for Program Objects and not for Load
Modules.
- Numerical map, which presents information approximately in the order in which
it appears in the module.
- Numerical cross-reference.
- Alphabetical map, which presents information alphabetically by symbol name.
- Alphabetical cross-reference.

In z/OS V1R10, the format of the LISTLOAD numerical and alphabetical
cross-references for program objects was significantly changed. The information is
now presented in a more concise format that more closely resembles the format
produced by the binder.

In the listing shown in "Example: Output for LISTLOAD OUTPUT=XREF for a
program object with class names: B_PRV and B_TEXT" on page 618, page 1 shows
the numerical map of the module. Page 2 shows the numerical cross-reference list
of the module. Page 4 shows the alphabetical map, and page 5 the alphabetical
cross-reference list.

Note: The module shown in the "Example: Output for LISTLOAD OUTPUT=XREF
for a program object with class names: B_PRV and B_TEXT" on page 618 is not in
overlay format; for overlay modules, each segment is formatted separately.
As with the other output from AMBLIST, each page begins with a standard heading. The first line of each page contains a page number and begins with one of the following heading constants:

- SEGMENT MAP OF PROGRAM OBJECT ....
- NUMERICAL MAP OF PROGRAM OBJECT ....
- NUMERICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT ....
- ALPHABETICAL MAP OF PROGRAM OBJECT ....
- ALPHABETICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT ....

The member name will appear following the heading. If the name is more than sixteen characters, its formatted 16-bytes abbreviation name is printed instead. An optional second line will be used to print the title information, provided by the user on the LISTLOAD control statement. Each of the four parts has its own subheading line(s), to describe the detail that follows.

Example: Output for LISTLOAD OUTPUT=XREF for a program object with class names: B_PRV and B_TEXT

```
** SEGMENT MAP TABLE ** PAGE 1

<table>
<thead>
<tr>
<th>CLASS</th>
<th>SEGMENT</th>
<th>OFFSET</th>
<th>LENGTH</th>
<th>LOAD</th>
<th>TYPE</th>
<th>ALIGNMENT</th>
<th>RMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>B_TEXT</td>
<td>1</td>
<td>0</td>
<td>2030</td>
<td>INITIAL</td>
<td>CAT</td>
<td>PAGE</td>
<td>31</td>
</tr>
</tbody>
</table>

** NUMERICAL MAP OF PROGRAM OBJECT LOADMOD1 ** PAGE 2

<table>
<thead>
<tr>
<th>CLAS LOC</th>
<th>ELEM LOC</th>
<th>LENGTH</th>
<th>TYPE</th>
<th>ALIGNMENT</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>430</td>
<td></td>
<td>430</td>
<td>ED</td>
<td>21</td>
<td>PAGE</td>
</tr>
<tr>
<td>238</td>
<td></td>
<td>238</td>
<td>ED</td>
<td>21</td>
<td>PAGE</td>
</tr>
<tr>
<td>668</td>
<td></td>
<td>668</td>
<td>ED</td>
<td>21</td>
<td>DOUBLE WORD</td>
</tr>
<tr>
<td>66C</td>
<td></td>
<td>66C</td>
<td>LD</td>
<td>21</td>
<td>PAGE</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>1000</td>
<td>ED</td>
<td>31</td>
<td>PAGE</td>
</tr>
</tbody>
</table>

** NUMERICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT LOADMOD1 ** PAGE 3

<table>
<thead>
<tr>
<th>CLAS LOC</th>
<th>ELEM LOC</th>
<th>FROM CLASS</th>
<th>FROM SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48 B_TEXT</td>
<td>SD1</td>
<td>SD1</td>
</tr>
<tr>
<td>12E</td>
<td>12E B_TEXT</td>
<td>SD1</td>
<td>B</td>
</tr>
<tr>
<td>196</td>
<td>196 B_TEXT</td>
<td>SD1</td>
<td>C</td>
</tr>
<tr>
<td>1FE</td>
<td>1FE B_TEXT</td>
<td>SD1</td>
<td>D</td>
</tr>
<tr>
<td>266</td>
<td>266 B_TEXT</td>
<td>SD1</td>
<td>E</td>
</tr>
<tr>
<td>2CE</td>
<td>2CE B_TEXT</td>
<td>SD1</td>
<td>F</td>
</tr>
<tr>
<td>336</td>
<td>336 B_TEXT</td>
<td>SD1</td>
<td>G</td>
</tr>
<tr>
<td>39C</td>
<td>39C B_TEXT</td>
<td>SD1</td>
<td>$CLASS_LEN</td>
</tr>
<tr>
<td>47B</td>
<td>47B B_TEXT</td>
<td>SD2</td>
<td>430</td>
</tr>
<tr>
<td>554</td>
<td>554 B_TEXT</td>
<td>SD2</td>
<td>CM1</td>
</tr>
<tr>
<td>604</td>
<td>604 B_TEXT</td>
<td>SD2</td>
<td>CM1</td>
</tr>
</tbody>
</table>

** ALPHABETICAL MAP OF PROGRAM OBJECT LOADMOD1 ** PAGE 5

<table>
<thead>
<tr>
<th>CLAS LOC</th>
<th>ELEM LOC</th>
<th>FROM CLASS</th>
<th>FROM SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48 B_TEXT</td>
<td>SD1</td>
<td>SD1</td>
</tr>
<tr>
<td>12E</td>
<td>12E B_TEXT</td>
<td>SD1</td>
<td>B</td>
</tr>
<tr>
<td>196</td>
<td>196 B_TEXT</td>
<td>SD1</td>
<td>C</td>
</tr>
<tr>
<td>1FE</td>
<td>1FE B_TEXT</td>
<td>SD1</td>
<td>D</td>
</tr>
<tr>
<td>266</td>
<td>266 B_TEXT</td>
<td>SD1</td>
<td>E</td>
</tr>
<tr>
<td>2CE</td>
<td>2CE B_TEXT</td>
<td>SD1</td>
<td>F</td>
</tr>
<tr>
<td>336</td>
<td>336 B_TEXT</td>
<td>SD1</td>
<td>G</td>
</tr>
<tr>
<td>39C</td>
<td>39C B_TEXT</td>
<td>SD1</td>
<td>$CLASS_LEN</td>
</tr>
<tr>
<td>47B</td>
<td>47B B_TEXT</td>
<td>SD2</td>
<td>430</td>
</tr>
<tr>
<td>554</td>
<td>554 B_TEXT</td>
<td>SD2</td>
<td>CM1</td>
</tr>
<tr>
<td>604</td>
<td>604 B_TEXT</td>
<td>SD2</td>
<td>CM1</td>
</tr>
</tbody>
</table>
```

LENGTH OF PROGRAM OBJECT 2030
### Segment map

When the binder produces a program object executable, it is composed of one or more segments. The binder will always arrange the program so that segment 1 contains the entry point. To understand the importance of the information contained in the Segment Map Table, see [Understanding binder programming concepts in z/OS MVS Program Management: Advanced Facilities](https://www.ibm.com/support/docview.wss?uid=swg21233783) in particular Multiple-text class modules, load segments and ESD offsets. For an example of a Segment map for a multiple-text class module, see “Sample segment map table for LISTLOAD OUTPUT=XREF of mutiple-text class module” on page 623.

Each of the columns in a Segment map table is described as follows:

**CLASS**

The class name of this part of the segment. The segment will be composed of one or more classes. Note that a section may contribute to more than one class.
SEGMENT
The segment in which this class resides. A given class will reside in only one segment, and all classes in the same segment will have the same attributes.

OFFSET
The offset the beginning of the named class from the beginning of the segment. Note that every segment has its own origin (offset 0), because each may be independently loaded.

LENGTH
The length of the named class in the segment.

LOAD
The loading time attribute of the class, which will be the same for all classes with the same segment number. The possible values are:

INITIAL
An initial load class which is loaded when the program entry point or alias is loaded.

DEFER
A deferred load class which is not loaded when the program entry point or alias is loaded.

TYPE
The binding type of class. The possible values are:

CAT
A concatenated class. This is the usual way in which a section contributes to a class (known as elements). The elements are arranged one after the other.

MERGE
A "merge" class. This type of class is used for parts, which are for data, where one or more part comprises an element.

ALIGNMENT
The alignment of the class or elements contained in the class. The possible values are:
- BYTE
- HALF WORD
- FULL WORD
- DOUBLE WORD
- QUAD WORD
- 32 BYTE
- 64 BYTE
- 128 BYTE
- 256 BYTE
- 512 BYTE
- 1024 BYTE
- 2048 BYTE
- PAGE

RMODE
The residency mode of the class and the segment. If only one value is listed, it is the residency mode of both the class and segment. If a second value is given, the first value is the residency mode of the class and the second value is the residency mode of the segment (for example, 24,31 would indicate that a class with the attribute RMODE=24 is part of a...
segment that has RMODE=31). The segment is loaded according to the residency mode of the segment. The possible values are: 24, 31, and 64.

Numerical map
The Numerical Map prints one line for each defined element definition, part, or control section in the composite ESD. The detail line contains the class offset (in hex), either a section offset for (labels/parts) or a length (for control sections element definitions), the ESD record type, alignment (for LD/PD record type), and the label/part, section or element definition name. Sections generated by the binder, or binder-generated names for unnamed user sections, will be displayed as:

• $PRIVxxxxx - where xxxxx is numeric.
• $BLANKCOM
• $SEGTAB
• $ENTAB

All other entries will contain a valid name, assigned by the user for a label/part, a control section/element definition or named common. For label (LD) or part (PD) type ESD entries, the class offset and label/part name will be indented to show that the label/part is contained within the previous section entry.

If the module is in overlay format, the map and cross-reference will alternate for each segment. In this case, the map will begin with a segment identifier line:

```
SEG. nnnnn ----------------------------------------------------
```

and will end with a segment length line:

```
LENGTH OF SEGMENT nnnnn
```

The numerical map is functionally equivalent to the load map produced by the linkage editor or binder.

Numerical cross-reference
The numerical cross-reference listing contains one entry for each RLD record in the module, presented in sequence by the hexadecimal class offset of the related address constant. There is one RLD record for each A-, V-, Q-, and J-type address constant in the module, and one RLD record for each class reference (RLD type=21), class length (CXD), or loader token.

Each entry consists of one line that is described as follows:

1. The first part of the line describes the adcon itself, showing the class and element offsets, in hex, and the name of the section (FROM SECTION) containing the adcon. Because all adcons must reside within a section, there will always be either a user-defined section name or a representation of the binder-generated name, such as $PRIV000001 or $BLANKCOM.

2. The second part of the line describes the referenced, or target, symbol. It contains the name of the referenced symbol (REFERS TO SYMBOL), the class and element offsets of the referenced label/part or section/element definition, and the class name of the referenced class name, if the reference is resolved, or one of the following constants:
   • $UNRESOLVED - A strong reference (ER) could not be resolved during binding.
   • $UNRESOLVED(W) - A weak reference (WX) could not be resolved during binding.
   • $NEVER CALL - The symbol was marked never call, and no attempt was made during binding to resolve the symbol from the library.
$CLASS-OFFSET - The reference was to a class offset (Q-con).
$CLASS-LEN - The reference was to a class length (RLD type=40).

If the RLD item is for a class offset or class length, the constant string $CLASS_OFFSET or $CLASS_LEN will appear in place of a name.

The third part of the line will be blank except for resolved A-type and V-type address constants and displays the containing section name (IN SECTION). If the target section does not have a name, then a representation of the binder-generated name (SPRIV000005, $BLANKCOM) will be printed. If the target name in the second part of the line matches the containing section name, the containing section name will not be printed, since it would provide no additional information.

The last, or only, segment cross-reference will be followed by the length of the program object:

LENGTH OF PROGRAM OBJECT nnnn

If no RLD available in a class, the following message will appear instead of the formatted detail:

**** NO ADCONS IN THIS CLASS ****

Alphabetical map
The alphabetical map displays label definitions, part definitions, control sections and element definitions (except ER and PR) in alphabetical sequence, two print lines per ESD entry. It contains all of the same information as the Numerical Map, but in a different sequence. This part always begins on a new page, with a standard page heading of ALPHABETICAL MAP OF PROGRAM OBJECT.

The first detail line contains the label, section or common name.

The second detail line consists of the class offset, the element offset (type LD/PD records) or element definition length (all other types), the class name (name of the containing class), and the name of the containing section/element (type LD/PD records) or the ESD entry type (all other types). Element lengths are indented, to distinguish them from element offsets. If the module is in overlay format, the segment number is printed to the right of the section length.

Note: There is no preset order in which entries with identical names are output.

Alphabetical cross-reference
The alphabetical cross-reference listing provides the same information as the numerical cross-reference listing, but in a different sequence. This part of the report is in collating sequence by referenced name (the name of the symbol being referred to in the address constant).

The alphabetical cross-reference begins on a new page with a standard page heading ALPHABETICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT, and like the numerical cross-reference, contains nine columns:

1. Class offset. This is the hex offset of the named item within a class of the program object. Class offsets for the second and third detail lines have been indented.

2. Element offset. This is the hex offset of the named label or part within its section. Lines referring to an element, rather than a label, will always display zero for the element offset.

3. Overlay segment. This is displayed for overlay format modules only.
4. Symbol. This field varies between the three detail lines, as described in the following text. If the displayed name is a special section name, then one of the binder-generated names for example, $PRIVxxxxxx, described earlier, will replace the name.

If no RLD is available in a program object, the following message will appear instead of the formatted detail:

**** NO RLD DATA ***

The cross reference listing concludes with the line

** END OF MAP AND CROSS-REFERENCE LISTING

Sample segment map table for LISTLOAD OUTPUT=XREF of mutiple-text class module

Figure 185 is an example of a segment map for a multiple-text class module.

```
<table>
<thead>
<tr>
<th>CLASS</th>
<th>SEGMENT</th>
<th>OFFSET</th>
<th>LENGTH</th>
<th>LOAD</th>
<th>TYPE</th>
<th>ALIGNMENT</th>
<th>RMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_CODE</td>
<td>1</td>
<td>0</td>
<td>160</td>
<td>INITIAL</td>
<td>CAT</td>
<td>DOUBLE WORD</td>
<td>31</td>
</tr>
<tr>
<td>B_TEXT</td>
<td>1</td>
<td>160</td>
<td>99A</td>
<td>INITIAL</td>
<td>CAT</td>
<td>DOUBLE WORD</td>
<td>31</td>
</tr>
<tr>
<td>C_EXTERNTR</td>
<td>1</td>
<td>B00</td>
<td>28</td>
<td>INITIAL</td>
<td>CAT</td>
<td>DOUBLE WORD</td>
<td>31</td>
</tr>
<tr>
<td>C_DATA</td>
<td>1</td>
<td>B28</td>
<td>18</td>
<td>INITIAL</td>
<td>CAT</td>
<td>DOUBLE WORD</td>
<td>31</td>
</tr>
<tr>
<td>C_@@PPA2</td>
<td>1</td>
<td>B40</td>
<td>140</td>
<td>INITIAL</td>
<td>MERGE</td>
<td>DOUBLE WORD</td>
<td>31</td>
</tr>
<tr>
<td>B_LIT</td>
<td>1</td>
<td>B48</td>
<td>14</td>
<td>INITIAL</td>
<td>CAT</td>
<td>DOUBLE WORD</td>
<td>31</td>
</tr>
<tr>
<td>B_LIT</td>
<td>1</td>
<td>C88</td>
<td>98</td>
<td>INITIAL</td>
<td>CAT</td>
<td>DOUBLE WORD</td>
<td>31</td>
</tr>
<tr>
<td>C_WSA</td>
<td>2</td>
<td>0</td>
<td>24</td>
<td>DEFER</td>
<td>MERGE</td>
<td>DOUBLE WORD</td>
<td>31</td>
</tr>
</tbody>
</table>
```

Figure 185. Example: Segment map table for LISTLOAD OUTPUT=XREF of multiple-text class module

LISTLOAD OUTPUT=MAP

This section produces output identical to the SEGMENT MAP TABLE and NUMERICAL MAP sections of the LISTLOAD OUTPUT=XREF Output. See “Segment map” on page 619 and “Numerical map” on page 621 for more information.

LISTLOAD OUTPUT=XREF output (comparison of load module and program object version 1)

“Example: Output for LISTLOAD OUTPUT=XREF for a load module” shows an example of output produced by LISTLOAD OUTPUT=XREF for a load module.

Example: Output for LISTLOAD OUTPUT=XREF for a load module

LISTLOAD OUTPUT=XREF,DDN=DD1, MEMBER=MAINRTN

```
MEMBER NAME: MAINRTN
LIBRARY: DD1
NO ALIASES **
```

```
** LINKAGE EDITOR ATTRIBUTES OF MODULE **

<table>
<thead>
<tr>
<th>**</th>
<th>BIT STATUS</th>
<th>BIT STATUS</th>
<th>BIT STATUS</th>
<th>BIT STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NOT-RENT</td>
<td>NOT-REUS</td>
<td>NOT-DVLY</td>
<td>NOT-TEST</td>
</tr>
<tr>
<td>4</td>
<td>NOT-OL</td>
<td>BLOCK</td>
<td>NOT-EXE</td>
<td>MULTI-RCD</td>
</tr>
<tr>
<td>8</td>
<td>NOT-DC</td>
<td>0</td>
<td>EP-ZERO</td>
<td>RLD</td>
</tr>
<tr>
<td>12</td>
<td>EDIT</td>
<td>NO-SYMS</td>
<td>F-LEVEL</td>
<td>NOT-REFR</td>
</tr>
</tbody>
</table>
```

MODULE SSI: NONE
APFCODE: 00000000
RMODE: ANY
Example: Output for LISTLOAD OUTPUT=XREF for a program object shows an example of output produced by LISTLOAD OUTPUT=XREF for a program object.

**Example: Output for LISTLOAD OUTPUT=XREF for a program object**

```
LISTLOAD OUTPUT=XREF,DDN=DD1,
  MEMBER=(MAINRTN,
    THISISALONGALIASNAMEYOUMAYCHANGETHENAMEIFYOULIKEANYNAMEWILLDOOOO),
  TITLE=('XREF LISTINGS OF A LONG ALIAS NAME',10)
```

***** MODULE SUMMARY *****

MEMBER NAME: MAINRTN
MAIN ENTRY POINT: 00000000
LIBRARY: DD1
AMODE OF MAIN ENTRY POINT: 31

** ALIASES **

** ENTRY POINT ** AMODE
THISISALONGALIASNAME 00000000 31

** ATTRIBUTES OF MODULE **

** BIT STATUS **

<table>
<thead>
<tr>
<th>BIT</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NOT-RENT</td>
</tr>
<tr>
<td>1</td>
<td>NOT-REUS</td>
</tr>
<tr>
<td>2</td>
<td>NOT-OVLY</td>
</tr>
<tr>
<td>3</td>
<td>NOT-TEST</td>
</tr>
<tr>
<td>4</td>
<td>NOT-DL</td>
</tr>
<tr>
<td>5</td>
<td>BLOCK</td>
</tr>
<tr>
<td>6</td>
<td>EXCC</td>
</tr>
<tr>
<td>7</td>
<td>MULTI-RED</td>
</tr>
<tr>
<td>8</td>
<td>NOT-DC</td>
</tr>
<tr>
<td>9</td>
<td>ZERO-ORG</td>
</tr>
<tr>
<td>10</td>
<td>RESERVED</td>
</tr>
<tr>
<td>11</td>
<td>RLD</td>
</tr>
<tr>
<td>12</td>
<td>EDIT</td>
</tr>
<tr>
<td>13</td>
<td>NO-SMRS</td>
</tr>
<tr>
<td>14</td>
<td>RESERVED</td>
</tr>
<tr>
<td>15</td>
<td>NOT-RFR</td>
</tr>
<tr>
<td>16</td>
<td>RESERVED</td>
</tr>
<tr>
<td>17</td>
<td>16m</td>
</tr>
<tr>
<td>18</td>
<td>NOT-PL</td>
</tr>
<tr>
<td>19</td>
<td>NO-SSI</td>
</tr>
<tr>
<td>20</td>
<td>APF</td>
</tr>
<tr>
<td>21</td>
<td>POM OBJ</td>
</tr>
<tr>
<td>22</td>
<td>NOT-SIGN</td>
</tr>
<tr>
<td>23</td>
<td>RESERVED</td>
</tr>
<tr>
<td>24</td>
<td>NOT-ALTP</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
</tr>
<tr>
<td>26</td>
<td>RESERVED</td>
</tr>
<tr>
<td>27</td>
<td>MODERY</td>
</tr>
<tr>
<td>28</td>
<td>RESERVED</td>
</tr>
<tr>
<td>29</td>
<td>RESERVED</td>
</tr>
<tr>
<td>30</td>
<td>RESERVED</td>
</tr>
<tr>
<td>31</td>
<td>RESERVED</td>
</tr>
<tr>
<td>32</td>
<td>NON-MGR</td>
</tr>
<tr>
<td>33</td>
<td>NO-PRIME</td>
</tr>
<tr>
<td>34</td>
<td>NO-PACK</td>
</tr>
<tr>
<td>35</td>
<td>RESERVED</td>
</tr>
<tr>
<td>36</td>
<td>RESERVED</td>
</tr>
<tr>
<td>37</td>
<td>RESERVED</td>
</tr>
<tr>
<td>38</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

** END OF MAP AND CROSS-REFERENCE LISTING **

"Example: Output for LISTLOAD OUTPUT=XREF for a program object" shows an example of output produced by LISTLOAD OUTPUT=XREF for a program object.
LISTLOAD OUTPUT=BOTH Output

"Example: Output for LISTLOAD OUTPUT=BOTH for a PDSE" shows an example of output produced by LISTLOAD OUTPUT=BOTH for a PDSE.

Example: Output for LISTLOAD OUTPUT=BOTH for a PDSE

1 LISTLOAD MEMBER=TESTRS,OUTPUT=BOTH,SECTION1=NO
1
0 MEMBER NAME: TESTLRS
***** MODULE SUMMARY *****
MAIN ENTRY POINT: 00000028
AMBLIST

0 LIBRARY: SYSLIB AMODE OF MAIN ENTRY POINT: 24
0 NO ALIASES **

---

**** ATTRIBUTES OF MODULE ****

** BIT STATUS BIT STATUS BIT STATUS BIT STATUS **

0 NOT-REN 1 NOT-REUS 2 NOT-OVLY 3 NOT-TEST
4 NOT-OL 5 BLOCK 6 EXEC 7 MULTI-RCD
8 NOT-DC 9 ZERO-ORG 10 RESERVED 11 RL
12 EDIT 13 NO-SYMS 14 RESERVED 15 NOT-REFR
16 RESERVED 17 <16M 18 NOT-PL 19 NO-SIS
20 NOT-APF 21 PGM OBJ 22 NOT-SIG 23 RESERVED
24 NOT-ALTP 25 RESERVED 26 RESERVED 27 RMODE24
28 RESERVED 29 RESERVED 30 RESERVED 31 RESERVED
32 MIGRATE 33 NO-PRIME 34 NO-PACK 35 RESERVED
36 RESERVED 37 RESERVED 38 RESERVED 39 RESERVED

---

- MODULE SSI: NONE
APPCODE: 00000000
RMODE: 24
LONGPARM: NO
PO FORMAT: 2
XPLINK: NO

---

*****PROGRAM OBJECT PROCESSED BY BINDER

THE FOLLOWING ARE THE UNFORMATTED PSE DIRECTORY ENTRY SECTIONS (PMAR AND PMARL)

PMAR 001E0208 02C00400 00000000 00380000 00280000 00280000 00000000 0000
PMARL 00520080 00000000 00020000 00380000 02280000 07FC0000 09E00000 00400000
013C0000 00200000 013C0000 00200000 017C0001 00000000 00380000 00002000
013E0000 00200000 013F0193 505FC3D6 D7E84040 4040

---

LISTING OF PROGRAM OBJECT TESTLR5 PAGE 1

THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695PMB01 AT LEVEL 01.06 ON 01/13/2005 AT 19:35:05

---

MODULE SECTION: $SUMMARY

USABILITY: UNSPECIFIED AMODE: ANY OVERLAY SEGMENT: 0 OVERLAY REGION: 0

----- ESDs -----

OB_PRV(ED)
CLASS: B_PRV LENGTH: 0 (HEX) CLASS OFFSET: 0 (HEX) FORMAT: F(0001)
NAME SPACE: 2 ALIGNMENT: DOUBLE WORD BIND METHOD: MERGE RMODE: UNS
TEXT NOLOAD FILL: UNSPEC

Q1(PD)
CLASS: B_PRV LENGTH: 4 (HEX) CLASS OFFSET: 0 (HEX)
NAME SPACE: 2 ALIGNMENT: FULL WORD PRIORITY: 0 (HEX) SCOPE: MODULE
ATTRIBUTES: WEAK

-----

CONTROL SECTION: A

USABILITY: UNSPECIFIED AMODE: 24 OVERLAY SEGMENT: 0 OVERLAY REGION: 0

-----

TRANSLATOR VER MOD DATE TIME
566896201 02 01 01/13/2005

-----

DATE USER DATA
01/13/2005 MYDATA

----- ESDs -----

OB_TEXT(ED)
CLASS: B_TEXT LENGTH: 18 (HEX) CLASS OFFSET: 0 (HEX) FORMAT: F(0001)
NAME SPACE: 1 ALIGNMENT: DOUBLE WORD BIND METHOD: CATENATE RMODE: 24
TEXT LOAD FILL: UNSPEC

A(LD)
CLASS: B_TEXT TEXT TYPE: UNSPEC CLASS OFFSET: 0 (HEX)
NAME SPACE: 1 SCOPE: MODULE ELEMENT OFFSET: 0 (HEX) AMODE: 24
ATTRIBUTES: GENERATED,STRONG

ENTA(LD)
CLASS: B_TEXT TEXT TYPE: UNSPEC CLASS OFFSET: 14 (HEX)
NAME SPACE: 1 SCOPE: MODULE ELEMENT OFFSET: 14 (HEX) AMODE: 24
ATTRIBUTES: STRONG

ENTA(ER)

TARGET SECTION: A

TARGET CLASS: B_TEXT RESOLVED ATTRIBUTES: STRONG
CLASS: B_TEXT TEXT TYPE: UNSPEC CLASS OFFSET: 0 (HEX)
NAME SPACE: 1 SCOPE: LIBRARY ELEMENT OFFSET: 0 (HEX) AMODE: UNS

-----

CONTROL SECTION: A

----- ESDs -----

NAME SPACE: 2 ALIGNMENT: DOUBLE WORD BIND METHOD: MERGE RMODE: UNS
TEXT NOLOAD FILL: UNSPEC
Chapter 16. AMBLIST: Map load modules and program objects
**LISTIDR output**

Figure 186 on page 629 shows an example of LISTIDR output for a load module processed by the linkage editor or binder.
Figure 187 on page 630 shows an example of LISTIDR output for a program object processed by the binder.
The IDR listing, as in Figure 186 on page 629 and Figure 187, has four sections that are separated by dashed lines. The four sections contain:

- The linkage editor identification or binder identification record (IDRB). The identification record is displayed in a single line. This line shows the binder or linkage editor program identifier, version and release numbers, and the data and time of binding.

**Note:** The time of binding is listed only for a program object.
A list of SPZAP IDR entries (IDRZ), if any. The IDRZ records, if any, are formatted two or more lines per section. The first contains the associated CSECT name, and the second, and subsequent lines, a modification date and up to eight bytes of PTF number or other data entered on the SPZAP IDRDATA control statement. There will be one detail line for each modification to the control section. For load module output, the IDRZ records are formatted one line per section.

A list of language translator IDR records (IDRL). These entries are formatted only if OUTPUT=ALL was specified, or defaulted, on the LISTIDR control statement. The IDRL records, if any, are also formatted two or more lines per CSECT. The section name appears on the first line, and the translator program id, version and release, and date of translation on the second and subsequent lines. There will be one line of translator data for each compiler, assembler or other language product involved in the production of the object code for that section. For load module output, the IDRL records are formatted one line per section. Blank CSECT names in program objects will be seen as $BLANKCOM. They will be be seen as $BLANKCM in load modules.

A list of user-supplied IDR data (IDRU), if any. The IDRU records normally appear two lines per CSECT. The first line shows the section name, and the second line an entry date and up to 80 bytes of data, entered by the user on the binder IDENTIFY control statement. If the section name is a module level section (identified as '00000001'x), the constants $MODULE LEVEL DATA are printed in place of the section name.

For program objects, if no data is available in a section, one of the following messages will appear instead of the formatted detail:

- NO SPZAP DATA EXISTS FOR THIS PROGRAM OBJECT
- NO BINDER DATA EXISTS FOR THIS PROGRAM OBJECT
- NO TRANSLATION DATA EXISTS FOR THIS PROGRAM OBJECT
- NO IDENTITY/USER DATA EXISTS FOR THIS PROGRAM OBJECT

For load modules, if no SPZAP data is available, the following message will appear instead of the formatted detail:

- THIS LOAD MODULE CONTAINS NO INFORMATION SUPPLIED BY SPZAP

**Description of LISTIDR output**

**LISTLPA output**

Figure 188 on page 632 shows an example of LISTLPA output.
MODIFIED
NAME
IGC00020
IGC0006I

LINK PACK
LOCATION
---------------

AREA MAP
LENGTH
-----------

- ALPHABETICALLY BY NAME
MAJOR LPDE NAME
EP ADDR
00B42000
00B419C0

MODIFIED
NAME
IGC0005E
IGC0006I

LINK PACK
LOCATION
---------------

AREA MAP
LENGTH
-----------

- NUMERICALLY BY ENTRY POINT
MAJOR LPDE NAME
EP ADDR
00B37FA0
00B419C0

LOCATION LENGTH
NAME
IGC0005E ------------IGGC019BN ------------N 00.12SEC VIRT
LOCATION LENGTH
NAME
IGGC019BN ------------IGC00020 ------------N 00.12SEC VIRT

PAGEABLE LINK PACK
LOCATION
NAME
AHLACFV
AHLDSP
AHLFI0
AHLFRR
AHLFSVC
AHLPINT
AHLSBCU1
AHLSBUF
AHLSETEV 01928000
AHLSRB
AHLSTAE
AHLTACFV
AHLTDIR
AHLTEXT 01956920
AHLTF0R 01954570
AHLTLSR
01971468
AHLTPID
AHLTSRB
AHLSTAE
AHLTSYFL 0193A908
AHLTUSR 019299C0
AHLTXSYS 01963850
AHLVCON 01989EE8
AHLWTOMD
AMDSYS01 01936000
AMDSYS03 01939000
01975178
00F28000
00B8E230
00C4E730
AMDSYS05
00CB8078
AMDUSRFD
00C26318
AMDUSRFF
AMDUSRF9
CVAFGTF
DCMBE1
DCM181
DCM183
DCM271

AREA MAP - ALPHABETICALLY BY NAME
MAJOR LPDE NAME
EP ADDR
LENGTH
819B595E AHLTVTAM
AHLTXSYS
81963962
8193A926 AHLTSYFL
8198F7EA AHLTSYSM
8193A9D8 AHLTSYFL
8198F748 AHLTSYSM
81991F4A AHLWSMOD
81991A90 AHLWSMOD
00001998 81928000
819639EE AHLTXSYS
8198F8C6 AHLTSYSM
819B596A AHLTVTAM
81926A58 AHLSETD
000006E0 81956920
00000A90 81954570
819717D2 AHLTPID
00000B98 81971468
AHLTPID
AHLTFOR
000006F8
00000640
000007B0 81971770
00000118 819547B4
8193A908 AHLSETD
00002AD8 819299C0
00001828 81963850
99999358 81989EE8
00001E60 81926E4C
81936000
IMDUSRFF
000003F8 91039000
000008D0 81975178
00F28000 DCM3B3
00000F88 00C4C000
00000CE8 00B8E230
00C4E730 DCM270
00C56328
00CB8078

00C26318
00F24000

Figure 188. Sample LISTLPA output

632

z/OS V2R2 MVS Diagnosis: Tools and Service Aids

LOCATION
NAME
AHLDMPMD
AHLEXT
AHLFPI
AHLFSSCH
AHLMCER
AHLREADR 01977C08
AHLSBL0K
AHLSETD 01926000
AHLSFE0B
AHLSRM
AHLSVC
AHLTCCWG 0192A000
AHLTDSP
AHLTFCG 0192D000
AHLTFRR
AHLTPI
AHLTSLIP 0192F000
AHLTSRM
AHLTSVC 01931000
AHLTSYSM 0198F508
AHLTVTAM 019B5940
AHLVCOFF 019B6F40
AHLWSMOD 019916B0
AMDSYS00 01934000
AMDSYS02 019BB648
AMDSYS04 0193B000
01961C08
00BF1008

LENGTH

00F26000
AMDSYS06
00C54020
AMDUSRFE
AMDUSRF8
CCKRIUWT
DCMBE0
DCM180
DCM182
DCM270
00F24000
DCM272
00E1E830

00001360
00000FE0

000003F8
00001708

00002378
000016D0
00001C50
00002768
00000AF8
000006C0
000000C0
00000950
00001208
00000548
00002038
000003F8
000006C8

00000FE0
000014E0
000007D0

EP ADDR
00B37FA0
00B414D8
200K SYS

MAJOR LPDE NAME

EP ADDR
00B414D8
00B42000
200K SYS

MAJOR LPDE NAME

EP ADDR
81926EBE
8198F660
8193A9FC
8193A946
81926450
81977C08
819916B0
81926000
819917EE
81963A62
8198F61A
8191A000
81971658
8192D000
81954694
8197147E
8192F000

MAJOR LPDE NAME
AHLSETD
AHLTSYSM
AHLTSYFL
AHLTSYFL
AHLTSETD

8195458C
81931000
8198F508
819B5940
819B6F40
819916B0
81934000
819BB648
8193B000
81961C08
00BF1008
00C08590
00C48000
00C56328
00F26000
00C54020

00F24000
00E1E830

276K

276K

AHLWSMOD
AHLWSMOD
AHLTXSYS
AHLTSYSM
AHLTPID
AHLTFOR
AHLTPID
AHLTFOR

IMDUSRF8
ISTAICIR
DCM3B3


Chapter 17. SPZAP: Modify data in programs and VTOCs

SPZAP is a service aid program that operates in problem state. SPZAP allows you to dynamically update and maintain programs and data sets. SPZAP can be used to apply fixes to modules or programs that need to be at current levels of the operating system. The functions of SPZAP provide many capabilities, including:

- Using the inspect and modify functions of SPZAP, you can fix programming errors that require only the replacement of instructions in a load module member of a PDS or a program object member of a PDSE without recompiling the program.
- Using the modify function of SPZAP, you can set traps in a program by inserting incorrect instructions. The incorrect instructions will force abnormal ending; the dump of storage provided as a result of the abnormal ending is a valuable diagnostic tool, because it shows the contents of storage at a predictable point during processing.
- Using SPZAP to replace data directly on a direct access device, you can reconstruct VTOCs or data records that may have been destroyed as the result of an I/O error or a programming error.
- On the advice of the IBM Support Center, start tracing in system components that do not use component trace. The IBM Support Center will tell you how to use the SPZAP service aid to start traces in these components.
- Update the system status index (SSI) in the directory entry for any load module in a PDS or program object in a PDSE. Update the CSECT identification record (IDR) in any load module in a PDS or program object in a PDSE.

Planning for SPZAP

SPZAP is an application that provides editing capabilities for data on a direct access storage device (DASD). Protect against SPZAP (and other applications that can update data sets) being used to damage data through use of the installation's security protection scheme:

- In [z/OS DFSMS Using Data Sets] see the chapter, “Protecting Data Sets” for information pertaining to protecting data sets.
- In [z/OS DFSMSdfp Advanced Services] see the chapter, “Protecting the VTOC and VTOC Index” for information pertaining to protecting VTOCs.

Installations using RACF should employ a combination of GDASDVOL and DASDVOL resource profiles to establish this protection. See [z/OS Security Server RACF Security Administrator’s Guide] for more information regarding these profiles.

IBM recognizes the particular sensitivity of the VTOC. For a VTOC, the console operator must respond to message AMA117D before SPZAP will process an update request. This authorization must be supplied in addition to authorization through use of the installation's security protection scheme.

Inspecting and modifying data

The inspection function is controlled by the VERIFY statement. VERIFY allows you to check the contents of a specific location in a load module member of a PDS, a program object member of a PDSE or a z/OS UNIX file, a specific physical record of a direct-access data set, or a record of a member of a data PDSE before you
SPZAP

replace the contents. If the contents at the specified location do not agree with the contents as specified in the VERIFY statement, subsequent REP operations are not performed.

**Note:** A PDSE containing data other than a program object will be referred to as a PDSE data library.

The SPZAP modification function is controlled by the REP (replace) control statement. The REP control statement allows you to replace instructions or data at a specific location in a load module member of a PDS, a program object in a PDSE or a z/OS UNIX file, a physical record in a direct-access data set or a record of a member of a PDSE data library.

To avoid possible errors in replacing data, you should always precede any REP operation with a VERIFY operation.

SPZAP is often used to inspect and modify the contents of executable programs to correct errors. Executable programs can be in one of two forms:

- A load module is stored in a PDS and created by the linkage editor, binder, or IEBIMAGE utility.
- A program object is stored in a PDSE or z/OS UNIX file and created by the program management binder.

**Note:**
1. All subsequent references in this topic to a program object in a PDSE also apply to a program object in a z/OS UNIX file.
2. For program objects located in a PDSE, SPZAP supports only CSECTs assembled with the GOFF or SECTALGN(>8) options.

In addition, SPZAP can be used to inspect and modify data other than executable programs. Examples of such types of data are:

- A sequential (QSAM/BSAM and EXCP) data set.
- A direct organization (BDAM) data set.
- A VSAM data set in a conventional DASD volume.
- A partitioned data set extended (PDSE) program library (see preceding note).

There are several types of data sets that are not supported by SPZAP:

- An extended format sequential data set.
- A VSAM data set in an extended address volume after the first 65520 cylinders.
- A partitioned data set extended (PDSE) data library (see preceding note).
- A load module that includes a CSECT assembled with quadword alignment.

See the following topics for more information:

- "Inspecting and modifying a load module or program object"
- "Inspecting and modifying a data record" on page 642

**Inspecting and modifying a load module or program object**

To inspect or modify data in a load module or program object, you need a NAME statement to supply SPZAP the name of the appropriate member of the file. The load module must be a member of the PDS, identified by the SYSLIB DD
SPZAP

statement included in the JCL. The program object must be a member of the PDSE or a file in the z/OS UNIX directory identified by the SYSLIB DD statement included in the JCL.

To inspect or modify a program object that is in a z/OS UNIX file system, use the PATH parameter on the SYSLIB DD statement instead of the DSNAME parameter. Use PATH to identify the directory that contains the file that is the program object. Use the NAME statement to identify the file.

If the load module member of a PDS or program object member of a PDSE contains more than one control section (CSECT), you must also supply SPZAP with the name of the CSECT that is to be inspected or modified. If no CSECT name is given in the NAME statement, SPZAP assumes that the control section to be processed is the first one encountered in searching the load module.

Whenever SPZAP updates a CSECT in a load module member of a PDS or program object member of a PDSE in response to your NAME and REP control statements, it also puts descriptive maintenance data in a CSECT identification record (IDR) associated with the load module or program object. This function will be performed automatically after all REP statements associated with the NAME statement have been processed; any optional user data that has to be placed in the IDR will come from the IDRDATA statement. See “SPZAP control statements” on page 651 for an explanation of the IDRDATA statement.

Figure 189 shows how to inspect and modify a load module containing a single CSECT.

```
//ZAPCSECT JOB MIGLEVEL=(1,1)
//STEP EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN DD *
NAME IEEVLNKT
VERIFY 0018 C9C8,D2D9,D1C2,C7D5
REP 0018 E5C6,D3D6,E6F0,4040
SETSSI 01211234
IDRDATA 71144
DUMP IEEVLNKT
/*
```

Figure 189. Example: Inspecting and modifying a single CSECT load module

**SYSLIB DD Statement**

Defines the system library SYS1.LINKLIB containing the module IEEVLNKT that SPZAP is to process.

**NAME Control Statement**

Instructs SPZAP that the operations defined by the control statements that follow are to be performed on the module IEEVLNKT.

**VERIFY Control Statement**

Requests that SPZAP check the hexadecimal data at offset X’0018’ in the module IEEVLNKT to make sure that it is the same as the hexadecimal data specified in this statement. If the data is the same, SPZAP continues processing the subsequent statements sequentially. If the data is not identical, SPZAP will not perform the REP and SETSSI operations requested for the module. It will,
however, perform the requested DUMP operation before discontinuing the processing. It will also dump a hexadecimal image of the module IEEVLNKT to the SYSPRINT data set.

**REP Control Statement**
Causes SPZAP to replace the data at offset X'0018' in module IEEVLNKT with the data given in this control statement, provided the VERIFY statement was successful.

**SETSSI Control Statement**
Instructs SPZAP to replace the system status information in the directory entry for module IEEVLNKT with the SSI data given in the statement, if the VERIFY statement was successful. The new SSI is to contain:
- A change level of 01
- A flag byte of 21
- A serial number of 1234

**IDRDATA Control Statement**
Causes SPZAP to update the IDR in module IEEVLNKT with the data 71144, if the REP operation is successful.

**DUMP Control Statement**
Requests that a hexadecimal image of module IEEVLNKT be dumped to the SYSPRINT data set. Since the DUMP statement follows the REP statement, the image will reflect the changes made by SPZAP if the VERIFY operation was successful.

Figure 190 shows how to apply an IBM-supplied PTF in the form of an SPZAP fix, rather than a module replacement PTF.

---

```plaintext
//PTF40228 JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=SYS1.NUCLEUS,DISP=OLD
//SYSIN DD *
  NAME IEANUC01 IEWFETCH
  IDRDATA LOCFIX01
  VERIFY 01F0 47F0C018
  VERIFY 0210 5830C8F4
  REP 01F0 4780C072
  REP 0210 4130C8F4
  SETSSI 02114228
  DUMPT IEANUC01 IEWFETCH
/*
```

**Figure 190. Example: Modifying a CSECT in a load module**

**SYSLIB DD Statement**
Defines the library (SYS1.NUCLEUS) that contains input module IEANUC01.

**SYSIN DD Statement**
Defines the input stream.

**NAME Control Statement**
Instructs SPZAP that the operations defined by the control statements that immediately follow this statement are to be performed on the CSECT IEWFETCH contained in the load module IEANUC01.
**IDRDATA Control Statement**
Causes SPZAP to update the IDR in module IEANUC01 for CSECT IEFETCH with the data LOCFIX01, if either of the REP operations is successful.

**VERIFY control statements**
Requests that SPZAP compare the contents of the locations X'01F0' and X'0210' in the control section IEFETCH with the data given in the VERIFY control statements. If the comparisons are equal, SPZAP continues processing subsequent control statements sequentially. However, if the data at the locations does not compare identically to the data given in the VERIFY control statements, SPZAP dumps a hexadecimal image of CSECT IEFETCH to the SYSPRINT data set; the subsequent REP and SETSSI statements are ignored. The DUMPT function specified will be performed before SPZAP ends processing.

**REP control statements**
Causes SPZAP to replace the data at offsets X'01F0' and X'0210' from the start of CSECT IEFETCH with the hexadecimal data specified on the corresponding REP statements.

**SETSSI Control Statement**
Causes SPZAP to replace the system status information in the directory for module IEANUC01 with the SSI data given in the SETSSI statement after the replacement operations have been effected. The new SSI will contain a change level of 02, a flag byte of 11, and a serial number of 4228.

**DUMPT Control Statement**
Causes SPZAP to produce a translated dump for CSECT IEFETCH of load module IEANUC01.

Use the JCL in [Figure 191](#) to inspect and modify two CSECTs in the same load module.

```plaintext
//CHANGIT JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN DD *
NAME IEFX5000 IEFQMSSS
VERIFY 0284 4780,C096
REP 0284 4770,C096
IDRDATA PTF01483
SETSSI 01212448
DUMPT IEFX5000 IEFQMRAW
NAME IEFX5000 IEFQMRAW
VERIFY 0154 4780,C042
REP 0154 4770,C042
IDRDATA PTF01483
SETSSI 01212448
DUMPT IEFX5000 IEFQMRAW
/*
```

*Figure 191. Example: Inspecting and modifying two CSECTs*

**SYSLIB DD Statement**
Defines the system library SYS1.LINKLIB containing the load module IEFX5000 that is to be changed by SPZAP.
NAME Control Statement #1
Instructs SPZAP that the operations requested through the control statements immediately following it are to be performed on CSECT IEFQMSSS in load module IEFX5000.

VERIFY Control Statement #1
Requests that SPZAP check the hexadecimal data at offset X'0284' in CSECT IEFQMSSS to make sure it is the same as the data specified in this control statement. If the data is identical, SPZAP continues processing the control statements. If the data is not identical, SPZAP does not perform the REP or SETSSI for CSECT IEFQMSSS, but it does perform the DUMPT operation. It also provides a hexadecimal dump of CSECT IEFQMSSS.

REP Control Statement #1
Causes SPZAP to replace the data at offset X'0284' in CSECT IEFQMSSS with the hexadecimal data given in this control statement.

IDRDATA Control Statement #1
Causes SPZAP to update the IDR in module IEFX5000 for CSECT IEFQMSSS with the data PTF01483, if the first REP operation is successful.

SETSSI Control Statement #1
Instructs SPZAP to replace the system status information in the directory entry for module IEFX5000 with the SSI data given. The new SSI will contain a change level of 01, a flag byte of 21, and a serial number of 2448.

DUMPT Control Statement #1
Provides a translated dump of CSECT IEFQMSSS.

NAME Control Statement #2
Indicates that the operations defined by the control statements that immediately follow this statement are to be performed on CSECT IEFQMRAW in the load module IEFX5000.

VERIFY Control Statement #2
Requests that SPZAP perform the VERIFY function at offset X'0154' from the start of CSECT IEFQMRAW. If the VERIFY operation is successful, SPZAP continues processing the subsequent control statements sequentially. If the VERIFY is rejected, however, SPZAP does not perform the following REP or SETSSI operations, but it does dump a hexadecimal image of CSECT IEFQMRAW to the SYSPRINT data set and performs the DUMPT operation as requested.

REP Control Statement #2
Causes SPZAP to replace the data at hexadecimal offset X'0154' from the start of CSECT IEFQMRAW with the hexadecimal data that is specified in this control statement.

IDRDATA Control Statement #2
Causes SPZAP to update the IDR in module IEFX5000 for CSECT IEFQMRAW with the data PTF01483, if the second REP operation is successful.

SETSSI Control Statement #2
Causes SPZAP to perform the same function as the previous SETSSI, but only if the second VERIFY is not rejected.

DUMPT Control Statement #2
Causes SPZAP to perform the DUMPT function on control section IEFQMRAW.
Use the JCL shown in Figure 192 to inspect and modify control section PRINTF in z/OS UNIX System Services.

```clike
//ZAPUNIX EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD PATH='/sj/sjpl/binder/unixzap/', PATHDISP=(KEEP,KEEP)
//SYSIN DD *
NAME LOADMOD1 PRINTF
VERIFY 0000 58F0C210
REP 0000 68F0D210
DUMP LOADMOD1 PRINTF
/*
```

Figure 192. Example: Inspecting and Modifying a CSECT in z/OS UNIX System Services

**SYSLIB DD Statement**
- Defines the directory `/sj/sjpl/binder/unixzap/` containing the program object LOADMOD1 that SPZAP is to process.

**SYSIN DD Statement**
- Defines the input stream.

**NAME control statement**
- Instructs SPZAP that the operations defined by the control statements that follow are to be performed on the control section PRINTF of the program object LOADMOD1.

**VERIFY control statement**
- Requests that SPZAP compare the contents of the location X'0000' in the control section PRINTF with the data given on the VERIFY control statement. If the comparisons are equal, SPZAP continues processing subsequent control statements sequentially. If the data does not compare, SPZAP dumps a hexadecimal image of CSECT PRINTF to the SYSPRINT data set; the subsequent REP control statement is ignored.

**REP control statement**
- Causes SPZAP to replace the data at offset X'0000' from the start of the CSECT PRINTF with the hexadecimal data provided.

**DUMP control statement**
- Requests that a hexadecimal image of program object LOADMOD1, control section PRINTF be dumped to the SYSPRINT data set. Because the dump statement follows the REP statement, the image will reflect the changes made by SPZAP if the VERIFY operation was successful.

Use the JCL in Figure 193 on page 640 to inspect and modify a CSECT within a program object.
SYSLIB DD statement

Defines the library SYS1.USERLIB containing a program object with an alias of LONGALIASNAME. (Note the continuation character (#) following LONG.) One CSECT in this program object is being changed.

SYSIN DD statement

Defines the input stream.

NAME control statement

This control statement contains a ‘#’ in column 72 and is continued to a second control statement. The first 18 columns of the continued statement are blanks and are ignored. The string ALIASNAME on this continued statement is concatenated with the string LONG to form member name LONGALIASNAME. Note that this statement could have been contained in one record as NAME LONGALIASNAME PDSPROCR. Either way, the NAME statement indicates that SPZAP is to use the VERIFY and two REP statements to one CSECT PDSPROCR in the program object member whose alias is LONGALIASNAME.

Note: Leading blanks on the continued statement are ignored. No characters on the first card are skipped. Therefore, in order to split an operand, part on the first card and the rest on the second, it is important that the part of the operand on the first card extends to column 71. A blank in column 71 indicates that the non-blank string in the second card begins a new operand.

VERIFY control statement

Requests that SPZAP check the data at hexadecimal displacement X’000070’ from the start of the data record defined in the CCHHR statement to make sure it is the same as the hexadecimal data specified in this control statement. If the data is the same, SPZAP continues processing the following control statements sequentially. If the data is not identical, SPZAP does not perform the REP function but does perform the ABSDUMPT operation; it also dumps a formatted hexadecimal image of the data record defined by the CCHHR statement to the SYSPRINT data set.

REP Control Statement #1

Causes SPZAP to replace the data at offset X’000074’ in CSECT PDSPROCR with the hexadecimal data given in this control statement. Notice that this statement contains a non-blank (#) in column 72 indicating that it is continued to a second control statement.

REP Control Statement #2

Causes SPZAP to replace the data at offset X’00009A’ in CSECT PDSPROCR with the hexadecimal data given in this control statement.

Figure 193. Example: Using SPZAP to modify a CSECT

SPZAP
Accessing data in a CSECT

For a complete description of the control statements mentioned in the following discussion, see “SPZAP control statements” on page 651.

Once the CSECT has been found, the use of offset parameters in the VERIFY and REP statements allow SPZAP to locate the data that is to be verified and replaced. The offset parameters are specified in hexadecimal notation and define the displacement of the data relative to the beginning of the CSECT. For example, if a hexadecimal offset of X’40’ is specified in a VERIFY statement, SPZAP will find the location that is 64 bytes beyond the beginning of the CSECT identified by the NAME statement, and begin verifying the data from that point.

Normally, the assembly listing address associated with the instruction to be inspected or modified can be used as the offset value in the VERIFY or REP statement. However, if a CSECT has been assembled with other CSECTs so that its origin is not at assembly location zero, then the locations in the assembly listing do not reflect the correct displacements of data in the CSECT. You must compute the proper displacements by subtracting the assembly listing address delimiting the start of the CSECT from the assembly listing address of the data to be referenced.

You can, however, use the BASE control statement to eliminate the need for such calculations and allow you to use the assembly listing locations. The BASE control statement should be included in the input to SPZAP immediately following the NAME statement that identifies the CSECT. The parameter in the BASE statement must be the assembly listing address (in hexadecimal) at which the CSECT begins. SPZAP then subtracts this value from the offset specified on any VERIFY or REP statement that follows the BASE statement, and uses the difference as the displacement of the data.

Figure 194 on page 642 is a sample assembly listing showing more than one control section. To refer to the second CSECT (IEFCVOL2), you could include in the input to SPZAP a BASE statement with a location of 0398. Then, to refer to the subsequent LOAD instruction (L R2,CTJCTA0) you could use an offset of X’039A’ in the VERIFY or REP statements that follow in the SPZAP input stream.
Inspecting and modifying a data record

You will inspect and modify a data record differently depending on whether the data record is in a PDSE or some other type of data set, such as a VTOC or sequential data set.

**Note:** The following information does NOT apply to a PDS. SPZAP only supports a PDS that contains load modules.

**Record not in a PDSE:** To inspect or modify a specific data record that is not in a PDSE you must use a CCHHR control statement to specify its direct access address. This CCHHR address must be within the limits of the direct access data set defined in the SYSLIB DD control statement.

When you use the CCHHR control statement, SPZAP reads the physical record you want to inspect or modify. The offset parameters specified in subsequent VERIFY and REP statements are then used to locate the data that will be verified or replaced within the record. These hexadecimal offsets must define the displacement of data relative to the beginning of the record and include the length of any key field.

If you request a REP operation for a record identified by a CCHHR control statement, SPZAP issues message AMA112I to provide a record of your request.

In z/OS V1R7 and later DSNTYPE=LARGE data sets are supported when using V1R7 or a later release of SPZAP.

**Record in a PDSE:** To inspect or modify a specific data record in a PDSE data library, you must use the RECORD control statement preceded by a NAME control
statement to specify its direct access address. This combination of RECORD and
NAME serves as a pointer to a specific location in a PDSE data library member.

The CCHHR control statement does not apply to a PDSE. Any attempt to access
data in a PDSE with a CCHHR control statement will cause an error message. Any
VER|VERIFY, REP, IDRDATA, or SETSSI control statements immediately following
a CCHHR statement will be flagged in error and ignored.

To determine the relative record number for a specific record, invoke SPZAP,
specifying:
NAME membernam
ABSDUMP(T) 1 99999999

The results show a display of all records in the member, record length, relative
record number, and other pertinent information.

In Figure 195, the data set to be modified is a volume table of contents.

//ZAPIT JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=FORMAT4.DSCB,DISP=OLD,
//UNIT=3390,VOLUME=SER=111111,DCB=(KEYLEN=44)
//SYSIN DD *
CCHHR 0005000001
VERIFY 2C 0504
REP 2C 0A08
REP 2E 0001,03000102
ABSDUMPT ALL
/*

Figure 195. Example: Inspecting and modifying a data record

SYSPRINT DD Statement
Defines the message data set.

SYSLIB DD Statement
Defines the data set to be accessed by SPZAP in performing the operations
specified by the control statements. In this example, it defines the VTOC (a
Format 4 DSCB) on a 3390 volume with a serial number of 111111.
DCB=(KEYLEN=44) is specified so that the dump produced by the
ABSDUMPT control statement will show the dsname which is a 44-byte key.
Note that this is not necessary for the VERIFY and REP control statements.

CCHHR Control Statement
Indicates that SPZAP is to access the direct access record address “0005000001”
in the data set defined by the SYSLIB DD statement while performing the
operations specified by the following control statements.

VERIFY Control Statement
Requests that SPZAP check the data at hexadecimal displacement X’2C’ from
the start of the data record defined in the CCHHR statement to make sure it is
the same as the hexadecimal data specified in this control statement. If the data
is the same, SPZAP continues processing the following control statements
sequentially. If the data is not identical, SPZAP does not perform the REP
function but does perform the ABSDUMPT operation; it also dumps a
formatted hexadecimal image of the data record defined by the CCHHR
statement to the SYSPRINT data set.
**REP control statements**

Cause the eight bytes of data starting at displacement 2C from the beginning of the record to be replaced with the hexadecimal data in the REP control statements. The 2C displacement value allows for a 44-byte key at the beginning of the record.

**ABSDUMPT Control Statement**

Causes SPZAP to dump the entire data set to the SYSPRINT data set. Since DCB=(KEYLEN=44) is specified on the SYSLIB DD statement, the 44-byte dsname is also dumped.

**Note:** If the VTOC is to be modified, message AMA117D is issued to the operator, requesting permission for the modification.

Figure 196 shows how to inspect and modify a record within a PDSE data library.

---

```plaintext
//UPDDATA JOB  MSGLEVEL=(1,1)
//ZAPSTEP EXEC  PGM=AMASPZAP
//SYSPRINT DD  SYSOUT=A
//SYSLIB DD  DSN=IBMUSER.LMD.PDSE,DISP=OLD
//SYSIN DD  *
  NAME USERDATA
  RECORD 0003
  VER 000010 04B3,9017
  REP 000014 10C7,C5E3,C4E2
  ABSDUMP 3 3
```

Figure 196. Example: Using SPZAP to modify a data record

**SYSLIB DD statement**

Defines the data set that SPZAP is to access to perform the operations specified by the control statements. In this example, it defines a private PDSE data library. The NAME statement identifies the member as USERDATA, which is shown in Figure 206 on page 666.

**SYSIN DD statement**

Defines the input stream containing the SPZAP control statements.

**NAME control statement**

Instructs SPZAP that the control statements that immediately follow this statement are to be performed on the member whose name is USERDATA.

**RECORD control statement**

Indicates that SPZAP is to access relative record 3, the third record in the member USERDATA. Record 3 is the object of the VERIFY and REP operations that follow.

**VERIFY control statement**

Requests that SPZAP check the data at hexadecimal displacement X'0010' to compare it to the string specified. If there is a difference, this VERIFY is flagged with an error message, the contents of record 3 are displayed, and the following REP statement is flagged and ignored.

**REP control statement**

Causes SPZAP to replace the data at offset X'000014' in record 3 of member USERDATA with the data X'10C7C5E3C4E2' if the preceding VERIFY statement completed successfully. If the preceding VERIFY statement was flagged in error, then this statement is also flagged in error, and no data is replaced.
ABSDUMP control statement

Uses SPZAP to display record 3 of member USERDATA. Record 3 is displayed whether the VERIFY succeeded or failed.

Updating the System Status Index (SSI)

You can use the SETSSI control statement to overlay the existing data in the SSI with your own data. For a complete description of the SETSSI control statement, see "SPZAP control statements" on page 651.

The SSI is a 4-byte field created by the linkage editor in the directory entry of a load module. It is useful for keeping track of any modifications that are performed on a load module. SPZAP updates the system status index automatically whenever it replaces data in the associated module.

Not all load modules have system status information. In those that do, the SSI is located in the last four bytes of the user data field in the directory entry. Figure 197 shows the position of the SSI in load module directory entries.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>TTR</th>
<th>C</th>
<th>User Data Field</th>
<th>SSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure 197. SSI bytes in a load module directory entry

Figure 198 on page 646 shows the composition of the SSI field and the flag bits used to indicate the types of changes made to the corresponding load module program.

The first byte of SSI information contains the member’s change level. When a load module is initially released by IBM, its change level is set at one. Thereafter, the change level is increased by one for each release that includes a new version of that program. If you make a change to the SSI for any of the IBM-released programs, take care not to destroy this maintenance level indicator unless you purposely mean to do so. To keep the change level byte at its original value, find out what information is contained in the SSI before using the SETSSI function. The LISTLOAD control statement of the LIST service aid can give you the information you need.
The second byte of the SSI is called the *flag byte*. Bits within the flag byte contain information reflecting the member's maintenance status. You need only be concerned with two of the eight bits when you are using SPZAP:

- Bit 2, the local fix flag, indicates that the user has modified a particular member. (It is not used to reflect modifications made by IBM-supplied program temporary fix or a PTF.) SPZAP sets this local fix flag bit to one after successfully modifying a load module.
- Bit 3, the program temporary fix flag, is set to one when an IBM-authorized PTF is applied to a system library to correct an error in an IBM module.

All other bits in the flag byte should be retained in the SSI as they appeared before the SETSSI operation took place, so as not to interfere with the normal system maintenance procedures.

The third and fourth bytes of the system status index are used to store a serial number that identifies the first digit and the last three digits of a PTF number. SPZAP will not change these bytes unless you request a change by using the SETSSI control statement.

**Running SPZAP**

You can run SPZAP using control statements as input into the job stream or dynamically as part of selected macros:

- “Using JCL and control statements to run SPZAP” on page 647
- “Invoking SPZAP dynamically” on page 649

Consider the following points when you run SPZAP:
SPZAP uses the system OPEN macro. Therefore, SPZAP cannot modify or inspect RACF-protected data sets when SPZAP cannot successfully complete the access checks that occur during OPEN processing.

A module can be a load module in a PDS or a program object in a PDSE. SPZAP replaces a program object in a PDSE rather than updating the program object in place. Users who have used the BLDL macro to establish a connection to a particular copy of a program object must invoke BLDL again to gain access to the new copy.

If you are using LLA to manage a program object that has been changed through the use of SPZAP, then, to make the modified object available, the operator must refresh LLA for the directory entries for that program object. Otherwise, LLA continues to load the unmodified version of the program object. SPZAP itself cannot determine when a load module or a program object is in use by another user or is in the process of being loaded through LLA. See z/OS DFSMS Using Data Sets for more information about PDSEs and their data structure.

Unexpired data sets such as system libraries cannot be modified unless the operator replies r xx,'U' to the expiration message that occurs during OPEN.

If you use SPZAP to modify an operating system module that is made resident in virtual storage only at IPL time, you must IPL the system again to invoke the new version of the module you have modified. (Note that this requirement applies to all modules in SYS1.LPALIB, all data sets named in the LPALSTxx member of SYS1.PARMLIB, and all modules in SYS1.NUCLEUS. SPZAP itself cannot identify when a module is loaded only at IPL time.

The SYSLIB DD statement cannot define a concatenated or a multi-volume data set.

SPZAP supports only direct access storage devices (DASD) for the SYSLIB device.

When modifying a system data set, such as SYS1.LINKLIB, specify DISP=OLD on the SYSLIB DD statement.

If you use SPZAP for a digitally signed module, message AMA165I is issued. The control statement is to be processed, but the digital signature is no longer valid.

SPZAP supports placement of SYSIN and SYSPRINT data sets in cylinder-managed space.

SPZAP (AMASPZAP or IGWSPZAP) supports all data sets allocated in the extended addressing space (EAS) of an extended address volume (EAV).

SPZAP (AMASPZAP or IGWSPZAP) supports the following dynamic allocation (DYNALLOC or SVC 99) options for all data sets: S99TIOEX(XTIOT), S99ACUCB (NOCAPTURE), and S99DSABA (DSAB above the line).

Using JCL and control statements to run SPZAP

One way to invoke SPZAP is through the job stream. The JCL statements you need to use when running SPZAP are:

- JOB statement
- EXEC statement
- SYSPRINT DD statement
- SYSLIB DD statement
- SYSABEND DD statement
- SYSIN DD statement
These JCL statements, when used with the control statements in “SPZAP control statements” on page 651, allow greater function for SPZAP.

Also, when running SPZAP, you must consider the region size available to your program. The minimum region size needed to run AMASPZAP is 200 kilobytes.

Usually, no REGION parameter is required on the EXEC statement but REGION=120K (or any other value less than 200K) will cause SPZAP to issue message AMA154I and stop processing with a return code of 16. In addition, SPZAP will issue message AMA154I if the program management binder has too small a region size. This problem might occur if the SYSLIB member is extremely large, when REGION=4M or REGION=6M might be needed.

**JCL statements**

**JOB Statement**
Marks the beginning of the job.

**EXEC Statement**
Invokes SPZAP. You identify AMASPZAP as the program to be run by specifying either PGM=AMASPZAP or PGM=IMASPZAP, which is an alias name for AMASPZAP.

**Note:** You must ensure that the region size is at least 200K for SPZAP to complete processing normally.

The only valid parameter that you may specify is PARM=IGNIDRFULL, which enables SPZAP to override the standard restrictions placed upon CSECT updates (through NAME and REP) when IDR space for the module is found to be full.

**Note:**
1. Do not use PARM=IGNIDRFULL with IBM-maintained modules.
2. PARM=IGNIDRFULL has no meaning if SYSLIB is a program object library.
   There is no restriction on the number of IDRZ records associated with a program object library member.

**SYSPRINT DD Statement**
Defines a sequential output data set for messages that can be written on a system printer, a magnetic tape volume, or a direct access volume. This statement is required for each run of SPZAP.

**SYSLIB DD Statement**
Defines the direct access data set that will be accessed by SPZAP when performing the operations specified on the control statements. The DSNAME parameter and DISP=OLD or DISP=SHR are required. The VOLUME and UNIT parameters are necessary only if the data set is not cataloged. This statement cannot define a concatenated or multi-volume data set. It is required to run SPZAP.

**Note:**
1. When this data set is the VTOC, you must specify DSNAME=FORMAT4.DSCB. When you access a record in the VTOC (that is, a DSCB) for modification, SPZAP issues message AMA117D to the console. No message is issued, however, when an ABSDUMPT operation is performed on the VTOC.
2. Standard VSAM processing requires the use of an ACB to access the data set. However, because SPZAP only supports open with DCB, it does not
obtain the correct information needed to operate upon a VSAM data set. Where there is a blocksize mismatch reported by SPZAP, explicit specification of the blocksize on the SYSLIB DD statement will override SPZAP's normal size processing.

**SYSABEND DD Statement**
Defines a sequential output data set to be used in case SPZAP ends abnormally. The data set can be written to a printer, a magnetic tape volume, or a direct access volume. This statement is optional.

**SYSIN DD Statement**
Defines the input stream data set that contains SPZAP control statements.

**Return codes**
When SPZAP ends, one of the following return codes is placed in general purpose register 15:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>04</td>
<td>Warning of a condition. This may result in future errors if an action is not taken to correct the warning now.</td>
</tr>
<tr>
<td>08</td>
<td>A SPZAP input statement contains an error or was overridden by operator intervention. Check the syntax of the statements to determine the cause of the error.</td>
</tr>
<tr>
<td>12</td>
<td>A requested JCL statement is absent or specifies a data set that was not successfully opened. SPZAP ends immediately.</td>
</tr>
<tr>
<td>16</td>
<td>A permanent I/O error has occurred, perhaps caused by a JCL error, such as incorrect blocksize. SPZAP ends immediately. The region size might be too small. REGION=200K is the smallest permitted. However, the program management binder might require as much as 4M or 6M if the program object is very large.</td>
</tr>
<tr>
<td>20</td>
<td>Using DUMP, DUMPT, VER, or REP processing, SPZAP found a control record for a specific control section that was larger than the specified BLOCKSIZE. SPZAP ends immediately.</td>
</tr>
</tbody>
</table>

**Invoking SPZAP dynamically**
You can run SPZAP from selected macros. SPZAP can be invoked by an application program at run time through the use of the CALL, LINK, XCTL, or ATTACH macro. The program must supply a list of alternate DDNAMEs of data sets to be used by SPZAP if the standard DDNAMEs are not used.

A program must be running APF authorized in order to update a VTOC through SPZAP. Other SPZAP functions do not require the calling program to be authorized.

The following diagram shows the general form of these macros when used to invoke SPZAP.

```plaintext
(anyname) CALL AMASPZAP,(oplist,ddnamlst),VL
(anyname) XCTL EP=AMASPZAP
(anyname) LINK EP=AMASPZAP,PARAM=(oplist,ddnamlst),VL=1
(anyname) ATTACH EP=AMASPZAP,PARAM=(oplist,ddnamlst),VL=1
```
anyname
   Indicates an optional statement label on the macro statement.

EP  The entry point for the SPZAP program.

PARAM
   Specifies, as a sublist, parameters to be passed from the program to SPZAP.

oplis
   Specifies the name of either a halfword of zeros (indicating no options) or a
   non-zero halfword followed by a character string whose length is given in
   bytes. For the possible parameter value, see the information about the EXEC
   statement in ["JCL statements" on page 648].

ddnamlst
   Specifies the name of a variable-length list containing alternate ddnames for
   data sets to be used during SPZAP processing. If all the standard ddnames
   (SYSPRINT, SYSLIB, and SYSIN) are used, then you can omit this parameter.

   The DDNAME list must begin on a halfword boundary. The first two bytes
   contain a count of the number of bytes in the rest of the list. The format of the
   list is fixed, with each entry having eight bytes. Any name of less than eight
   bytes must be left justified and padded with blanks. If a name is left out in the
   list, the entry must contain binary zeros; the standard name is then assumed.
   Names can be omitted from the end of the ddname list by shortening the list.

   The sequence of 8-byte entries in the list is as follows:

   Entry    Standard name
   0-7      not applicable
   8-15     not applicable
   16-23    not applicable
   24-31    SYSLIB
   32-39    SYSIN
   40-47    SYSPRINT

VL | VL=1
   Indicates that the high-order bit is to be set to 1 in the last word of the address
   parameter list.

   Note: If you do not supply the name of a DDNAME list, you must ensure that
   the high-order bit of the oplist address is set on. Coding VL|VL=1 sets the bit
   correctly.

Figure 199 on page 651 is an example of two functionally-equivalent dynamic
invocations of SPZAP.
SPZAP control statements

SPZAP control statements (entered either through the user's input stream in the JCL or through the system console) define the processing functions to be performed during a particular run of SPZAP. To enter other SPZAP control statements through the system console, you can use the CONSOLE control statement. The control statements that define the running of SPZAP are:

- ABSDUMP or ABSDUMPT
- BASE
- CCHHR
- CHECKSUM
- Comment (*)
- CONSOLE
- DUMP or DUMPT
- IDRDATA
- NAME
- RECORD
- REP
- SETSSI
- VERIFY

Figure 199. Sample assembler code for dynamic invocation of SPZAP
Coding rules for SPZAP control statements: Follow these rules when coding the control statements for SPZAP:

- The size of a SPZAP control card is 80 bytes; it can contain 71 bytes of control information.
- Statements can begin in any column up to column 71.
- The operation name of the statement must precede the parameters and must be complete on the first statement; you cannot continue the operation name.
- There must be at least one blank between the specified operation name and the first parameter.
- All parameters must also be separated by at least one blank space.
- Data field parameters may be formatted with commas for easier visual check, but blanks within data fields are not permitted.
- Data and offset values must be specified as a multiple of two hexadecimal digits.
- Following the last required parameter and its blank delimiter, the rest of the space on most control statements can be used for comments. Exceptions to this are the NAME and DUMP control statements: if you omit the CSECT parameter from either of these statements, do not use the space following the load module parameter for comments.
- A record beginning with an asterisk is considered to be a comment statement.
- A comment statement (one that begins with a single asterisk) cannot be continued.
- Member names and CSECT names for program objects can be as long as 1024 characters.
- When SYSLIB refers to a PDSE or a z/OS UNIX file, you can continue any non-comment statement as follows:
  - Column 72 of the control card to be continued must contain a non-blank character.
  - The string of characters on the immediately following card (starting with the first non-blank character) is concatenated with column 71 of the preceding card. AMASPZAP ignores leading blanks in a continuation card, but it displays the cards on SYSPRINT unchanged.
  - You can continue statements as necessary. You cannot, however, continue a comment field that follows the last parameter.
  - Even though some parameters allow you to use a single asterisk (*) to indicate an omitted parameter, the first non-blank character on a continuation card cannot be an asterisk. Select the break point carefully to avoid starting a continuation statement with a single asterisk.
- In other words, for continuation:
  - When the SYSLIB is a PDSE or a z/OS UNIX file, IGWSPZAP is invoked, which supports continuation.
  - When the SYSLIB is a PDS, AMASPZAP is invoked, which does not support continuation.

Following are detailed descriptions of the SPZAP control statements, in alphabetical order.

\{ABSDUMP|ABSDUMPT\}{startaddr stopaddr | startrec stoprec | membername | ALL}\}

This statement can be used to dump the following, as defined in the SYSLIB DD statement:
The variables are:

`startaddr`

The absolute direct access device address of the first record to be dumped. This address must be specified in hexadecimal in the form `cccchhhhrr` (cylinder, track and record numbers). This parameter must be exactly 10 digits long.

`stopaddr`

The absolute direct access device address of the last record to be dumped, and it must be in the same format as the start address.

Both addresses must be specified when this method of dumping records is used, and both addresses must be within the limits of the data set defined by the SYSLIB DD statement. The record number specified in the start address must be a valid record number. If a record number of 0 is specified, SPZAP will change it to 1 since the READ routine skips over such records. The record number specified as the stop address need not be a valid record number, but if it is not, the dump will continue until the last record on the track specified in the stop address has been dumped.

Note: When the SYSLIB DD statement describes a data set placed in an extended address volume (EAV), the `startaddr` and `stopaddr` values must be specified in hexadecimal in the form `CCCCcccHRR`, where `CCCCccc` is referred to as a 28-bit cylinder address. The meanings of the codes are as follows:

- `CCCC` is the 16 low order bits of the cylinder number.
- `ccc` is the 12 high order bits of the cylinder number.
- `H` is the track number.
- `RR` is the record number.

`startrec`

The value of the first relative record of a member of a PDSE data library to display. This parameter can be 1 to 8 digits long. The first record of a member has a `startrec` value of 1.

Note: `ABSDUMP` or `ABSDUMPT` `startrec stoprec` is valid only following a `NAME member` statement where SYSLIB is a PDSE data library and `member` is a valid member of that library.

`stoprec`

The value of the last relative record of a member of a PDSE data library to display. This parameter can be 1 to 8 digits long. If the value of `stoprec`
SPZAP

specifies a relative record value greater than that of the last physical record, printing stops after the last record of the member is printed. If the value of stoprec is less than the value of startrec, no records are displayed.

One can display all the records of a member of a PDSE data library by using the following two statements:

```
NAME member
ABSDUMP|ABSDUMPT 1 99999999
```

**membername**

The name of a member of a PDS or a PDSE, as specified by the SYSLIB DD statement. The name can refer to a load module member of a PDS or a member of a PDSE data library. In each case, the entire member is dumped when this variable is specified. (Use DUMP/DUMPT for program object members of a PDSE.)

**ALL**

Specifies that the entire data set defined by the SYSLIB DD statement is to be dumped. How much of the space allocated to the data set is dumped depends on how the data set is organized:

- For a sequential data set, SPZAP dumps until it reaches end of file.
- For an indexed sequential and direct access data set, SPZAP dumps all extents.
- For a PDS, SPZAP dumps all extents, including all linkage editor control records, if any exist.
- For a PDSE data library, SPZAP displays a directory plus a listing of all members of the library. If the data set is a PDSE that contains program objects, SPZAP displays only the directory.

**BASE xxxxxxx**

Used by SPZAP to adjust offset values that are to be specified in any subsequent VERIFY and REP statements. This statement should be used when the offsets given in the VERIFY and REP statements for a CSECT are to be obtained from an assembly listing in which the starting address of the CSECT is not location zero.

For example, assume that CSECT ABC begins at assembly listing location X'000400', and that the data to be replaced in this CSECT is at location X'000408'. The actual displacement of the data in the CSECT is X'08'. However, an offset of X'0408' (obtained from the assembly listing location X'000408') can be specified in the REP statement if a BASE statement specifying X'000400' is included prior to the REP statement in the SPZAP input stream. When SPZAP processes the REP statement, the base value X'000400' will be subtracted from the offset X'0408' to determine the proper displacement of data within the CSECT. The variable is:

```
xxxxxxx
```

A 6-character hexadecimal offset that is to be used as a base for subsequent VERIFY and REP operations. This value should reflect the starting assembly listing address of the CSECT being inspected or modified.

**Note:** The BASE statement should be included in the SPZAP input stream immediately following the NAME statement that identifies the control section that is to be involved in the SPZAP operations. The specified base value remains in effect until all VERIFY, REP, and SETSSI operations for the CSECT have been processed.
Figure 200 shows how to use the BASE control statement to inspect and modify a CSECT whose starting address does not coincide with assembly listing location zero.

```plaintext
//MODIFY JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN DD *
//NAME IEFMCVOL IEFCVOL2
BASE 0398
IDRDATA MOD04
VERIFY 039A 5820C010
REP 039A 47000000
DUMP IEFMCVOL IEFCVOL2
```

**SYSLIB DD Statement**: Defines the system library, SYS1.LINKLIB, that contains the module IEFMCVOL in which the CSECT to be changed, IEFCVOL2, resides.

**SYSIN DD Statement**: Defines the input stream that contains the SPZAP control statements.

**NAME Control Statement**: Instructs SPZAP that the operations defined by the control statements that immediately follow it are to be performed on CSECT IEFCVOL2 in the load module IEFMCVOL.

**BASE Control Statement**: Provides SPZAP with a base value that is to be used to readjust the offsets on the VERIFY and REP statements that follow it.

**IDRDATA Control Statement**: Causes SPZAP to update the IDR in module IEFMCVOL for CSECT IEFCVOL2 with the data MOD04, if the REP operation is successful.

**VERIFY Control Statement**: Requests that SPZAP inspect the data at offset X'039A'. The base value X'0398' given in the previous BASE statement is subtracted from this offset to determine the proper displacement of the data within CSECT IEFCVOL2. Therefore, SPZAP checks the data at the location that is actually displaced X'0002' bytes from the beginning of CSECT IEFCVOL2 to ensure that it is the same as the hexadecimal data specified in this control statement. If the data is the same, SPZAP continues processing the following statements in the order in which they are encountered. If the data is not identical, SPZAP does not perform the REP, SETSSI, or IDRDATA functions, but it does perform the DUMP's operation; it also dumps a hexadecimal image of CSECT IEFCVOL2 to the SYSPRINT data set.

**REP Control Statement**: Causes SPZAP to replace the data at displacement X'0002' (offset 039A minus base value 0398) into CSECT IEFCVOL2 with the hexadecimal data specified in this control statement.

**DUMP Control Statement**: Requests that SPZAP dump a hexadecimal image of CSECT IEFCVOL2 to the SYSPRINT data set. Since the DUMP statement follows the REP statement, the image will reflect the changes made by SPZAP (assuming no verification has been rejected).

**CCHHR record address**
Identifies a physical record on a direct access device that is to be modified or
verified. The record must be in the data set defined by the SYSLIB DD statement. Any immediately following REP or VERIFY statements will reference the data in the specified record. The variable is:

record address

The actual direct access address of the record containing data to be replaced or verified. It must be specified as a 10-digit hexadecimal number in the form cccchhhhrr, where cccc is the cylinder, hhhh is the track, and rr is the record number. For example, 0001000A01 addresses record 1 of cylinder 1, track 10. A zero record number is incorrect and defaults to 1.

Note:

1. You can define more than one CCHHR statement in your input to SPZAP. However, the VERIFY, REP and SETSSI statements associated with each CCHHR statement must immediately follow the specific CCHHR statement to which they apply.

2. When the SYSLIB DD statement describes a data set placed in an extended address volume (EAV), the record address value must be specified in hexadecimal in the form CCCCcccHRR, where CCCCccc is referred to as a 28-bit cylinder address. The meanings of the codes are as follows:
   - CCCC is the 16 low order bits of the cylinder number.
   - ccc is the 12 high order bits of the cylinder number.
   - H is the track number.
   - RR is the record number.

CHECKSUM [hhhhhhhh]

Used to print or verify a fullword checksum (parity-check). All of the valid hexadecimal operands since the preceding CHECKSUM statement or SPZAP initialization are logically concatenated into a single string divided into fullwords, the sum of which is the checksum. For example, the string 12345678FACE produces the checksum 0D025678. Each CHECKSUM statement resets the accumulated checksum value to zeros.

The CHECKSUM statement is effective in detecting clerical errors that may occur when transcribing an SPZAP type of fix. CHECKSUM does not prevent errors; it only causes a message to be issued. By the time the CHECKSUM statement is processed, all prior replaces have been done.

hhhhhhhh

8 hexadecimal characters that are compared with the checksum. If the two values are equal, a message is written indicating that the checksum was correct and has been reset.

If the operand field is blank, a message is written giving the actual value of the checksum, and indicating that the checksum has been reset.

When the CHECKSUM control statement is provided with an incorrect operand, the REP and SETSSI statements processed already are not affected.

If the operand is not valid or is not equal to the checksum, a message is written indicating incorrect operand or checksum error. All subsequent REP and SETSSI statements are ignored until the next NAME or CCHHR statement is encountered. The results of previously processed statements are not affected.

* (Comment)

When the first non-blank character in a statement is an asterisk, SPZAP recognizes the statement as a comment, used to annotate the SPZAP input
stream and output listing. You can specify the asterisk in any position, but at least one blank space must follow the asterisk. You can include any number of comment statements in the input stream, but you cannot continue a comment statement. When SPZAP recognizes a comment, it writes the entire statement to the data set specified for SYSPRINT.

**CONSOLE**

Indicates that SPZAP control statements are to be entered through the system console. When this statement is encountered in the input stream, the following message is written to the operator:

```
AMA116A ENTER AMASPZAP CONTROL STATEMENT OR END
```

The operator may then enter in any valid SPZAP control statement conforming to the specifications described in the beginning of this control statement discussion. After each operator entry through the console is read, validated, and processed, the message is reissued, and additional input is accepted from the console until “END” is replied. SPZAP will then continue processing control statements from the input stream until an end-of-file condition is detected.

**Note:**

1. You can enter control statements through the console in either uppercase or lowercase letters, but AMASPZAP does not fold lowercase input to uppercase.
2. You cannot continue a control statement entered through the console.

Figure 201 shows how to enable SPZAP control statements to be entered through the console.

---

**SYSLIB DD Statement:** Defines the data set that contains the module to be updated.

**SYSIN DD Statement:** Defines the input stream.

**CONSOLE Control Statement:** Indicates that SPZAP control statements are to be entered through the console.

```
{DUMP|DUMPT} member [csect | ALL | * ] [class-name]
```

Dumps a specific control section or all control sections in a load module in a PDS, a program object in either a PDSE or a z/OS UNIX file. DUMP produces a hexadecimal printout only, while DUMPT prints the hexadecimal data, the EBCDIC translation, and the mnemonic equivalent of the data (see “Reading SPZAP output” on page 662). The variables are:

- **member**
  
  The member name of the load module in a PDS or program object in a PDSE that contains the control section(s) to be dumped. (Note: This...
variable, ‘member’, must correspond to the name of a member of the PDS or PDSE that is defined by the SYSLIB DD statement.

\textbf{csect | ALL | *}

Defines the name of the particular control section that is to be dumped. To dump all the CSECTs of a load module in a PDS or a program object in a PDSE, specify “ALL” instead of the CSECT name. If you omit the variable entirely, or, for program objects only, code “*”, SPZAP assumes that you mean to dump only the first CSECT in the load module or program object. If you specify a CSECT name that SPZAP does not find in the member, SPZAP dumps all of the CSECTs in the member.

\textbf{Note:} DUMP or DUMPT applied to a CSECT consisting only of space allocations (DS statements) will produce no output between the statement printback and the dump-completed message.

\textbf{class-name}

Indicates, for program objects only, the class of text that you want to dump. The default is B_TEXT. Specifying B_, C_, or D_* causes SPZAP to dump all text classes beginning with the string that precedes the asterisk. If you want to omit the CSECT name and supply a class-name, code a single asterisk for the CSECT name followed by the class-name.

For information about the values you can specify for class name, see \textit{z/OS MVS Program Management: User's Guide and Reference}.

\textbf{Note:} SPZAP does not fold lowercase input to uppercase; be sure to enter class-name in the correct case.

\textbf{Figure 202} shows how to use the DUMPT and DUMP control statements to inspect CSECTs in a program object with multiple text classes in a z/OS UNIX file:

```
//ZAPDUMP EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*  
//SYSLIB DD PATH='/u/mydir',PATHDISP=(KEEP,KEEP)  
//SYSIN DD *  
DUMPT hwz CEEMAIN C_CODE  
DUMP hwz CEESTART B_*  
DUMP hwz ALL C_*  
/*  
```

\textit{Figure 202. Example: Using the DUMP control statement with a class name}

**SYSLIB DD Statement:** Defines the z/OS UNIX directory that contains the module hwz in which the CSECTs to be inspected reside.

**SYSIN DD Statement:** Defines the input stream that contains the SPZAP control statements.

**DUMPT hwz CEEMAIN C_CODE:** This control statement requests that the contents of class C_CODE in csect CEEMAIN in module hwz be dumped.

**DUMP hwz CEESTART B_*:** This control statement requests that the contents of all classes in csect CEESTART in module hwz whose class name begins with B_ be dumped.
DUMP hwz ALL C_*: This control statement requests that the contents of all classes whose class name begins with C_ in any csect in module hwz be dumped.

IDRDATA xxxxxxxx
Causes SPZAP to place up to eight bytes of user data into the SPZAP CSECT identification record of the load module; this is only done if a REP operation associated with a NAME statement is performed and the load module was processed by the linkage editor to include CSECT identification records. The variable is:

xxxxxxxx
Eight (or fewer) bytes of user data (with no embedded blanks) that are to be placed in the user data field of the SPZAP IDR of the named load module. If more than eight characters are in the variable field, only the first eight characters will be used.

Note: The IDRDATA statement is valid only when used in conjunction with the NAME statement. It must follow its associated NAME statement, or the BASE statement associated with a NAME statement, and precede any DUMP, DUMPT, ABSDUMP or ABSDUMPT statement. IDRDATA statements associated with CCHHR statements will be ignored.

NAME member [csect | *] [class-name]
Identifies a CSECT in a load module member of a PDS, a program object member of a PDSE, or a z/OS UNIX file that is to be the object of subsequent VERIFY, REP, SETSSI, or IDRDATA operations. The variables are:

member
The member name of the load module belonging to a PDS, the program object belonging to a PDSE, or a z/OS UNIX file that includes the CSECT that contains the data to be inspected or modified. The load module or the program object must be a member of the data set defined by the SYSLIB DD statement.

csect | *
The name of the particular control section that contains the data to be verified or replaced. If you omit this variable, or, for program objects only, code "*", SPZAP assumes that the first CSECT in the load module contained in a PDS, the program object contained in a PDSE, or a z/OS UNIX file is the one to be used. If there is only one CSECT in the load module or program object, this variable is not necessary.

If you specify a CSECT name that SPZAP does not find in the member you name, then SPZAP does not perform any requested processing. Instead, it produces hexadecimal dumps of all CSECTs in the member. (The class of text dumped is specified on the class-name variable, and the default is B_text.)

class-name
Indicates, for program objects only, the class of text that you want to modify. The default is B_text. If you want to omit the CSECT name and supply a class-name, code a single asterisk for the CSECT name, followed by the class-name.

For information about the values you can specify for class name, see
Note: SPZAP does not fold lowercase input to uppercase; be sure to enter class-name in the correct case.

Note that you can define more than one NAME statement in your input to SPZAP. However, the VERIFY, REP and SETSSI statements associated with each NAME statement must immediately follow the NAME statement to which they apply.

NAME member
Identifies the member of a data library that is to be the object of subsequent VERIFY, REP, ABSDUMP, ABSDUMPT or RECORD operations. The variable is:

member
The member name of a data library whose contents are to be displayed, verified and/or replaced.

RECORD nnnnnnnn
This statement identifies a particular record in a member of a PDSE data library and must follow a NAME member statement where member specifies the name of the member. The combination of NAME and RECORD defines the record for which VER|VERIFY and possible REPs are to be performed.

nnnnnnnn consists of 1 to 8 decimal digits and specifies the relative record of interest. Leading zeroes are ignored. For example, the first record of a member may be specified as 1 or 01 or 00000001.

REP offset data
Modifies data at a specified location in a CSECT or physical record that was previously defined by the NAME, NAME/RECORD combination, or CCHHR statement. The data specified on the REP statement will replace the data at the record or CSECT location stipulated in the offset variable field.

SPZAP issues message AMA122I to record the contents of the specified location as they were before the change was made.

Note: IBM recommends that, before you replace any data, you always use VER/VERIFY to make sure that the contents you are going to change with the REP function are what you expect. The offset and length that you specify on the VER/VERIFY statement, however, do not need to match any following REP statement exactly; a single successful VERIFY can validate multiple following REP statements.

offset
Provides the hexadecimal displacement of data to be replaced in a CSECT or data record. This displacement need not address a fullword boundary, but it must be specified as a multiple of two hexadecimal digits (0D, 02C8, 001C52).

If the offset value is outside the limits of data record (physical block) or CSECT being modified, the replacement operation will not be performed. When replacing data in a record with a key, the length of the key should be considered in the calculation of the displacement; that is, offset zero is the first byte of the key, not of the data.

data
Defines the bytes of data to be inserted at the location. As with the offset variable, the number of bytes of data defined must be specified as a multiple of two hexadecimal digits. If desired, the data within the variable would be specific.
may be separated by commas (never blanks); but again, the number of
digits between commas must also be a multiple of two. For example, a
REP data variable may look like this:

4160B820 (without commas)
or like this:
4160,0820 (with commas).

If all the data to be modified does not fit into one REP statement (72
bytes), you can code another REP statement.

Note:
1. Remember that SPZAP automatically updates the system status index (SSI)
when it successfully modifies the CSECT named or implied on the previous
NAME statement.
2. If you are performing multiple VERIFY and REP operations on a CSECT,
make sure that all the VERIFY statements precede all the REP statements.
This procedure ensures that all REP operations are ignored if one VERIFY
reject occurs.
3. You are not required to supply a VERIFY statement before the first REP
statement; however, when SPZAP encounters a VERIFY statement, it must
be satisfied before SPZAP processes any following REP requests.
4. When you access a record in the VTOC (for example, the data set control
block (DSCB)) for modification, SPZAP issues the message AMA117D to the
console. No message is issued, however, when an ABSDUMPT operation is
performed on the VTOC.

SETSSI xxyynnnn
Places user-supplied system status information in the directory entry for the
load module member in a PDS or program object member in a PDSE. The SSI
entry must have been created when the load module or program object
member was link edited. The variable is:

xxyynnnn
Four bytes of system status information the user wishes to place in the SSI
field for this member. Each byte is supplied as two hexadecimal digits
indicating the following:

xx - change level
yy - flag byte
nnnn - modification serial number

If SPZAP detects an error in any previous VERIFY or REP operation, the
SETSSI function is not performed.

Note: Because all bits in the SSI entry are set (reset) by the SETSSI statement,
be very careful when using it to avoid altering the vital maintenance-status
information. SPZAP issues message AMA122I to record the SSI as it was before
the SETSSI operation was performed. See “Updating the System Status Index
(SSI)” on page 645.

{VERIFY|VER} offset expected-content
Causes the data at a specified location within a CSECT or physical record to be
compared with the data supplied in the statement.

offset
The hexadecimal displacement of data to be inspected in a CSECT or
record. This displacement does not have to be aligned on a fullword
boundary, but it must be specified as a multiple of two hexadecimal digits, such as 0D, 021C, 014682. If this offset value is outside the limits of the CSECT or data record defined by the preceding NAME, NAME/RECORD, or CCHHR statement, the VERIFY statement is rejected. If this offset value plus the length of the expected-content string is outside the limits of the CSECT or record defined by the preceding NAME, NAME/RECORD combination, or CCHHR statement, the VERIFY statement is rejected and flagged in error. When inspecting a record with a key, the length of the key should also be considered in the calculation of the displacement; that is, offset zero is the first byte of the key.

**expected-content**

Defines the bytes of data that are expected at the specified location. As with the offset variable, the number of bytes of data defined must be specified as a multiple of two hexadecimal digits. If desired, the data within the parameter may be separated by commas (never blanks), but again, the number of digits between commas must also be a multiple of two. For example, expected content might look like this:

5840C032 (without commas),
or like this:
5840,C032 (with commas)

If all the data does not fit into one VERIFY statement (80-byte logical record), then another VERIFY statement must be defined.

**Note:** If the two fields being compared are not in agreement, that is, if the VERIFY operation is rejected, no succeeding REP or SETSSI operations are performed until the next NAME or CCHHR control statement is encountered. SPZAP provides a formatted dump of each CSECT or record for which a VERIFY operation failed.

**Reading SPZAP output**

SPZAP provides two different dump formats for the purpose of checking the data that has been verified or replaced. These dumps (written to the SYSPRINT data set specified by the user) may be of the formatted hexadecimal type or the translated form. Both formats are discussed below in detail with examples showing how each type will look.

**Formatted hexadecimal dump:** When DUMP or ABSDUMP is the control statement used, the resulting printout is a hexadecimal representation of the requested data. Figure 203 on page 663 gives a sample of the formatted hexadecimal dump. A heading line is printed at the beginning of each block. This heading consists of the hexadecimal direct access address of the block (ABSDUMP only), the length of the record, the class of text (program objects only), and the names of the member and the CSECT that contain the data being printed (if the dump is for specific CSECT or load module). Each printed line thereafter has a three-byte displacement address at the left, followed by eight groups of four data bytes each. The following message is printed under the last line of the dump printout:

AMA113I COMPLETED DUMP REQUIREMENTS

**Translated dump:** The control statements DUMPT and ABSDUMPT also provide an operation code translation and an EBCDIC representation of the data contained in the dump. Not all characters are translated to EBCDIC, only upper case and a few special characters are translated. Others, such as lowercase letters are not
translated and their translations are substituted by periods.

<table>
<thead>
<tr>
<th><strong>CCHR</strong></th>
<th><strong>RECORD LENGTH</strong></th>
<th><strong>MEMBER NAME</strong></th>
<th><strong>IEHMVSN</strong></th>
<th><strong>CSECT NAME</strong></th>
<th><strong>IEHMVSN</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>4F7FO014</td>
<td>0ECE2E05</td>
<td>60E6D9C1</td>
<td>D760E4D7</td>
<td>60E6B000</td>
</tr>
<tr>
<td>00002000</td>
<td>D0485000</td>
<td>10D05100</td>
<td>D0918D1</td>
<td>98100000</td>
<td>9200000C</td>
</tr>
<tr>
<td>00004000</td>
<td>9201C2F4</td>
<td>D20E2C25</td>
<td>C2F9108</td>
<td>C20C4710</td>
<td>90E69500</td>
</tr>
<tr>
<td>00006000</td>
<td>9201C2FC</td>
<td>D20E3C320</td>
<td>C31C95FF</td>
<td>C32A1770</td>
<td>98A41180</td>
</tr>
<tr>
<td>00008000</td>
<td>951E89ED</td>
<td>9846E620</td>
<td>9670280</td>
<td>C640700</td>
<td>45109898</td>
</tr>
<tr>
<td>0000A000</td>
<td>0A1495FF</td>
<td>C3274780</td>
<td>90A9108</td>
<td>C20C4710</td>
<td>91668520</td>
</tr>
<tr>
<td>0000C000</td>
<td>C20E4710</td>
<td>9068E110</td>
<td>C2084710</td>
<td>9568B0700</td>
<td>45109806</td>
</tr>
<tr>
<td>0000E000</td>
<td>0A1447FF</td>
<td>E1FC4789</td>
<td>909A0780</td>
<td>C040700</td>
<td>45109808</td>
</tr>
<tr>
<td>00010000</td>
<td>005210000</td>
<td>90201000</td>
<td>0A1495FF</td>
<td>C334780</td>
<td>96DC41A0</td>
</tr>
<tr>
<td>00012000</td>
<td>C2F94200</td>
<td>C30094FF</td>
<td>A0429701</td>
<td>C2094780</td>
<td>91689120</td>
</tr>
<tr>
<td>00014000</td>
<td>0148E861</td>
<td>01748E90</td>
<td>004E45E5</td>
<td>C0839101</td>
<td>10204710</td>
</tr>
<tr>
<td>00016000</td>
<td>01041110</td>
<td>F0000A08</td>
<td>1A444340</td>
<td>C2254810</td>
<td>22458310</td>
</tr>
<tr>
<td>00018000</td>
<td>90ACE658</td>
<td>03041000</td>
<td>47F91192</td>
<td>D5053010C</td>
<td>10044781</td>
</tr>
<tr>
<td>0001A000</td>
<td>00011B14</td>
<td>41400001</td>
<td>D0301000</td>
<td>301195FF</td>
<td>30042710</td>
</tr>
<tr>
<td>0001C000</td>
<td>D2051004</td>
<td>301C1333</td>
<td>043096FD</td>
<td>D201100A</td>
<td>96FC1310</td>
</tr>
<tr>
<td>0001E000</td>
<td>96FCD2001</td>
<td>100AB9FC</td>
<td>5010C224</td>
<td>4240C224</td>
<td>9110C202</td>
</tr>
<tr>
<td>00020000</td>
<td>47F709238</td>
<td>501C0224</td>
<td>90830102</td>
<td>47071126</td>
<td>D20196FC</td>
</tr>
<tr>
<td>00022000</td>
<td>00011032</td>
<td>47709282C</td>
<td>41220001</td>
<td>042096FC</td>
<td>D20110A0</td>
</tr>
<tr>
<td>00024000</td>
<td>01015822</td>
<td>00286183</td>
<td>00100580</td>
<td>92B44780</td>
<td>92B81233</td>
</tr>
<tr>
<td>00026000</td>
<td>91023030</td>
<td>47F927000</td>
<td>47F502028</td>
<td>D2023295</td>
<td>9205820</td>
</tr>
<tr>
<td>00028000</td>
<td>D25020400</td>
<td>C4122</td>
<td>000C50220</td>
<td>C2099640</td>
<td>C20947F0</td>
</tr>
</tbody>
</table>

**Figure 203. Sample formatted hexadecimal dump**

**Figure 204 on page 664** shows the format of the translated dump. The first byte of each halfword of data is translated into its mnemonic operation code equivalent, provided such a translation is possible. If there is not equivalent mnemonic representational value to be given, the space is left blank. This translated line of codes and blanks is printed directly under the corresponding hexadecimal line. An EBCDIC representation of each byte of data is printed on two lines to the right of the corresponding line of text, with periods substituted for those bytes that have been set not to be translated to valid printable characters.
DUMPT IEANUC05 SUTFPL59

**CCHHR- 01AB000416  RECORD LENGTH- 000068  MEMBER NAME IEANUC05  CSECT NAME SUTFPL59
000000 47F0 F01C 16E2 E4E3 C6D7 D3F6 F940 F9F8 01F0 F540 F8C2 C2F6 F6F0 F600 98EC D00C 8B3C 0000 D010
000020 BC SRP OR MVZ CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CM
000040 8850 41F0 0000 5800 0064 18A1 5000 0000 50A0 0000 50F1 0000 5010 0000 8385 0000 0000 0000 8385
000060 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
AMA113I COMPLETED DUMP REQUIREMENTS
AMA100I AMASPZAP PROCESSING COMPLETED

Figure 204. Sample translated dump

Figure 205 on page 665 shows CSECT output (obtained through DUMP/DUMPT) for a program object module.

**Note:**
1. There are no **CCHHR** values. The program management binder manages its own DASD storage and returns no physical location.
2. **RECORD LENGTH**: indicates length of the CSECT or module, not the length of the physical record containing the CSECT or module.
3. Program management binder returns no text for named or unnamed common areas. The length of the common section will be indicated. Message AMA152I indicates that no text has been returned.
4. SPZAP displays MEMBER NAME and CSECT NAME on as many lines as necessary. The names can be as long as 1024 characters.
5. SPZAP labels common storage in a program object with the tag COMMON NAME instead of CSECT NAME. Named common displays that name. Unnamed common is flagged as $BLANK COMMON. Private code is displayed with the subheading CSECT NAME: $PRIVATE CODE.
6. **UNINITIALIZED DATA SKIPPED** may appear.
7. IGWSPZAP is the part of SPZAP that receives control when accessing or updating program objects in a PDS/E or z/OS UNIX file, or data in a PDS/E. Listings and messages refer to AMASPZAP when processing a PDS or IGWSPZAP when processing a PDSE or a z/OS UNIX file.
**Figure 206 on page 666** shows output for a member of a PDSE data library.

ABSDUMPT 0001 0500 would have been preceded by a **NAME membername** statement (not shown).

**Note:** There are no **CCHHR** values. RECORD NUMBER: shows the 8 digit value of the relative record number of the member being printed. RECORD LENGTH: shows the length of the record, while MEMBER NAME: shows the member name as it appears on the **NAME membername** statement.
Figure 206. Sample translated dump for PDSE data library
Chapter 18. AMATERSE: Pack and unpack a data set

AMATERSE is a service aid program that operates in problem state. You can use AMATERSE to pack a data set before transmitting a copy to another site, typically employing FTP as the transmission mechanism. A complementary unpack service is provided to create a similar data set at the receiving site.

Note: IBM also supports z/OS Problem Documentation Upload Utility (PDUU), which is a utility that sends large amounts of documentation to IBM. AMATERSE is useful for compressing (packing) and unpacking relatively small amounts of service data, but is incompatible with PDUU (output and input), and offers no data transfer capability. For information about PDUU, see Chapter 19, "AMAPDUPL: Problem Documentation Upload Utility," on page 673.

AMATERSE supports Direct Access Storage Device (DASD) and tape data sets:
- Sequential data sets, which can be unpacked by VM TERSE.
- Partitioned data sets (PDS), and partitioned data sets extended (PDSE) that do not contain program objects.
- Large format (DSNTYPE=LARGE) data sets.
- Fixed and variable, blocked and unblocked, spanned and unspanned record formats (RECFM) =F, FB, FBS, V, VB, VBS where the logical record length (LRECL) is less than 32K; and RECFM=VBS where the LRECL may be more than 32K but less than 64K.
- Data sets with records containing ISO/ANSI or machine code printer control characters.
- Placement of data sets into cylinder-managed space is also supported.

Planning for AMATERSE

AMATERSE is an application that prepares diagnostic materials, such as z/OS dumps and traces, for transmission to IBM and independent software vendor sites. When the materials arrive, AMATERSE also provides a means to create similar data sets to support diagnosis of problems.

If you have previously used the TRSMAIN utility (see http://techsupport.services.ibm.com/390/trsmain.html), note the following changes made to prepare AMATERSE for formal inclusion in z/OS:
- Use AMATERSE as the preferred application program name rather than TRSMAIN. TRSMAIN ships as an alias entry point to AMATERSE.
- Use the replacements for the DDNAMES, which are SYSUT1 and SYSUT2.
  When the TRSMAIN entry point of AMATERSE is invoked, DDNAMES INFILE and OUTFILE remain as the defaults.
- AMATERSE is in MIGLIB, a library that is part of the link list. No STEPLIB DDNAME is necessary to invoke AMATERSE.
- In nearly all cases, you can use AMATERSE, the TRSMAIN utility, and VM terse interchangeably. See "Restrictions for AMATERSE" on page 670 for the exceptions to this rule.
Invoking AMATERSE

Figure 207 shows an example of the JCL to invoke AMATERSE. Lower case text reflects the data that you must alter.

```plaintext
//jobname JOB ...
//stepname EXEC PGM=AMATERSE,PARM=aaaaa
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=12901)
//SYSUT1 DD DISP=bbb,DSN=your.input.dataset.name
//SYSUT2 DD DISP=ccc,DCB=ddd,DSN=your.output.dataset.name,
// SPACE=space_parameters
//SYSUT3 DD DISP=ccc,SPACE=space_parameters
```

Figure 207. Example: AMATERSE JCL

Specifying the JCL statements for AMATERSE

If you have previously used the TRSMAIN program to invoke AMATERSE, you can continue using it along with the old DDNAMES. However, if you choose to use AMATERSE instead of TRSMAIN, realize that the DDNAMES are changed: SYSUT1 replaces INFILE and SYSUT2 replaces OUTFILE.

A missing SYSUT1 DD statement results in an RC X’10’ error message:

RC X‘10’, AMA522E INPUT DATASET HAS AN UNSUPPORTED DATASET ORGANIZATION

A missing SYSUT2 DD statement will result in an RC X’28’ error message:

RC X‘28’, AMA518E UNABLE TO OPEN OUTPUT DATASET

AMATERSE requires the following JCL statements. The required DD statements are SYSPRINT, SYSUT1 and SYSUT2. SYSUT3 is optional. Replace aaaaa in the example with one of the following values:

**EXEC**

Marks the beginning of the job.

**PACK**

Compresses records in a data set so that the output is known as the simple format.

**SPACK**

Compresses records in a data set so that the output is known as the complex format. The SPACK option is more time-consuming than the PACK option by a factor of about three, but in many cases produces much smaller output.

**Note:** A data set compressed by either PACK or SPACK should not be modified in any way. If such a data set is modified, the UNPACK routines are unable to reconstruct the original data set.

**UNPACK**

Reverses the PACK or SPACK operation. If you inadvertently packed a data set multiple times, restore it using the UNPACK function the same multiple number of times.

**SYSPRINT statement**

This DD defines where all messages from the program are sent. It must be RECFM=FBA and an LRECL between 121 and 133. Any block size that is a legal multiple of the LRECL is supported.
Note: The DCB information does not have to be specified on the DD statement and will default to the correct values.

SYSUT1 statement
This DD defines the data set to be compressed if PACK or SPACK parameter is specified on the EXEC statement. If UNPACK is specified, then it defines the compressed data set to be restored. See “Restrictions for AMATERSE” on page 670 for special considerations.

Note: If TRSMAIN entry point is used, INFILE statement is used instead of SYSUT1.

SYSUT2 statement
This DD defines the data set to receive the compressed output. If you specify PACK or SPACK on the EXEC statement, this is the data set that receives the compressed output. If you specify UNPACK, this is the data set that receives the restored output. See “Allocation considerations” on page 671 and “Space considerations” on page 671.

SYSUT3 statement
This optional DD defines the temporary data set to use when the PACK or UNPACK operation is performed against large PDS data sets. This data set acts as an intermediate form between PDS and PACKED data set. Only the DISP and SPACE parameters are necessary to be supplied by JCL. If the SYSUT3 DD statement is missing, AMATERSE allocates this data set by itself and deletes it automatically after AMATERSE ends.

AMATERSE return codes
When AMATERSE ends, one of the following return codes is placed in general purpose register 15:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>4</td>
<td>Error in file operation.</td>
</tr>
<tr>
<td>8</td>
<td>Error in file operation.</td>
</tr>
<tr>
<td>10</td>
<td>Unsupported data set format.</td>
</tr>
<tr>
<td>12</td>
<td>Operation cannot be performed with the specified data set.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid input specified.</td>
</tr>
<tr>
<td>20</td>
<td>Invalid input specified.</td>
</tr>
<tr>
<td>24</td>
<td>Severe error in file operation.</td>
</tr>
<tr>
<td>28</td>
<td>Severe error occurred during file open.</td>
</tr>
<tr>
<td>32</td>
<td>Invalid device.</td>
</tr>
<tr>
<td>36</td>
<td>Buffer storage obtain failure.</td>
</tr>
<tr>
<td>64</td>
<td>Severe error. Abend with 1111.</td>
</tr>
<tr>
<td>99</td>
<td>System or user abend occurred.</td>
</tr>
</tbody>
</table>

Invoking AMATERSE from a problem program
To invoke AMATERSE from a program, specify the following information:
- PACK, SPACK, or UNPACK on the PARM parameter of the EXEC statement
AMATERSE

- The DDNAMES of the data sets to be processed by the AMATERSE program, if the calling program is to override the DDNAMES.

Figure 208 shows how to invoke AMATERSE using alternate DDNAMES:

```plaintext
* Invoke AMATERSE to perform SPACK processing
  LINK EP=AMATERSE,PARAM=(PARM,DDNAMES),VL=1
  ...
* Request SPACK option
  PARM DS OH EXEC PARM= data
  PARMLEN DC Y('PARMTEXT') Length of data
  PARMTEXT DC C'SPACK' AMATERSE processing option
* Request MYPRINT, MYSYSUT1, and MYSYSUT2 instead of SYSPRINT, SYSUT1, and SYSUT2 respectively
* DDNAMES DS OH DDNAME override data
  DDNAMEL DC Y('DDNAMES9-DDNAMET') Length of data
  DDNAME9 DS 0C DDNAME override list
  DC SXL8'0' Not used by AMATERSE
  SYSPRINT DC CL8'MYPRINT' Instead of SYSPRINT
  DC XL8'0' Not used by AMATERSE
  SYSUT1 DC CL8'MYSYSUT1' Instead of SYSUT1
  SYSUT2 DC CL8'MYSYSUT2' Instead of SYSUT2
  SYSUT3 DC CL8'MYSYSUT3' Instead of SYSUT3
  DDNAMES DS 0C End of list
```

Figure 208. Example: AMATERSE JCL from a problem program

For a description of the parameter list, see the topic on "Invoking Utility Programs from an Application Program" in z/OS DFSMSdfp Utilities. From that information, AMATERSE supports the optionsaddr and ddnameaddr, but not hdingaddr.

AMATERSE also supports DDNAMES SYSPRINT, SYSUT1, SYSUT2, and SYSUT3, but not DDNAMES SYSIN and SYSUT4, which are shown in the "ddname parameter list (DDNMESTOP)" in that information.

Additional considerations for AMATERSE

When using AMATERSE, certain restrictions apply as well as allocation and space considerations, as well as restrictions to consider when you are using AMATERSE.

Restrictions for AMATERSE

The following restrictions apply to AMATERSE:

- VSAM data sets and direct (DSORG=DA) data sets are not supported.
- Data sets with keys (KEYLEN) are not supported.
- A partitioned data set (PDS) compressed by AMATERSE on MVS cannot be unpacked by VM TERSE. This results in a 1007 or 1009 return code from VM TERSE. A PDS must be compressed to a DASD.
- Partitioned data sets extended (PDSE) containing program objects are not supported.
- AMATERSE handles data sets with a LRECL of more than 32K but less than 64K only when RECFM=VBS DASD data sets are processed.
- A data set with the FB record format can be packed and unpacked to a FBS data set. However, during the UNPACK operation, extending a non-empty output data set with DISP=MOD is not possible because this results in a FB data set. An error message is issued for this.
AMATERSE

- AMATERSE does not support large block interface (LBI).

**Allocation considerations**

The data set compressed by AMATERSE (produced by PACK or SPACK) must be of fixed or fixed-blocked record format (RECFM) with a record length (LRECL) of 1024 and any legal block size (BLKSIZE). These values do not have to be specified explicitly on the DD statement.

The data set restored by AMATERSE (produced by UNPACK) must match the original RECFM and LRECL. Leave the DCB information off the DD statement for AMATERSE program to set it up. If it unpacks to an already existing data set then the DCB parameters are checked for compatibility. RECFM must be the same in all cases except for Variable to Undefined and Undefined to Variable. If you specify the DCB parameters to force data that was originally variable (V) format into undefined (U) format, or conversely, a warning message is written and the operation is performed.

**Space considerations**

When allocating space for the output data set SYSUT2, you must estimate the required size information:

- For the PACK or SPACK option a data set compressed by AMATERSE is expected to be about half the size of the original. Allocate more than expected and use the RLSE (partial release parameter) function of the SPACE value to release the unused portion back to the system.

- For the UNPACK option: If the data set contains random data, allocate three to five times the size of the compressed data set. If the data set contains Listing, Document, or Messages type data, allocate five to ten times the size of the compressed data set.

If there is not enough allocated space, the program issues ABEND X'B37':

*Not Enough Space Allocated for the Output Data Set*
AMATERSE
Chapter 19. AMAPDUPL: Problem Documentation Upload Utility

The IBM z/OS Problem Documentation Upload Utility (PDUU) is a utility that sends large amounts of documentation in a more efficient manner than using File Transfer Protocol (FTP) to send one large data set to IBM FTP sites. This utility sections the input data set (such as stand-alone dump data set) into smaller data sets that are compressed and sent in parallel using multiple, simultaneous FTP sessions. This results in shorter transmission time for very large data sets. You can also encrypt the data sets.

There are two work files for each FTP session (the "A" file and the "B" file). Each "A" work file is filled by copying records from the input file. When the "A" file is full, the FTP sessions are started in parallel. At the same time, each "B" work file is filled by copying records from the input file. When the "B" file is full and the transfer of the "A" file is complete, transfer of the next "B" file starts. This process continues between the "A" and the "B" files, until everything in the input file is sent.

You can have up to 20 FTP sessions running simultaneously. The work data sets are dynamically allocated and can range in size from 1 MB to 9,999 MB. You can experiment to see what works best in your environment, but here are some guidelines:

- Start with three or four parallel FTP sessions. Too many parallel FTP sessions can saturate the network link.
- Use medium size work data sets.

If the work data sets are very small in relationship to the input data set, you can end up with too many files on the IBM FTP sites. For example, if you are sending a 100 GB z/OS stand-alone dump and make the work data set size 1 MB, you create 100,000 files on the IBM FTP site, which exceeds the IBM limit of 99,999 files. This also causes a lot of delay by starting and stopping the FTP sessions for each file.

If the work data sets are very large in relationship to the input file size, the amount of overlap time is decreased. When the program first starts, it must fill the "A" work files before it starts transmitting any data, which means the copy time is not overlapping with data that needs to be sent through FTP. For example, if you were sending a 1 GB dump and you set the work data set size to 1 GB (1,000 MB), there is no overlap between copying the records and sending the work files.

If the input data set is a partitioned data set (PDS/PDSE), the PDUU unloads it first into a sequential data set using the IECOPY utility.

The PDUU typically compresses the input data before it is written to the work data sets, therefore it is not necessary to use a tool such as AMATERSE or TRSMAIN to compress the input data set before using the PDUU to send it to the IBM FTP site. If a file is tersed, PDUU will copy and send the files without further compression. Overall performance of using AMATERSE with PDUU to send the file takes longer than if an un tersed file is compressed and sent using PDUU. In addition, 192-bit triple Data Encryption Standard (DES) can be requested by using the
**CIPHER_KEY** keyword. Without the keyword, the data is just compressed. With the keyword, the data is compressed, and then encrypted.

Encryption is provided by the CP Assist for Cryptographic Functions (CPACF), DES/TDES Enablement (feature 3863) and is available on all processors starting with the z990 (2084) and z890 (2086). CPACF, feature 3863 enables clear key DES and TDES instructions on all supported CPs. The encryption algorithm used for 3DES is documented in *z/Architecture Principles of Operation*.

---

**Planning to use PDUU**

Use the PDUU as the primary utility for sending large volumes of documentation, such as stand-alone dumps, to the IBM FTP site. The encryption capability ensures that the transfer occurs in a secure manner. AMATERSE is useful for compressing (packing) and unpacking relatively small amounts of service data, but is incompatible with PDUU (output and input), and offers no data transfer capability. If you have previously used the PDUU, shipped as the MTFTPS stand-alone program from [http://www-01.ibm.com/support/docview.wss?uid=isg3T1011823](http://www-01.ibm.com/support/docview.wss?uid=isg3T1011823), you must understand the following changes made to package the PDUU utility as part of z/OS:

- The PDUU utility name is AMAPDUPL; however, MTFTPS ships as an alias entry point to AMAPDUPL.
- AMAPDUPL resides in SYS1.MIGLIB (which must be a data set in the LNKLST concatenation), so a STEPLIB DDNAME is not necessary to invoke AMAPDUPL.
- AMA messages are described in *z/OS MVS System Messages, Vol 1 (ABA-AOM)*.

**Prerequisites and restrictions for PDUU**

You must properly configure z/OS Communications Server to use the FTP client program. See the topic on Transferring files using FTP in *z/OS Communications Server: IP Configuration Guide*. The PDUU uses active FTP mode as the default, unless another mode is requested with the corresponding FTP subcommands defined in FTPCMDS data set.

The PDUU supports the following types of data sets:

- Members of partitioned data sets (PDS) and partitioned data sets extended (PDSE)
- Large format (DSNTYPE=LARGE) and traditional sequential data sets
- Extended format sequential data sets
- Fixed, variable, and undefined-length, blocked and unblocked, spanned and unspanned record formats (RECFM) = F, FB, FBS, V, VB, VS, VBS, U
- Data sets with records containing ISO/ANSI or machine code control characters
- Data sets in cylinder-managed space.
- Partitioned data sets (PDS) and partitioned data sets extended (PDSE).

PDUU does not support the following types of input data sets:

- Large block interface (LBI) (no BLKSIZE value).
- VSAM and direct (DSORG=DA) data sets
- Data sets with keys (KEYLEN)
- z/OS UNIX files
- Concatenated data sets of any type
The JCL statements for the PDUU are:

**SYSPRINT**

The data set can be either SYSOUT or a sequential data set. The data set must be RECFM=FB, LRECL=134. For additional details, see “Prerequisites and restrictions for PDUU” on page 674.

**SYSUT1**

The sequential or partitioned data set to transfer to IBM. For additional details, see “Prerequisites and restrictions for PDUU” on page 674.

**SYSUT2**

This data set is optional and can be used only when transferring partitioned data sets (PDS/PDSE). It defines a sequential unload data set for IEBCOPY output produced during the unload operation. If the SYSUT2 statement is omitted the unload data set will be allocated dynamically. This parameter can be used if you want to directly control the allocation of the unload data set, for example, to specify a particular volume or certain amount of volume space. For additional details about usage and allocation parameters of the unload data set see the topic on Unloading (Backing up) Data Sets in z/OS DFSMShsm Utilities. See also Example 9: Using SYSUT2 to allocate an unload data set on page 682.

**SYSIN**

A sequential data set that uses the following control statements. The data set must be RECFM=FB, LRECL=80. For additional details, see “Prerequisites and restrictions for PDUU” on page 674.

PDUU is managed through the following SYSIN statements with these guidelines:

- Use an asterisk (*) in the first column of each comment line to indicate comments.
- Keywords must start in column one.
- Use control statements that are in form VERB=OPERAND.
- Mixed case verbs and operands are allowed.
- The operand starts in the column after the equal sign and goes to the first blank column except TARGET_SYS, DIRECTORY, CIPHER_KEY, ACCOUNT, USERID, and PASSWORD, which can contain blanks.
- Anything after the first blank is ignored except for any operands that can contain blanks. In those cases, do not use blanks from column one to the end of the operand.
- Control statements can be coded on one or more (up to 6) consecutive records. Control statements with operands that allow blanks must not extend beyond column 71, but can continue on the following record in columns 16 through 71. Columns 1 through 15 of the continuation record must be blank. See Example 10: Using a multiple record control statement in SYSIN” on page 683. Control statements with operands that do not allow blanks can occupy columns 1 through 80.

**TARGET_SYS**

The name of the TCP/IP system to transfer the files to using FTP. One through 256 characters, dotted decimal format is allowed, no default value, can contain blanks, and it must be specified.

If using a proxy server, this should be the name of the proxy server.
You can include additional FTP command parameters on the TARGET_SYS parameter by using the z/OS UNIX specifications as shown in the topic FTP command -- Entering the FTP environment in z/OS Communications Server: IP User’s Guide and Commands. For example, to trace output (-d) and use a specific ftpdata_filename (-f"//'WES.MYFTP.DATA'":)

```
TARGET_SYS=-d -f"//'WES.MYFTP.DATA'": testcase.boulder.ibm.com
```

Use the -p parameter to specify an alternate IP stack.

**USERID**

The user ID on the target system that is used to send the files. One through 64 characters, no default value, does not have to be specified, and can contain imbedded blanks. If USERID and PASSWORD are not supplied and NETRCLEVEL=2, the values from the NETRC data set is used for the FTP sessions.

If using a proxy server, this can be the full login to the remote system in the format `userid@remote.system.name`.

**PASSWORD**

The password for the USERID on the target system. One through 64 characters and the default value is blanks.

If using a proxy server, this can be the USERID and PASSWORD for the proxy server in the format `userid@password`.

**ACCOUNT**

The account data that is sent when an FTP session is started. One through 64 characters with no default value.

**TARGET_DSN**

The variable (descriptive) portion for the file names on the target system. One through 64 characters, no default value, and it must be specified.

**WORK_DSN**

The prefix for the data set names of work files on the sending system. One through 40 characters, no default value, and it must be specified. The work data sets are large format, sequential, data sets and cannot have the compaction attribute.

**Note:** Because work files are dynamically allocated with large format and do not support compressed format, if you specify data class for work files with the compaction attribute or N, the following message is issued for all work files:

`IGD17163I COMPRESSION REQUEST NOT HONORED FOR DATA SET work_file_dsnname BECAUSE DATA SET CHARACTERISTICS DO NOT MEET COMPRESSION CRITERIA, ALLOCATION CONTINUES`

**WORK_DSN_SIZE**

The maximum size of the work files in megabytes. One through four decimal digits and the default is 100 MB.

**KEEP_WORK**

The parameter to save the work data sets that are dynamically allocated for each FTP session. If you omit the KEEP_WORK parameter, the program does not save the work data sets. Y is the only value for the KEEP_WORK parameter.

**Note:** Only specify this parameter when debugging a problem.
z/OS Problem Documentation Upload Utility

**DATAclas**
The data class to use when allocating the work files on the sending system.
One through eight characters with no default value.

**MGMTCLAS**
The management class to use when allocating the work files on the
sending system. One through eight characters with no default value.

**STORCLAS**
The storage class to use when allocating the work files on the sending
system. One through eight characters with no default value.

**CC_FTP**
The number of parallel FTP sessions to use when transmitting the files.
One or two decimal digits, the value must be between one and 20, and the
default is two.

**DIRECTORY**
The directory on the target system where the files will be sent with FTP.
One through 32 characters, with no default, can contain blanks, and you
must specify the directory.

**PMR**
The PMR number with which this file is associated. This field must be 13
numeric or uppercase characters, specify in the form `xxxxx.yyy.zzz`, and
define the variables as:

<table>
<thead>
<tr>
<th>Field</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXXX</td>
<td>PMR Number</td>
<td>34143</td>
</tr>
<tr>
<td>YYY</td>
<td>Branch office</td>
<td>055</td>
</tr>
<tr>
<td>ZZZ</td>
<td>IBM Country Code</td>
<td>724</td>
</tr>
</tbody>
</table>

**CIPHER_KEY**
The encryption key to use for 192-bit triple DES encryption. The 24
characters following `CIPHER_KEY=` are used as the key. The key can
include imbedded and/or trailing blanks. For example, `CIPHER_KEY=HERE IS CIPHER KEY IN 24` or `CIPHER_KEY=Shortkey`. If you do not specify
`CIPHER_KEY=`, no encryption is performed. If you encrypt the data set
using `CIPHER_KEY`, you must provide IBM with the encryption key so
they can perform problem diagnosis.

**Note:** If `CIPHER_KEY=` is followed by 24 blanks, the file will be encrypted
with a key of 24 blanks.

**NO_FTP**
A value of 'Y' specifies that PDUU compress, optionally encrypt, separate
files into parts, and move the part to a local z/OS Unix Systems Services
directory, without FTP'ing the files. PDUU uses TSO services to send data
sets to USS directories. When you specify `NO_FTP=Y`:

- `TARGET_SYS`, `CC_FTP`, `USERID`, and `PASSWORD` are not required
- `CC_FTP` settings are ignored and set to 1
- `FTPCMDS DD` will be ignored
- The system allocates the `SYSTSPRT DD` to receive messages from TSO.
Once transferred, you can browse the files and extract them from the z/OS Unix Systems Service directory. The file names used depend on the PMR and TARGET_DSN input:

```
PMR.TARGET_DSN.Tdate.MTFTP.F00002 to ...Fnnnnn
```

The size and number of files you end up with depends on the WORK_DSN_SIZE SYSIN control statement and how well the input file compresses.

The date section is a random string based on the time the job is run, and PDUU creates different output file names if the same file is sent at a different time. In addition there will be a small file with suffix F001 containing the control record with information necessary to recreate the file on the receiving end.

Please package and send all generated files to IBM with the file names created. Use the NO_FTP=Y option if you have a closed data center (with no outside internet access). PDUU then prepares files you can send to IBM service on removable media as desired.

**DEBUG**

An optional DD statement that gathers debug information such as messages issued to the SYSPRINT data set and the FTP protocol messages. The data set must be RECFM=FB, LRECL=134.

**FTPCMDS**

An optional DD statement that provides additional flexibility for traversing firewall or proxy servers. When this DD statement is provided, after the initial USERID and PASSWORD are sent, the specified sequential data set is read by the application and the commands contained in the data set are included as FTP commands. The data set must be RECFM=FB and LRECL=80.

**SYSTSPRT**

An optional DD statement that shows messages from TSO operations when you have NO_FTP=Y specified. If you receive message AMA778I with a return code of X'04' and function code of X'C', add this DD statement to your JCL to receive messages and rerun if necessary to receive messages.

**JCL examples for PDUU**

Use the following JCL examples as a guideline for creating your own JCL. Consider storing your user ID and password in a separate concatenated data set. Doing so provides added security because the user ID and password are not directly in the JCL. It is also makes it much easier to change the user ID and password across multiple jobs.

You can use some of the JCL examples as a starting point to traverse a firewall or proxy server. There are very few common characteristics for firewall or proxy servers with local customization. If you are able to traverse the firewall or proxy server with a plain FTP statement, modifications to the parameters USERID, PASSWORD, ACCOUNT, and TARGET_SYS, in conjunction with commands in the FTPCMDS data set, the ftp.data file, or both can permit the z/OS PDUU to traverse your firewall or proxy server.

**Example 1: Simple FTP connection**

The JCL example in Figure 209 on page 679 invokes the AMAPDUPL program to transfer file H441PCS.WESSAMP.TRKS055K to the testcase.boulder.ibm.com system as a set of work files stored in /toibm/mvs with the shared prefix.
Each of the three work files is 500 MB.

Example 2: FTP connection using a proxy server

In Figure 210, the USERID control statement has the format user@hostname, where hostname is the name of the TCP/IP system to transfer files to, and user is the user name on the hostname system. The PASSWORD control statement has format proxyuser@proxypass, where proxyuser is the user name on the proxy server and proxypass is the user password on the proxy server. TARGET_SYS is the name of the TCP/IP proxy server.

Example 3: FTP connection using a proxy server with proxy user ID

In Figure 211 on page 680 example, the USERID control statement has format user@proxyuser@hostname, where hostname is the name of the TCP/IP system to transfer the files to, user is the user name on the hostname system, and proxyuser is the user name on the proxy server. The PASSWORD control statement has format proxyuser@proxypass, where proxyuser is the user name on the proxy server and proxypass is the user's password on the proxy server. TARGET_SYS is the name of the TCP/IP proxy server.
Example 4: Using a proxy server with the FTPCMDS DD statement

In Figure 212, the USERID control statement has format proxyuser@hostname, where hostname is the name of the TCP/IP system to transfer the files to, and proxyuser is the user name on the proxy server. The PASSWORD control statement defines the user password on the proxy server. The data set name WES.FTPCMDS.DATA contains an additional user command with an anonymous user name and password on the hostname system.

Example 5: Using a proxy server with a port specification on the TARGET_SYS parameter

The example in Figure 214 on page 681 uses a proxy server with a port specification of 2121 on the TARGET_SYS parameter and inline FTPCMDS DD statement. This example is similar to the previous one, the only difference are the FTPCMDS is an in-stream data set and the port specification is included on the
TARGET_SYS parameter.

Example 6: Forcing PASSIVE mode using the FTPCMDs inline DD statement

Figure 215 shows the FTP connection set up from the FTP client to the FTP server using the FTP locsite fwfriendly command.

The FWFriendly parameter specifies that the FTP client is firewall-friendly. For additional details, see the topic about LOCSite subcommand—Specify site information to the local host in z/OS Communications Server: IP User’s Guide and Commands.

Note: When the FTP server has an IPv6 address, data connections are always set up from the FTP client to the FTP server without reference to the FWFriendly setting.

Example 7: Using a userid.NETRC data set

The example in Figure 216 on page 682 uses the proxy login and password stored in the userid.NETRC data set (you can submit this as a surrogate job where the userid.NETRC data set is invisible to the job originator). Use of the userid.NETRC
**Example 8: Using the DEBUG statement**

The example in Figure 217 adds the DEBUG DD statement, which creates a data set that contains the message data, as described in "DEBUG" on page 678.

```
//FTP EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN DD *
TARGET_SYS=-f"//WES.MYFTP.DATA" testcase.boulder.ibm.com
PMR=99999.999.000
DIRECTORY=/toibm/mvs/
TARGET_DSN=wes.gb01tst
work_dsn=wes.ftpout
cc_ftp=03
WORK_DSN_SIZE=500
```

![Figure 216. FTP using a userid.NETRC data set](image)

For this instance, the userid.NETRC data set consists of one line:
```
machine testcase.boulder.ibm.com login anonymous password ibmusr@ibm.com
```

The FTP.DATA data set contains:
```
;***********************************************************************
;*
NETRCLEVEL 2
;*
```

**Example 9: Using SYSUT2 to allocate an unload data set**

The JCL example in Figure 218 on page 683 invokes the AMAPDUPL program to transfer partitioned data set H44IPCS.PDS.DATA to the testcase.boulder.ibm.com. The optional SYSUT2 statement is used to allocate an unload sequential data set.
for the IEBCOPY utility invoked by the PDUU.

```plaintext
//FTP EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSUT1 DD DSN=H441PCS.PDS.DATA,DISP=SHR  
//SYSUT2 DD DSN=H441PCS.UNLOAD.DASD,DISP=(NEW,CATLG),  
// DCB=(RECFM=VS),  
// SPACE=CYL,(1,1),RLSE),UNIT=SYSDA  
//SYSIN DD *  
USERID=anonymous  
PASSWORD=anonymous  
TARGET_SYS=testcase.boulder.ibm.com  
TARGET_DSN=wessamp.bigfile  
WORK_DSN=wes.ftpout  
CC_FTP=03  
WORK_DSN_SIZE=500  
DIRECTORY=/toibm/mvs/  
PMR=12345.123.123  

Figure 218. Using SYSUT2 statement for allocating an unload data set
```

**Example 10: Using a multiple record control statement in SYSIN**

The following JCL example in Figure 219 invokes the AMAPDUPL program with FTP_DATA data set specified by the -f parameter on TARGET_SYS control statement coded on 3 consecutive SYSIN records.

```plaintext
--------1--------2--------3--------4--------5--------6--------7-- 
//FTP EXEC PGM=AMAPDUPL  
//SYSUDUMP DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSUT1 DD DSN=H441PCS.SEQ.DATA,DISP=SHR  
//SYSIN DD *  
USERID=anonymous  
PASSWORD=anonymous  
TARGET_SYS=-d -f /u/directoryForFtpData/subdirectoryForFtpData1/subdir 
/TARGET_DSN=wessamp.bigfile  
WORK_DSN=wes.ftpout  
CC_FTP=03  
WORK_DSN_SIZE=500  
DIRECTORY=/toibm/mvs/  
PMR=12345.123.123  

Figure 219. Using a multiple record control statement
```

**Example 11: Using the NO_FTP option**

The following JCL invokes the AMAPDUPL program to compress and separate the file into chunks that can be brought to IBM for documentation. File H441PCS.WESSAMP.TRKS055K will be compressed and stored into several files at:

```
/u/nickj/pduu/12345.123.123.WESSAMP.BIGFILE.Txxxxx.MTFTP.Fnnnnn
```
z/OS Problem Documentation Upload Utility

The following files will be stored on your local systems directory:

```
/u/nickj/pduu/
12345.123.123.WESSAMP.BIGFILE.TE3246.MTFTP.F00002
12345.123.123.WESSAMP.BIGFILE.TE3246.MTFTP.F00003
12345.123.123.WESSAMP.BIGFILE.TE3246.MTFTP.F00004
... 
12345.123.123.WESSAMP.BIGFILE.TE3246.MTFTP.F001
```

**Example 12: Using a proxy server with multiple FTPCMDS DD statements**

The example in Figure 221 uses a proxy server with multiple statements in an inline FTPCMDS DD. In this example, FTPCMDS DD runs the USER and PASS commands, and the USERID and PASSWORD SYSIN statements are omitted. The user also specifies PASSIVE mode.

```
//FTP EXEC PGM=AMAPDUPL
//SYSPRINT DD SYSOUT=* 
//SYSSPRRT DD SYSOUT=* 
//SYSUT1 DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K 
//SYSIN DD * 
NO_FTP=Y 
TARGET_DSN=WESSAMP.BIGFILE 
WORK_DSN=WES.FTPOUT 
WORK_DSN_SIZE=500 
DIRECTORY=/u/nickj/pduu/ 
PMR=12345.123.123
```

Figure 220. Using the NO_FTP option

```
//FTP EXEC PGM=AMAPDUPL 
//SYSDUMP DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSSPRRT DD SYSOUT=* 
//SYSUT1 DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K 
//FTPCMDS DD * 
USER 'anonymous@testcase.boulder.ibm.com proxyuser' 
PASS userid@company.com 
ACCT proxypassword 
LOCSite FWFriendly 
//SYSSIN DD * 
TARGET_SYS=proxy.server.name 
TARGET_DSN=SVCD 
WORK_DSN=HLQ.FTPOUT 
CC_FTP=03 
WORK_DSN_SIZE=500 
DIRECTORY=/toibm/mvs/ 
PMR=99999.000.000 
CIPHER_KEY=PMR99999sd 
```

Figure 221. FTP specifying multiple statements in an inline FTPCMDS DD

**Return codes for PDUU**

Upon completion, PDUU places one of the return codes listed in Table 72 in general purpose register (GPR) 15.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
<tr>
<td>4</td>
<td>Potential successful completion. If not, investigate messages.</td>
</tr>
</tbody>
</table>
Table 72. Return codes for z/OS Problem Documentation Upload Utility (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Invalid parameters in control statement</td>
</tr>
<tr>
<td>10</td>
<td>Unsupported data set format</td>
</tr>
<tr>
<td>12</td>
<td>Storage obtain failure</td>
</tr>
<tr>
<td>16</td>
<td>Required input parameters are missing</td>
</tr>
<tr>
<td>20</td>
<td>Invalid input data set specified</td>
</tr>
<tr>
<td>24</td>
<td>Severe error occurred during dictionary building</td>
</tr>
<tr>
<td>28</td>
<td>Severe error occurred during file open</td>
</tr>
<tr>
<td>32</td>
<td>Severe error occurred during compression process</td>
</tr>
<tr>
<td>36</td>
<td>Error in FTP operation</td>
</tr>
<tr>
<td>40</td>
<td>Severe error during IKJTSOEV operation for NO_FTP=Y invocations, see message AMA777I</td>
</tr>
<tr>
<td>44</td>
<td>Severe error during IKJEFTSR operation for NO_FTP=Y invocations, see message AMA778I and SYSTSPRT DD</td>
</tr>
<tr>
<td>64</td>
<td>Severe error in file operation</td>
</tr>
<tr>
<td>99</td>
<td>System or user abend occurred</td>
</tr>
</tbody>
</table>
z/OS Problem Documentation Upload Utility
Chapter 20. Dump suppression

The system requests dumps you might not need. To keep from using your system's resources on unneeded dumps, you should suppress them. The reasons for the unneeded dumps and ways to suppress them are:

- **Duplicate dumps:** The system can request a dump for a problem that recurs. The dump written when the problem first occurs can be used for diagnosis; additional dumps are unneeded. Also, sometimes a system can request several dumps for one instance of a problem. A recurring problem may have been diagnosed, but the fix has not yet been incorporated into the system.

To eliminate the duplicate dumps, use dump analysis and elimination (DAE). See "Using DAE to suppress dumps.”

- **Dumps for certain abend codes:** For some abend codes, the accompanying messages provide the needed problem data. To eliminate the dumps for these abend codes, use a SLIP command. See "Using a SLIP command to suppress dumps” on page 697.

- **A dump for an abend in an application program:** If the dump is not needed. See “Using an ABEND macro to suppress dumps” on page 697.

- **Dumps the installation decides are not needed:** If you decide that certain dumps are not needed, you can code a routine for an installation exit to suppress these dumps. See “Using installation exit routines to suppress dumps” on page 698.

This topic lists the ways that an expected dump can be suppressed, so that you can determine why you did not receive an intended dump. See "Determining why a dump was suppressed” on page 698.

Using DAE to suppress dumps

Dump analysis and elimination (DAE) suppresses dumps that match a dump you already have. Each time DAE suppresses a duplicate dump, the system does not collect data for the duplicate or write the duplicate to a data set. In this way, DAE can improve dump management by only dumping unique situations and by minimizing the number of dumps.

For more information about the topics described in this section, refer to the following references:

- See z/OS MVS Diagnosis: Reference for symptoms and symptom strings.
- See z/OS MVS Initialization and Tuning Reference for the ADYSETxx and IEACMD00 parmlib members.
- See z/OS MVS IPCS Commands for the VERBEXIT DAEDATA subcommand.
- See z/OS MVS Planning: Global Resource Serialization for data set serialization.
- See z/OS Security Server RACF Command Language Reference to control access to data sets.

Performing dump suppression

To perform dump suppression, DAE builds a symptom string, if the data for it is available. If the symptom string contains the minimum problem data, DAE uses the symptom string to recognize a duplicate SVC dump or SYSMDUMP dump.
Dump suppression

requested for a software error. When installation parameters request suppression, DAE suppresses the duplicate dump. The following describes DAE processing.

1. **DAE obtains problem data.** DAE receives the data in the system diagnostic work area (SDWA) or from values in a SYMREC parameter on the SDUMP or SDUMPX macro that requested the dump.
   - The ESTAE routine or the functional recovery routine (FRR) of the failing program supplies module-level information, such as the failing load module name and the failing CSECT name.
   - The system supplies system-level data, such as the abend and reason codes, the failing instruction, and the register/PSW difference.
     
     If the failing component does not supply the failing load module name or CSECT name, the system determines the name, if possible. In this case, the name may be IEANUC0x.

2. **DAE forms a symptom string.** DAE adds a descriptive keyword to each field of problem data to form a symptom. DAE forms MVS symptoms, rather than RETAIN symptoms. DAE combines the symptoms for a requested dump into a symptom string.

The following tables show the required and optional symptoms. SDWA field names are given for the symptoms the failing program must provide to enable dump suppression. The tables have both MVS and RETAIN symptoms so that you can relate the MVS symptoms DAE uses to the RETAIN symptoms you might use to search the RETAIN data base. An MVS symptom string must contain at least five symptoms that are not null. DAE places symptoms into strings in the order shown in the tables.

Table 73 summarizes the required symptoms, which are first and must be present.

<table>
<thead>
<tr>
<th>Symptom</th>
<th>SDWA Field</th>
<th>MVS Keyword</th>
<th>RETAIN Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of the failing load module</td>
<td>SDWAMODN</td>
<td>MOD/name</td>
<td>RIDS/name#L</td>
</tr>
<tr>
<td>Name of the failing CSECT</td>
<td>SDWACSCT</td>
<td>CSECT/name</td>
<td>RIDS/name</td>
</tr>
</tbody>
</table>

Table 74 summarizes the optional symptoms, which must follow the required symptoms. DAE needs at least three of these optional symptoms to make a useful symptom string.

<table>
<thead>
<tr>
<th>Symptom</th>
<th>SDWA Field</th>
<th>MVS Keyword</th>
<th>RETAIN Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product/component identifier with the component identifier base</td>
<td>SDWACID, SDWACIDB</td>
<td>PIDS/name</td>
<td>PIDS/name</td>
</tr>
<tr>
<td>System completion (abend) code</td>
<td>AB/S0hhh</td>
<td>AB/S0hhh</td>
<td></td>
</tr>
<tr>
<td>User completion (abend) code</td>
<td>AB/Uddd</td>
<td>AB/Uddd</td>
<td></td>
</tr>
<tr>
<td>Recovery routine name</td>
<td>SDWAREXN</td>
<td>REXN/name</td>
<td>RIDS/name#R</td>
</tr>
<tr>
<td>Failing instruction area</td>
<td>FI/area</td>
<td>VALU/Harea</td>
<td></td>
</tr>
<tr>
<td>PSW/register difference</td>
<td>REGS/hhhhh</td>
<td>REGS/hhhhh</td>
<td></td>
</tr>
</tbody>
</table>
Table 74. Summary of optional symptoms (continued)

<table>
<thead>
<tr>
<th>Symptom</th>
<th>SDWA Field</th>
<th>MVS Keyword</th>
<th>RETAIN Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason code, accompanying the abend code or from the REASON parameter of the macro that requests the dump</td>
<td>HRC1/nnnn</td>
<td>PRCS/nnnn</td>
<td></td>
</tr>
<tr>
<td>Subcomponent or module subfunction</td>
<td>SDWASC</td>
<td>SUB1/name</td>
<td>VALU/Cname</td>
</tr>
</tbody>
</table>

3. **DAE tries to match the symptom string from the dump to a symptom string for a previous dump** of the same type, that is, SVC dump or SYSMDUMP. When DAE finds a match, DAE considers the dump to be a duplicate.

When DAE is started, it selects active symptom strings to be used to determine which dumps to suppress. An active symptom is one where either the string was created for a unique dump within the last 60 days, or its dump count was updated within the last 60 days.

The systems in a sysplex can share the DAE data set to suppress duplicate dumps across the sysplex. While each system in a sysplex can use its own DAE data set, IBM recommends that systems in a sysplex share a DAE data set so that:

- DAE can write a dump on one system and suppress duplicates on other systems in the sysplex.
- Only one DAE data set is required, rather than a data set for each system.

See “Defining a DAE data set” on page 693 for more information, including recommended names for the data set.

4. **DAE updates the symptom strings in storage and, when the dump is written to a dump data set, in the DAE data set**, if updating is requested.

- For a unique symptom string, DAE adds a new record. The record contains the symptom string, the dates of the first and last occurrences, the incidence count for the number of occurrences, and the name of the system that provided the string.
- For a duplicate symptom string, DAE updates the incidence count for the string, the last-occurrence date, and the name of the last system that found the string.

If updating is requested, DAE examines the incoming dump requests against captured dumps. If the incoming dump's symptom string matches any dump on the captured dump queue, it is suppressed. Updates are done when the DAE data set is updated.

In a sysplex, changes to the in-storage strings of other systems are made after the shared DAE data set is updated. If an incident is occurring at about the same time on multiple systems, multiple dumps will be generated — but only one per system. Dumps on other systems are suppressed after one of the dumps is written, the DAE data set updated, and the updates propagated to the other systems.

If the system with the original dump fails before it writes the captured dump, the dump will not be suppressed the next time it is requested.

5. **DAE suppresses a duplicate dump**, if DAE is enabled for dump suppression.

Note that, if you specify an ACTION of SVCD, TRDUMP, or NOSUP on a SLIP command, the command overrides DAE suppression and the system writes the dump. Also, dumps requested by the DUMP operator command are not eligible for suppression.
Dump suppression

When DAE does not suppress a dump, the symptom string is in the dump header; you can view it with the IPCS VERBEXIT DAEDATA subcommand. DAE also issues informational messages to indicate why the dump was not suppressed.

DAE suppresses a dump when all of the following are true:

- DAE located in the dump the minimum set of symptoms.

  **Note:** If more than the minimum number of symptoms are passed, ALL symptoms are used for comparison, up to 150 characters.

- The symptom string for the dump matches a symptom string for a previous dump of the same type.

- Either of the following is true:
  - The current ADYSETxx parmlib member specifies SUPPRESS for the type of dump being requested and the VRADAE key is present in the SDWA.
  - The current ADYSETxx parmlib member specifies SUPPRESSALL for the type of dump being requested and the VRANODAE key is absent from the SDWA.

Table 75 shows the effect of the VRADAE and VRANODAE keys on dump suppression when SUPPRESS and SUPPRESSALL keywords are specified in the ADYSETxx parmlib member. For SUPPRESS, the VRANODAE key can be present or absent; the system does not check it. The table assumes that the symptom string from the dump has matched a previous symptom string.

<table>
<thead>
<tr>
<th>ADYSETxx Option</th>
<th>VRADAE Key in SDWA</th>
<th>VRANODAE Key in SDWA</th>
<th>Dump Suppressed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPPRESS</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>SUPPRESS</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>SUPPRESSALL</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SUPPRESSALL</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SUPPRESSALL</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SUPPRESSALL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

The only way to ensure that a dump is not suppressed, regardless of the contents of the ADYSETxx parmlib member, is to specify the VRANODAE key in the SDWA.

**Managing rapidly recurring dumps**

DAE can suppress rapidly recurring dumps automatically and the support staff does not need to be aware when a dump request recurs. However, a surge of dump requests could affect system performance, even though the dumps are suppressed. The surge could go unnoticed for hours. To help the support staff take actions to avoid impact to users, the system can notify you of high-frequency dump requests.

To obtain notification, add a NOTIFY parameter to the SVCDUMP statement on the ADYSETxx parmlib member to establish a threshold for notification. The SVCDUMP statement must also specify UPDATE. The default threshold is 3 dumps requested in 30 minutes for the same symptom string. The notification time is measured from completion or suppression of dumps, rather than from initiation of dumps.
The notification is made by the event notification facility (ENF). You can use an ENF exit to:

- Notify the support staff by a message or a signal to a beeper
- Use automation

Any program can receive the ENF signal. If active, the First Failure Support Technology™ (FFST™) issues a generic alert in response to this ENF signal.

If DAE is stopped and restarted, DAE begins counting dumps again to reach the threshold.

If each system has its own DAE data set, notification is for a system. If the systems of a sysplex share the DAE data set, notification is for the sysplex. For example, with a shared DAE data set, four dumps for the same symptom string on the same or different systems in 25 minutes would cause notification if the ADYSETxx parmlib member contains NOTIFY(4,25).

Note: The system in the sysplex that crosses the notification threshold is the system that does the notify.

Planning for DAE dump suppression

Planning for DAE dump suppression consists of tasks to be done before an initial program load (IPL). The system programmer performs the following tasks:

- Selecting or creating an ADYSETxx parmlib member
- Defining a DAE data set

Selecting or creating an ADYSETxx parmlib member

Select or create an ADYSETxx parmlib member to be used at IPL. IBM supplies three ADYSETxx members:

- **ADYSET00**, which starts DAE and keeps 400 symptom strings in virtual storage. The IBM-supplied ADYSET00 member contains:
  
  DAE=START, RECORDS(400),
  SVCdump(MATCH, SUPPRESSALL, UPDATE, NOTIFY(3,30)),
  SYSMDump(MATCH, UPDATE)

  ADYSET00 does not suppress SYSMDump dumps because installation-provided programs deliberately request them. If desired, change the ADYSETxx member being used to suppress SYSMDump dumps.

- **ADYSET01**, which stops DAE processing. The IBM-supplied ADYSET01 member contains:
  
  DAE=STOP

  When using the DAE Display facility’s TAKEDUMP (T) action in a sysplex where DAE is active, you must change the contents of ADYSET01 to:
  
  DAE=STOP, GLOBALSTOP

- **ADYSET02**, which contains the same parameters as ADYSET00.

The IBM-supplied IEACMD00 parmlib member issues a SET DAE=00 command, which activates ADYSET00 during IPL. If you do not want DAE to start during IPL, change IEACMD00 to specify SET DAE=01.

For a sysplex, IBM recommends that you use the same ADYSETxx parameter values in each system. To use the same values, use a shared SYS1.PARMLIB. If your installation does not share a SYS1.PARMLIB, make the ADYSETxx and
Dump suppression

IEACMDxx members in the SYS1.PARMLIB for each system identical. A shared
ADYSETxx or identical ADYSETxx members should specify SHARE(DSN) to share
the DAE data set.

IBM recommends that the ADYSETxx member specify SUPPRESSALL, which
requests that dumps be suppressed even though the component or program did
not request dump suppression with a VRADAE key in the system diagnostic work
area (SDWA). SUPPRESSALL is useful because it allows more dumps to be eligible
for suppression.

In the example shown in Figure 222 the systems in the sysplex share a DAE data
set, SYS1.DAESHARE, so DAE can suppress a duplicate of a previous dump from
any system. This member also specifies SUPPRESSALL.

| DAE=START, RECORDS(400),  |
| SVCDUMP(MATCH, SUPPRESSALL, UPDATE, NOTIFY(3,30)), |
| SYSDUMP(MATCH, UPDATE)    |
| SHARE(DSN, OPTIONS)       |
| DSN(SYS1.DAESHARE)        |

Figure 222. Example: An ADYSETxx Member for a System in a Sysplex

The ADYSET00 member specifies RECORDS(400). If your system does not
suppress a dump when the matching symptom string is in the DAE data set, you
might need more than 400 records in storage; the IBM Support Center can advise
you.

Changing Parmlib Members to Change DAE Processing: While the system is
running, change the DAE data set or parameters for the dumps by creating a new
ADYSETxx parmlib member. See “Changing DAE processing in a Sysplex” on page
696 for the operator actions needed to change the parmlib member.

There is another benefit when all the systems in a sysplex are sharing the DAE
data set. That is, after DAE is started on each system using an ADYSETxx member
which at least contains SHARE(DSN). One operator command can set the DAE
values to be the same on all systems. This is accomplished by issuing the SET
DAE= command, for an ADYSEYxx member which includes the GLOBAL
parameter. ALL systems sharing the DAE data set will be effected.

In the following ADYSET04 member changes the DAE data set being
used on all systems to SYS1.DAESH2 and changes the dump options on all
systems.

| DAE=START, RECORDS(400),  |
| SVCDUMP(MATCH, SUPPRESSALL, UPDATE, NOTIFY(3,30)), |
| SYSDUMP(MATCH, UPDATE)    |
| SHARE(DSN, OPTIONS)       |
| DSN(SYS1.DAESH2)          |
| GLOBAL(DSN, OPTIONS)      |

Figure 223. Example: An ADYSETxx Member with GLOBAL

None of the changes made using operator commands are kept across an IPL of a
system. At IPL, each system will again use the member specified in IEACMD00 or
the COMMNDxx member being used. To make the changes permanently effective,
do one of the following:
Dump suppression

- Make the changes in ADYSET00 and the default IEACMD00 will start DAE.
- Make the changes in the ADYSETxx member and update a COMMNDxx member to start the ADYSETxx using the SET DAE=xx statement. Then update the appropriate IEASYSxx member to include the CMD=xx statement. Once complete, remove the SET DAE=00 within the IEACMD00 member to ensure that the two commands do not complete out of sequence.

Defining a DAE data set
Define a DAE data set when defining system data sets. When the system is IPLed or if DAE is stopped and restarted, DAE should continue using the DAE data set previously used.

1. **Define the DAE data set in a DD statement.** Use the default name of SYS1.DAE for a single system; use a different name for a DAE data set shared by systems in a sysplex.

   The sample DD statement shown in Figure 224 is for a DAE data set used by a single system.

   ```
   //DAE DD DSN=SYS1.DAE,DISP=(,CATLG),VOL=(,RETAI,SER=SG2001),
   // DCB=(RECFM=FB,LRECL=255,DSORG=PS,BKSIZE=0),
   // UNIT=3390,SPACE=(TRK,(6,2))
   ```

   **Figure 224. Example: DAE Data Set for Single System**

   In a sysplex, each system can have its own DAE data set, but IBM recommends that all systems in a sysplex share a DAE data set.

   The sample DD statement in Figure 225 is for a DAE data set shared by the systems in a sysplex. The statement will catalog the DAE data set in the shared master catalog or in the master catalog on each system that uses it.

   ```
   //DAE DD DSN=SYS1.DAESHARE,DISP=(,CATLG),VOL=(,RETAI,SER=SG1055),
   // DCB=(RECFM=FB,LRECL=255,DSORG=PS,BKSIZE=0),
   // UNIT=3390,SPACE=(TRK,(12,2))
   ```

   **Figure 225. Example: DAE Data Set Shared by Sysplex Systems**

   If you manage your dumps with the hierarchical storage manager (HSM), consider using an HSM purge time of 60 days to correspond to the DAE record aging of 60 days.

2. **Provide DAE data set integrity through a serialization component**, such as global resource serialization.

   For a single system, the DAE data set is a local resource. The default DAE data set, SYS1.DAE, is defined as a local resource in the default global resource serialization resource name list (RNL). If you give the DAE data set another name, add the name to the SYSTEMS exclusion RNL to avoid contention when more than one system uses the same DAE data set name for physically different data sets.

   For systems in a sysplex, the shared DAE data set is a global resource. To make global resource serialization treat it as a global resource, do one of the following:
   - Give the DAE data set a name other than SYS1.DAE. For example, SYS1.DAESHARE.
Dump suppression

- If you use the name SYS1.DAE, delete the DAE data set entry from the default SYSTEMS exclusion RNL. The DAE data set entry is SYS0SN SYS1.DAE.

For information, see [z/OS MVS Planning: Global Resource Serialization](#).

3. **Control access to the DAE data set.** On a single system or on all systems sharing the DAE data set in a sysplex, use Resource Access Control Facility (RACF) to control access. Enter a RACF ADDSD command to define a data set profile for the DAE data set.

## Accessing the DAE data set

A DAE data set that is used by one system or is shared by systems in a sysplex is accessed by:

- Invoking the IPCS DAE Display panel
- Generating a suppressed dump
- Editing the DAE data set

### Invoking the IPCS DAE display panel

For the ways to invoke the panel, see IPCS option 3.5 in the [z/OS MVS IPCS User’s Guide](#). On the panel, you can:

- View the symptom strings the data set contains by entering:
  - The date of the dump,
  - The last date for the string,
  - The number of times the dump has been requested,
  - And the last system that requested the dump.
- Search the Entry list for symptoms, system names, dates, etc.
- Navigate through the sysplex dump directory (or whatever dump directory is active) for the symptom string.
- View the dump title for a symptom string.

### Generating a suppressed dump

You may want to obtain a dump that is being suppressed. Perhaps the first dump was ignored and thrown out, but since then the dump has been requested often enough so that you would like to analyze the dump. Do the following to obtain the suppressed dump through the IPCS TAKEDUMP option:

1. Customize the TSO user ID that will invoke the TAKEDUMP action. Make sure it:
   - has authority to issue an MVS operator SET command and, if DAE is active in a sysplex, the ROUTE command
   - has RACF UPDATE access to the DAE data set.
2. Ensure that the ADYSET01 member(s) contains DAE=STOP (or DAE=STOP,GLOBALSTOP in a sysplex).
3. Check that the active IKJTSOxx member includes the program name ADYOPCMD in the AUTHCMD NAMES section.
4. In a sysplex, the maximum benefit is realized when DAE is started using ADYSETxx members which contain at least SHARE(DSN) — enabling shared data set activities.
5. Use the IPCS DAE dialog Panel to issue action code T (the TAKEDUMP option) on the line showing the symptom string of interest.

To process the TAKEDUMP option of the IPCS DAE dialog, DAE processing is stopped, dialog processing occurs, and DAE processing is restarted on the systems...
Dump suppression

involved. There are some cases where a particular system may end up using different DAE parameters from those it was previously using. Table 76 illustrates possible results.

For this discussion there are two systems (SY1 and SY2), and five ADYSETrxx members involved. Members ending with G1 and G2 include GLOBAL(DSN,OPTIONS) parameters. Members ending with S1 and S2 have SHARE(DSN,OPTIONS) without GLOBAL options. The center column in Table 76 indicates the system where the TSO user is issuing the TAKEDUMP request.

Table 76. Examples of when DAE parameters may change

<table>
<thead>
<tr>
<th>Start State</th>
<th>TSO</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>SY1 SY2</td>
<td>SY1</td>
<td>SY1 SY2</td>
</tr>
<tr>
<td>G1 G1</td>
<td>G1</td>
<td>G1 G1</td>
</tr>
<tr>
<td>S1 S2</td>
<td>S1</td>
<td>S1 S2</td>
</tr>
<tr>
<td>S1 S2</td>
<td>S2</td>
<td>S2 S2</td>
</tr>
<tr>
<td>S1 G1</td>
<td>SY1</td>
<td>S1 S1</td>
</tr>
<tr>
<td>S1 G2</td>
<td>SY2</td>
<td>S2 S2</td>
</tr>
<tr>
<td>G2 S2</td>
<td>SY2</td>
<td>S2 S2</td>
</tr>
<tr>
<td>00 *</td>
<td>SY1</td>
<td>00 *</td>
</tr>
</tbody>
</table>

Note:
1. GLOBAL(DSN) systems remain synchronized.
2. When all systems are NOT in GLOBAL(DSN) mode, the system will be started using the member last active on the system where the IPCS TAKEDUMP dialog runs.
3. No change to other systems if all work is done on a system which is not sharing the DAE data set. Here, ADYSET00 contains the default IBM—supplied values.
4. The commands necessary to accomplish the task are issued by the TSO user at their dispatching priority. It is possible that the system may not dispatch the TSO user due to that dispatching priority, and therefore the action may not complete in a timely manner.

The system will generate the next dump for the symptom string. After that dump, DAE resumes suppressing dumps for the symptom string.

Note: Despite specifying action code T, the dump might still be suppressed. See “Determining why a dump was suppressed” on page 698 for the reasons, other than DAE suppression.

Editing the DAE data set

Edit the DAE data set, using Interactive System Productivity Facility (ISPF) edit. For ISPF edit, see z/OS V2R2 ISPF Dialog Developer’s Guide and Reference. You must have WRITE access to the DAE data set. Once in ISPF Edit, use the Edit macro ADYUPDAT as described below.

In the edit session, type one of the following on the command line, place the cursor on the symptom string line for the dump, and press ENTER. If the cursor is on the command line, the first symptom string is used. Note that DAE must be stopped before these actions and started again after.

ADYUPDAT TAKEDUMP
ADYUPDAT NODUMP
Dump suppression

ADYUPDAT TAKEDUMP requests that the next dump be generated for this symptom string. SLIP can still suppress the dump.

ADYUPDAT NODUMP undoes the effect of TAKEDUMP, if it was in effect. Otherwise, NODUMP results in no action.

In the edit session, you can also delete every symptom string that has not been updated within a specified number of days. You must SAVE the DAE data set for the deletions to take effect. To request the deletions, enter on the command line:

```
ADYUPDAT CLEANUP nnn
```

`nnn` number of days a record has not been updated for it to be selected for deletion. The default is 60 days.

ADYUPDAT always issues a status message reflecting the outcome of the command.

Stopping, starting, and changing DAE

If an ADYSET00 parmlib member is used and the DAE data set is allocated, DAE starts during IPL. Normally, DAE runs at the same time as the system. However if DUMPSRV is ever cancelled or restarted, DAE restarts as well albeit with dump suppression inactive. You must manually reactivate dump suppression using the SET DAE=xx command.

An operator can stop and start DAE with the following steps. One reason to use these steps would be to change to a different ADYSETxx parmlib member with different parameters.

Stopping DAE

You can stop DAE with a SET DAE command that specifies the ADYSET01 parmlib member, which contains a DAE=STOP statement:

```
SET DAE=01
```

Starting DAE

You can start DAE with a SET DAE command that specifies an ADYSETxx parmlib member that contains the DAE=START parameter, such as an installation-provided ADYSET03 parmlib member:

```
SET DAE=03
```

Changing DAE processing in a Sysplex

The operator can change all DAE processing in a sysplex, if desired. For example, the operator can do the following to make all systems in a sysplex use a different ADYSETxx member:

1. Stop DAE processing using the IBM-supplied ADYSET01 member:
   ```
   ROUTE +ALL,SET DAE=01
   ```
   Another way to stop DAE processing on all systems in a sysplex is to specify in the SET DAE command an ADYSETxx member containing a GLOBALSTOP parameter.

2. Start DAE processing using, for example, the ADYSET04 member:
   ```
   ROUTE +ALL,SET DAE=04
   ```
Using a SLIP command to suppress dumps

Some dumps are almost never needed. For example, some abend codes tell the diagnostician enough to solve the problem. For these codes, place SLIP operator commands in an IEASLPxx parmlib member to suppress the unneeded dumps. The IBM-supplied IEASLP00 member contains the SLIP commands to suppress abend dumps that are seldom needed.

Using SLIP to suppress dumps also suppresses message IEA995I, which contains symptom dump information. The system may document the abend in a LOGREC error record.

To suppress dumps for an abend code, specify a SLIP operator command with one of the following ACTION parameters. Place all the SLIP commands in an IEASLPxx parmlib member and activate the IEASLPxx member with the following command in a COMMNDxx or IEACMDxx member that is always used:

CMD='SET SLIP=xx'

**SLIP Parameter**

<table>
<thead>
<tr>
<th>Dumps Suppressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION=NODUMP</td>
</tr>
<tr>
<td>All dumps</td>
</tr>
<tr>
<td>ACTION=NOSVCD</td>
</tr>
<tr>
<td>SVC dumps</td>
</tr>
<tr>
<td>ACTION=NOSYSA</td>
</tr>
<tr>
<td>ABEND SYSABEND dumps</td>
</tr>
<tr>
<td>ACTION=NOSYSM</td>
</tr>
<tr>
<td>ABEND SYSMDUMP dumps</td>
</tr>
<tr>
<td>ACTION=NOSYSU</td>
</tr>
<tr>
<td>ABEND SYSUDUMP dumps</td>
</tr>
</tbody>
</table>

For example, to suppress SVC dumps and ABEND SYSMDUMP dumps for abend code 'X'B37', add the following to IEASLPxx:

SLIP SET,COMP=B37,ACTION=(NOSVCD,NOSYSM),END

For more information about the following topics, see the following references:

- See [z/OS MVS Initialization and Tuning Reference](#) for the IEASLPxx member.
- See [z/OS MVS System Commands](#) for the SLIP operator command.

Using an ABEND macro to suppress dumps

A program can suppress a dump by issuing an ABEND macro without a DUMP parameter. Application programmers should not specify a DUMP parameter when a symptom dump can provide enough information for diagnosis.

See [z/OS MVS Programming: Assembler Services Reference ABE-HSI](#) for more information about the ABEND macro.
Dump suppression

Using installation exit routines to suppress dumps

An installation can add installation exit routines, summarized in Table 77, to suppress dumps. Use IEAVTABX if you want to suppress ABEND dumps based on the job name, abend code, or other information in the system diagnostic work area (SDWA). Use IEAVTSEL if you want to discard an SVC or SYSDUMP dump based on information in the dump header or from DAE. Use JES2 exit 4 or JES3 exit IATUX34 to suppress different types of dumps.

Table 77. Summary of installation exit routines for dump suppression

<table>
<thead>
<tr>
<th>Exit</th>
<th>Processing</th>
<th>Dump Suppression</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAVTABX</td>
<td>Before any ABEND dump</td>
<td>Routine(s) can place a return code of 8 in register 15 to suppress the requested dump.</td>
</tr>
<tr>
<td>IEAVTSEL</td>
<td>After an SVC dump or ABEND SYSDUMP dump, if the dump was not suppressed by DAE</td>
<td>Routine(s) can clear the dump data set.</td>
</tr>
<tr>
<td>JES2 exit 4 or JES3 IATUX34</td>
<td>For any JCL statement</td>
<td>Can change the DSNAME parameter on a dump DD statement to DUMMY to suppress the dump.</td>
</tr>
</tbody>
</table>

For more information about the following topics, see the following references:
- See z/OS MVS Installation Exits for IEAVTABX and IEAVTSEL.
- See z/OS JES2 Installation Exits for the JES2 exit 4 routine.
- See z/OS JES3 Customization for the JES3 IATUX34 exit routine.

Determining why a dump was suppressed

If an intended dump is missing, use this list to decide why. The list gives reasons why dumps are suppressed, including the ways discussed in this topic. In planning for problem determination, be aware of all of these ways so that your installation does not suppress intended dumps.

- **DAE suppression of dumps.** See “Using DAE to suppress dumps” on page 687.
- **SLIP command that suppresses all dumps for an abend code.** See “Using a SLIP command to suppress dumps” on page 697.
- **An ABEND macro without a DUMP parameter.** See “Using an ABEND macro to suppress dumps” on page 697.
- **An MVS installation exit routine that suppresses the dump.** See “Using installation exit routines to suppress dumps.”
- **Resource Access Control Facility (RACF) control of programs in an address space to be dumped:** Beginning with RACF 1.8.1, the installation can protect ABEND dumps of programs using the FACILITY class. The protection can keep you from accessing a dump.
- **Dump on another system blocked by SYSDCOND in the PROBDESC area and the IEASDUMP.QUERY routine.** A dump on another system in a sysplex is requested by a DUMP command or SDUMPX macro with a REMOTE parameter. If the area specified by the PROBDESC parameter contains SYSDCOND, the dump on the other system is not written because of either of the following on the other system:
  - No IEASDUMP.QUERY routine exists
  - No IEASDUMP.QUERY routine returns a code of 0
Dump suppression

- **Dump suppressed by CHNGDUMP command.** If a CHNGDUMP command specifies NODUMP for SVC dumps:
  - All SVC dumps on the system are suppressed.
  - If a DUMP command or SDUMPX macro includes a REMOTE parameter, the dump on the local system and the dumps on other systems in the sysplex are suppressed.

- **Dump on another system suppressed by CHNGDUMP command on the other system.** If a DUMP command or SDUMPX macro includes a REMOTE parameter and a CHNGDUMP command previously entered on another system in the sysplex specifies NODUMP for SVC dumps, the SVC dump on the other system is suppressed. The dump on the local system is written.

The system can also place the dump in another data set, so that it is not in the original data set specified in a message you received:

- **An installation exit routine at JES2 exit 4 or at JES3 exit IATUX34 can change the dump data set name.**
- **DUMPDS operator command can redirect SVC dump output.** The command can redirect SVC dump output to other SYS1.DUMPxx data sets.

For more information about the following topics, see the following references:

- See [z/OS Security Server RACF Security Administrator's Guide](#) for the FACILITY class to control access to program dumps.
- See [z/OS MVS System Commands](#) for the DUMP, DUMPDS, and SLIP commands.
- See [z/OS MVS Installation Exits](#) for the IEAVTABX and IEAVTSEL exit routines.
- See [z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU](#) for the SDUMPX macro.
- See [z/OS MVS Programming: Authorized Assembler Services Guide](#) for the IEASDUMP .QUERY routine.
- See [z/OS JES2 Installation Exits](#) for the JES2 exit 4 routine.
- See [z/OS JES3 Customization](#) for the JES3 IATUX34 exit routine.
Dump suppression
Chapter 21. Messages

The system issues messages to do the following:

- Tell the operator or system programmer of progress and problems in system processing
- Ask the operator to take actions and make decisions
- Tell the application programmer how the system ran the application program and of problems in the application program

The system issues messages from the base control program components and a variety of subsystems, products, and applications. Applications running under the system can also issue their own messages.

Producing messages

You can get the system to produce a message by issuing a macro in any program or by asking an operator to enter a command. The macros and command are:

- LOG operator command to write a message to the SYSLOG and OPERLOG
- WTL macro to write a message to the SYSLOG and OPERLOG

Note: Use WTO specifying MCSFLAG=HRDCPY instead of using WTL, which provides additional information with the WTO message that is not with the WTL message.
- WTO macro to write a message to the operator
- WTOR macro to write a message to the operator and request a reply

Use the following for related activities:

- DOM macro to delete an operator message or group of messages from the display screen of a console
- REPLY operator command to answer a message
- WRITELOG operator command to start, stop, or print the SYSLOG and to change the output class for the SYSLOG

For additional information, see the following resources:

- See z/OS MVS System Commands for the LOG, REPLY, and WRITELOG commands.

Receiving messages

The system issues messages through WTO and WTOR macros to the following locations. Routing codes determine the display and print location of the messages.

- Console
- Extended console
- Hard-copy log
- Job log
Messages

- SYSOUT data set

The system issues messages to the SYSLOG and OPERLOG using the WTL macro.

The access methods issue messages directly to one of the following locations:
  - Display terminal
  - Output data set

Console

Messages sent to a console with master authority are intended for the operators. The system writes in the hard-copy log all messages sent to a console, regardless of whether the message is displayed.

Hard-Copy log

The hard-copy log is a record of all system message traffic:
  - Messages to and from all consoles
  - Commands and replies that are entered by the operator.

In a dump, these messages appear in the master trace. With JES3, the hard-copy log is written to the SYSLOG, the OPERLOG, or both. With JES2, the hard-copy log is written to the SYSLOG, the OPERLOG, or both; it can also be viewed using a product like System Display and Search Facility (SDSF). For more information about SDSF, see z/OS SDSF Operation and Customization.

System log

The SYSLOG is a SYSOUT data set provided by the job entry subsystem (either JES2 or JES3). SYSOUT data sets are output spool data sets on direct access storage devices (DASD). Use SDSF to view the SYSLOG to check for problems. The SYSLOG consists of the following:
  - All messages issued through WTL macros
  - All messages entered by LOG operator commands
  - Usually, the hard-copy logs
  - Any messages routed to the SYSLOG from any system component or program

Job log

Messages sent to the job log are intended for the programmer who submitted a job. Specify the system output class for the job log in the MSGCLASS parameter of the JCL JOB statement.

SYSOUT data set

Messages sent to a SYSOUT data set are intended for a programmer. These messages are issued by an assembler or compiler, the binder and loader, and an application program. To make all messages about a program appear in the same SYSOUT listing, specify the same class for the SYSOUT data set and in the MSGCLASS parameter on the JCL JOB statement.

Receiving symptom dumps

A symptom dump is a system message, either message IEA995I or a numberless message, which provides some basic diagnostic information for diagnosing an abend. Often the symptom dump information can provide enough information to diagnose a problem.
The example in Figure 226 shows the symptom dump for an abend X'0C4' with reason code X'4'. This symptom dump shows the following information:

- Active load module ABENDER is at address X'00006FD8'.
- The failing instruction was at offset X'12' in load module ABENDER.
- The address space identifier (ASID) for the failing task was X'000C'.

Symptom dumps appear in the following places:

- For SYSUDUMP and SYSABEND ABEND dumps: in message IEA995I, which is routed to the job log.
- For a SYSMDUMP ABEND dump: in message IEA995I in the job log and in the dump header record.
- For an SVC dump: in the dump header record.
- For any dump in a Time Sharing Option/Extensions (TSO/E) environment: displayed on the terminal when requested by the TSO/E PROFILE command with the WTPMSG option.
- In response to a DISPLAY DUMP,ERRDATA operator command, which displays information from SYS1.DUMPxx data sets on direct access.

If the information in a symptom dump is enough for diagnosis, do not provide a DD statement for a dump.

For additional information, see one of the following resources:

- See Chapter 9, “Master trace,” on page 221 for information on master trace.
- See z/OS MVS JCL Reference for the JOB statement.
- See z/OS TSO/E Command Reference for the PROFILE command.

Planning message processing for diagnosis

Your installation can change message processing in a number of ways to optimize diagnosis. Your installation can do the following tasks:

- Control message location
- Suppress messages
- Automate message processing
Messages

- Not retain action messages
- Suppress the symptom dump message (IEA995I)

This section can help you find the information you need to optimize message processing for your installation.

**Controlling message location**

An installation can change the following:

- The routing codes for specific messages to control
- On which console to display a message.

Change or specify the routing codes using the following methods:

- A WTO or WTOR macro specifies the routing code for the message that the macro creates.
- A WTO/WTOR installation exit routine changes the routing code for any WTO or WTOR message. This exit routine is the routine named in the USEREXIT parameter in the MPFLSTxx parmlib member or IEAVMXIT.
- The JES3 MSGROUTE initialization statement changes the routing code of JES3 console messages.
- The JES3 CONSOLE initialization statement can control which messages a JES3 console receives.
- Subsystem interface listeners, such as NetView, can change the routing codes.

**Suppressing messages**

An installation can use the following to suppress messages. Suppressed messages do not appear on a console.

- An MPFLSTxx parmlib member can specify message suppression. A suppressed message does not display on a console, but writes to the hard-copy log.
- A WTO/WTOR installation exit routine can suppress any WTO/WTOR messages or override suppression. This exit routine is the routine named in the USEREXIT parameter in the MPFLSTxx parmlib member or IEAVMXIT.
- The JES2 installation exit routine, Exit 10, can suppress messages issued by the JES2 main task.
- The JES3 installation exit routine, IATUX31, can suppress messages routed to JES3 consoles.
- The CONTROL V operator command can suppress messages by specifying the message levels to display at a console.
- VARY operator command can change the messages received by a console by specifying the routing codes of messages to be displayed.
- The LEVEL keyword on the CONSOLE statement in the CONSOLxx parmlib member also can suppress messages by specifying the message levels to be displayed at a console. The ROUTCODE keyword can specify the routing codes of messages that are displayed at the console.

**Handling message floods**

A MSGFLDxx parmlib member can specify the criteria required to recognize that a message flooding situation is occurring and the actions to be taken to handle the message flood. The message flood automation policy can prevent flood messages from being:

- prevent flood messages from being displayed on a console
- prevent flood messages from being queued for automation
Messages

• prevent flood messages from being written to the SYSLOG or OPERLOG

Messages are written to the SYSLOG or OPERLOG identifying the job or specific message that is causing the message flood; messages are also written to the logs when action is taken against specific messages or messages from that job.

Automating message processing
An MPFLSTxx parmlib member can specify that a message is to be passed to an automation subsystem, such as NetView. The automation subsystem can perform actions that the operator would have performed without operator intervention.

Not retaining action messages
The MPFLSTxx parmlib member can specify not to retain a message using the Action Message Retention Facility (AMRF). An operator cannot recall to the screen action messages that are not retained.

Suppressing symptom dumps (IEA995I)
Table 78 lists ways to suppress symptom dumps for ABEND dumps. These ways suppress only message IEA995I; symptom dumps continue to appear in other locations.

Table 78. Suppressing symptom dumps

<table>
<thead>
<tr>
<th>Dump Type</th>
<th>CHNGDUMP Operator Command used to Suppress Symptom Dump</th>
<th>Parmlib Member used to Suppress Symptom Dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSABEND ABEND</td>
<td>CHNGDUMP SET,SYSABEND,SDATA=(NOSYM)</td>
<td>IEAABD00</td>
</tr>
<tr>
<td>SYSMDUMP ABEND</td>
<td>CHNGDUMP SET,SYSMDUMP=(NOSYM)</td>
<td>IREADMR00</td>
</tr>
<tr>
<td>SYSUDUMP ABEND</td>
<td>CHNGDUMP SET,SYSUDUMP,SDATA=(NOSYM)</td>
<td>IREADMP00</td>
</tr>
</tbody>
</table>

For more information, see one of the following references:
• See z/OS MVS Planning: Operations for message flood automation, controlling message display, suppressing messages, AMRF, and automating messages in a sysplex.
• See z/OS MVS Programming: Assembler Services Reference IAR-XCT for the WTO and WTOR macros.
• See z/OS MVS Initialization and Tuning Reference for the MPFLSTxx and CONSOLxx members.
• See z/OS MVS Installation Exits for the exit routine named in the USEREXIT parameter and for IEAVMXIT.
• See z/OS JES3 Initialization and Tuning Reference for the JES3 initialization statements.
• See z/OS JES2 Installation Exits for Exit 10.
• See z/OS JES3 Customization for the IATUX31 exit.
• See z/OS MVS System Commands for the CONTROL V and the CHNGDUMP operator command.
Chapter 22. Hardware Instrumentation Services

Hardware instrumentation services (HIS) is a function that collects hardware event data for IBM System z10™ or later machines.

IBM may request that you provide hardware event data for diagnostics. Before you start the HIS data collection, you may first need to authorize the sampling facilities and counter set types you want to use through the support element (SE) console. For each LPAR of interest, you will need to verify each of the following settings:

- Basic counter set authorization control
- Problem state counter set authorization control
- Crypto activity counter set authorization control
- Extended counter set authorization control
- Basic sampling authorization control

The LPAR to be measured may be already activated and running. A dynamic update can be done to enable the CPU Measurement Facility without disrupting this LPAR by selecting the Change LPAR Security icon on the SE or HMC. The Save and Change options on the LPAR security panel will dynamically change the running system and also save the change to the activation profiles. See Figure 227 for an example.

For LPARs that have not yet been activated, the CPU Measurement Facility can be enabled using the security tab for the activation profile. See Figure 228 on page 708 for an example.
For information about how to set up the authorization of the sampling facilities and counter sets, see Support Element Operations Guide for System z10 machine on the Resource Link home page (http://www.ibm.com/servers/resourcelink).

In addition, with the enhanced-monitor facility hardware released with z196 machines, the HIS function expands into a z/OS software event data collector that will be used by IBM for improved problem analysis. The z/OS event counters are viewed as an additional counter set, but there is no authorization required to use the hardware. For information on the enhanced-monitor facility hardware, see z/Architecture Principles of Operation.
Appendix. Accessibility

Accessible publications for this product are offered through [IBM Knowledge Center](http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the ["Contact us" web page for z/OS](http://www.ibm.com/systems/z/os/zos/webqs.html) or use the following mailing address.

IBM Corporation  
Attention: MHVRCFS Reader Comments  
Department H6MA, Building 707  
2455 South Road  
Poughkeepsie, NY 12601-5400  
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- [z/OS TSO/E Primer](#)
- [z/OS TSO/E User’s Guide](#)
- [z/OS V2R2 ISPF User’s Guide Vol 1](#)

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out
punctuation. All the syntax elements that have the same dotted decimal number
(for example, all the syntax elements that have the number 3.1) are mutually
exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your
syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a
syntax element with dotted decimal number 3 is followed by a series of syntax
elements with dotted decimal number 3.1, all the syntax elements numbered 3.1
are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add
information about the syntax elements. Occasionally, these words and symbols
might occur at the beginning of the element itself. For ease of identification, if the
word or symbol is a part of the syntax element, it is preceded by the backslash (\)
character. The * symbol is placed next to a dotted decimal number to indicate that
the syntax element repeats. For example, syntax element *FILE with dotted decimal
number 3 is given the format 3\* FILE. Format 3\* FILE indicates that syntax
element FILE repeats. Format 3\* \* FILE indicates that syntax element * FILE
repeats.

Characters such as commas, which are used to separate a string of syntax
elements, are shown in the syntax just before the items they separate. These
characters can appear on the same line as each item, or on a separate line with the
same dotted decimal number as the relevant items. The line can also show another
symbol to provide information about the syntax elements. For example, the lines
5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the
LASTRUN and DELETE syntax elements, the elements must be separated by a comma.
If no separator is given, assume that you use a blank to separate each syntax
element.

If a syntax element is preceded by the % symbol, it indicates a reference that is
defined elsewhere. The string that follows the % symbol is the name of a syntax
fragment rather than a literal. For example, the line 2.1 %OP1 means that you must
refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted
decimal number followed by the question mark symbol (?) indicates that all
the syntax elements with a corresponding dotted decimal number, and any
subordinate syntax elements, are optional. If there is only one syntax element
with a dotted decimal number, the ? symbol is displayed on the same line as
the syntax element, (for example 5? NOTIFY). If there is more than one syntax
element with a dotted decimal number, the ? symbol is displayed on a line by
itself, followed by the syntax elements that are optional. For example, if you
hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements
NOTIFY and UPDATE are optional. That is, you can choose one or none of them.
The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted
decimal number followed by the ! symbol and a syntax element indicate that
the syntax element is the default option for all syntax elements that share the
same dotted decimal number. Only one of the syntax elements that share the
dotted decimal number can specify the ! symbol. For example, if you hear the
lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the
default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used.

However, if you hear the lines 2? FILE, 2.1, 2.1.1 (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

* indicates an optional syntax element that is repeatable

The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.

2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.

3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.
Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

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
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel  
IBM Corporation  
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.

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 or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted.
for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

**Minimum supported hardware**

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS](http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

**Trademarks**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [http://www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.
Index

Special characters
* control statement
in SPZAP 656
<element name>
content, new xx

A
AB= parameter
in GTF 234
ABDUMP= parameter
in GTF 234
abend analysis
obtain abend code 154
obtain reason code 154
abend code
in STATUS FAILDATA report 43
ABEND dump
analysis 153
contents 142
customize 147
displaying options 142
reasons for selection 3
summary dump contents 146
SYSABEND dump analysis 153
SYSUDUMP dump analysis 153
ABEND dumps
obtaining 137
synopsis 136
ABEND macro
for requesting ABEND dumps 140
in dump customization 147, 149, 150, 152
to suppress dumps 697
abnormal end
indicated in logrec error record 551
ABSDUMP/ABSDUMPT control statement
example 644
in SPZAP 644, 652, 653, 654
parameter 653, 654
abstract xv
accessibility 709
contact IBM 709
features 709
ACR (alternate CPU recovery)
problem data 167
system trace event 174
ACR trace entry
in system trace 174
ADATA= parameter
LISTLOAD control statement 580
address
commonly bad 164
address space
buffers for component trace 384
addressing mode
system trace event 188
ADDSUMM keyword 84
ADDSUMM parameter
of AMDSADMP macro 81, 83
ADINT trace record
formatted 270
unformatted 325
Advanced Program-to-Program Communication
See APFC/MVS 399
AINT (adapter interruption)
system trace event 176
AINT trace entry
in system trace 176
ALIB parameter
AMDSADMP macro 79
ALL parameter
dump option 143
ALLNUC parameter
dump option 26, 60, 143
allocation
automatically of dump data set 8, 56
component trace 445
pre-allocation of dump data set 13
ALLPA parameter
dump option 143, 160
ALLPDATA parameter
dump option 143
ALLPSA parameter
dump option 26, 60
ALLSDATA parameter
dump option 143
ALLVNUC parameter
dump option 143, 160
ALTR trace entry
in system trace 177
AMAPDUPL
overview 673
see also z/OS Problem Documentation Upload Utility (PDUU) 673
AMATERSE
allocation considerations 671
alternate DDNAMES 670
DASD support 667
DD statements
SYSPRINT 669
SYSUT1 669
SYSUT3 669
exec statement 668
incompatible with z/OS Problem Documentation Upload Utility (PDUU) 674
PACK 668
parameters 668, 670
planning 667
restrictions 670
return codes 669
sample JCL 668
space considerations 671
SPACK 668
starting
problem state 669
tape support 667
troubleshooting 668
UNPACK 668
using 668
z/OS Problem Documentation Upload Utility (PDUU) 667
AMATERSE service aid 667
reasons for selection 5
AMBLIST output
obtain 578
AMBLIST service aid
control statement
r rules for coding 578
description 577
functions 583
JCL statement 578
LISTIDDR control statement 581
LISTLOAD control statement 579
LISTLPA control statement 582
LISTOBJ control statement 580
mapping CSECTs in a load module or program object 585
mapping the contents of the nucleus 588
mapping the modules in the link pack area 588
output 590
reasons for selection 5
tracing modifications to the executable code in a
CSECT 587
AMDD029 parameter
of AMDSADMP macro 80
AMDSADDD utility 88
catalog requirement 92
characteristics 89
CLEAR option 90
DEFINE option 90
invocation 89
REALLOC option 90
space requirement 91
syntax 90, 91, 92
unit requirement 91
vollist requirement 91
volser requirement 91
AMDSADMP macro 75
assembly 103
DUMP keyword 84
example 81
format for high-speed dump 75
multiple versions, assembling 104
one-stage generation 96
parameter
ADDSUMM 83
ADDSUMM 81
ALIB= 79
AMDD029= 80
COMPACT= 79
CONSOLE= 77
DDSPROMPT= 80
DUMP= 83
DUMP= 78
IPL= 76
PLEXIST= 81
LNKLIB= 80
MINASID 79
MODLIB= 80
MSG= 79
NUCLIB= 80
OUTPUT= 77
PROMPT 78, 83
REUSED= 79
SYSLIB= 77
ULABEL= 77
VOLSER= 77
sample JCL 103
stage-two generation 103
AMDSADMP macro (continued)
symbol 76, 104
syntax
for high-speed dump 75
SYS1.MACLIB data set
assembly 103
two-stage generation 96
AMDSADNG
DDNAMES 97
AMRF (active message retention facility)
for message retention 705
ANR record 558
APPC/MVS (Advanced Program-to-Program
Communications/MVS)
component trace 399
ASIDP trace option
in GTF 247
prompting 254
assistive technologies 709
authorized program
request dump 19
automation
of messages 705
B
BAKR instruction
system trace event 178
BALR instruction
system trace event 178
BASE control statement
element 652
in SPZAP 641, 652, 654
parameter 654
basic hyperswap socket support Component
See basic hyperswap socket support Component 420
basic hyperswap socket support Component (basic hyperswap
socket support Component)
component trace 420
BASR instruction
system trace event 178
BASM instruction
system trace event 178
BCPii
component trace 417
BR trace entry
in system trace 178
branch
system trace event 178
branch instruction
trace 170
BRANCH parameter
to control summary dump 30
BSG trace entry
in system trace 179
buffer
for component trace 384
logrec 567
C
CALL trace entry
in system trace 181
CALLRTM macro
for requesting ABEND dumps 140
in dump customization 149, 150, 152
CANCEL operator command  
to request ABEND dump 141

catalog  
 rebuild 649
 CB parameter  
dump option 143, 160
 CCHHR control statement  
in SPZAP 642, 655, 656
 parameter 656
 CCW trace option  
in GTF 247
 CCW trace record  
formatted 271
 unformatted 326
 CCWN parameter of GTF CCWP 254, 255
 CCWP trace option  
 DATA parameter 255
 in GTF 247, 254, 255
 I parameter 254
 IOSB parameter 255
 PCITAB parameter 256
 S parameter 254
 SI parameter 254
 prompting 254
 central storage dump  
description 65  
of stand-alone dump 69
 channel program data  
record 247
 CHECKSUM control statement  
in SPZAP 656
 parameter 656
 CHNGDUMP operator command  
in dump customization 34, 63, 149, 151, 153
 to change dump options 22, 142
 to suppress symptom dump 705
 clear  
 SVC dump 24
 Transaction dump 59
 CLKC trace entry  
in system trace 181
 combination of AMBLIST control statement 578
 combination of GTF trace options 252
 common event adapter Component  
See common event adapter Component 425
 common event adapter Component (common event adapter Component)  
 component trace 425
 common storage tracking  
 reasons for selection 5
 COMPACT parameter  
of AMDSADMP macro 79
 component trace  
description 375
 IBM Health Checker for z/OS 441
 options  
 for OPS 490
 reasons for selection 4
 start the external writer 392
 start the trace 387, 392, 394
 stop the external writer 392
 stop the trace 388, 392, 395
 sublevel 375, 377
 sublevel trace  
 verifying 396
 SYSAPPc for APPC/MVS component 399
 SYSAXR for System REXX component 413
 component trace (continued)  
 SYSBCP for BCPii 417
 SYSBHI for basic hyperswap socket support  
 component 420
 SYSCEA for common event adapter component 425
 SYSDF for data lookaside facility (DLF) 427
 SYSDOM for distributed SOM 429
 SYSDUMP 432
 SYSGRS for global resource serialization 436
 SYSEFAL for allocation component 445
 SYSICS for IOS component trace 451
 SYSJES for JES common coupling services 457
 SYStESA for JES2 subsystem 467
 SYSLLA for LLA 469
 SYSLGR for system logger 470
 SYSOMVS for z/OS UNIX 478
 SYSupS for OPS component 489
 sysplex 393
 SYSSRRS for RRS component 494
 SYSSRM for RSM 501
 SYSSSPI for SPI component 518
 SYSVLF for VLF component 520
 SYSWLM for WLM component 523
 SYSXCF for XCF component 526
 SYSEXES for cross-system extended services 531
 verifying 395
 viewing trace data 398
 compression  
 how it works for PDUU 674
 use for dumps 15
 considerations  
one-stage generation 99
 CONSOLE control statement  
 example 657
 in SPZAP 657
 console supported by stand-alone dump 77
 CONSOLE= parameter  
of AMDSADMP macro 77
 specifying the SYSC constant 77
 consolidate  
data from GTF traces 261
 contact  
z/OS 709
 control records  
 GTF trace record  
 unformatted 321
 control registers  
 format in a dump 50
 control statement  
 for AMBLIST service aid 578
 for SPZAP 651
 Control statement  
 for AMBLIST service aid 589
 z/OS UNIX 589
 copy  
 SVC dump 24
 Transaction dump 59
 copy a dump  
 DASD to DASD 117
 multiple devices to DASD 118
 tape to DASD 117
 COPYDUMP subcommand  
to obtain component trace data 398
 COPYTRC subcommand  
for component traces 398
 counter and sampling collection  
 overview 707

Index 719
COUPLE parameter
dump option 26, 60
CP Assist for Cryptographic Functions (CPACF) 674
cross memory instruction
system trace event 179
cross memory processing mode
problem data 166
cross-system coupling facility (XCF)
component trace 526
cross-system extended services
component trace 531
CRW record 558
CSA dumped by stand-alone dump 69
CSA parameter
dump option 143
CSCH trace entry
in system trace 184
CSCH trace option
in GTF 248
CSCH trace record
formatted 273
CSECT
accessing data 641
tracing modifications to the executable code in a
CSECT 587
CSECT identification record (IDR)
print 628
CSECT name
NAME control statement
of SPZAP 635
CSECTs in a load module or program object
mapping with AMBLIST service aid 585
CTIAXRxx parmlib member
for SYSAXR component trace 413
CTIBHXx parmlib member
for SYSBHI component trace 421
CTICEAx parmlib member
for SYSCEA component trace 425
CTIHWW00 parmlib member
for SYSBPII component trace 417
CTIHZS00 parmlib member
for SYSHZS component trace 442
CTIIIEFx parmlib member
for SYSSION component trace 446
CTIRSM01 502
CTIRSMxx parmlib member 502
CTnAPPxx parmlib member
for SYSAPPC component trace 400
CTBPWxx parmlib member
for SYSCEA component trace 478
CTnGRSxx parmlib member
for SYSSRS component trace 436
CTnIOSxx parmlib member
for SYSSOS component trace 453
CTnJESON parmlib member
for SYSJES component trace 459
CTnJESxx parmlib member
for SYSSOS component trace 459
CTnLOGx parmlib member
for SYSSLOGR component trace 473
CTnOPSxx parmlib member
for SYSSOPS component trace 490
CTnRRSxx parmlib member
for SYSRRS component trace 495
CTnRSMxx parmlib member
for SYSRSRSM component trace 502
CTnXCFxx parmlib member 526
for SYSXCF component trace 526
CTnXESxx parmlib member
for SYSXES component trace 533
CTRACE
component trace 375
CTRACE subcommand
for SYSAPPC component trace 403
for SYSAXR component trace 415
for SYSBHI component trace 423
for SYSCEA component trace 427
for SYSCEAS component trace 481
for SYSOPS component trace 492
for SYSWLM component trace 524
for SYSXCF component trace 529
for SYSXES component trace 535
FULL report 493
SHORT report 493
CTWRSM05 parmlib member 502
customization
of nucleus area in dump 33, 62, 147
 customize
master trace 221

D

DAE (dump analysis and elimination)
description 687
DAE data set
editing 694
managing 694
removing old symptom strings 694
DAE Display panel
invoking 694
DAE service aid
reasons for selection 5
DASD (direct access storage device) 70, 76, 77, 97, 104
allocation 88
AMDSADDD utility 88
IPCS SADMP utility 88
data set 87
stand-alone dump 65
types 88
DASD-SIM recovery record 558
data
for diagnosis 163
inspect with SPZAP 633
modify with SPZAP 633
data inspection
use SPZAP 633
data lookaside facility
See DLF 427
data modification
use SPZAP 633, 634, 642, 652
DATA parameter of GTF CCWP 254, 255
data set
automatically allocated for dumps 8, 56
to receive ABEND dump 138
data space
buffers for component trace 384
DCB parameter
for component trace data set 390
DD statement
in AMBLIST service aid
anyname 578
SYSIN data set 578
SYSPRINT data set 578
DUMPDS operator command  
to clear SYS1.DUMPxx data set  24  
DUMPOPT or DUMPOPX parameter  
in dump customization  149, 150, 152  
dumps  
description  2  
dvolser  120  
dynamic invocation  
of SPZAP  649

E
EID (event identifier) for GTF  317  
EMS trace entry  
in system trace  181  
enabled  
ENQ parameter  
dump option  143  
entry  
system trace  171  
environmental data  
definition  556  
record header information  557  
environmental record  556  
EOD record  558  
RDE option  558  
ERR parameter  
dump option  143, 161  
error  
record in logrec  551  
error identifier  
in STATUS WORKSHEET report  42  
in VERBEXIT LOGDATA report  48  
error statistic  
definition  556  
ESTAE or ESTAEX macro  
in dump customization  149, 151, 152  
ESTAI parameter  
in dump customization  149, 151, 152  
ETR recovery record  558  
example  
CTWRSM05 parmlib member  502  
exit 4  
for JES2  
to suppress dumps  698  
ext routine  
to suppress dumps  698  
EXT trace entry  
in system trace  181  
EXT trace option  
in GTF  248  
EXT trace record  
comprehensive  
unformatted  327  
formatted  280  
minimal  
unformatted  327  
extended format sequential data set  
to hold large dumps  13  
external interruption  
record  248  
external writer  
source JCL  388  
extact  
GTF trace data from dumps  261

F
failing instruction  43  
FID (format identifier)  319  
FID (format identifier) for GTF  319  
fixed link pack area  
map  582  
FLIH (first level interrupt handler)  
problem data  
saved by external FLIH  130  
saved by I/O FLIH  130  
saved by SVC FLIH  130  
FLPA parameter  
of LISTLPA control statement  583  
FORCE ARM command  246  
format of the logrec buffer  568  
formatting  
master trace  223  
formatting the Logrec buffer  568  
FRR (functional recovery routine)  
in dump customization  149, 151, 152  
FRR (functional recovery routine) data  
record  250  
FRR trace record  
formatted  282  
FTPCMDS  680  
FTPCMDS data set example  680

G
general purpose registers  
format in a dump  50  
generating from AMDSADMP macro  
stand-alone dump  96  
generic tracking facility  343  
GFS trace  
requesting  545  
global resource serialization  
component trace  436  
goff listing  
AMBLIST output for LISTOBJ with GOFF records.  595  
GRSQ parameter  
dump option  26, 60, 143  
GTF (generalized trace facility)  
combining trace options  252  
prompting  252  
prompting keywords in SYS1.PARMLIB  258  
specification of a system event  258  
specification of GTF trace for a system event  258  
starting  241  
starting GTF  244  
prompting  252  
START command  241  
with internal tracking mode  242  
starting with data recorded on a device  245  
STOP command  245  
storing trace options in SYS1.PARMLIB  242, 243  
trace VTAM remote network activity  244  
GTF (generalized trace facility) trace  
ADINT trace record  
formatted  270  
ADINT trace records  
unformatted  325  
cataloged procedure  230, 231  
CCW trace record  
formatted  271
GTF (generalized trace facility) trace (continued)

CCW trace records
  unformatted 326
control record 321
CSCH trace record
  formatted 273
customization 232
definition of trace options 232
DSP comprehensive trace record
  unformatted 326
DSP minimal trace record
  unformatted 326
DSP trace record
  formatted 274
EID (event identifier) for USR trace records 317
EXT comprehensive trace record
  unformatted 327
EXT minimal trace record 327
EXT trace record
  formatted 280
FID (format identifier) for USR trace records 319
formatted output 267
FRR trace record
  formatted 282
generation of trace record 230
HEXFORMAT trace record 283
HSCH trace record
  formatted 273
I/O trace record
  unformatted 331
IBM defaults 230
IBM-supplied catalogued procedure 230
in GTF 229
IOX trace record
  formatted 284
lost data record 322
lost event record 320
LSR trace record
  formatted 287
merge with component traces 262
MSCH trace record
  formatted 288
output direction 229
parmlib member options 230
PCIDMX trace record
  formatted 289
PCIDMX trace records
  unformatted 332
PCILG trace record
  formatted 290
PCILG trace records
  unformatted 332
PCISTG trace record
  formatted 291
PCISTG trace records
  unformatted 332
PGM trace record
  formatted 292
PI comprehensive trace record
  unformatted 332
PI minimal trace record
  unformatted 333
PI trace record
  formatted 292
program event 229
receive 260
GTF (generalized trace facility) trace (continued)
records 262
request reasons 229
RNIO trace record
  formatted 293
RR comprehensive trace record
  unformatted 333
RR minimal trace record
  unformatted 334
RSCH trace record
  formatted 294
SDSP trace record
  formatted 274
setting up a catalogued procedure 232
SLIP DEBUG trace record
  unformatted 338
SLIP trace record
  formatted 295
  unformatted 334
SLIP user trace record
  unformatted 337
source index record 269
SRB trace record
  formatted 301
SRM comprehensive trace record
  unformatted 338
SRM minimal trace record
  unformatted 338
SRM trace record
  formatted 302
SSCH trace record
  formatted 303
  unformatted 339
STAE trace record
  formatted 304
starting GTF
  how to start 229
  START command 239
storage requirement determination 237
SUBSYS trace record
  formatted 283
SVC comprehensive trace record
  unformatted 339
SVC minimal trace record
  unformatted 340
SVC trace record
  formatted 306
system data record
  unformatted 324
system data records 325
system event 229
SYSTEM trace record
  formatted 283
time stamp record 269
trace record format 247
unformatted output 320
use of IPCS to print output 230
use with system trace 230
USR trace record
  formatted 312
XSCCH trace record
  formatted 317
GTF (Generalized trace facility) trace
SLIP standard trace record
  unformatted 337
GTF cataloged procedure
  IBM defaults 230, 231
GTF START command parameter
parm member
  BLOK= 234
  SIZE= 234
GTF trace
ASIDP option 247
CCWP option 247
CSCH option 248
DSP option 248
EXT option 248
HSCH option 248
IO option 248
IOX option 248
JOBNAMEP option 248
MSCH option 249
PCI option 249
PCIE option 249
PFIDP option 249
PI option 249
PIP option 249
reasons for selection 4
RNIO option 249
SI option 250
SIOP option 250
SLIP option 250
SRM option 250
SSCH option 250
SSCHP option 250
SVC option 250
SVCP option 250
SYS option 250
SYSP option 250
SYR option 251
HSCH trace entry
  in system trace 184
  in GTF 248
  HSCH trace record
    formatted 273
  HSCH trace option
    combining options 252
    prompting for 252
    USRP option 251
    XSCH option 252
GTFTRACE subcommand
to format GTF trace 260

H
halt subchannel operation
  GTF record 248
hardcopy log
  and master trace 221
hardware
  error 551
hardware counter and sampling collection
    overview 707
hardware instrumentation services
  how to collect counter and sampling data 707
Hardware Instrumentation Services (HIS)
  overview 707
header record
  for incident record 557
  for logrec data set
    format 557
    used by EREP 557
    used by recording routine 557
HEXFORMAT trace record
  formatted 283

I
I/O interruption
  GTF record 248
I/O operation
  system trace event 184
I/O trace entry
  in system trace 181
I/O trace record
  unformatted 331
IATUX34 installation exit
  for JES3
    to suppress dumps 698
IBM Health Checker for z/OS
  component trace 441
  TRACE command 443
identifer
  for formatted GTF formatted trace record 265
  for system trace entries 172
IDRC (improved data recording capability) feature
  COMPACT parameter 79
IDRDATA control statement
  example 636, 637, 638, 652
  in SPZAP 636, 637, 638, 652, 659
  parameter 659
IEAABD00 parmlib member
  in dump customization 148
  in dump suppression 705
IEACMD00 parmlib member
  in dump customization 34, 63
IEADMP00 parmlib member
  in dump customization 152
  in dump suppression 705
IEADMRO0 parmlib member
  in dump customization 150
  in dump suppression 705
IEASLPxx parmlib member
  for SLIP operator command 697
IEAVADF00 parmlib member
  for JES3
    to suppress dumps 698
IEAVADF00 exit
  in dump customization 150, 153, 160
IEAVADUS exit
  in dump customization 150, 153, 160
IEAVTabx exit
  in dump customization 150, 151, 153
IEAVTABX installation exit
  to suppress dumps 698
IEAVTSDT program 42
IEAVTSEL installation exit
to suppress dumps 698
IFCDIP00 - disk initialization program
application 552
changing logrec data set size 552
reinitializing logrec data set 552
IFCDIP00 service aid
function 552
initializing logrec data set 552
reallocate space on logrec data set 553
JCL example 553
reinitializing logrec data set 553
JCL example 553
IGD17163I 676
IMPEXP= parameter
LISTLOAD control statement 580
incident record
on logrec data set
content 556
record header 557
size 556
on logrec log stream
content 556
size 556
initialization error messages
in stand-alone dump 70
installation exit routine
in dump customization 150, 151, 153, 160
to suppress dumps 698
instruction address trace
reasons for selection 4
interpretation of software record 569
interruption
code 257
I/O 256
program 249, 257
supervisor 250, 258
SVC interrupt 250, 258
system trace event 181
introduction
SNAP dumps 157
IO parameter
dump option 143, 161
IO trace option
in GTF 248
IO=SSCH= keyword
in GTF 257
prompting 257
IOP trace option
in GTF 256
prompting 256
IOS
component trace 451
IOS recovery record 558
IOSB parameter of GTF CCWP 254, 255
IOX trace option
in GTF 248
IOX trace record
formatted 284
IPCS SADMP utility 68
IPCS service aid
reasons for selection 5
IPCS subcommand
VERBEXIT LOGDATA
to format in-storage logrec buffer 568
IPL record 558
RDE option 558
IPL/outage recorder
function 557
IPL= parameter
of AMDSADMP macro 76
IPLEXIST parameter
of AMDSADMP macro 81
J
JCL statement
AMBLIST service aid 578
SPZAP service aid 647
JES common coupling services
compartment trace 457
JES2 subsystem
compartment trace 467
job control language statement in IFCDIP00 553
JCL example 553
JOBNAMEP trace option
in GTF 248, 257
prompting 257
JPA parameter
dump option 143, 161
K
key-length-data format
SDWAVRA 574
keyboard
navigation 709
PF keys 709
shortcut keys 709
L
large dumps
how to send to IBM 673
library lookaside
See LLA 469
link pack area
AMBLIST service aid 582
map 582
mapping using AMBLIST service aid 628
mapping with AMBLIST service aid 588
linkage stack
analysis for diagnosis 164
list a link pack area 582
list CSECT identification record 582
LIST service aid
control statement
LISTLOAD statement 645
description 577
planning 577
LISTHDR control statement
eample 586, 587, 588
format 581
in AMBLIST service aid 581, 582, 586, 587, 588
OUTPUT= 582
MODLIB parameter 582
parameter
DDN= 582
MEMBER= 582
TITLE= 582
LISTLOAD control statement
eample 585, 586, 588, 589
format 579
LISTLOAD control statement (continued)
in AMBLIST service aid 579, 580, 585, 586, 588, 589
parameter
  ADATA= 580
  DDN= 579
  IMPEXP= 580
  MEMBER= 579
  OUTPUT= 579
  RELOC= 580
  TITLE= 579
to list the SSI  645
LISTLPA control statement
example 588, 589
FLPA parameter 583
format 582
in AMBLIST service aid 582, 583, 588, 589
MLPA parameter 583
PLPA parameter 583
LISTOBJ control statement
DDN parameter 581
example 584, 586
format 580
in AMBLIST service aid 580, 581, 584, 586
parameter
  MEMBER= 581
  TITLE= 580
LLA (library lookaside)
component trace 469
LMI recovery record 558
LNKLIB parameter
  of AMDSADMP macro 80
load module
  AMBLIST service aid output 579, 580
  listing 579
  listing current information using AMBLIST service aid 585
load module list
  AMBLIST service aid output 585
locating the logrec buffer 568
locating the WTO buffer 568
locked processing mode
problem data 166
log stream
  JCL specification 562
  log stream data set
    subsys-options2 565
LOGDATA verb 568
logrec
  buffer, recording control 567
  format of buffer 568
  formatting 568
  how to print 561
  recording control buffer 567
LOGREC Buffer
  format in a dump  48
logrec data set
  error recording 556
  purpose 551
header record
  format 557
  function 557
  initializing 552
  must be permanently mounted volume 553
  non-sharable data set 557
  purpose of error record  551
logrec data set (continued)
  reinitialization
    JCL example 553
    reinitializing 552
    reIPL
    JCL example 553
    size 552
    space allocation 553
    JCL example 553
    reallocating 553
    time stamp record 557
    format 557
    type of record 558
    type of record recorded 558
logrec data set - IFCDIP00 reallocation
  description 553
  JCL example 553
logrec data set - IFCDIP00 reinitialization
  description 553
  JCL example 553
logrec data set initialization 552, 553
Logrec data set service aid
  reasons for selection 5
logrec data set software record
  interpretation 569
  SDWA-type 569
    output 569
    symptom record 569
    output 575
logrec error record
  contents 557
  customize 576
  for diagnosis 551
logrec log stream
  defining 554
  error recording 556
  purpose 551
  example of creating a history data set 566
  example of producing an event history 562, 567
  example of using system logger utility 554
  obtaining record 562
  purpose of error record 551
  type of record 558
  type of record recorded 558
logrec recording medium
  changing 556
  planning 552
lost data records
  GTF trace record
    unformatted 322
lost event record
  in GTF trace 270
LPA parameter
  dump option 143, 161
LSQA dumped by stand-alone dump 69
LSQA parameter
  dump option 143, 161
LSR trace record
  formatted 287

M
machine check
  problem data 167
macro expansion messages
  in stand-alone dump 70
main storage dump
description 65
of stand-alone dump 69
map
link pack area 582, 588
nucleus 588
master trace 221, 224
and hardcopy log 221
customize 221
dump output 223
entry in trace table 226
formatting 223
header in trace table 225
receive 222
request 222
start 222
stop 222
MCH record 558
MCH trace entry
in system trace 181
MCIC (machine check interrupt code)
problem identification 168
MDR record 558
MEMBER= parameter
LISTDIR control statement 582
LISTLOAD control statement 579
LISTOBJ control statement 581
membername
for GTF 239
merge
component and GTF trace 262
MERGE subcommand
combining and formatting component trace data 398
message
customize location 704
customize processing 703
from SPZAP 662
produce 701
receive 701
message display
3480 device 120
3490 device 120
3590 device 120
stand-alone dump program 120
messages 701
migrating
one-stage generation
stand-alone dump 100
migration considerations
one-stage generation
stand-alone dump 100
MIH record 558
MINASID parameter
of AMDSADMP macro 79
MLPA parameter
of LISTLPA control statement 583
MOBR trace entry
in system trace 188
mode
cross memory processing mode 166
locked processing mode 166
of processing 165
physically disabled processing mode 166
service request processing mode 166
task processing mode 166
MODE trace entry
in system trace 188
MODE= parameter
in GTF 233
modified link pack area
map 582
MODLIB parameter
of AMDSADMP macro 80
of LISTDIR control statement 582
module
problem data 165
module summary
AMBLIST output for module processed by binder 590, 630
AMBLIST output for module processed by linkage editor 590, 629
MSCH trace entry
in system trace 184
MSCH trace option
in GTF 249
MSCH trace record
formatted 288
MSG= parameter
of AMDSADMP macro 79
MTFIPS replacement
z/OS Problem Documentation Upload Utility (PDUU) 674
multiple error events 48

N
NAME control statement
example 635, 636, 638, 652
in SPZAP 634, 635, 636, 638, 652, 659
parameter 659
navigation
keyboard 709
NOALL parameter
dump option 26, 60
NOALLPSA parameter
dump option 26, 60
NODEFAULTS parameter
dump option 26, 60
NOPROMPT parameter
in GTF 234
NOPS parameter
dump option 26, 60
NOSQA parameter
dump option 26, 60
NOSUM parameter
dump option 26, 60
NOTIFICATIONS parameter
dump option 143
Notices 713
notification
of rapidly recurring dumps 690
NP parameter
in GTF 234
NUC parameter
dump option 143, 161
nucleus
dump 33, 62, 147
mapping using AMBLIST service aid 588
NUCIB parameter
of AMDSADMP macro 80
Index 727
Object module
   for AMBLIST service aid 589
   z/OS UNIX 589
object module list
   obtain 583
object module or GOFF
   listing current information using AMBLIST service
   aid 583
OBR record 558
OPS (operations services component)
   component trace 489
option
   for dump content 22, 142
output 115, 116
   of AMBLIST service aid 590
   of SPZAP 662
output to DASD dump program
   example 81
OUTPUT= parameter
   LISTIDR control statement 582
   LISTLOAD control statement 579
   of AMDSADMP macro 77
overlay, storage
   in pattern recognition 163
overview
   information xv

P
pageable link pack area
   map 582
paging activity
   RSM 502
PARM option
   IGNIDRFULL 648
   of JCL EXEC statement
   for SPZAP service aid 648
parmlib
   CTnXCFxx member 526
parmlib library
   IEADMCxx member 20
   IEASLPxx member 20
PASSIVE mode 681
PC trace entry
   in system trace 179
PCDATA parameter
   dump option 143, 161
PCI trace option
   in GTF 249
PCDMX trace record
   formatted 289
   unformatted 332
PCIE trace option
   in GTF 249
PCIL (PCI load)
   system trace event 189
PCIL trace entry
   in system trace 189
PCILG trace record
   formatted 290
   unformatted 332
PCIS (PCI store)
   system trace event 190
PCIS trace entry
   in system trace 190
PCISTG trace record
   formatted 291
   unformatted 332
PCITAB parameter of GTF CCWP 254, 256
PDMX (PCIIE adapter interruption de-multiplexing)
   system trace event 191
PDMX trace entry
   in system trace 191
PDS/PDSE 673
PFIDP trace option
   in GTF 249
PGM trace entry
   in system trace 192
PGM trace record
   formatted 292
physically disabled processing mode
   problem data 166
PI trace option
   in GTF 249
PI trace record
   comprehensive
   unformatted 332
   formatted 292
   minimal
   unformatted 333
PIP trace option
   in GTF 249
PLPA parameter
   of LISTLPA control statement 583
PMR
   number variables for PDUU 677
PR trace entry
   in system trace 179
print
   ABEND dump 141
   SNAP dump 157, 159
   stand-alone dump 119
   SVC dump 24
   SYSABEND dump 141
   Transaction dump 59
problem
   hardware-detected 154
   software-detected 153
problem data
   from dump 163
   from linkage stack 164
   from module 165
problem state
   AMATERSE 669
program
   system trace event 192
program check
   saved problem data 130
program event
   trace with GTF 229
program object
   AMBLIST service aid output 579, 580
   listing 579
   listing current information using AMBLIST service
   aid 585
   using SPZAP to modify a CSECT
   example 634
Program object
   for AMBLIST service aid 589
   z/OS UNIX 589
REP control statement (continued)
in SPZAP 633, 636, 637, 638, 652, 660
variable 660
resume subchannel data
record 250
retention
message
by AMRF 705
retrieve information from logrec data set
using the EREP program 559
return code 99
from SPZAP 649
return codes
AMATERSE 669
z/OS Problem Documentation Upload Utility (PDUU) 684
REUSEDS parameter
of AMDSADMP macro 79
RGN parameter
dump option 143
RNIO formatted trace record
formatted 293
RNIO trace option
in GTF 249
RR trace record
comprehensive
unformatted 333
minimal
unformatted 334
RRS (resource recovery services)
component trace 494
RSCH trace entry
in system trace 184
RSCH trace record
formatted 294
RSM
trace data 502
RSM (real storage manager)
component trace 501
RST trace entry
in system trace 181
S
S11SI parameter of GTF CCWP 254
SA parameter
dump option 143, 161
SA= parameter
in GTF 234
SADMP message
display example 120
MSADMP#U 120
NTRDY message 120
RSADMP# 120
RSADMP# U 120
SADMP# 120
status information 3480 device 120
SADMP= parameter
in GTF 234
SAH parameter
dump option 143, 161
sample JCL
one-stage generation 96
DASD 98
tape 98
sample JCL for migrating
one-stage generation
  DASD  100
  tape  102
scheduled SVC dump  42
SD= parameter
  in GTF  234
SDSP trace record
  formatted  274
SDUMP= parameter
  in GTF  234
SDUMPX 4K SQA buffer
  content  53
SDUMPX macro
to request SVC dump  19
SDWA (system diagnostic work area)
  recordable extension  574
SDWAVRA (SDWA variable recording area)
  key-length-data format  574
secondary extent
  allocation  14
calculation  14
  SPACE requirement  14
security
  problems with APPC/MVS component  409
self-dump
  of stand-alone dump  113
sending comments to IBM xvii
sequential data set
  extended
to hold large dumps  13
service aid selection
  AMATERSE  2
  AMBLIST  2
  DAE  2
  IPCS  2
  Logrec data set  2
  SLIP  2
  SPZAP  2
service aids and tools
descriptions  2
  selection  1
service request processing mode
  problem data  166
SETRP macro
for requesting ABEND dumps  140
  in dump customization  149, 151, 152
SETSSI control statement
  example  636, 637, 638
  in SPZAP  636, 637, 638, 645, 661
  parameter  661
shortcut keys  709
SIGA trace entry
  in system trace  184
SIO trace option
  in GTF  250
SIOP trace option
  in GTF  250
SLH record  559
SLIP command
  SDUMPX 4K SQA buffer data  53
  SLIP work area data  129
SLIP data
  record  250
SLIP debug trace record
  formatted  300
  unformatted  338
SLIP operator command
  control dump contents  33, 62
  to request SVC dump  20
  to suppress dumps  697
SLIP service aid
  reasons for selection  5
SLIP standard trace record
  formatted  299
  unformatted  337
SLIP trace option
  in GTF  250
SLIP user trace record
  formatted  299
  unformatted  337
SLIP work area
  content  129
SNAP dump
  contents  160
  customization  160
  introduction  157
  reasons for selection  3
  request  157
SNAP macro
  for requesting dumps  159
SNAP or SNAPX macro
  in dump customization  149, 150, 152
software
  error  551
  software record  559, 569
detail edit report  569
  interpretation  569
source index record
  in GTF trace  269
source JCL
  for component trace external writer  388
SPER trace entry
  in system trace  192
SPI (service processor interface)
  component trace  518
SPIN trace entry
  in system trace  199
SPLS option
  in dump customization  148, 149, 152
SPLS parameter
dump option  143, 161
SPR2 trace entry
  in system trace  192
SPZAP example
  inspect and modify CSECT in z/OS UNIX  634
SPZAP service aid
  accessing a load module  641
  control statement  656
  ABDUMP  644, 652
  ABDUMPT  644, 652
  BASE statement  641, 652, 654
  CCHHR statement  642, 655
  CHECKSUM statement  656
  CONSOLE statement  657
  DUMP  636, 637, 638, 652, 657
  DUMPT  636, 637, 638, 652, 657
  IDRDATA statement  636, 637, 638, 652, 659
  NAME statement  634, 635, 636, 638, 652, 659
  RECORD statement  642
  REP statement  633, 636, 637, 638, 652, 660
  rules for coding  651
Index 731
SYSIN DD statement (continued)
used in SPZAP 657
SYSIOS component trace
  CTnIOSxx parmlib member 453
  SHORT report 456
  TRACE command 453
  trace request options 453
SYSJES component trace
  CTnJESxx parmlib member 459
  format options 462
  request sublevels 458
  requesting 458
  TRACE command 459
  verifying 396
SYSjes2 component trace
  format options 467
  request sublevels 467
  requesting 467
SYSLIB DD statement 635, 636, 637, 643, 647, 648, 652
  used in SPZAP 657
SYSLOGR component trace
  CTnLOGxx parmlib member 473
  output 477
  trace request options 475
SYSM trace option
  in GTF 251
SYSDUMP
  analysis approach 135
SYSDUMP dump
  customization 150
  preallocate data set 138
  suppressing 687
  symptom dump 703
  to VIO data set 138
SYSMVS component trace
  CTnBPXX parmlib member 478
  format options 481
  FULL report 483
  SUMMARY report 487
  TRACE command 479
  trace request options 479
SYSP component trace
  CTnOPSxx parmlib member 490
  format options 492
  TRACE command 490
  trace options 490
  trace request options 490
SYSPRINT
  z/OS Problem Documentation Upload Utility (PDUU) 675
  component trace data sets 389
  component tracing in systems 393
SYSPRINT DD
  used in AMBLIST service aid 578
SYSPUNCH DD statement 643, 648
  in stand-alone dump 97
SYSSP component trace
  CTnRRSxx parmlib member 495
  FULL report 499
  SHORT report 499
  SUMMARY report 499
SYSSRRS component trace (continued)
  TALLY report 499
  TRACE command 495
  trace request options 495
SYSSRSM component trace
  CTnRSMxx parmlib member 502
  FULL report 517
  TRACE command 502
  trace request options 502
SYSSSPI traces
  OPTIONS 519
  system data records
    GTF trace record
      unformatted 324
  system event
    trace with GTF 229
  System logger component
    component trace 470
  system mode
    in STATUS FAILDATA report 43
  system resource manager data record 250
  system restart
    for stand-alone dump 112
  System REXX Component
    See System REXX Component 413
  System REXX Component (System REXX Component)
    component trace 413
  system service
    system trace event 204
  system spin
    system trace event 199
  system status index
    function 645
  system trace 169
    ACR trace entry 174
    addressing mode trace entries 188
    AINT trace entry 176
    altering options 177
    ALTR trace entry 177
    branch trace entries 178
    BSG trace entry 179
    CALL trace entry 181
    CLKC trace entry 181
    CSCH trace, entry 184
    customizing 169
    DSP trace entry 186
    EMS trace entry 181
    entries 171
    entry identifiers 172
    example in a dump 172
    EXT trace entry 181
    format in a dump 50
    formatting output 171
    HSCCH trace entry 184
    I/O trace entry 181
    increasing the size of trace table 169
    MCH trace entry 181
    MOBR trace entry 188
    MODE trace entry 188
    MSCH trace entry 184
    PC trace entry 179
    PCIL trace entry 189
    PCIS trace entry 190
    PDMX trace entry 191
    PGM trace entry 192
    PR trace entry 179

trace selection (continued)
  GTF trace 2
  master trace 2
  system trace 2
trace table
  for master trace 224
trace table in storage 224
trace VTAM network activity 249
traces
  description 2
trademarks 715
Transaction dump
  asynchronous 55
  automatically allocated data set 56
  clearing 59
  contents 59
  copying 59
  customization 59
  introduction 55
  planning dump data set 55
  printing 59
  request 58
  summary dump contents 62
  synchronous 55
  viewing 59
transaction trace
  SYSTTRC for transaction trace 520
Transaction Trace
  introduction 539
TRC trace option
  in GTF 251
troubleshooting
  AMATERSE 668
TRT parameter
  dump option 143, 161
TTRC (transaction trace)
  transaction trace 520
two-stage generation
  migration 105
  overriding 107
type of record
  ANR record 558
  CRW record 558
  DASD-SIM recovery record 558
  EOD record 558
  ETR recovery record 558
  IOS recovery record 558
  IPL record 558
  LMI recovery record 558
  MCH record 558
  MDR record 558
  MIH record 558
  OBR record 558
  SLH record 559
  software record 559

U
ULABEL= parameter
  of AMDSADMP macro 77
unformatted dump
  data set 55
  of GTF 320
unformatted GTF records
  control records 321
  lost data records 322
  system data records 324
user trace records 323
unformatted output
  of GTF 320
user
  system trace event 218
user data records
  GTF trace record 323
  unformatted 323
user interface
  ISPF 709
  TSO/E 709
user trace data
  record 251, 252
userid.NETRC data set 681
USR trace option
  in GTF 251
USR trace record
  formatted 312
USRn trace entry
  in system trace 218
USRp trace option
  in GTF 251, 258
prompting 258

V
VERBEXIT MTRACE subcommand
  dump output 223
  to format master trace 223
VERBEXIT subcommand
  LOGDATA report 48
  TRACE report 50
verify
  component trace 395
  external writer 395
VERIFY control statement
  example 633, 635, 637, 638, 652
  in SPZAP 633, 635, 637, 638, 652, 661, 662
  parameter 661, 662
view
  ABEND dump 141
  SVC dump 24
  Transaction dump 59
viewing
  component trace data 398
  virtual lookaside facility
    See VLF 520
virtual lookaside facility
  component trace 520
VIRTUAL= parameter
  of AMDSADMP macro 77
VSAM object
  access 649
VTAM network activity
  record 249
W
    wait state code
        issued by stand-alone dump 108, 113
    WAIT trace entry
        in system trace 186
    wait-reason code
        issued by stand-alone dump 110
        unload a tape 112
    WLM (workload manager)
        component trace 523
    wrap
        of component trace data sets 390
    WRAP parameter
        TRACE CT command 390
    WTI trace entry
        in system trace 181
    WTO recording control buffer location 568

X
    XCF (cross-system coupling facility)
        component trace 526
    XSCH trace entry
        in system trace 184
    XSCH trace option
        in GTF 252
    XSCH trace record
        formatted 317
    xsd listing
        AMBLIST output for LISTOBJ with XSD record 594

Z
    z/OS MVS Diagnosis: Tools and Service Aids
        summary of changes xix, xx
    z/OS Problem Documentation Upload Utility (PDUU)
        AMADUPL 673
        AMATERSE 667
        compression 673
        data set
            supported 674
            unsupported 674
        DEBUG 678
        encryption 673
        encryption key 677
        example
            DEBUG data set 682
            FTP connection using a proxy server 679
            FTP proxy server with FTPCMDs DD 680
            FTP proxy server with user ID 679
            multiple FTPCMDs DD statements 684
            PASSIVE mode 681
            port specification 680
            simple FTP connection 678
            SYSIN continuation record 683
            SYSUT2 682
            userid.NETRC data set 681
            FTPCMDs 678
            JCL examples 678
            JCL statements 675
            messages 674
            number of sessions 673
            overview 673
            partitioned data sets 673
            planning use 674
            PMR variables 677

z/OS Problem Documentation Upload Utility (PDUU) (continued)
    prerequisites 674
    restrictions 674
    return codes 684
    SYSIN
        ACCOUNT 676
        CC-FTP 677
        CIPHER_KEY 677
        DATACLAS 677
        DIRECTORY 677
        KEEP_WORK 676
        MGMTCLAS 677
        NO_FTP 677
        PASSWORD 676
        PMR 677
        STORCLAS 677
        TARGET_DSN 676
        TARGET_SYS 675
        USERID 676
        WORK_DSN 676
        WORK_DSN_SIZE 676
        SYSTSPRT 678
z/OS UNIX
    component trace 478
    example 589
    File support 589
    in AMBLIST service aid 589
    NAME control statement 634
    REP statement 633
    SPZAP example
        inspect and modify CSECT 634
        VERIFY statement 633
        zeroed page
dump suppression 69