EMS Manual

Abstract
This manual describes the Event Management Service (EMS). EMS is a collection of processes, tools, and interfaces that provide event-message collection and distribution on a system running the HP NonStop™ Kernel operating system. This manual is intended for programmers, operators, and those who configure and manage systems and networks.

Product Version
EMS H02

Supported Release Version Updates (RVUs)
This manual supports D30.02 and all subsequent D-series RVUs, G01.00 and all subsequent G-series, H01.00 and all subsequent H-series and J01.00 and all subsequent J-series RVUs until otherwise indicated by its replacement publication.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>426909-005</td>
<td>October 2012</td>
</tr>
<tr>
<td>Part Number</td>
<td>Product Version</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>426909-001</td>
<td>SPI F40, EMS F40</td>
</tr>
<tr>
<td>426909-002</td>
<td>SPI G05, EMS G06</td>
</tr>
<tr>
<td>426909-003</td>
<td>SPI EMS G06</td>
</tr>
<tr>
<td>426909-004</td>
<td>SPI EMS H02</td>
</tr>
<tr>
<td>426909-005</td>
<td>SPI EMS H02</td>
</tr>
</tbody>
</table>
Legal Notices

© Copyright 2012 Hewlett-Packard Development Company L.P.
Confidential computer software. Valid license from HP required for possession, use or copying.
Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software
Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under
vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP
products and services are set forth in the express warranty statements accompanying such products
and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be
liable for technical or editorial errors or omissions contained herein.

Export of the information contained in this publication may require authorization from the U.S.
Department of Commerce.

Microsoft, Windows, and Windows NT are U.S. registered trademarks of Microsoft Corporation.

Intel, Itanium, Pentium, and Celeron are trademarks or registered trademarks of Intel Corporation or its
subsidiaries in the United States and other countries.

Java® is a U.S. trademark of Oracle and/or its affiliates.

Motif, OSF/1, UNIX, X/Open, and the "X" device are registered trademarks and IT DialTone and The
Open Group are trademarks of The Open Group in the U.S. and other countries.

Open Software Foundation, OSF, the OSF logo, OSF/1, OSF/Motif, and Motif are trademarks of the
Open Software Foundation, Inc.

OSF MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE OSF MATERIAL PROVIDED
HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

OSF shall not be liable for errors contained herein or for incidental consequential damages in
connection with the furnishing, performance, or use of this material.

© 1990, 1991, 1992, 1993 Open Software Foundation, Inc. This documentation and the software to
which it relates are derived in part from materials supplied by the following:

1992 International Business Machines Corporation. © 1988, 1989 Massachusetts Institute of

This software and documentation are based in part on the Fourth Berkeley Software Distribution
under license from The Regents of the University of California. OSF acknowledges the following
individuals and institutions for their role in its development: Kenneth C.R.C. Arnold,
Gregory S. Couch, Conrad C. Huang, Ed James, Symmetric Computer Systems, Robert Elz. © 1980,

Printed in the US
Part I: Introduction to EMS

1. Introduction to EMS
   EMS in the System Environment 1-2
   EMS Communications in the DSM Environment 1-3
   EMS Capabilities and Features 1-5
   Basic Capabilities 1-6
   Key Features 1-6
   Applications of EMS 1-8

2. EMS Components and Architecture
   Event Messages 2-2
   Information Contained in Event Messages 2-2
   Special Kinds of Event Messages 2-3
   Flow of Event Messages 2-5
   Event Message Collectors 2-7
   Primary Collector ($0) 2-7
   Alternate Collectors 2-7
   Event Message Collector Functions 2-7
   Subsystem Support 2-8
Part II: Using EMS

3. Retrieving Event Messages Interactively

Displaying Current Event Messages 3-2
Displaying Saved Event Messages 3-2
Creating Simple Compiled Filters 3-3
Creating Simple Filter Tables 3-4

4. Retrieving Event Messages Programmatically

Getting Started 4-2
Changing the Environment 4-4
Filters and Filter Parameters 4-4
Event-Message Sources 4-6
Log File Position 4-10
Specifying Multiple Parameters in One Command 4-12

Retrieving Event Information 4-12
Obtaining an Event Message (GETEVENT) 4-12
Extracting Tokens From the Response (SSGET) 4-16
Extracting Tokens From the Event Message (EMSGET) 4-16
Generating Display Text (EMTEXT) 4-18
5. Compiled Filters

Writing and Compiling Filters 5-1
Filter Operation 5-1
Introduction to the Filter Language 5-2
Creating a Compiled Filter 5-3
Considerations for Writing Filters 5-5

The Filter Language 5-7
Overview of Filter Operation 5-8
TACL Environment 5-8
Basic Components 5-9
Tokens 5-10
Constants 5-15
Bit-Extraction Operator 5-19
Comparisons 5-19
Boolean Expressions 5-25
Declarations 5-26
Statements 5-30
Functions 5-38

The Filter Compiler 5-44
Compiler Input and Output 5-44
Compiler Directives 5-45
Compiler Invocation 5-46
Compiler Errors and Warnings 5-47
Completion Codes 5-62

6. Filter Tables and Burst Filters

Filter Tables 6-1
Filter Table Features and Functions 6-1
Filter Table Format 6-2
Filter Table Keywords 6-3
Filter Table Errors 6-5
Multiple Filters 6-6
Recovery Actions 6-6
Loading Filter Tables 6-6
Logical Connection Is AND 6-7
Restrictions 6-7

Burst Filters 6-7
7. Burst Detection and Suppression

Event Suppression Strategies  7-1
Suppressing Event Bursts  7-1
Suppressing All Events of a Specific Type  7-1

BDS Features  7-2
BDS Parameters  7-3
BDS Configuration Examples  7-5

Implementing BDS  7-6
Implementing BDS From a Primary Collector  7-6
Implementing BDS From an Alternate Collector  7-6
Implementing BDS From a Distributor  7-7
Implementing Specific-Event Suppression  7-7
Implementing Specific-Event Suppression From a Primary Collector  7-8
Implementing Specific-Event Suppression From an Alternate Collector  7-8
Implementing Specific-Event Suppression From a Distributor  7-8

Types of Collector Events BDS and PLF Do Not Suppress  7-9

Part III: Designing and Implementing Event Reporting

8. Reporting Events

Task 1: Decide What Events to Report  8-2
Task 1.1: Decide Which Detectable Events to Report to EMS  8-2
Task 1.2: Decide Which Reported Events Are Critical or Action Events  8-3

Task 2: Decide What to Include in Event Messages  8-5
General Guidelines  8-5
What Is Provided for You  8-6
What You Must Provide  8-6
What You Can Optionally Provide  8-10

Task 3: Create a Data Definition File  8-11

Task 4: Write the EMS Interface in Your Program  8-12
Example: Writing an EMS Interface Into a Program  8-12

9. Standard Events

Introduction to Standard Events  9-2
Requirements for Standard Events  9-3
Object State Monitoring Functions  9-4
Reactive Problem Management Functions  9-5
10. Generating Standard Events

Task 1. Determine Your Subsystem ID and Acronym 10-2
Task 2. Analyze Your Subsystem Environment 10-2
  Task 2.1: Identify Types of Objects to Manage in Your Subsystem 10-3
  Task 2.2: Identify the Characteristics of Your Objects 10-4
  Task 2.3: Identify the State Transitions of Your Objects 10-4
  Task 2.4: Identify the Events for Your Subsystem 10-5
  Task 2.5: Design Your Asynchronous Management Interface 10-6
  Task 2.6: Design Your Command and Control Management Interfaces 10-6
Task 3. Generate Standard Events for Your Subsystem 10-6
Task 3.1: Determine the Operational States of Your Objects and Subsystem Functions 10-6
Task 3.2: Customize Your Standard Events 10-8

Task 4. Define Private Event Types for Your Subsystem 10-9
Task 4.1: Determine Management Functions and If EMS Is the Appropriate Platform 10-10
Task 4.2: Specify Which Operations Are Defined for Your Management Functions 10-11
Task 4.3: Specify the System Data Needed to Automate These Operations 10-11
Task 4.4: Specify the EMS Event Types for Your System Data 10-13

Task 5. Migrate Existing Events 10-13
Migration Rules 10-13
Incorporating Old Events Into New Events 10-14
Revising Old Events 10-14

Task 6. Design Your Event Messages 10-15

Task 7. Write Your Event External Specification 10-17
Task 7.1: Prepare the Subsystem Event External Specification Template 10-17
Task 7.2: Describe Your Subsystem Environment 10-17
Task 7.3: List the Standard Management Functions Your Subsystem Supports 10-18
Task 7.4: Describe the Private Management Functions You Support 10-19
Task 7.5: Define Your Event Subjects 10-19
Task 7.6: Define the Event Types You Support 10-19
Task 7.7: Define Your Event Numbers and Their CDMT Attributes 10-19
Task 7.8: Enumerate Your Private Values to Standard EMS Tokens 10-20
Task 7.9: List Private Tokens From Other Subsystems 10-20
Task 7.10: List Private Tokens From Your Subsystem 10-20
Task 7.11: Describe the Details of Each Event Message 10-20

Task 8. Create and Build Your DDL Definitions 10-22
Task 8.1: Prepare the DDL Definitions File Template 10-22
Task 8.2: Specify Subsystem Product Information 10-22
Task 8.3: Define Your Subsystem ID (SSID) 10-22
Task 8.4: Define Your External SSID 10-23
Task 8.5: Specify Event Number Definitions ZEMS-TKN-EVENTNUMBER 10-23
Task 8.6: Specify Private Event Type Definitions ZEMS-TKN-CONTENT-USER 10-23
Task 8.7: Specify Private Enumerations for Standard Tokens 10-23
Task 8.8: Specify Private Enumerations of Other Subsystems' Tokens 10-24
Task 8.9: Specify Private Enumerations of Your Subsystem's Tokens 10-24
Task 8.10: Define Event Subjects 10-24
11. Procedure Calls for Standard Events

**Introduction to Procedure Calls** 11-2

**EMS_TRANSIENT_FAULT_EVT_BLD** Procedure 11-2
  - Condition Code Settings 11-5
  - Considerations 11-6

**EMS_OBJ_UNAVAIL_EVT_BLD** Procedure 11-7
  - Condition Code Settings 11-11
  - Considerations 11-11

**EMS_OBJ_AVAIL_EVT_BLD** Procedure 11-13
  - Condition Code Settings 11-16
  - Considerations 11-16

**EMS_OTHER_STATE_CHANGE_EVT_BLD** Procedure 11-18
  - Condition Code Settings 11-21
  - Considerations 11-21

**EMS_OPER_ATTN_NEEDED_EVT_BLD** Procedure 11-23
  - Condition Code Settings 11-26
  - Considerations 11-26

**EMS_OPER_ATTN_COMPED_EVT_BLD** Procedure 11-27
  - Condition Code Settings 11-30
Part IV: Configuring and Maintaining EMS

12. Configuring EMS

Basic Attributes of EMS Components 12-1
   Primary Collectors 12-2
   Alternate Collectors 12-2
   Log Files 12-3
   Consumer, Forwarding, and Printing Distributors 12-12
   Compatibility Distributor ($Z0) 12-14
   Other EMS Components 12-15

Configuration Issues 12-16
   Task Requirements 12-16
   Networking Considerations 12-16
   Logging Integrity 12-17
   Delivery Integrity 12-19
   Performance 12-19
   Reliability 12-20
   Resources 12-20
   Security 12-20

Installation and System Generation Considerations 12-21
   The Primary Collector 12-21
   The Compatibility Distributor ($Z0) 12-22
   Template Files 12-23

Log File Operation 12-23
   Collector Context 12-23
13. EMS Programs

EMSACOLL—Alternate Collector Program  13-2
  Startup Error Messages  13-6
  Startup Warning Messages  13-8
EMSCCTRL—Control Collector Utility  13-9
  Referencing Collector Attributes  13-17
EMSCINFO—Collector Information Utility  13-18
  EMSCINFO Display Examples (No DETAIL Parameter)  13-18
  Definition of Terms  13-19
  EMSCINFO Display Examples (With DETAIL Option)  13-22
EMSDINFO—Distributor Information Utility  13-29
EMSDIST—Distributor Program  13-34
  Startup Error Messages  13-40
  Translating an EMS Event Into an SNMP Trap  13-51

14. EMS Definitions

Event-Message Overview  14-1
  Terminology  14-1
  Header Tokens Overview  14-2
  Header Tokens Usage Restrictions  14-2
  Event Message Requirements and Conventions  14-2
  Event-Message Restrictions  14-3
  Definitions of Event-Message Tokens  14-3
  Definitions of SPI Tokens in the Event-Message Header  14-3
  Definitions of EMS Tokens in the Event Message Header  14-4
  Definitions of EMS Special Token Codes  14-8
  Definitions of EMS Data-Portion Tokens  14-9
  SPI Data-Portion Tokens  14-10

15. EMS Procedures

Which Procedures to Use  15-2
  Passing Token Parameters by Value or by Reference  15-2
  Procedure Summary Tables  15-2
  Examples  15-4
  Required Declarations and Standard Definitions  15-7
  Standard Definitions in TAL Programs  15-7
16. Event Routing

Routing Capability 16-1
Launching of Destination Processes 16-1
Formatting Selection 16-2
Multiple Filters 16-2
User Interfaces 16-3
  Startup of Multiple Filters 16-3
  Filter PASS and DESTINATION Statements 16-3
Selection of Collectors for Internal Events 16-3
Distributor Generated Messages 16-4

Part V: Using EMS Distributors and Collectors

17. Distributor Commands and Responses

Extended Programmatic Interface 17-2
Object Support Summary 17-3
Common Definitions for ZCOM- Commands 17-5
  Common Command Tokens for ZCOM- Commands 17-5
  Common Response Tokens for ZCOM- Commands 17-7
Common Definitions for ZEMS- Commands 17-9
  Common Command Tokens 17-9
  Distributor Errors 17-10
  Error Lists 17-10
  Distributor Error Numbers 17-11
Distributor Command Descriptions 17-12
  ADD Command (ZCOM-CMD-ADD) 17-12
  ALTER Command (ZCOM-CMD-ALTER) 17-13
  CONTROL Command (ZEMS-CMD-CONTROL) 17-17
  DELETE Command (ZCOM-CMD-DELETE) 17-24
  GETEVENT Command (ZEMS-CMD-GETEVENT) 17-25
  GETVERSION Command (ZEMS-CMD-GETVERSION) 17-28
18. Distributor Event Messages

Event Message Descriptions 18-3
Token and Data Type Definitions 18-3

SPI Token Codes 18-3
EMS Token Codes 18-4

538: ZEMS-EVT-BURST-START 18-7
539: ZEMS-EVT-BURST-END 18-10
1000: ZEMS-EVT-LOG-ACCESS 18-13
1001: ZEMS-EVT-COLL-ACCESS 18-15
1002: ZEMS-EVT-DEST-ACCESS 18-17
1003: ZEMS-EVT-LOGFILE-EOF 18-19
1005: ZEMS-EVT-BAD-FILTER 18-21
1006: ZEMS-EVT-COLL-PROTOCOL 18-23
1007: ZEMS-EVT-BAD-EVENT 18-25
1008: ZEMS-EVT-DEVTYPE 18-27
1009: ZEMS-EVT-INTERNAL-ERROR 18-29
1010: ZEMS-EVT-CHECKOPEN-FAILED 18-30
1011: ZEMS-EVT-TAKEOVER 18-32
1012: ZEMS-EVT-CREATEBACKUP-FAILED 18-34
1013: ZEMS-EVT-BACKUP-CREATED 18-37
1014: ZEMS-EVT-BACKUP-ABENDED 18-39
1015: ZEMS-EVT-BACKUP-DELETED 18-41
1016: ZEMS-EVT-CHECKPOINT-FAILED 18-42
1017: ZEMS-EVT-BAD-LOG 18-44
1018: ZEMS-EVT-FILES-LOST 18-46
1019: ZEMS-EVT-COL-DISCONNECT 18-49
1020: ZEMS-EVT-STARTUP-FAILED 18-51
1021: ZEMS-EVT-STARTUP-OK 18-54
1022: ZEMS-EVT-WRITE-FAILED 18-56

19. Collector Commands and Responses

SPI Command and Response Buffers 19-1
Sending a SPI Command 19-2
Summary of EMS Collector Commands 19-2
Common Definitions of ZCOM- Commands 19-4
Common Command Tokens for ZCOM- Commands 19-5
20. Collector Event Messages

Types of Collector-Generated Messages  20-1
  Tokenized Operator-Console Messages  20-1
  Tokenized Text Messages  20-1
  Primary Collector Event Messages  20-1
  Alternate Collector Event Messages  20-2

Summary of Collector-Generated Messages  20-2

Tokenized Operator-Console Messages (1–511)  20-4

Tokenized Text Messages (512)  20-7

Collector-Specific Event Messages  20-9
  513: ZEMS-EVT-COLD-LOAD  20-9
  514: ZEMS-EVT-FILESWITCH  20-11
  515: ZEMS-EVT-COLL-DISC-FAILED  20-14
  517: ZEMS-EVT-COMPAT-DISTR-STopped  20-16
  518: ZEMS-EVT-COL-EVENT-DISCARcAUDS  20-18
  519: ZEMS-EVT-MSGR-EVENTS-DISCARcAED  20-20
  520: ZEMS-EVT-FILE-ROTAcE-PURGE  20-22
  521: ZEMS-EVT-LOGGING-STOpPEd  20-24
  522: ZEMS-EVT-COLLECTOR-RUN  20-26
  523: ZEMS-EVT-ACOL-EVENT-DISCARcAUDS  20-28
21. Distributor Errors

Token and Data Type Definitions 21-1

SPI Token Codes 21-1
EMS Token Codes 21-3

Distributor Warning Codes 21-4

501: ZEMS-WRN-EOF 21-4
502: ZEMS-WRN-TOO-EARLY 21-4
503: ZEMS-WRN-TOO-LATE 21-5
504: ZEMS-WRN-STARTUP-OK 21-5

Distributor Error Codes 21-6

1001: ZEMS-ERR-VERSION 21-6
1002: ZEMS-ERR-INV-CMD 21-6
1003: ZEMS-ERR-INV-SSID 21-6
1004: ZEMS-ERR-INV-TKN 21-7
1005: ZEMS-ERR-INV-VALUE 21-7
1006: ZEMS-ERR-DUP-TKN 21-7
1007: ZEMS-ERR-MODE-TKN 21-8
21. Distributor Errors

1008: ZEMS-ERR-INV-OBJECT 21-8
1014: ZEMS-ERR-INV-OP 21-8
1015: ZEMS-ERR-REQ-TKN 21-8
1016: ZEMS-ERR-INV-HEADERTYPE 21-9
1018: ZEMS-ERR-COLL-ACCESS 21-9
1019: ZEMS-ERR-FLT-FORM 21-9
1020: ZEMS-ERR-FLT-LOAD 21-10
1022: ZEMS-ERR-REQ-PARAM 21-10
1024: ZEMS-ERR-HIST-MODE 21-11
1025: ZEMS-ERR-MAX-COLLECTOR 21-11
1026: ZEMS-ERR-COLLECTOR-EXISTS 21-11
1027: ZEMS-ERR-COLL-NOT-FOUND 21-12
1031: ZEMS-ERR-LOG-ACCESS 21-12
1032: ZEMS-ERR-EOF 21-13
1033: ZEMS-ERR-FORWARD-SEARCH 21-13
1035: ZEMS-ERR-MAX-DEST 21-13
1036: ZEMS-ERR-DEST-ACCESS 21-14
1037: ZEMS-ERR-DEST-EXISTS 21-14
1038: ZEMS-ERR-DEST-NOT-FOUND 21-15
1039: ZEMS-ERR-CONTEXT 21-15
1041: ZEMS-ERR-ZSPI 21-15
1042: ZEMS-ERR-BAD-FILTER 21-16
1043: ZEMS-ERR-NO-POOL 21-16
1044: ZEMS-ERR-NO-EVENT-SOURCE 21-16
1045: ZEMS-ERR-DEVTYPE 21-17
1046: ZEMS-ERR-COLLECTOR-PROTOCOL 21-17
1047: ZEMS-ERR-BAD-EVENT 21-18
1049: ZEMS-ERR-BAD-LOG 21-18
1050: ZEMS-ERR-COLL-DISCONNECT 21-19
1051: ZEMS-ERR-FILES-LOST 21-19
1052: ZEMS-ERR-STATUS-ONLY 21-20
1059: ZEMS-ERR-MAXFLT 21-20
1060: ZEMS-ERR-FLT-ALLOC 21-20
1061: ZEMS-ERR-DIST-ALLOC 21-21
1062: ZEMS-ERR-INV-PROFILE 21-21
1063: ZEMS-ERR-DEST-CONFLICT 21-21
1064: ZEMS-ERR-STARTUP-FAILED 21-22
1065: ZEMS-ERR-WRITE-FAILED 21-22
22. Collector Errors

ZCOM- Errors (Over Extended SPI Interface) 22-2
Token and Data Type Definitions for ZCOM- Errors 22-2
-3: ZCOM-ERR-CMD-INV-IN-SUMSTATE 22-4
-4: ZCOM-ERR-CMD-MISMATCH 22-4
-5: ZCOM-ERR-CMD-NOT-SUPP 22-5
-15: ZCOM-ERR-OBJ-ALRDY-DEF 22-5
-17: ZCOM-ERR-OBJ-NOT-FOUND 22-5
-22: ZCOM-ERR-SECUR-VIOL 22-6
-23: ZCOM-ERR-SPI-ERROR 22-6
-24: ZCOM-ERR-SSID-INV 22-6
-25: ZCOM-ERR-SUB-NOT-FOUND 22-7
-26: ZCOM-ERR-TKN-CODE-INV 22-7
-27: ZCOM-ERR-TKN-DUP 22-7
-28: ZCOM-ERR-TKN-LEN-INV 22-8
-29: ZCOM-ERR-TKN-RECH 22-8
-30: ZCOM-ERR-TKN-VAL-INV 22-8
-32: ZCOM-ERR-VSN-INCOMP 22-9
-33: ZCOM-ERR-EMPT-RSP 22-9

ZEMS- Errors (Over Extended SPI Interface) 22-10
Token and Data Type Definitions for ZEMS- Errors 22-10
1008: ZEMS-ERR-INV-OBJECT 22-12
1009: ZEMS-ERR-INV-CPU 22-12
1010: ZEMS-ERR-CPU-RANGE 22-13
1013: ZEMS-ERR-CDIST-CPU 22-13
1019: ZEMS-ERR-FLT-FORM 22-13
1020: ZEMS-ERR-FLT-LOAD 22-14
1034: ZEMS-ERR-OPN-LOG 22-15
1036: ZEMS-ERR-DEST-ACCESS 22-15
1040: ZEMS-ERR-ZFIL 22-16
1053: ZEMS-ERR-INV-MODE 22-16
1054: ZEMS-ERR-CDIST-DOWN 22-17
1059: ZEMS-ERR-MAXFLT 22-17
1060: ZEMS-ERR-FLT-ALLOC 22-17
1067: ZEMS-ERR-ZOPR-SEND 22-18
1068: ZEMS-ERR-ZOPR-SYNC 22-18

ZEMS- Errors (Over Basic SPI Interface) 22-19
Token and Data Type Definitions 22-19
Part VI: EMS Example Files

A. Example of Retrieving Event Messages

- Running the TAL Version A-2
- Running the TACL Version A-2
- Running the COBOL Version A-2
- Running the C Version A-2
- Overview of Application Logic A-3
- DDL Source File A-4
- Filter Source File A-5
- TAL Source File A-5
- TACL Source File A-14
- COBOL Source File A-19
- C Source File A-24
- Program Modifications A-34
  - Processing Retrieved Event Messages A-34
B. Example of Reporting Events

Testing the Program   B-1
The MAKE TACL Macro File   B-3
The DDL Compile-Control File   B-3
The DDL Source File   B-4
The SOURTMPL File   B-5
The C Source File   B-6
The TACL Source File   B-10

C. Standard Event Sample Files

Event External Specification Sample File   C-1
DDL Definitions Sample File   C-20
EMS Templates Sample File   C-30

D. EMS Usage Best Practices and Recommendations

Introduction   D-1
Event Requirements Guidelines   D-1
   Who is the audience for the events?   D-1
   How will the events be used?   D-1
Event Content Guidelines   D-1
Event Destination Guidelines   D-3
Event Subject Guidelines   D-3
Event Severity Guidelines   D-3
Event Emphasis Guidelines   D-4
Event Text Guidelines   D-4
Event Frequency Guidelines   D-5
Event Specifications Guidelines   D-5
Event Sustaining Guidelines   D-5

Index
What’s New in This Manual

Manual Information

Abstract
This manual describes the Event Management Service (EMS). EMS is a collection of processes, tools, and interfaces that provide event-message collection and distribution on a system running the HP NonStop™ Kernel operating system. This manual is intended for programmers, operators, and those who configure and manage systems and networks.

Product Version
SPI EMS H02

Supported Release Version Updates (RVUs)
This manual supports D30.02 and all subsequent D-series RVUs, G01.00 and all subsequent G-series, H01.00 and all subsequent H-series and J01.00 and all subsequent J-series RVUs until otherwise indicated by its replacement publication.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>426909-005</td>
<td>October 2012</td>
</tr>
</tbody>
</table>

Document History

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Product Version</th>
<th>Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>426909-001</td>
<td>SPI F40, EMS F40</td>
<td>February 2001</td>
</tr>
<tr>
<td>426909-002</td>
<td>SPI G05, EMS G06</td>
<td>May 2002</td>
</tr>
<tr>
<td>426909-003</td>
<td>SPI EMS G06</td>
<td>June 2004</td>
</tr>
<tr>
<td>426909-004</td>
<td>SPI EMS H02</td>
<td>February 2012</td>
</tr>
<tr>
<td>426909-005</td>
<td>SPI EMS H02</td>
<td>October 2012</td>
</tr>
</tbody>
</table>

New and Changed Information

Changes to the 426909-005 Manual

This manual contains these changes:

- Updated the programs in Appendix B.
- Added Appendix D.
Changes to the 426909-004 Manual

This manual contains these changes:

- Added a Note in EMS Procedures.
About This Manual

This manual describes the Event Management Service (EMS)—a collection of processes, tools, and interfaces that provide event-message collection and distribution in the Distributed Systems Management (DSM) environment. This manual is intended for programmers, operators, and those who configure and manage systems and networks.

This manual describes information for:

<table>
<thead>
<tr>
<th>Server Type</th>
<th>Product Version</th>
<th>Supported RVUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP NonStop S-series</td>
<td>T9631F40</td>
<td>G00.00 and later G-series RVUs</td>
</tr>
<tr>
<td>HP NonStop K-series</td>
<td>T9631D31</td>
<td>D30.02 and later D-series RVUs</td>
</tr>
</tbody>
</table>

Manual Organization

This manual contains introductory, explanatory, and reference material for all aspects of EMS. Not all sections are of interest to every reader. Later in this section, you will find some guidelines to help you select which sections are likely to be of interest to you.

Part I, Introduction to EMS

Section 1, Introduction to EMS

Introduces the EMS product within the NonStop Kernel and DSM environments.

Section 2, EMS Components and Architecture

Introduces the components of EMS and describes their relationships.

Part II, Using EMS

Section 3, Retrieving Event Messages Interactively

Describes how event messages can be viewed.

Section 4, Retrieving Event Messages Programmatically

Describes the task of writing a program to retrieve event information from EMS.

Section 5, Compiled Filters

Describes compiled filters, the EMF filter language, and the EMF compiler.

Section 6, Filter Tables and Burst Filters

Describes filter tables and burst filters.

Section 7, Burst Detection and Suppression

Describes burst detection and suppression (BDS) functionality and configuration guidelines.

Part III, Designing and Implementing Event Reporting

Section 8, Reporting Events

Describes how to write a subsystem to create and send event messages.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Section 9, Standard Events</strong></td>
<td>Describes the EMS events that subsystems and applications generate to support operations management on NonStop Kernel operating systems.</td>
</tr>
<tr>
<td><strong>Section 10, Generating Standard Events</strong></td>
<td>Describes the steps involved in generating standard events and provides an example using a fictitious subsystem.</td>
</tr>
<tr>
<td><strong>Section 11, Procedure Calls for Standard Events</strong></td>
<td>Describes procedure calls that help subsystems and applications format standard events.</td>
</tr>
<tr>
<td><strong>Part IV, Configuring and Maintaining EMS</strong></td>
<td><strong>Section 12, Configuring EMS</strong> Describes configuration requirements, options, and strategies to help you make decisions about configuring EMS components, based on the tasks to be performed and the system resources available to perform them. <strong>Section 13, EMS Programs</strong> Describes the programs you can run to change the configuration of or display status information about various EMS components. <strong>Section 14, EMS Definitions</strong> Provides information that all programmers retrieving or producing event messages need, such as definitions of the standard EMS tokens. <strong>Section 15, EMS Procedures</strong> Describes the procedures used by management applications to retrieve information from event messages and the procedures used by subsystems to produce event messages. <strong>Section 16, Event Routing</strong> Describes distribution of events to selected destinations.</td>
</tr>
<tr>
<td><strong>Part V, Using EMS Distributors and Collectors</strong></td>
<td><strong>Section 17, Distributor Commands and Responses</strong> Describes the programmatic interfaces you use to perform functions such as retrieving event messages from, controlling, and inquiring about EMS distributors. <strong>Section 18, Distributor Event Messages</strong> Describes the event messages produced by the distributor. <strong>Section 19, Collector Commands and Responses</strong> Describes the programmatic interfaces you use for tasks such as controlling and inquiring about EMS collectors. <strong>Section 20, Collector Event Messages</strong> Describes the event messages produced by the primary and alternate collectors. <strong>Section 21, Distributor Errors</strong> Describes the error numbers and error lists returned as part of the programmatic interface to EMS distributors. <strong>Section 22, Collector Errors</strong> Describes the error numbers and error lists returned as part of the programmatic interface to EMS collectors.</td>
</tr>
<tr>
<td><strong>Part VI, EMS Example Files</strong></td>
<td><strong>Appendix A, Example of Retrieving Event Messages</strong> Provides a sample program that illustrates how to retrieve event messages from an EMS distributor.</td>
</tr>
</tbody>
</table>
Reading Guidelines

The sections of this manual of greatest use to you depend on what network or system management task you are undertaking.

All readers should be familiar with general aspects of the NonStop Kernel environment and should read:

- Section 1, Introduction to EMS
- Section 2, EMS Components and Architecture

D-series users should read the Introduction to Distributed Systems Management (DSM) as background to this manual.

Programmers

The five major programming tasks related to EMS involve writing:

- A management application to retrieve event messages reported by one or more subsystems (management-application programming)
- A management application to retrieve event messages reported by EMS and possibly to control EMS as a subsystem (EMS-management programming)
- A subsystem that reports event messages to EMS (subsystem programming)
- Filters for EMS event message distributors (filter programming)
- Filter tables for pre-log filtration or burst filters for burst detection and suppression

For these tasks, be familiar with the SPI Programming Manual.

Management-Application Programmers

You need the computer language manual for the language you are using, the Guardian Procedure Calls Reference Manual, the Guardian Procedure Errors and Messages Manual, and the DSM Template Services Manual. You also need the management programming manual for the subsystems you are interested in.

You should also read:

- Appendix B, Example of Reporting Events
- Appendix C, Standard Event Sample Files
- Appendix D, EMS Usage Best Practices and Recommendations

Provides a sample program that illustrates how to use the EMS procedures to build an event message and how to send the message to the EMS collector.

Provides sample files to help you create your external specification, DDL, and templates.

Describes best practices for creating event messages.
EMS Manual—426909-005

EMS-Management Programmers

You need the computer language manual for the language you are using, the Guardian Procedure Calls Reference Manual, the Guardian Procedure Errors and Messages Manual, and the DSM Template Services Manual. Because you are monitoring and controlling the EMS subsystem, you do not need a separate management programming manual. This manual describes the EMS programmatic interfaces.

Read the sections listed for management-application programmers and:

- Section 18, Distributor Event Messages
- Section 20, Collector Event Messages

Subsystem Programmers


Also read:

- Section 8, Reporting Events
- Section 9, Standard Events
- Section 10, Generating Standard Events
- Section 11, Procedure Calls for Standard Events
- Section 14, EMS Definitions
- Section 15, EMS Procedures
- Appendix A, Example of Retrieving Event Messages
Filter Programmers

Because the filter language exists in the context of TACL, you need the TACL Reference Manual.

Also read:
- Section 5, Compiled Filters
- Section 6, Filter Tables and Burst Filters
- Appendix A, Example of Retrieving Event Messages

System Managers and Operators

Both system managers and operators need access to these related manuals:
- System Generation Manual
- Operator Messages Manual
- Guardian User’s Guide
- ViewPoint Manual (if you use the ViewPoint application)
- SNMP Configuration and Management Manual

Operators should read:
- Section 3, Retrieving Event Messages Interactively
- Section 13, EMS Programs

System managers should read the sections listed for operators and:
- Section 12, Configuring EMS

If you plan to write filters, refer to the recommendations for filter programmers.

Additional Information

EMS event information tends to emphasize the software environment—hardware conditions are usually reported from the perspective of their impact on the software and their effects on the operating environment of the system or network.

In contrast, related products, the TMDS on systems running the D-series, and Compaq TSM on systems running the G-series, are primarily oriented toward maintaining and diagnosing hardware. This manual does not describe TMDS or TSM in any detail. For information on TMDS or TSM, see their respective manuals in the NonStop Technical Library.

This manual describes a combination of standard and optional products. If you have any questions about which products are included in the product packages you have purchased, contact your HP representative.
Notation Conventions

General Syntax Notation

This list summarizes the notation conventions for syntax presentation in this manual:

**UPPERCASE LETTERS.** Uppercase letters indicate keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

`MAXATTACH`

**lowercase italic letters.** Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

`file-name`

**computer type.** Computer type letters within text indicate C and Open System Services (OSS) keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

`myfile.c`

**italic computer type.** Italic computer type letters within text indicate C and Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

`pathname`

**[ ] Brackets.** Brackets enclose optional syntax items. For example:

`TERM [\system-name.$terminal-name`

`INT[ERRUPTS]`

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list may be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

`LIGHTS [ ON ]
[ OFF ]
[ SMOOTH [ num ] ]`

`K [ X | D ] address-1`

**{ } Braces.** A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list may be arranged either vertically, with aligned
braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }  
{ $process-name }
```

```
ALLOWSU { ON | OFF }
```

| **Vertical Line.** A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

| **Ellipsis.** An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address-1 [ , new-value ]...
[ - ] {0|1|2|3|4|5|6|7|8|9}...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char..."
```

| **Punctuation.** Parentheses, commas, semicolons, and other symbols not previously described must be entered as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;
LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must enter as shown. For example:

```
"[" repetition-constant-list "]"
```

| **Item Spacing.** Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In this example, there are no spaces permitted between the period and any other items:

```
$process-name.#su-name
```

| **Line Spacing.** If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] CONTROLLER

[ , attribute-spec ]...
```
**Notation for Messages**

This list summarizes the notation conventions for the presentation of displayed messages in this manual:

**Nonitalic text.** Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

**lowercase italic letters.** Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

```
p-register
process-name
```

**[ ] Brackets.** Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value ]
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list might be arranged either vertically, with aligned brackets on each side of the list, or

---

**!i and !o.** In procedure calls, the !i notation follows an input parameter (one that passes data to the called procedure); the !o notation follows an output parameter (one that returns data to the calling program). For example:

```
CALL CHECKRESIZESEGMENT ( segment-id !i, error !o );
```

**!i,o.** In procedure calls, the !i,o notation follows an input/output parameter (one that both passes data to the called procedure and returns data to the calling program). For example:

```
error := COMPRESSEDIT ( filenum ) ; !i,o
```

**!i:i.** In procedure calls, the !i:i notation follows an input string parameter that has a corresponding parameter specifying the length of the string in bytes. For example:

```
error := FILENAME_COMPARE_ ( filename1:length !i:i, filename2:length ) ; !i:i
```

**!o:i.** In procedure calls, the !o:i notation follows an output buffer parameter that has a corresponding input parameter specifying the maximum length of the output buffer in bytes. For example:

```
error := FILE_GETINFO_ ( filenum !i, [ filename:maxlen ] ) ; !o:i
```
Notation for Management Programming Interfaces

**UPPERCASE LETTERS.** The ![r] notation following a token or field name indicates that the token or field is required. For example:

```
ZCOM-TKN-OBJNAME token-type ZSPI-TYP-STRING. ![r]
```

**!o.** The ![o] notation following a token or field name indicates that the token or field is optional. For example:

```
ZSPI-TKN-MANAGER token-type ZSPI-TYP-FNAME32. ![o]
```

**Change Bar Notation**

Change bars are used to indicate substantive differences between this revision of the manual and the preceding revision. Change bars are vertical rules placed in the right margin of changed portions of text, figures, tables, examples, and so on. Change bars highlight new or revised information. For example:
The message types specified in the REPORT clause are different in the COBOL85 environment and the Common Run-Time Environment (CRE).

The CRE has many new message types and some new message type codes for old message types. In the CRE, the message type SYSTEM includes all messages except LOGICAL-CLOSE and LOGICAL-OPEN.

HP Encourages Your Comments

HP encourages your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to docsfeedback@hp.com.

Include the document title, part number, and any comment, error found, or suggestion for improvement you have concerning this document.
This part of the manual introduces the Event Management Service (EMS) and provides an overview of its components and architecture:

- **Section 1, Introduction to EMS**
- **Section 2, EMS Components and Architecture**
Part I: Introduction to EMS
1 Introduction to EMS

The Event Management Service (EMS) is a collection of processes, tools, and interfaces that support the reporting and retrieval of event information. Information retrieved from EMS helps you:

- Monitor your system or network environment
- Analyze circumstances that led to a problem
- Detect failure patterns
- Adjust for changes in the run-time environment
- Recognize and handle critical problems
- Perform many other tasks required to maintain a productive computing operation

EMS is a major component of NonStop Kernel System Management and was originally released on the D-series as part of the HP Distributed Systems Management (DSM) product group, which is described in the *Introduction to Distributed Systems Management (DSM)*.

EMS is used on both D-series and G-series RVUs of the NonStop Kernel.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMS in the System Environment</td>
<td>1-2</td>
</tr>
<tr>
<td>EMS Communications in the DSM Environment</td>
<td>1-3</td>
</tr>
<tr>
<td>EMS Capabilities and Features</td>
<td>1-5</td>
</tr>
<tr>
<td>Applications of EMS</td>
<td>1-8</td>
</tr>
</tbody>
</table>
EMS in the System Environment

EMS runs as a layer between the subsystem environment and the operations environment:

- In the subsystem environment, subsystem software directly controls resources (such as communication lines, files, and processes). Subsystems can be HP products such as Pathway, components of DSM such as ViewPoint, or user-written programs.

- In the operations environment, operations staff and specialized application programs (known as management applications) obtain information needed to manage the system or network.

In the requester-server model, EMS acts as a server to both layers. It responds to requests from the subsystem layer to accept and log event information, and responds to requests from the operations environment to retrieve certain event information as needed.

Because events can occur at any time, a process that logs (or collects) events must run continuously. This process, the primary collector ($0), is installed when the system is generated. EMS allows only one primary collector on each node of a network.

In addition to the primary collector, you can define alternate collectors to provide separate logging facilities for sorting selected events in separate log files. You can define as many alternate collectors on a system as you want, and you can start or stop them whenever you want.

You retrieve events from log files by running EMS processes called distributors. You can initiate and terminate distributors as dictated by your operations environment needs.

*Figure 1-1* is a simplified diagram of the EMS layer, its components, and their relationships to the subsystem and operations environments.
EMS Communications in the DSM Environment

EMS uses two types of interfaces to communicate with other components of DSM (for example, ViewPoint) and with processes that use DSM to manage a system or network:

- **EMS Interface.** This interface manages event reporting in a system or network. It is the mechanism used to send event messages to EMS. Subsystems, HP or user-written applications, and DSM processes (such as ViewPoint and DNS) use it to send event information to the EMS primary collector by subsystems. EMS processes themselves use this interface to report event information to the primary collector.

- **Command-Response Interface.** This interface is used to issue commands and receive responses about resources in a system or network. HP or user-written management applications can use the command-response interface. For example, a management application can issue a command to a subsystem to open a communication line.

*Figure 1-2* is a simplified diagram of the lines of communication between the components of a DSM system.
The major EMS processes (such as collectors and distributors) are controlled through the command-response interface.

Both the EMS and command-response interfaces are based on the Subsystem Programmatic Interface (SPI) message format. (See the SPI Programming Manual.) The common message format for event messages and commands supported by SPI simplifies the learning and programming tasks for programmers who deal with both interfaces.

Because the interfaces differ in purpose and use, they also differ in certain key characteristics:

- **Initiator**
  
  In the command-response interface, the management application initiates communication with the subsystem. The operator or management application also controls the flow of information, issuing the commands necessary to accomplish the desired task.
  
  In the EMS environment, the subsystem initiates communication, at least for reporting an event to EMS. Although operators and management applications can control how much information they request from EMS, they do not control the flow of information into EMS itself.

- **Communication pattern (synchronous or asynchronous)**
  
  The command-response interface supports a dialog between a management application and a subsystem. The management application starts a synchronous session with a subsystem by opening the appropriate subsystem process and
sending command messages. For each command, the subsystem returns a response message. The management application and the subsystem communicate synchronously, in a one-command, one-response dialog.

The EMS interface is unidirectional and asynchronous. Operators and management applications cannot use EMS to send information to subsystems, and subsystems report event information regardless of whether or not an operator or management application is interested in that information at the moment.

- Variable urgency

In general, commands are treated as equally urgent, in that each must be completed before the next one is issued. The operator or management application determines what command to issue next. To the subsystem, a request for status information is no less important or urgent than a request to stop all objects immediately.

In EMS, although operators and management applications control the retrieval and use of EMS information, subsystems can use certain EMS-defined conventions to convey significance or urgency. These conventions are needed to emphasize certain conditions that the operator or management application might otherwise not recognize as significant in the continuous flow of event information.

Because of its unique characteristics, the EMS interface is a valuable complement to the command-response interface. The two interfaces together can provide a wide range of management services. For example, you can use EMS to monitor a large array of subsystems. Then, when a significant situation arises, you can use the command-response interface to initiate a dialog with the appropriate subsystem.

**EMS Capabilities and Features**

The primary function of EMS is to collect and distribute event messages. Event messages are a special category of SPI messages used to convey information about events, which are significant occurrences in the subsystem environment.

Event messages report many occurrences and conditions, such as:

- Changes in the subsystem environment
- Errors encountered during continuous operation (as opposed to errors encountered during an interaction with a user or application, which are usually reported directly to the user or application)
- Conditions that might lead to a problem if not corrected
- Situations requiring operator intervention
- Significant losses of function or resources
- Indications of why a process terminated
Event messages can be reported by subsystems for the NonStop server or subsystems you write. They are subsystem specific, and hundreds of different event messages are produced by subsystems for the NonStop server alone. Although this wide range of messages provides the diversity and depth of information required to manage systems and networks, it also creates the need for tools to manage the event messages themselves.

**Basic Capabilities**

EMS supports many aspects of managing event messages, from generation of a message in the subsystem environment to generation of text for display in the operations environment. This list describes the basic EMS capabilities, in the approximate order in which they would apply to a single event message:

- **Message building.** EMS includes several library procedures that subsystems for the NonStop server, and subsystems you write use to build event messages. One procedure initializes a buffer, ensuring that a message has the correct format and providing some of the standard information found in all event messages. The subsystem then calls other procedures to add different kinds of information to the event message.

- **Message collection, filtering, and logging.** The EMS primary and alternate collectors accept event messages from subsystems, use pre-log filtration (PLF) for specific event messages, use burst detection and suppression (BDS) to suppress specified types of event bursts, and write unfiltered events to log files. Pre-log filtration makes use of compiled filters or filter tables to totally suppress the logging of certain event types.

- **Message filtering and distribution.** The EMS distributor processes provide:
  - Selection (or filtering) of event messages
  - Use of burst filters to implement BDS on specified types of event bursts
  - Distribution of event messages to their destination devices

  As with collector filters, filtering in distributors is the process of evaluating event messages and selecting only those in which a particular requester has expressed interest. Distribution is the returning of a selected event message to a requester through the appropriate interface. EMS can distribute event messages to processes, files, collectors on other nodes, and display devices such as terminals and printers.

- **Text formatting.** EMS includes a procedure that returns text suitable for display to an operator.

**Key Features**

EMS has many significant characteristics that help you manage your system or network:
灯具的一般性。每个EMS消息包含多个名为元体的元素。每个元体由一个数据值及其关联的标签组成，该标签标识数据值。所有子系统使用元体来构建事件消息，从而确保消息具有相同的通用格式并以类似的方式呈现相似信息。例如，EMS要求所有子系统指定事件的主体（事件的主要参与组件）。因此，管理应用可以以标准方式请求主体元体。此外，EMS定义了报告关键事件和操作事件（需要操作员采取行动的状况）的规范，以确保这些类型的事件在操作环境中得到识别。

- 简易性。接口到EMS使用标准文件系统过程调用，且在适用情况下使用标准SPI消息格式和规范。为帮助您编写报告事件的子系统，EMS包括初始化和添加元体到事件消息的程序。为帮助您获取用于显示的文本，EMS包括一个程序，该程序返回定义为事件消息的文本。

- 多样性。您可以使用EMS为多种应用程序，原因在于它提供了多种选项：
  - EMS支持监控事件的应用，以及分析报告事件记录的过去消息的应用。
  - 分发过程用于通信事件消息到管理应用程序，将事件消息从一个节点转发到另一个节点，并在打印机和其他显示设备上显示事件消息的文本。
  - 过滤器让您根据您选择的任何事件消息元体和值来选择感兴趣的事件消息。
  - EMS以令牌形式提供事件信息，用于程序化使用或者在文本形式用于操作员显示。以令牌形式的事件信息也称为令牌化事件信息。未令牌化消息用于与预-EMS系统兼容。
  - EMS支持HP和用户编写的子系统。如果用户编写的子系统在您的系统或网络上遵循与NonStop服务器子系统的相同规范，EMS可以为您提供一个子系统环境的综合视图。

- 配置灵活性。EMS支持各种配置，您可以根据数据处理环境的需求选择配置：
  - EMS允许在网路节点和同一点内处理器之间分布事件消息处理。事件消息可以分布到其他节点。SPI接口，如用于控制EMS过程的接口和用于检索事件消息的接口，可以跨越节点边界。
Applications of EMS

EMS is a set of tools that you can apply to any of several system or network management tasks, including these application areas:

- Monitoring a running network or system. You can use the ViewPoint application or your own management application to recognize situations needing attention as they arise. Depending on the problem and the sophistication of the application, the operator or application can resolve the problem through the appropriate command-response interface.

- Managing operator tasks. You can use a management application, or a distributor that routes messages to a display device, to select and display action event messages—those requesting operator attention. If operators see an integrated picture of the required tasks, they can focus on accomplishing the tasks, not on finding out what they are.

- Analyzing problems. To determine what went wrong and why, you must often retrace a series of events leading up to the problem condition. Log files, filters, and other EMS components can help you or your service provider by providing the historical record and the tools needed to sift through it.

- Detecting potential problems in advance. You can write a management application to analyze collector log files. Have it look for repeated warnings or persistent minor problems that singly would not require a recovery action but might disclose a correctable condition when taken together.
EMS Components and Architecture

This section describes the processes, files, and interfaces that together provide the event management capabilities described in Section 1, Introduction to EMS.

Later sections describe how to perform various tasks using these components, including standard events (those that subsystems and applications must generate to support operations management). Other sections include detailed specifications for the interfaces mentioned here.

Before working with the process architecture, you must understand more about event messages themselves because they are what EMS exists to manage.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Messages</td>
<td>2-2</td>
</tr>
<tr>
<td>Information Contained in Event Messages</td>
<td>2-2</td>
</tr>
<tr>
<td>Event Message Collectors</td>
<td>2-7</td>
</tr>
<tr>
<td>Event Message Collector Functions</td>
<td>2-7</td>
</tr>
<tr>
<td>Event Message Distributors</td>
<td>2-9</td>
</tr>
<tr>
<td>EMS Filters</td>
<td>2-11</td>
</tr>
<tr>
<td>Text Formatting Tools</td>
<td>2-12</td>
</tr>
<tr>
<td>Definition Files</td>
<td>2-14</td>
</tr>
<tr>
<td>EMS Object Programs</td>
<td>2-14</td>
</tr>
<tr>
<td>EMS Utility Programs</td>
<td>2-15</td>
</tr>
<tr>
<td>EMS Procedures</td>
<td>2-15</td>
</tr>
<tr>
<td>Standard Events</td>
<td>2-16</td>
</tr>
</tbody>
</table>
Event Messages

Event messages are based on tokens that convey information about events, which are significant occurrences in the subsystem environment.

Because of their token structure, event messages are much more convenient and efficient for applications to process than text messages. Information is easier to find and in a more convenient form. Character-by-character scans or conversions to internal form—typically required by text messages—are not required by event messages. In addition, EMS provides facilities that derive display text from message tokens. Therefore, with EMS, event information can be presented both to operators and to applications in the format most appropriate to each.

Another advantage of event messages is that they are subsystem specific, so the information they contain can be tailored to provide an exact and complete specification of the event circumstances.

Before EMS, the NonStop Kernel supported only text messages (numbered and unnumbered console messages). These messages are still supported for the benefit of subsystems and applications that have not yet migrated to EMS. These messages have the subsystem ID of EMS and have event numbers in the range 1 through 512.

Information Contained in Event Messages

A subsystem generates an event message whenever it detects a significant occurrence. The subsystem sends the event message to EMS, which stores the event message indefinitely (depending on how long the log files are kept).

An event message includes many kinds of information besides what happened, such as:

- Information about the message itself
  
  For example, the message contains a value identifying it as an event message (as opposed to a command or response message), a count of how long the message is, and a time stamp indicating when the message was generated.

- Information about the subsystem reporting the message
  
  All event messages include a subsystem identifier, process identifiers, and other relevant information about the creator of the message.

- Information specifying how the message should be handled
  
  EMS includes some conventions that determine how each message is interpreted after it is reported. These conventions, triggered by certain token values, include whether the message is critical and whether operator action is required. (For details, see Special Kinds of Event Messages on page 2-3.)

- Information about the actual event (occurrence)
Every event message includes an event number that uniquely identifies that type of event message among all the types that the subsystem is capable of generating. The event number is the primary indication of what happened (printer out of paper, line down, takeover by backup process, and so on).

Also, every event message includes a subject: the name or number of the hardware or software component most directly involved in the event. Beyond this information, the subsystem usually includes relevant details, such as error codes, disk addresses, pool sizes, or whatever information might serve to explain the error further.

- Additional context

Because an event message might be examined long after the event it is reporting occurred, the subsystem sometimes includes details about its environment, such as the active path to the device or the name of a file or process it has open, even if the path, file, or process is not directly involved in the event. This contextual information can help you when analyzing a problem to link seemingly unrelated event messages into a meaningful pattern.

The exact contents of all messages generated by a subsystem are described in the management programming manual for that subsystem (which is often a separate book with that title). For example, the event messages for Pathway are described in the \textit{TS/MP Management Programming manual} and \textit{Pathway Management Programming manual}. This manual describes event messages for EMS itself.

### Special Kinds of Event Messages

There are two special categories of events:

- Critical events are indicated by assigning the value TRUE to the emphasis token.
- Action events are indicated by including an action-needed token.

Messages with neither of these indicators are reporting events that are neither critical events nor action events. Subsystems rarely report an event that is both a critical event and an action event.

### Critical Events

A subsystem designates an event as critical when the consequences of the event might be severe. The subsystem itself cannot accurately determine what conditions you would consider severe for your particular system or network.

For example, you might not consider loss of a rarely used data communication line between two test systems a critical situation, but you would consider loss of a heavily used line between two production systems critical. These situations look identical to the subsystem. The best the subsystem can do is identify potentially critical situations and let you (with the help of programmatic tools you select) make the final determination.
Subsystems typically identify these events as critical:

- Potential or actual loss of data
- Loss of a major subsystem function
- Loss of a fault-tolerance capability, such as a redundant resource or failure-recovery function
- Loss of subsystem integrity (for example, an unrecoverable internal error in a subsystem)

To indicate a critical event, the subsystem assigns the value TRUE to the ZEMS-TKN-EMPHASIS token. For a description of EMS tokens and conventions, see Section 14, EMS Definitions.

The ViewPoint application highlights critical events in its event display. (For details, see the ViewPoint Manual.)

**Action Events**

A subsystem reports an action event when a situation arises that the subsystem cannot resolve without operator intervention. For example, a subsystem might report an action event when it cannot proceed until a tape is mounted or a printer ribbon is replaced.

Action events are reported in pairs of event messages. The first message, called an action-attention message, reports the problem. The second message, called an action-completion message, reports that the appropriate action has been taken. (In some cases, the subsystem can directly detect that the situation has been corrected. In other cases, the operator has to inform the subsystem of the action taken.)

Action events are designated by the presence of the ZEMS-TKN-ACTION-NEEDED token:

- The value TRUE designates an action-attention message.
- The value FALSE designates an action-completion message.

Another token, ZEMS-TKN-ACTION-ID, should be included in each pair of action-attention and action-completion messages with the same value in each message. (If the subsystem generates the completion message, it includes the correct action ID automatically.)

The action ID lets the operator or management application retrieving an action-completion message identify the action-attention message to which it corresponds. For a detailed description of EMS tokens and conventions, see Section 14, EMS Definitions.

The ViewPoint application counts and highlights action-attention messages in its event display, decrementing the count and dimming the message when the corresponding action-completion message is retrieved. (For details, see the ViewPoint Manual.)
Flow of Event Messages

Two types of EMS processes manage the flow of event messages from the subsystem environment to the operations environment:

- Event Message Collectors accept event messages from subsystems and log them. Pre-log filtration (PLF) lets a primary or alternate collector use one or more filters to detect and discard all event bursts with specified attributes or specific unwanted events that would otherwise be sent to the collector’s log file.

- Event Message Distributors filter event messages and return selected messages to the operations environment.

Figure 2-1 shows the flow of event messages in a system:

1. Messages originate in the subsystems.
2. The subsystems send the messages to the primary and alternate collector processes.
3. Collectors write passed event messages to the log files.
4. Distributors retrieve selected messages from the log files and send them to processes, printers, terminals, and other destinations in the operations environment.

The four kinds of distributors are each suited to a particular set of destination requirements:

- Forwarding
- Consumer
- Printing
- Compatibility
Figure 2-1. Flow of Event Messages

Legend

- Flow of event messages
- Original event-message stream
- Filtered event-message stream

Compaq Subsystems
User-written Subsystems
Alternate Collectors Filters
Primary Collector ($0)
Filters
Log Files
Log Files
Compatibility Distributor ($Z0)
Console
Forwarding Distributor Filters
Consumer Distributor Filters
Printing Distributor Filters
To Remote Collector
Management Application
Log Files
Event Message Collectors

EMS supports two types of event message collectors.

Primary Collector ($0)

Each system (or node) has only one primary event message collector, named $0. The primary collector, initiated at system generation, provides a central collection point for all events from all subsystems in a system. $0 uses the $ZOPR process to perform waited operations. $ZOPR is a continuous process that, like $0, exists on every node and is initiated at system generation. $ZOPR has no user interface and is used only as described here.

Alternate Collectors

Alternate collectors offer an alternative to the central collection point provided by the primary collector, $0. Alternate collectors provide functions similar to those supplied by $0, but each alternate collector maintains its own log files. The separation of events into several log files speeds up event processing because a network application program does not have to read a single large file containing many events unrelated to that application.

Some features of alternate collector usage are:

- One or more alternate collectors can be started on a system.
- An alternate collector is typically started by entering a TACL RUN command.
- An alternate collector is a named process (or process pair) with a name you define when you initiate it.
- An alternate collector accepts EMS events sent to it rather than to the primary collector.
- An alternate collector writes tokenized events to a log file. This file is identical in structure to the log files maintained by the primary collector.

Event Message Collector Functions

The primary and alternate collectors provide:

- Subsystem support
- Pre-log filtration (PLF)
- Log file management
- Distributor support
Subsystem Support

Subsystems and user-written applications send event information to the primary and alternate collectors by means of the WRITEREAD procedure call. Event messages are based on tokens rather than text.

Pre-Log Filtration (PLF)

Pre-log filtration lets primary and alternate collectors use EMS filters to determine whether specific events should be logged to the EMS log files. Judicious use of PLF conserves disk space and system resources that might otherwise be required to write unnecessary or redundant events to the EMS logs.

PLF is implemented by adding one or more filters to a collector:

- Compiled filters and filter tables can fail one or more particular events, thereby preventing the failed events from being forwarded to the collector’s log files.

- Burst filters can suppress event bursts based on burst size, duration, and other criteria (rather than the event type). Burst filters use the EMS burst detection and suppression (BDS) feature. For details, see Section 7, Burst Detection and Suppression.

Log-File Management

The collector writes messages to log files. The collector either writes messages one at a time or writes a number of messages to a buffer then writes that entire message group to the log file. The process of writing a group of messages to a buffer is known as blocking. Blocking significantly increases the speed at which messages are written to the log file.

Tasks associated with log switches (switching to using a new log file when the current file becomes full) are performed when the collector is idle; that is, when there are no events to log. The tasks include purging and deallocating (disk space) the old log file and preallocating (disk space for) the next log file.

Each collector maintains its own log files according to parameters you specify. These parameters can vary for each collector, and they control these log file variables:

- Disk volume and subvolume in which the log files are to be created
- Number of log files that can exist concurrently in one subvolume
- Primary-extent and secondary-extent sizes for each file
- Whether or not event messages are blocked before they are written to the log file
- Whether or not the file label is updated on disk each time the end of file is advanced
- Security attributes of the log files the collector creates
Primary Collector Log-File Attributes

The primary collector assumes default values for log file parameters at system generation. You can change the values programmatically through the collector's SPI ALTER or CONTROL commands or interactively through the EMSCCTRL utility.

For a description of the collector's command-response interface, see Section 19, Collector Commands and Responses. For a description of EMSCCTRL, see EMSCCTRL—Control Collector Utility on page 13-9.

Alternate Collector Log-File Attributes

You can assign values to the parameters that control alternate collector log files when you issue the RUN command to start the collector. If you do not specify a value for a log-file parameter, EMS uses a default value. You can change the values of some attributes programmatically by using the SPI CONTROL command or interactively through the EMSCCTRL program.

Distributor Support

In addition to logging event messages for retrieval by distributors, both the primary and alternate collectors support the forwarding, consumer, and printing distributors by:

- Notifying distributors when new event messages are logged
- Accepting event messages forwarded from another node by a forwarding distributor

Event Message Distributors

Event-message distributors have three basic functions:

- Accessing one or more log files
- Filtering event messages
- Routing the selected event messages to the appropriate destination
Depending on the nature of the destination, you can use one of four types of distributors:

**Consumer** Sends selected event messages to a requesting management application.

**Forwarding** Sends selected event messages to an EMS collector on another node in the network or to an alternate collector on the same node.

**Printing** Obtains formatted text for selected event messages and writes it to a printer or other display device, or to a file. When a routing filter is used, unformatted events can be routed to selected distributor processes.

**Compatibility** Filters event messages according to a fixed (rather than a user-specified) criterion, obtains formatted text for the selected primary collector event messages, and writes it to a console device.

### Consumer, Forwarding, and Printing Distributors

In general, you can initiate and terminate consumer, forwarding, and printing distributors as needed. When you initiate a distributor, you can (in some cases, must) provide:

- The source of the event messages you want the distributor to process: the current primary or alternate collector log files (on a local or remote node) or a specific log file
- The destination of event messages selected by the distributor (a display device, an EMS collector on another node, or a management application)
- The generation time stamp of the earliest event message you want the distributor to process (or none, if you want only the most recent event messages)
- The filter or filters you want the distributor to use to select event messages to send to the destination
- The burst detection and suppression (BDS) criteria you want the distributor to use

Not all option combinations are allowed, and some option combinations are allowed but not commonly used. For more information about configuring distributors, see Section 12, Configuring EMS.

### Compatibility Distributor ($Z0)

The compatibility distributor provides continuity between EMS and the prior console-message mechanism and allows message viewing during system load. $Z0 is the only mechanism for viewing messages during system load.

Because its purpose is so specialized, the compatibility distributor is not initiated and controlled in the same way as other types of distributors. You can configure the compatibility distributor only during system generation, and only one compatibility distributor is allowed per system (node). The distributor can be stopped by the CDISTSTOP option of the EMSCCTRL utility.
The compatibility distributor has no interactive or programmatic command interface of its own. It is controlled indirectly through commands to the primary collector. The primary collector can be instructed, through its own commands, to change certain parameters that affect the compatibility distributor.

Compared to other types of distributors, the compatibility distributor has a few simple characteristics:

- Its source of event messages is always the current primary collector log (or logs, on a network)—not the saved log files.
- The console prints or displays each message in the DSM display format. The pre-EMS compatible format is not supported.
- It has a selection criterion, CRITICAL-ONLY, which you can set to ON or OFF:
  - When CRITICAL-ONLY is ON, the compatibility distributor suppresses the printing of certain operator messages on the operator console. This setting decreases the number of noncritical messages displayed on the console, particularly during a system load. The messages that are suppressed are numbered operator messages, subsystem EMS, event number 6 (LDEV UP), 141 (CLIP DOWNLOADED), and 150 (CSS ACTIVATE PATH).
  - When CRITICAL-ONLY is OFF, the compatibility distributor displays all messages.
- You can set the selection criterion during system generation. If you do not set the selection criterion at system generation, the value CRITICAL-ONLY ON is used as the default. You can use EMSCCTRL to change the selection criterion after the distributor is started.

EMS Filters

Every EMS collector and distributor (except the compatibility distributor) can use one or more filters to determine whether to forward specific event messages to their destinations. These filters can select messages in a way that is as detailed and complex as you want. For example, you can design an alternate collector that filters out all events of a certain type from its event log, a forwarding distributor that forwards only critical event messages, a printing distributor that prints only action event messages, and a consumer distributor that returns only Pathway event messages.

EMS collectors and distributors can use three types of filters.

Compiled Filters

You generate compiled filters from filter source statements contained in an EDIT file written in the Event Management Service filter (EMF) language. You then compile this filter specification to create the filter, which is returned as an object file. For details on writing compiled filters, see Writing and Compiling Filters on page 5-1. For full
Filter Tables

Filter tables are EDIT files you load into a collector or distributor, at which time the table is automatically converted to an object representation that is then evaluated for each event by a collector or distributor. For a detailed description of filter tables, see Filter Tables on page 6-1.

Burst Filters

Burst filters are filter tables that contain burst detection and suppression (BDS) configuration parameters. These parameter directives are saved in an EDIT file, which the collector or distributor converts to a filter object like they do for standard filter tables. When you add a burst filter to a collector or distributor, as a startup parameter or through the SPI ADD command, all event bursts that conform to the BDS configuration parameter values in the filter are suppressed. For details, see Burst Filters on page 6-7.

Compared to compiled filters, filter tables (including burst filters) simplify the process of filter installation and modification. You can use filter tables instead of or combined with compiled filters. Because a compile step is not required, they are easier to maintain. Filter tables are especially useful when you need to filter events by relatively simple criteria (for example, subsystem owner, subsystem ID, and event number) or when you need frequent and fast online updates of filter contents.

To make a filter for a collector or distributor, create an EDIT file containing the filter-language constructs that express your selection criteria. For a compiled filter, you then use the filter-language compiler (EMF) to generate an object file suitable for loading to the collector or distributor. After the filter specification is loaded, the collector or distributor uses it to decide whether to pass each event message to its destination. The EMF compiler does not compile filter tables and burst filters. The collector or distributor converts them when they are loaded.

Text Formatting Tools

When passed an event message, the EMSTEXT procedure returns text in DSM display format. For parameter definitions and syntax, see EMSTEXT Procedure on page 15-19.

EMSTEXT performs these steps when it generates text for an event message:

1. If the event message points to a format template or contains a TEXT token:
   a. EMSTEXT checks first for a template.

   For many HP and user-written subsystems, EMSTEXT generates display text from an event message by applying a format template found in a template file installed at system generation. The subsystem ID and event number of the specifications of the EMF language and the filter compiler, see The Filter Language on page 5-7 and The Filter Compiler on page 5-44.
message uniquely identify its template. For information on format templates and template files, see the DSM Template Services Manual.

b. If no template is invoked for this message, EMSTEXT checks for the TEXT token.

If EMSTEXT finds ZEMS-TKN-TEXT, EMSTEXT generates displayable message text from the token’s contents. This token appears in the tokenized messages that the collector generates from the pre-EMS-style messages issued by HP and user-written subsystems.

2. If there is no template for the message and it does not contain a TEXT token, but it was formerly a pre-EMS style (console) message, EMSTEXT assigns a subsystem ID of EMS and an event number between 1 and 512, and produces text based on what it was before EMS.

3. Finally, if it cannot produce any text, EMSTEXT returns one of four error messages:

   EMSTEXT—No template and no TEXT token for event. SSID = subsystem-ID Event number = event-msg-ID [Subject = first-subject-value]

   This message means no format template exists for the event message, it lacks the token ZEMS-TKN-TEXT, and it is not identifiable as one of the pre-EMS messages. first-subject-value is the formatted representation of the first subject token in the event message. This part of the error message does not appear if the data type of the first subject is one that EMSTEXT cannot format.

   EMSTEXT—Bad event buffer: buffer-contents

   This message means that the event message buffer contains invalid data. Some of it is displayed to help you diagnose the problem. buffer-contents is up to the first 60 words of the buffer in hexadecimal notation.


   This message means an error occurred in opening or reading the template file. extstat1 and extstat2 are the two halves of the EMSTEXT extended-status code that defines the error. They are followed by “File open error,” or “File error” if the problem was a READ error, and then the template file name and a brief description of the reason for the error.

   first-subject-value is the formatted representation of the first subject token in the event message. This part of the error message does not appear if the data
type of the first subject is one that EMSTEXT cannot format. (EMSTEXT can format most token types, such as file, device, string, and integer.)

**Note.** Applications should retry a call to open or read a template file if a timeout message is returned. A timeout is not a fatal error.

---

**EMSTEXT error—(extstat1,extstat2) prevented display of event.**

SSID = subsystem ID  Event number = event-msg ID  [Subject = first subject value]

This message covers any other case in which EMSTEXT cannot produce text from the event message. extstat1 and extstat2 are the two halves of the EMSTEXT extended-status code that defines the error. first subject value is the formatted representation of the first subject token in the event message. This part of the error message does not appear if the data type of the first subject is one that EMSTEXT cannot format.

A printing distributor or compatibility distributor calls EMSTEXT to create text for an output device and the ViewPoint application to create the text it displays. Any user application can call EMSTEXT. Applications that call EMSTEXT should have a minimum of five data pages (stack).

---

### Definition Files

Like all subsystems that support SPI interfaces, EMS has its own DDL (Data Definition Language) file that defines the tokens, values, and data types used in event messages and in command and response messages relating to EMS. The DDL file is called ZEMSDDL, and its language-specific derivative files are called ZEMSTAL, ZEMSTACL, ZEMSOCOB, and ZEMSC.

These files are of particular importance to EMS because they include definitions for some widely used tokens. For example, all subsystems that report event messages use the token ZEMS-TKN-EVENTNUMBER, and all programs that need to extract the event number from any event message also use the ZEMS-TKN-EVENTNUMBER token. Thus, the EMS definition files pertain to many subsystems besides EMS.

### EMS Object Programs

HP provides two object programs for starting EMS components:

- **EMSACOLL (EMS Alternate Collector)** starts an alternate collector by invoking the TACL RUN command.

  Several command parameters are available to customize an alternate collector, including:

  - SUPPRESS, which enables burst detection and suppression (BDS)
EMS Components and Architecture

EMS Utility Programs

HP provides three utility programs as part of EMS that provide an interactive means for controlling and inquiring about collectors and distributors:

- EMSCCTRL (EMS Collector Control) lets you change the values of options controlling the primary collector ($0), alternate collectors, and the compatibility distributor ($Z0). Among other things, these options control how a collector maintains its log files.
- EMSCINFO (EMS Collector Information) displays the current settings of the primary and alternate collector options, information about the status of the compatibility distributor, and statistics describing file operations and occurrences of event messages.
- EMSDINFO (EMS Distributor Information) displays the current settings of the distributor options, the status of event message sources and destinations, and statistics about filter operations and errors.

For more details on these utility programs, see Section 13, EMS Programs.

EMS Procedures

Several EMS-related procedures are available as part of the procedures library:

- Retrieve information from event messages
- Create event messages

The two procedures that relate to retrieving information are EMSTEXT (see Text Formatting Tools on page 2-12) and EMSGET. EMSGET retrieves tokens from event messages, as SSGET retrieves tokens from command messages.

Several procedures build event messages:

- EMSINIT initializes an event-message buffer, creating the correct header format for event messages and inserting standard information it derives from parameters and from other sources.
- EMSADDTOKENS and EMSADDSUBJECT put additional tokens and event-message subjects in the buffer. Subsystems that report events use these procedures to create their event messages. (For details, see Section 15, EMS Procedures.)

### Standard Events

**For a description of...**

- The small set of events that subsystems and applications must generate to support operations management
- Generating these events
- The procedure calls that help subsystems and applications format standard events
- Samples of a data definition language (DDL) file, an EMS template, and a subsystem event external specification

**See...**

- Section 9, Standard Events
- Section 10, Generating Standard Events
- Appendix C, Standard Event Sample Files
Part II: Using EMS

This part of the manual describes basic procedures for using EMS and provides supplemental reference information:

- Section 3, Retrieving Event Messages Interactively
- Section 4, Retrieving Event Messages Programmatically
- Section 5, Compiled Filters
- Section 6, Filter Tables and Burst Filters
- Section 7, Burst Detection and Suppression
Part II: Using EMS
Retrieving Event Messages Interactively

This section describes how to retrieve event messages with little, if any, configuration or programming. Displaying messages, either as they are generated or in a saved log file, is a good way to become familiar with the EMS environment before you undertake a larger project, such as writing a management application or configuring a large network of EMS components.

The three ways to see an event’s display text as it is reported:

- For D-series RVUs, use the ViewPoint application, which offers several event-display screens and many options for changing them. For a description of ViewPoint, see the ViewPoint Manual.
- Run a printing distributor, as described in this section.
- Read the event-message display produced on the operator console. If $Z0 is configured, it sends the event messages to the operator console. (For more information about the event-message display, see the Guardian User’s Guide.)

These three alternatives use the DSM display format.

This section describes techniques for running printing distributors to produce some simple event-message displays:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displaying Current Event Messages</td>
<td>3-2</td>
</tr>
<tr>
<td>Displaying Saved Event Messages</td>
<td>3-2</td>
</tr>
<tr>
<td>Creating Simple Compiled Filters</td>
<td>3-3</td>
</tr>
<tr>
<td>Creating Simple Filter Tables</td>
<td>3-4</td>
</tr>
</tbody>
</table>
Displaying Current Event Messages

Assuming the log files are secured so that you can access them, it is easy to run a printing distributor. The simplest case is to run a printing distributor to your terminal by entering this TACL command:

```
EMSDIST/NAME $dist/TYPE PRINTING, COLLECTOR $0, TEXTOUT $TERM1
```

This example runs a printing distributor and sends the output to a terminal called $TERM1. If you try this example, substitute the name of your terminal. In this example, the source of event messages is the collector $0. (The primary collector on your local system.) The filter is the default filter, because none is specified. The default filter (for the printing distributor) passes all event messages. The distributor displays messages for all events reported from when you start the distributor until you stop it.

You can use similar commands to run printing distributors to other terminals or printers by substituting other names for $TERM1.

To stop the distributor, press the BREAK key on your terminal. Then, assuming the distributor is the process you most recently started, use the TACL STOP command to terminate it. The EMSDIST command examples in this section use the NAME RUN parameter (/NAME $dist/), which assigns a name to the distributor process and lets you subsequently STOP the process by name. You must provide a distributor process name in the EMDIST command.

Displaying Saved Event Messages

You can run a printing distributor to display event messages saved in a previously created log file. The TACL command is much the same as the previous example, except the LOGFILE parameter replaces the COLLECTOR parameter, and a TIME parameter is added to specify the generation time of the earliest message you want to display:

```
EMSDIST/NAME $dist/TYPE PRINTING, LOGFILE log1, TEXTOUT $TERM1, TIME 15 jul 1999 7:30
```

This example runs a printing distributor to $TERM1 to process event messages stored in the previously generated log file (named log1) and generated since 7:30 a.m. on July 15, 1999. Because this example does not specify a filter, the distributor uses the default filter, which passes all events.

You can stop the distributor at any time by pressing the BREAK key and using the TACL STOP command. Or you can let the distributor run until it reaches the end of the log file.
Creating Simple Compiled Filters

A printing distributor needs a filter to specify which event messages to print and which to ignore. If you do not specify a filter, it uses its default filter, which passes all event messages.

To try running a printing distributor with another filter if you do not have access to a filter object file (a compiled filter specification), either write your own or try one of these simple compiled filters:

- Pass only critical event messages:

  ```tcl
  FILTER critical;
  BEGIN
  IF ZEMS^TKN^EMPHASIS = [ZSPI^VAL^TRUE] THEN PASS;
  END;  --ELSE FAIL is implied
  ```

- Pass all event messages from a specific subsystem (Pathway, in this example):

  ```tcl
  FILTER passpwy;
  BEGIN
  --initialize SSD structure
  --(Note: Pathway must be installed for this to work
  [ #set ZPWY^VAL^SSID  TANDEM.PATHWAY.C00 ]
  --pass if SSID = PATHWAY
  IF ZSPI^TKN^SSID = SSID(ZPWY^VAL^SSID) THEN PASS;
  END;  --ELSE FAIL is implied
  ```

To create these compiled filters, copy the previous filter-language statements to two EDIT files, one for each filter. For example, you could name the files SCRIT and SPWY. (S stands for source for filter.) Then pass the EDIT files to the EMF (filter language) compiler. These TACL statements load the necessary SPI definitions and compile the two filters into object files called FCRIT and FPWY:

```tcl
PUSH X
#LOAD / KEEP 1, LOADED X / $SYSTEM. ZSPIDEF.ZSPITACL
#LOAD / KEEP 1, LOADED X / $SYSTEM. ZSPIDEF.ZEMSTACL
#LOAD / KEEP 1, LOADED X / $SYSTEM. ZSPIDEF.ZPWYTACL
POP X
EMF /IN SCRIT/ FCRIT
EMF /IN SPWY/ FPWY
```

You could also use this ATTACHSEG TACL command:

```tcl
ATTACHSEG SHARED $SYSTEM.ZSPISEGF.ZxxxTACL :Zxxx
#SET #USELIST [#USELIST] :Zxxx
```

Zxxx could be any of ZSPI, ZEMS, or ZPWY.

After you have a compiled filter specification, direct the distributor to load that filter by including the FILTER parameter and specifying the file name of the compiled filter:

```tcl
EMSDIST TYPE PRINTING, COLLECTOR $0, FILTER FCRIT,
TEXTOUT [ #MYTERM ]
```
For information about compiled filters, see Section 5, Compiled Filters. For information about creating filter tables and burst filters, see Section 6, Filter Tables and Burst Filters.

Creating Simple Filter Tables

To try running a printing distributor with a filter table, you can write one of your own. A filter table is represented as an EDIT file and does not need to be compiled. The filter table can be loaded directly into the distributor.

These two examples show filter tables that perform the same functions as the two simple compiled filters in the previous example:

- To pass only critical event messages:

  ```
  ?COMMENT
  EMS FILTER TABLE
  ?PASS
  ?EMPHASIS 1
  *
  ```

- To pass all event messages from any specified subsystem (Pathway, in this case):

  ```
  ?COMMENT
  EMS FILTER TABLE
  ?PASS
  TANDEM PWY  *
  ```

In both examples, the asterisk (*) causes all specified events to be passed. For a detailed description of formatting guidelines for filter tables, see Section 6, Filter Tables and Burst Filters.
Retrieving Event Messages Programmatically

This section describes the interface between a management application and a consumer distributor.

Because event messages follow the SPI format, and the interface to the distributor follows the command-message protocol, you should be familiar with the SPI Programming Manual.

Section 3, Retrieving Event Messages Interactively, also applies to this section. This section starts from the point where the system is correctly configured, EMS is running, and compiled filter specifications are available to you.

This section provides a perspective about the programmatic interface to the consumer distributor:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started</td>
<td>4-2</td>
</tr>
<tr>
<td>Changing the Environment</td>
<td>4-4</td>
</tr>
<tr>
<td>Retrieving Event Information</td>
<td>4-12</td>
</tr>
</tbody>
</table>

Detailed explanations are provided in later sections. You might want to read this section again after reading them.

For sample code (in several programming languages) that retrieves event messages, see Appendix A, Example of Retrieving Event Messages.

To retrieve event messages:

1. Start a distributor process interactively (using a TACL RUN command) or programmatically (using the NEWPROCESS procedure).
2. Open the distributor’s command-response interface.
3. Establish the desired environment: the filter and, optionally, its parameters; the event-message source (or sources); and the log-file position.
4. Request each event message from the distributor, and extract the desired information from each event message.
5. Close the distributor.
1. Start the consumer distributor interactively or programmatically:

- If you start the distributor interactively, you can provide all of its environment specifications when you start it, or you can leave some of the specifications for the management application to establish when it has opened the distributor.

Enter a TACL command to RUN EMSDIST (the distributor):

```tcl
RUN EMSDIST / NAME $DIST1, CPU 3 / TYPE CONSUMER, BACKUP 4, COLLECTOR $0, FILTER FCRIT
```

For a detailed description of the RUN EMSDIST command, see EMSDIST—Distributor Program on page 13-34. This sample command runs a consumer distributor as a process pair named $DIST1 in processors 3 and 4. The distributor accesses the current collector log using the filter contained in file FCRIT.

To let the management application specify some of the environment specifications, omit one of these parameters:

- FILTER parameter, letting the application specify a filter later or default to "pass all event messages"
- COLLECTOR parameter, in which case the application would have to specify either a collector or a saved log file before it could retrieve any event messages

Although this example does not show the TIME parameter, you can also specify it in the RUN command, allow it to default, or have the management application specify it later.

- To start the distributor programmatically, use the NEWPROCESS procedure or the PROCESS_CREATE_ procedure. You must use PROCESS_CREATE_ to create high-PIN processes. This TAL example initiates a consumer distributor called $MYDST as a process pair with the primary process running in processor 7:

```tcl
distprog ':=' "[$SYSTEM SYSTEM EMSDIST "];
distname ':=' "[$MYDST "];
CALL NEWPROCESS(distprog,
    !priority--default!
    ,
    !memory pages--default!
    ,
    !primary process in cpu 7!
    emsdist^pid, !process ID returned!
    error, !error returned!
    distname, !process name!
    , !home term defaults!
    , !inspect flag defaults!
);

! Check for NEWPROCESS error
IF error THEN ... !unsuccessful!
```
2. Use the OPEN procedure to open the process. You can also use the FILE_OPEN procedure, which is mandatory if the process is started at a high PIN.

3. Use the WRITE procedure to send the process a start-up message.

When you send a start-up message, the only parameter you must specify is the distributor type ("TYPE C" for consumer). The other parameters then use their default values, and later you can use a distributor CONTROL command-message to change any values you want to change. Or, as shown in this code, you can specify other parameters in the start-up message, using the same syntax as for the RUN EMSDIST command.

In this example, the parameter string is set to "TYPE C, BACKUP 8" to specify that this is a consumer distributor whose back-up process is to run in processor 8:

```
! Open distributor process to send start-up message
process^name ':= ["$MYDST ", 8*[
"
"];
CALL OPEN(process^name, fnum, , 1);
IF <> THEN ...

startup^message.id        := -1;
startup^message.infile   ':=' [12*[
"
"];
startup^message.outfile  ':=' [12*[
"
"];
startup^message.defaults ':=' my^file^defaults for 8 words;
startup^message.params   ':=' ["TYPE C, BACKUP 8",0,0];
CALL WRITE(fnum,
    startup^message,
    $offset(startup^message.params) + 18);
CALL FILEINFO(fnum, error);
IF NOT (error = 0 OR error = 70) THEN ... !error
CALL CLOSE(fnum);
```

```
! Open distributor process for SPI communication
process^name[4] ':= "#ZSPI ";
CALL OPEN(process^name, fnum, , 1);
IF <> THEN ...
```

**Note.** You can use FILE_GETINFO_ instead of FILEINFO.
Changing the Environment

Your management application can dynamically control its distributor environment—the source (or sources) of event messages, the log file position, and the filter and its parameters—by using the distributor’s subsystem programmatic interface (SPI). To control the distributor environment, use the ADD, ALTER, DELETE, and REPLACE commands described in Section 17, Distributor Commands and Responses.

Filters and Filter Parameters

This command message example uses the non-object-oriented interface, which uses a maximum of one filter per distributor. For distributors that require multiple filters, the alternative object-oriented ADD, DELETE, or REPLACE command messages let you add or delete filters or replace the current filter or return to the default filter (pass all event messages). These commands also let you pass parameters to a filter when you load it or change the parameter values the current filter is using.

To load a filter, pass the distributor the name of the file containing the compiled filter specification. Use the token ZEMS^TKN^FILTERFILE to specify the file name.

For example, to construct the appropriate command message to load a filter stored in a file called FCRIT and send it to the distributor using the WRITEREAD procedure (the distributor will contain only this filter):

```plaintext
! Initialize buffer for distributor CONTROL command message
IF (error := SSINIT(spibuffer,
  ZEMS^VAL^BUFLEN,
  ZEMS^VAL^SSID,
  ZSPI^VAL^CMDHDR,
  ZEMS^CMD^CONTROL)) THEN ...

! Get external file name to use as token value
sbuf ':=' ['FCRIT", 0];
CALL FNAMEEXPAND(sbuf, filter^file^name,
  my^file^defaults);

! Place token with token value in spibuffer
IF (error := SSPUTTKN(spibuffer,
  ZEMS^TKN^FILTERFILE,
  filter^file^name)) THEN ...

! Determine how many bytes of spibuffer have been used
spi^error := SSGETTKN(spibuffer,
  ZSPI^TKN^USEDLEN,
  buffer^length);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! And send the command message
CALL WRITEREAD(fnum,
  spibuffer,
  buffer^length,
  buffer^length);
```
Retrieving Event Messages Programmatically

Filters and Filter Parameters

```
ZEMS^VAL^BUFLEN,
count^read);
IF <> THEN ... ! Handle the error
! Reset buffer length to what you declared for spibuffer
spi^error := SSPUTTKN(spibuffer,
    ZSPI^TKN^RESET^BUFFER,
    ZEMS^VAL^BUFLEN);

! See if anything wrong with response buffer
IF spi^error <> ZSPI^ERR^OK THEN ... 

! Buffer ok, was distributor CONTROL command ok?
spi^error := SSGETTKN(spibuffer,
    ZSPI^TKN^RETCODE,
    error,
    1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...
IF error <> 0 THEN ... ! Handle error

! Filter correctly installed !

To reset the filter to the default (pass all event messages), send the same basic
message, omitting ZEMS^TKN^FILTERFILE and instead including
ZEMS^TKN^RESET^FILTER with value ZSPI^VAL^TRUE.

To pass parameters to a filter, you must have a DDL definition file that defines the
subsystem ID and the necessary tokens. The filter uses the subsystem ID and the
tokens to select the messages you want. If this definition file is not already available to
you, for a discussion of creating this DDL file, see the SPI Programming Manual. For a
complete description of the filter language aspects of parameter passing, see
Section 5, Compiled Filters, and Section 6, Filter Tables and Burst Filters.

After the definition file is available, use a CONTROL command message to pass the
parameters or to modify their values. To pass the parameters when you initially load
the filter, include the ZEMS^TKN^FILTERFILE parameter and the parameter tokens
and values you want to load with the filter. To change the values of the parameters
used by the current filter without reloading the filter, include the
ZEMS^TKN^REPLACE^PARAM token instead of the FILTERFILE token, and include
the parameter tokens and their new values. For information about more options and
restrictions, see Section 17, Distributor Commands and Responses.
```
Event-Message Sources

You must specify the source of the event messages the distributor is to access: either the current log files of one or more primary or alternate collectors, or a saved log file. Event message sources can be added or deleted. To change from one source to another, delete the current source and add the new source. You can access up to ten collectors (current log files), but only one saved log file.

To add a collector, issue a CONTROL command message that includes the token ZEMS^TKN^CONNECT^SRC^COLL. You can include more than one of these tokens, as long as the total number of collectors is always ten or fewer.

For example, to add two collectors, $0 (the local collector) and \CITY.$0 (the collector on the node \CITY):

! Initialize buffer for distributor CONTROL command message
IF (error := SSINIT(spibuffer,
   ZEMS^VAL^BUFLEN,
   ZEMS^VAL^SSID,
   ZSPI^VAL^CMDHDR,
   ZEMS^CMD^CONTROL)) THEN ...

! Get external name of collector for use as token value
sbuf ':=' ["$0", 0];
CALL FNAMEEXPAND(sbuf, coll^name1, my^file^defaults);

! Place token--with token value--in spibuffer
spi^error := SSPUTTKN(spibuffer,
   ZEMS^TKN^CONNECT^SRC^COLL,
   coll^name1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! Do the same thing for the second collector
sbuf ':=' ["\CITY.$0", 0];
CALL FNAMEEXPAND(sbuf,
   coll^name2,
   my^file^defaults);

spi^error := SSPUTTKN(spibuffer,
   ZEMS^TKN^CONNECT^SRC^COLL,
   coll^name2);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! Determine how many bytes of spibuffer have been used
spi^error := SSGETTKN(spibuffer,
   ZSPI^TKN^USEDLEN,
   buffer^length);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...
! And send the command message
CALL WRITEREAD(fnum,
  spibuffer,
  buffer^length,
  ZEMS^VAL^BUFLEN,
  count^read);

IF <> THEN ... ! Handle the error

! The response message is now in spibuffer.

! Reset buffer length to what you declared for spibuffer
spi^error := SSPUTTKN(spibuffer,
  ZSPI^TKN^RESET^BUFFER,
  ZEMS^VAL^BUFLEN);

! See if anything wrong with response buffer
IF spi^error <> ZSPI^ERR^OK THEN ...  

! Buffer ok, was distributor CONTROL command ok?
spi^error := SSGETTKN(spibuffer,
  ZSPI^TKN^RETCODE,
  error,
  1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ... 

IF error <> 0 THEN ...

For the other operations, send the same basic command message, but substitute a different token for ZEMS^TKN^CONNECT^SRC^COLL:

● To delete a collector, substitute the ZEMS^TKN^DISCONNECT^SRC^COLL token.
● To add a log file, substitute the ZEMS^TKN^CONNECT^LOGFILE token.
● To delete a log file, substitute the ZEMS^TKN^DISCONNECT^LOGFILE token.

These restrictions apply to CONTROL command messages that change the event-message source:

● Up to ten collectors can be connected simultaneously.
● Only one log file can be connected at a time.
● A collector cannot be added if a log file is connected, and a log file cannot be added if one or more collectors are connected.
This example illustrates some of these restrictions by showing how to start with the previous two-collector example and end up with the distributor connected to a saved log file called $OPS.LOG.SV930315:

! Build CONTROL command message to delete both collectors
IF (error := SSINIT(spibuffer,
   ZEMS^VAL^BUFLEN,
   ZEMS^VAL^SSID,
   ZSPI^VAL^CMDHDR,
   ZEMS^CMD^CONTROL)) THEN ... 

! Place token to delete first collector in spibuffer
spi^error := SSPUTTKN(spibuffer,
   ZEMS^TKN^DISCONNECT^SRC^COLL,
   coll^name1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ... 

! Place token to delete second collector in spibuffer
spi^error := SSPUTTKN(spibuffer,
   ZEMS^TKN^DISCONNECT^SRC^COLL,
   coll^name2);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ... 

! Determine how many bytes of spibuffer have been used
spi^error := SSGETTKN(spibuffer,
   ZSPI^TKN^USEDLEN,
   buffer^length);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ... 

! And send the command message
CALL WRITEREAD(fnum,
   spibuffer,
   buffer^length,
   ZEMS^VAL^BUFLEN,
   count^read);
IF <> THEN ... 

! The SPI response is now in spibuffer.

! Reset buffer length to what you declared for spibuffer
spi^error := SSPUTTKN(spibuffer,
   ZSPI^TKN^RESET^BUFFER,
   ZEMS^VAL^BUFLEN);

! Check if anything wrong with response buffer
IF spi^error <> ZSPI^ERR^OK THEN ... 

! Buffer ok, was distributor CONTROL command ok?
spi^error := SSGETTKN(spibuffer,
ZSPI^TKN^RETCODE, error, 1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

IF error <> 0 THEN ...

! Build and send CONTROL command message to add log file
! Initialize spibuffer for distributor CONTROL command
IF (error := SSINIT(spibuffer,  
ZEMS^VAL^BUFLEN,  
ZEMS^VAL^SSID,  
ZSPI^VAL^CMDHDR,  
ZEMS^CMD^CONTROL)) THEN ...

! Get external name of log file for use as token value
sbuf ':=' ["$OPS.LOG.SV930315", 0];
CALL FNAMEEXPAND(sbuf,  
logname,  
my^file^defaults);

! Place token--with token value--in spibuffer
spi^error := SSPUTTKN(spibuffer, 
ZEMS^TKN^CONNECT^LOG, 
logname);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! Determine how many bytes of spibuffer have been used
spi^error := SSGETTKN(spibuffer, 
ZSPI^TKN^USEDLEN, 
buffer^length);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! And send the command message
CALL WRITEREAD(fnum,  
spibuffer,  
buffer^length,  
ZEMS^VAL^BUFLEN,  
count^read);

IF <> THEN ... ! Handle the error

! The SPI response is now in spibuffer.

! Reset buffer length to what you declared for spibuffer
spi^error := SSPUTTKN(spibuffer, 
ZSPI^TKN^RESET^BUFFER, 
ZEMS^VAL^BUFLEN);
! Check if anything wrong with response buffer
IF spi^error <> ZSPI^ERR^OK THEN ...

! Buffer ok. Was distributor CONTROL command ok?
spi^error := SSGETTKN(spibuffer,
   ZSPI^TKN^RETCODE,
   error,
   1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...
IF error <> 0 THEN ...  ! Handle error

! Distributor now uses log file source

Log File Position

You can use either of two time tokens in the CONTROL command message to specify
where in the collector log or the saved log file you want the distributor to start retrieving
event messages:

- To position the distributor by event-message generation time, use the token
  ZEMS^TKN^GMTTIME.

- To position the distributor by event-message log time, use the token
  ZEMS^TKN^LOGTIME.

Both parameters require that you specify the time as Greenwich mean time (GMT).

If you do not specify GMTTIME or LOGTIME before the distributor starts examining
messages, the distributor uses these positioning specifications:

- If the event-message source is the current log file of one or more collectors, the
distributor retrieves incoming event messages as they are logged by the collector
(or collectors), starting at the time the CONTROL command message is
processed.

- If the event-message source is a saved log file, the distributor retrieves event
messages starting at the beginning of the log file.

This example positions the distributor so that it returns all event messages generated
at or after 7:30 a.m. Local Civil Time:

! Compute time to use as token value
time^array[0]  := 1996;  !year
time^array[1]  := 3;     !month
time^array[2]  := 21;    !day of the month
time^array[3]  := 7;     !hour
time^array[4]  := 30;    !minute
time^array[5]  := 0;     !second
time^array[6]  := 0;     !millisecond
time^array[7]  := 0;     !microsecond
local^time     := COMPUTETIMESTAMP(time^array);
greenwich^time := CONVERTTIMESTAMP(local^time, 2);

! Initialize spibuffer for distributor CONTROL command
! message
IF (error := SSINIT(spibuffer,
             ZEMS^VAL^BUFLEN,
             ZEMS^VAL^SSID,
             ZSPI^VAL^CMDHDR,
             ZEMS^CMD^CONTROL)) THEN ...

! Place token--with time token-value--in spibuffer
spi^error := SSPUTTKN(spibuffer,
             ZEMS^TKN^GMTTIME,
             greenwich^time);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! Determine how many bytes of spibuffer have been used
spi^error := SSGETTKN(spibuffer,
             ZSPI^TKN^USEDLEN,
             buffer^length);

! And send the command message
CALL WRITEREAD(fnum,
               spibuffer,
               buffer^length,
               ZEMS^VAL^BUFLEN,
               count^read);

IF <> THEN ...

! The SPI response is now in spibuffer.
! Reset buffer length to what you declared for spibuffer
spi^error := SSPUTTKN(spibuffer,
             ZSPI^TKN^RESET^BUFFER,
             ZEMS^VAL^BUFLEN);

! Check if anything wrong with response buffer
IF spi^error <> ZSPI^ERR^OK THEN ...

! Buffer ok. Was distributor CONTROL command ok?
spi^error := SSGETTKN(spibuffer,
             ZSPI^TKN^RETCODE,
             error,
             1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

IF error <> 0 THEN ...

! Log file positioned!
Setting the value of GMTTIME or LOGTIME to -1 causes the distributor to use the default positioning specifications (for collectors, the messages as they are logged; for a saved log file, the beginning of the file).

**Note.** If a SETTIME command resets the system time to earlier than it was, one or more event messages will have time stamps out of sequence: earlier than that of the last message before the reset. This might mean that later the distributor either positions itself in the log file after the out-of-sequence messages or skips them. To alert the operator to this, the collector compares the time stamp of each event message to that of the previous message. If the time stamp of the later message is earlier than that of the earlier message, the collector issues an event message ZEMS-EVT-LOGTIME-DECREASE. The distributor, in turn, checks each LOGTIME-DECREASE message, regardless of its time stamp, and (if it is not filtered out) passes it to its destination. For details, see the description of event message 524: ZEMS-EVT-LOGTIME-DECREASE on page 20-30.

**Specifying Multiple Parameters in One Command**

If you stay within the allowed parameter combinations (for example, if you do not try to add a log file and a collector at the same time), you can include many environment parameters in one CONTROL command message. For example, you could load a filter, specify its parameters, delete three collectors, add a log file, and specify a log file position, all in one command. However consider this strategy carefully.

The advantage of including several operations in one CONTROL command is fewer messages. But it has one important disadvantage. If an error occurs—even if you note the order in which you enter the parameters—it might be hard to determine which operations completed before the error occurred and which were skipped because of it.

**Retrieving Event Information**

After you establish the distributor environment, you can begin to retrieve and process event messages.

**Obtaining an Event Message (GETEVENT)**

To get an event message from the distributor, issue a GETEVENT command message. For details on the GETEVENT command, see GETEVENT Command (ZEMS-CMD-GETEVENT) on page 17-25.

The GETEVENT command has only two parameters, and they have an effect only in special cases. All other information governing the selection of the event message to be returned, such as the filter, its parameters, and the log file to access, are already specified as part of the distributor environment. The two parameters you can specify with GETEVENT:

- ZSPI^TKN^CONTEXT. This token lets the distributor maintain its context in the sequence of event messages it is returning to your management application. The context token is returned in every GETEVENT response message, and you need
only carry it forward to the next GETEVENT command message. You do not need to examine the token.

You must not include the context token when its contents are not valid—that is, when the distributor has just been opened and no message has been returned yet or when a log-file positioning operation has just been performed. For all other GETEVENT command messages, you can include the context token you received from the previous GETEVENT response message. Use of the context token is optional. However, once a context token is submitted, subsequent messages must also contain it.

- ZEMS^TKN^EOFSTOP (nonshared). If the distributor is accessing current collector logs, this parameter lets you specify that you do not want to wait for the next incoming message when you have reached the end of the log file. If you set EOFSTOP to TRUE and there are no further event messages to return, the distributor returns an end-of-file warning in the response. The ZSPI^TKN^RETCODE is zero and, in the accompanying error list, the z^error field of ZSPI^TKN^ERROR is ZEMS^WRN^EOF.

If you set EOFSTOP to FALSE (or omit it) and there are no further event messages to return, the distributor does not respond to the GETEVENT command message until the collector has logged a new event message.

This parameter is ignored if the distributor is accessing a saved log file.

This example uses GETEVENT to retrieve two event messages. No CONTEXT token is included in the first GETEVENT because it is the first one issued since the distributor was opened. But the second GETEVENT command message includes the CONTEXT token from the response to the first one.

! Build a GETEVENT command message that includes EOFSTOP but excludes CONTEXT.

! Initialize spibuffer for GETEVENT command message
IF (error := SSINIT(spibuffer,
    ZEMS^VAL^BUFLEN,
    ZEMS^VAL^SSID,
    ZSPI^VAL^CMDHDR,
    ZEMS^CMD^GETEVENT)) THEN ...

! Place token--with token value--in spibuffer
eofstopvalue := ZSPI^VAL^TRUE;
spi^error := SSPUTTKN(spibuffer,
    ZEMS^TKN^EOFSTOP,
    eofstopvalue);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...
! Save an image of this buffer for later GETEVENT commands
spibuffer^image := spibuffer for ZEMS^VAL^BUFLEN Bytes;

! Determine how many bytes of spibuffer have been used
spi^error := SSGETTKN(spibuffer,
ZSPI^TKN^USEDLEN,
buffer^length);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! And send the command message
CALL WRITEREAD(fnum,
spibuffer,
buffer^length,
ZEMS^VAL^BUFLEN,
count^read);

IF <> THEN ... ! Handle the error

! The SPI response is now in spibuffer.

! Reset buffer length to what you declared for spibuffer
spi^error := SSPUTTKN(spibuffer,
ZSPI^TKN^RESET^BUFFER,
ZEMS^VAL^BUFLEN);

! See if anything wrong with response buffer
IF spi^error <> ZSPI^ERR^OK THEN ...

! Buffer ok. Was distributor command message ok?
spi^error := SSGETTKN(spibuffer,
ZSPI^TKN^RETCODE,
error,
1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

IF error = 0 THEN
BEGIN
! Command completed OK, but might have reached EOF
IF SSGETTKN(spibuffer,
ZEMS^TKN^EVENT,
embuffer,
! an integer array
1) = ZSPI^ERR^OK THEN
BEGIN
! Event message returned, stored in emsbuffer ...
END ELSE
BEGIN
! EOF reached, no event message returned
END;

Obtaining an Event Message (GETEVENT)

END
ELSE

! Command completed without event message
BEGIN
! An error occurred on the command
END;

! Move context into spibuffer^image
spi^error := SSMOVETKN(ZSPI^TKN^CONTEXT,
    spibuffer, 1,
    spibuffer^image, 1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! Build a GETEVENT command message that includes EOFSTOP
! and CONTEXT.
spibuffer ':=' spibuffer^image for ZEMS^VAL^BUFLEN Bytes;

! Determine how many bytes of spibuffer have been used
spi^error := SSGETTKN(spibuffer,
    ZSPI^TKN^USEDLEN,
    buffer^length);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! And send the command message
CALL WRITEREAD(fnum,
    spibuffer,
    buffer^length,
    ZEMS^VAL^BUFLEN,
    count^read);

IF <> THEN ... ! Handle the error

! The SPI response is now in spibuffer.
! Reset buffer length to what you declared for spibuffer
spi^error := SSPUTTKN(spibuffer,
    ZSPI^TKN^RESET^BUFFER,
    ZEMS^VAL^BUFLEN);

! See if anything wrong with response buffer
IF spi^error <> ZSPI^ERR^OK THEN ...

! Buffer ok. Was distributor GETEVENT command ok?
spi^error := SSGETTKN(spibuffer,
    ZSPI^TKN^RETCODE,
    error,
    1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! Check retcode as appropriate
Extracting Tokens From the Response (SSGET)

At the highest level, the GETEVENT response includes only a few basic tokens, among them the event message itself. To extract these tokens from the response buffer, use the SSGET procedure. (For a description of SSGET, see the SPI Programming Manual.)

- **ZSPI^TKN^RETCODE.** The return token is used to report the overall result of the command processing, as it is for all other subsystem command-response interfaces. A nonzero value means that a serious error has occurred. A value of zero indicates that the command completed without errors. However, a warning might have been issued (for example, an end-of-file warning, as described in Obtaining an Event Message (GETEVENT) on page 4-12 under ZEMS^TKN^EOFSTOP).

- **ZEMS^TKN^EVENT (nonshared).** This token contains the event message you requested by issuing the GETEVENT command message.

- **ZSPI^TKN^CONTEXT.** This token contains the current distributor context, as described in Obtaining an Event Message (GETEVENT) on page 4-12.

- **ZEMS^TKN^PASSVAL (nonshared).** If the filter executed a PASS statement that included a pass value (for example, PASS 3), the PASSVAL token contains the pass value. If the filter executed a simple PASS statement (with no specified value), this token is not present.

For example, to extract the event message from the response buffer:

```plaintext
spi^error := SSGETTKN(spibuffer,
                      ZEMS^TKN^EVENT,
                      emsbuffer,        !an integer array
                      1);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ...

! Reset buffer length to what you declared for evtbuffer
@evtbuffer := @emsbuffer '+' 1; !skip over length field
spi^error := SSPUTTKN(evtbuffer,
                      ZSPI^TKN^RESET^BUFFER,
                      ZEMS^VAL^BUFLEN);

! See if anything wrong with evtbuffer
IF spi^error <> ZSPI^ERR^OK THEN ...
```

Extracting Tokens From the Event Message (EMSGET)

To extract tokens from an event message, use the EMSGET procedure. For a detailed description, see Section 15, EMS Procedures. EMSGET is analogous to SSGET, but EMSGET retrieves tokens from event messages. SSGET retrieves tokens from command and response messages.
The tokens you choose to extract depend on your application. For example, to get the value of the event number token and get the subject token from an event message:

! Retrieve the event number from the event message
spi^error := EMSGETTKN(emsbuffer,
    ZEMS^TKN^EVENT^NUMBER,
    event^number);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ... 

! Find the first subject token in the event message
spi^error := EMSGETTKN(emsbuffer,
    ZEMS^TKN^SUBJECT,
    subject^token,
    1,
    subject^index,
    subject^ssid);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ... 

! Get the length of the subject to check against the
! length you declared to receive it
spi^error := EMSGETTKN(emsbuffer,
    ZSPI^TKN^LEN,
    subject^token,
    subject^index,
    subject^length,
    subject^ssid);

! Handle any error from SPI procedure
IF spi^error <> ZSPI^ERR^OK THEN ... 

IF subject^length > declared^size^for^subject^value THEN
    ! Cannot ssget this subject value
else
    BEGIN
        spi^error := EMSGETTKN(emsbuffer,
            subject^token,
            subject^value,
            subject^index,
            subject^ssid);

        ! Handle any error from SPI procedure
        IF spi^error <> ZSPI^ERR^OK THEN ... 
    END;

    ! Now, subject^value contains the value of the subject.
Generating Display Text (EMSTEXT)

Event messages contain information in a form most suitable for programmatic access, rather than for display. To obtain text suitable for display, use the EMSTEXT Procedure on page 15-19.

EMSTEXT lets you specify several parameters that control the form of its output:

- The event message buffer.
- The length of each line of text.
- The maximum number of lines of text.
- The amount that the second and subsequent lines of text are indented relative to the first line. A positive integer specifies the number of characters of indentation. Zero indicates no indentation. A value of -1 specifies that the lines be indented by the width of the header information in the first line. If you omit the parameter, -1 is used.
- The format of the text message: a user-selected format or DSM display format. User-selected formats are possible with format templates. For information, see the DSM Template Services Manual.

EMSTEXT returns the text and the number of characters used in each line.

This example shows an EMSTEXT call to generate up to five lines of display-format text. Each line can be 80 characters long and is indented by the width of the message header.

Literal maxlines = 5,
    maxlength = 80;
String .textbuffer [0 : maxlines*maxlength - 1];
Int    .linelengths[0 : maxlines-1];
Int(32) emstext^err;

emstext^err :=
    EMSTEXT(emsbuffer, !event message for text generation
            textbuffer, !holds the generated text
            maxlength, !80-char-length lines
            maxlines, !up to five lines
            linelengths, !lengths of lines in textbuffer
            , !for Compaq use only
            , !default indent
          );

!Use of emstext^err is optional.
if emstext^err <> 0D then ... 

This EMSTEXT call results in returned parameters containing this information.
Example 4-1. Normal-Format Text Produced by EMSTEXT

93-07-09 16:31:45 \COMM.$ZTNT TANDEM.TELNET.D10 000002 Socket read error file number 9 error 0 terminal #PTY9
5 Compiled Filters

This section describes compiled filters. For a detailed description of filter tables and burst filters, see Section 6, Filter Tables and Burst Filters.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing and Compiling Filters</td>
<td>5-1</td>
</tr>
<tr>
<td>The Filter Language</td>
<td>5-7</td>
</tr>
<tr>
<td>The Filter Compiler</td>
<td>5-44</td>
</tr>
</tbody>
</table>

Writing and Compiling Filters

The filter language and compiler are used to construct compiled filters.

Note. Symbolic names in this section are in TACL form, using circumflex (^) symbols rather than hyphens, because of the interface to TACL of the filter language.

Filter Operation

This subsection describes the environment in which compiled filters run.

Event Message Filters

A compiled filter is a program that runs interpretively in a collector or distributor while that collector or distributor reads event messages. Filters select or reject messages, depending on criteria that you specify in a filter specification. When compiled filters are used, the filter specification must be compiled by the filter compiler, and the filter object (or filter) must be loaded into the collector or distributor before filter operation can begin.

EMS filters, including filter tables and burst filters, reduce message traffic within a system and over the network by placing part of the program logic as close to the source of event messages (a collector or distributor) as possible. Rejected messages are skipped, while selected messages proceed to their destinations. For collector filters, the destination for selected events is the collector’s log file. The destination for messages selected by distributor filters depends on the distributor type:

- Consumer distributors return selected event messages in response to GETEVENT command messages. Each response message contains at most one event message.
- Printing distributors send the selected event messages—in the form of display text—to TEXTOUT destinations.
- Forwarding distributors send the selected event messages to a primary collector on another system (node) or to an alternate collector on the same node.
Source of Event Messages

The collector or distributor executes the filter from the beginning for each event message that it reads. The collector or distributor reads messages sequentially from its event message source. Event messages are sent from subsystems to primary and alternate collectors by WRITE or WRITEREAD procedures. For distributors, the event message source is either a saved log file or the log files of one or more collectors. The messages are arranged in the source in the order that they were logged. If more than one collector is used as a message source for a distributor filter, the messages are merged in log-time order and examined by the filter one by one.

Operations Not Controlled by the Distributor Filter

You must control the distributor to manage these aspects of message examination, which are closely linked to filter operation:

- The source of event messages, whether collector log-files or a saved log-file
- The position in the event-message source of the first message to be examined by the filter
- The destination of event messages that have passed the filter or what processing those messages undergo

For more information on start-up options for a distributor, see Section 13, EMS Programs. For information on how to control a distributor programmatically, see Section 17, Distributor Commands and Responses.

Introduction to the Filter Language

The filter language that is used to construct compiled filters provides features to examine and compare message tokens and their values.

Filter Language Features

- Filter parameters, which let you defer specification of token values until the filter is loaded
- A MATCH function, which determines whether a value in the message matches a pattern; the pattern can include wild-card characters
- PASS and FAIL statements, which select or reject the event message that the filter is currently examining
- An interface to TACL, which provides two advantages:
  - You do not need to declare the names of tokens in the filter; names of tokens for NonStop Kernel subsystems are defined in standard definition files, which you make available to the compiler. Many other definitions, for user subsystems as well as NonStop Kernel subsystems, are available in the same way.
You can supplement the filter language facilities with your own TACL commands and macro invocations. During filter compilation, TACL expands any text in the filter source that is surrounded by brackets ([]).

- A TOKENPRESENT function, which returns TRUE or FALSE, depending on whether the current event message contains the specified token
- A LITERALLY function, which makes comparisons case-sensitive, overriding the standard comparison convention, which is to ignore case
- Relational operators ( =, <>, >, >=, <, and <=), which allow comparisons between token values and constants
- Boolean operators (AND, OR, and NOT) and Boolean expressions
- Boolean variables and assignment statements
- IF-THEN-ELSE and compound statements

**Sample Filter Specification**

This filter selects event messages local to the \ABCD system:

```tcl
FILTER forward^only^local^events;
BEGIN
  IF ZEMS^TKN^SYSTEM = [#SYSTEMNUMBER \ABCD] THEN
    PASS
  ELSE
    FAIL;
END;
```

The filter specification does not declare the token name ZEMS^TKN^SYSTEM, because—like all names used by the EMS subsystem—it is defined in the standard definition file for EMS, $SYSTEM.ZSPIDEF.ZEMSTACL. For details, see Loading Standard Definitions on page 5-4.

The brackets that surround the #SYSTEMNUMBER macro-call direct TACL to expand the enclosed text. This expansion occurs during filter compilation, not during filter execution. In the filter listing, the brackets and their contents are replaced by a constant (116, in this case), the system number of \ABCD.

**Creating a Compiled Filter**

To create a compiled filter:

1. Identify the conditions that event messages must meet for the task at hand.
2. Create an EDIT file in the Event Management Service filter (EMF) language that describes these conditions. This is the filter specification.
3. Compile the filter specification to create the filter, which is returned in an object file.
Loading Standard Definitions

You must load the standard definitions of SPI and of each subsystem whose tokens you use. These TACL statements load the standard definitions for SPI and EMS:

```tcl
#PUSH dummy
#LOAD /KEEP 1, LOADED dummy/ $SYSTEM.ZSPIDEF.ZSPITACL
#LOAD /KEEP 1, LOADED dummy/ $SYSTEM.ZSPIDEF.ZEMSTACL
#POP dummy
```

You could also use this ATTACHSEG TACL command:

```tcl
ATTACHSEG SHARED $SYSTEM.ZSPISEGF.ZxxxTACL :Zxxx
#SET #USELIST [#USELIST] :Zxxx
```

where Zxxx is either ZSPI or ZEMS.

If your site places the standard definition files on another volume, change $SYSTEM to the name of the other volume.

You can enter these TACL statements, exactly as shown, at the TACL that runs the filter compiler. You can also include the TACL statements in the filter specification if you surround each statement with brackets.

For information on how to create and attach a TACL segment file—a fast way to save and restore standard definitions—see the TACL Reference Manual.

⚠️ Caution. Always re-create TACL segment files that save standard definitions whenever your site installs a new release of the programmatic interface. Using private copies of such files can cause errors if they are not kept current.

Compiling a Filter Specification

After you have loaded the standard definition files that your filter requires, you can compile the filter.

To compile the filter specification shown in Sample Filter Specification on page 5-3 (in EDIT file FWDFLT) and to place the compiled filter in object file OFILTER, enter this TACL command:

```tcl
EMF / IN FWDFLT / OFILTER
```

The compiler listing is:

```
EMS Filter Compiler - T9634C00 - (15JUL87) - (19JUN87)
COPYRIGHT TANDEM COMPUTERS INCORPORATED 1987
1 FILTER forward^only^local^events;
2 BEGIN
3  IF ZEMS^TKN^SYSTEM = 116 THEN
4     PASS
5  ELSE
6     FAIL;
7 END;
8
Object file name is $HUBS.MYSUB.OFILTER
```
Considerations for Writing Filters

When you write compiled filters, consider several factors to retrieve exactly the event messages you need while making optimum use of system and network resources, and to develop a library of filters that are maintainable:

Selectivity

You can exclude large numbers of irrelevant messages if your application can determine that the messages to be examined are in a restricted time interval. If such an interval is known, position the distributor to the beginning of the interval. For details, see the distributor CONTROL command in Section 17, Distributor Commands and Responses.

To provide an early stop for event-message examination by a consumer distributor, consider one of these methods:

- To make the distributor stop when it reaches the end of the event messages already logged, include the ZEMS^TKN^EOFSTOP token when sending the GETEVENT command message. (For more information, see GETEVENT Command (ZEMS-CMD-GETEVENT) on page 17-25.)

- If the application can determine a particular time at which to stop (a log time, for example), the application can pass that time to the filter as a parameter. The filter can signal that it has seen a message with such a time by executing a PASS statement with a special PASS value; a PASS 0 statement, for example. This strategy can save significant computer time. The special PASS value tells your application to stop sending GETEVENT command messages, so the filter does not execute a FAIL statement for each event message from the stop time to the end of the log files.

If a compiled filter is to select messages from one subsystem only, or if the filter has a compound statement for selecting those messages, place an IF statement near the beginning of the filter or the compound statement to exclude messages from other subsystems.
Efficiency

Filters help a forwarding distributor make the network more efficient. Unneeded event messages that you exclude with a well-written filter help reduce network message traffic.

⚠️ **Caution.** Do not use a forwarding distributor to return a message to its system of origin. Doing so can cause a message loop that rapidly consumes system and network resources and causes event-message flooding. Unexamined, relevant collector log files might be overwritten if this continues with ROTATEFILES selected.

Filters can help operator efficiency in the case of printing distributors. Extra messages can delay the printing of important messages, and irrelevant messages can clutter a listing, obscuring relevant messages.

Filters can reduce the amount of interprocess communication in the case of consumer distributors.

To help decide whether to perform an operation in the filter or to perform it in an application, consider:

- The filter processes every message that the distributor reads. The application processes only the selected messages.
- Each time a filter examines a new event message, its Boolean variables are set to FALSE. Therefore, if part of your selection logic requires saving the value of comparisons across event messages, place that logic in the application, not in the filter.
- If the filter has categorized event messages while selecting them, the integer on the PASS statement can be used to return the category to an application.

Some compiled filters are written with substantial TACL computations included in brackets in the filter source. This strategy might appear inefficient but is not because TACL expands these bracketed constructs during filter compilation, not during filter execution.

Compatibility

Consider these points, which should help your filter perform without changes across different versions of the SPI interface:

- When the same information exists in an event message in two forms—a text form for display purposes and an encoded form—comparisons in the filter should use the encoded form. When information is duplicated in this way, HP reserves the right to change the display text at any time.
- Use volume names as opposed to logical device numbers where possible.
When checking the subsystem ID of an event message, use the SSID function—which ignores the version—as in this example:

```plaintext
IF NOT ( ZSPI^TKN^SSID = SSID(ZPWY^VAL^SSID) ) THEN FAIL;
```

### Correctness

To help you write accurate compiled filters:

- Check that the event messages come from the correct subsystem. That is, begin the filter with an IF statement to exclude any event messages whose main subsystem ID (the value of the ZSPI^TKN^SSID token) is not the one you want. Then check for tokens you know the subsystem has placed in the message.

- When searching for a token, you must know the subsystem ID that qualifies the token within the message. You must inform the filter of this subsystem ID: the subsystem ID that you would pass to the EMSGET procedure to get the token from an application. Most event-message tokens are qualified by one of two subsystem IDs:
  - Subsystems place most tokens in their event messages without explicit subsystem-ID qualification. Such tokens are implicitly qualified, by the subsystem ID of the subsystem that generated the message. In the filter, you can refer to such tokens by simple (unqualified) names.
  - You can also refer to any header tokens by simple names because header tokens are shared by all subsystems.
  - The subsystem might place some tokens in the event message qualified by the subsystem ID of EMS. Certain tokens, such as ZEMS^TKN^ACTION^NEEDED, are defined by EMS but used by other subsystems. (ZEMS^TKN^ACTION^NEEDED is not a header token.) You can refer to these tokens in a filter in two ways:
    - Explicitly qualify the token name by the subsystem ID for EMS:
      ```plaintext
      IF SSID ( ZEMS^VAL^SSID, ZEMS^TKN^ACTION^NEEDED) & = -1 THEN ...
      ```
    - Place the unqualified token name in a compound statement that starts:
      ```plaintext
      BEGIN SSID ( ZEMS^VAL^SSID )
      ```

### The Filter Language

This subsection provides detailed information about the filter language used with compiled filters, which lets you select some event messages and screen out others.

For an introduction to using the filter language and the filter compiler, see [Writing and Compiling Filters](on page 5-1). For a description of how to produce an object filter program from a filter specification, see [The Filter Compiler](on page 5-44). For
Overview of Filter Operation

A filter operates in a collector or a distributor, which might already be running or must be started. You configure filters in an alternate collector or distributor when they are first started by specifying the filters in the startup options. Use the EMSACOLL program to start an alternate collector. For details, see "EMSACOLL—Alternate Collector Program" on page 13-2. Use the EMSDIST program to start a distributor. For details, see "EMSDIST—Distributor Program" on page 13-34.

Alternatively, filters can be configured in a primary or alternate collector or a distributor dynamically (that is, while they are running) by using object-oriented SPI commands to ADD, ALTER, REPLACE, or DELETE filters.

Filter parameters are analogous to procedure parameters. They let you delay the specification of certain token values until you install the filter or let you change the values during operation. You can change these filter parameters only with the object-oriented SPI ADD, ALTER, REPLACE, or DELETE commands.

The collector in which a filter operates reads event messages sequentially as they are received through WRITE or WRITEREAD procedures from a subsystem. The distributor in which a filter operates reads event messages sequentially from a series of one or more log files. Each time the collector or distributor reads an event message, it executes the filter.

Each token name refers to a parameter token if the name matches the name of a declared parameter token. Otherwise, the name refers to a token in the current event message—that is, the event message that the distributor read last.

Filter execution terminates if a PASS statement is executed, if a FAIL statement is executed, or if control reaches the last statement of the filter without executing a PASS or FAIL, which is an implicit FAIL. Consequently, each event message either passes or fails the filter.

The collector or distributor uses the event message—sends it to a log file, prints it, forwards it, or sends it to an application—if the message passes the filter. Otherwise, the collector or distributor ignores the event message.

Pass or fail, the collector or distributor is then ready to read another event message and to begin the cycle again.

TACL Environment

The filter language uses TACL as a preprocessor. TACL expands source text within brackets before the compiler processes it. You must treat characters that are special to TACL in a special way.

---

Note. Symbolic names in this section are in TACL form, using circumflex (^) symbols rather than hyphens, because of the interface of the filter language to TACL.
The compiler uses TACL to translate token names to token codes. The TACL process used is the one that calls the compiler. For more information about the relationship between the filter language and TACL, see The Filter Compiler on page 5-44.

Basic Components

This subsection describes the lexical elements of the filter language—such as names, operators, and constants—with out much discussion of their relationship to each other. The following subsections describe the way these elements combine to form larger structures and ultimately a complete filter specification.

Comments

If you include two dashes (--) on a line (but not inside a string), the rest of the line is a comment. (You can write dashes in a string as is, without an escape character. In strings, dashes do not start a comment.)

Use comments freely to improve the readability of your source files or compiler listings.

Although comments using the TACL comment characters == and { } improve the readability of your source file, they do not affect the compiler listing of your source text. TACL strips TACL comments out of the language text. They do not appear in the compiler listing.

Escape Character (~)

If you require a TACL special character in the filter specification for a usage not understood by TACL, you must precede that character (or character group) with a tilde (~), the TACL escape character. Specifically, when you need these characters:

```
[     ]     {     |     ==
```

Write them as:

```
~[    ~]    ~{    ~|    ~==
```

(TACL processes the filter specification before the compiler sees it. The tilde is not kept in the string constant. It warns TACL not to treat these characters as TACL special characters.)

Reserved Words

Do not use the keywords reserved for filter language use as identifiers either in uppercase or lowercase characters.

<table>
<thead>
<tr>
<th>Table 5-1. Reserved Words (page 1 of 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
</tr>
<tr>
<td>BEGIN</td>
</tr>
<tr>
<td>BOOLEAN</td>
</tr>
</tbody>
</table>
Names

Names in the filter language follow the conventions used in TACL, except for length. Names contain letters, digits, underscores (_), and circumflexes (^), and begin with a letter. Case is not significant. TACL names can contain up to 31 characters. Names in the filter language cannot be longer than 30 characters.

Names represent Boolean variables and filter specifications, both of which you must declare in the filter language.

Names also represent tokens, structures, fields (structure subcomponents), and subsystem IDs. These names are actually names of TACL variables and must be defined in TACL. You define most of these names through standard definition files that you must load. For information on how to load standard definition files, see The Filter Compiler on page 5-44.

Names that represent constants, such as ZSPI^VAL^TRUE, cannot be used as is. You must surround such names with brackets, as in [ZSPI^VAL^TRUE]. TACL expands these bracketed expressions to the values that they represent. ([ZSPI^VAL^TRUE] is displayed in your source listing as -1, the value of ZSPI^VAL^TRUE.)

Tokens

In a comparison, a token name represents the value of the token. For details, see Comparisons on page 5-19. This example compares the value of the header token ZEMS^TKN^CPU in the current event message to 3:

IF ZEMS^TKN^CPU = 3 THEN PASS;

The current event message passes the filter if the comparison is TRUE.

A token name refers to a parameter token if you so declare it. (For details, see Filter Parameters on page 5-29.) Otherwise it refers to an event-message token, as in this example.

To specify a token uniquely, you usually need to specify both the subsystem ID of the token and the token name. You must do so because the token code that represents something in subsystem \( x \) usually represents something else in subsystem \( y \). (The subsystem ID is not needed for token codes that are shared by all subsystems, such as header token-codes.)

For information about the subsystem ID that the language uses when none is specified, see EMF Default Subsystem ID on page 5-12.
When you refer to a token, specify a subsystem ID using syntax that:

- Qualifies the token name with a subsystem ID. For details, see **Qualified Tokens** on page 5-12.
- Refers to a token name without qualification in contexts in which the subsystem ID of the token matches the EMF default subsystem ID. For details, see **Tokens (Unqualified)** on page 5-11.

**Tokens (Unqualified)**

To refer to tokens when qualification by subsystem ID is unnecessary, use this syntax:

\[
\text{token-name} \ [ \ ( \text{index} \ ) \ ]
\]

- **token-name** is the token name.
- **index** if present, is an integer that specifies an occurrence of the token in the event message (or in the set of parameter tokens). index must have a value between 1 and 1023, inclusive. If you omit index, you select the first occurrence of the token.

Use this syntax to refer to:

- All event-message header tokens
- All parameter tokens
- A data-portion token if the subsystem ID of the token matches the EMF default subsystem ID at the place of reference

For a description of how to change the EMF default subsystem ID, see **Compound Statement** on page 5-31.

For information about how to specify a subsystem ID where the above syntax is inappropriate, see **Qualified Tokens** on page 5-12.

**Example 1**

To compare the value of the ZEMS^TKN^EMPHASIS token with the value ZSPI^VAL^TRUE:

\[
\text{IF ZEMS^TKN^EMPHASIS} = \ [\text{ZSPI^VAL^TRUE}] \ \text{THEN PASS};
\]

The PASS statement is executed if the ZEMS^TKN^EMPHASIS token has a value of ZSPI^VAL^TRUE.
Example 2
This example shows an indexed token reference:

ABC^TKN^STUFF(3) = 5

In this example, the value of a stuff token is compared to 5.

Specifically, the event message is searched, from left to right, until the third stuff token is found. The value of that token is then compared with 5.

If the event message contains fewer than three stuff tokens, the value of the comparison is FALSE. For details, see Comparisons With Missing Values on page 5-25.

EMF Default Subsystem ID

If you refer to a token without qualifying the token name by a subsystem ID, the token owner is provided by the EMF (EMS filter) default subsystem ID. By using this default, you can avoid explicit token qualification in most cases. The default subsystem ID is defined everywhere within the filter specification. Unless you change it, the EMF default is the NULL subsystem ID. To change the EMF default subsystem ID, use a compound statement. For details, see Compound Statement on page 5-31.

NULL Subsystem ID

When you specify a NULL subsystem ID, the filter supplies the value of the ZSPI^TKN^SSID token in the event message it is currently examining. ZSPI^TKN^SSID identifies the subsystem that reported the event. Therefore, if the filter is examining a Pathway or HP NonStop Transaction Management Facility (TMF) event message, the NULL subsystem ID represents Pathway or TMF, respectively. Within EMF, 0.0.0 designates the NULL subsystem ID.

Qualified Tokens

To refer to tokens by names that are qualified by subsystem IDs, use this syntax:

```
SSID ( ssid , token-name [ ( index ) ] )
```

ssid

is the subsystem ID that specifies the token owner. (Include ZEMS^VAL^SSID or TANDEM.EMS.0 to represent the EMS subsystem, for example.)

token-name

is the token name.
index

is an integer that specifies an occurrence of the token in the event message. index must have a value between 1 and 1023, inclusive. By omitting index, you select the first occurrence of the token.

If you specify a value for index that is less than 1 or greater than 1023, the filter compiler issues this error message:

*** Error ***
This number must be between 1 and 1023, inclusive.

You must qualify the name of a token with the name of the subsystem ID if the EMF default subsystem ID at the place of reference does not match the subsystem ID of the token in the event message.

Considerations

- Use qualified token names to override the EMF default subsystem ID.
- Do not use this syntax to refer to parameter tokens.
- For information on how to specify a token when qualification is unnecessary, see Tokens (Unqualified) on page 5-11.
- Remember to qualify EMS tokens that are not header tokens by ZEMS^VAL^SSID. Or you can qualify all EMS tokens and ignore the distinction between header and data-portion tokens.
- You might ask whether the subsystem ID of a token can be derived from the token name to avoid the need for name qualification. The answer is that some subsystems use different naming conventions, so the compiler must use the token name only to determine the token code.
- If you specify a value for index that is less than 1 or greater than 1023, the filter compiler issues this error message:

*** Error ***
This number must be between 1 and 1023, inclusive.

Example

To qualify the EMS token in this statement (which is incorrect in most contexts):

IF ZEMS^TKN^ACTION^NEEDED = [ZSPI^VAL^TRUE] THEN PASS;

Write this statement which is always correct:

IF SSID ( ZEMS^VAL^SSID, ZEMS^TKN^ACTION^NEEDED ) = [ZSPI^VAL^TRUE] THEN PASS;
Fields

A field name refers to a simple subcomponent (containing an integer, for example) of a structured token within the event message. In a comparison, the field name represents the value of the subcomponent or of an array of subcomponents.

The syntax to refer to a field:

```
token-name.struct-name [ : field-name [ ( index ) ] ] ...
                [ : field-name [ ( index ) ] ]
                [ ( range ) ]
```

- `token-name`  
  is the name of a structured token. That is, `token-name` is the name of a TACL variable that contains the token map or token code for the structured token. You can qualify the token name and include an index. For details, see Tokens (Unqualified) on page 5-11 and Qualified Tokens on page 5-12.

- `struct-name`  
  is the name of a TACL variable of type STRUCT that presents the structure of `token-name`.

- `field-name`  
  is the name of a field or substructure within the structure.

- `index`  
  is an integer that specifies one element of an array of fields. If you omit both `index` and `range`, `index` is zero.

- `range`  
  is a pair of indexes (integers) separated by a colon; 2 : 4, for example. This notation represents an array of fields, which includes the field at the first index through the field at the second index.

Considerations

- The field syntax is that used in TACL.

- The TACL variable `token-name` sometimes contains a token code rather than a token map. A token code is appropriate if the token (ZSPI^TKN^ERROR, for example) is a structured token but not an extensible structured token.
Example 1

To use a field reference to exclude all messages except the ones created by the $ZVPT process:

[#DEF fname32^struct STRUCT  
  BEGIN  
  CHAR  systemname (0:7);  
  FNAME  name;  
  END;  
]

FILTER fld_ref;

BEGIN
-- Refers to field of a SENDERID token
IF ZEMS^TKN^SENDERID.fname32^struct:name = $ZVPT THEN PASS;
END;

As this example shows, to refer to a subcomponent of a structured token, you must first define the structure in TACL (within brackets, as usual).

The structured ZEMS^TKN^SENDERID token, which is of data type fname32, contains the name of the process that created the event message as one field.

Example 2

This example compares the value of a subcomponent of an extensible structured token with the enumerated constant ZEMS^VAL^FORWARD^DIST. ZEMS^VAL^FORWARD^DIST indicates that the message was generated by a forwarding distributor.

FILTER events^from^forwarding^dist;

-- Select all event messages that were  
-- generated by a forwarding distributor

BEGIN
  IF ZSPI^TKN^SSID = SSID (ZEMS^VAL^SSID) AND 
    ZEMS^MAP^DIST^STATUS.ZEMS^DDL^DIST^STATUS:ZDIST^TYPE 
    = [ZEMS^VAL^FORWARD^DIST] 
    THEN PASS;
END;

Constants

The filter language supports the following types of constants: integers, strings, file names, subsystem IDs, and constant lists.
Integers

You can use any signed integer constant that can be expressed internally in 64 bits. For example:

1     99     -523     32769    2114678910123456708

You write unsigned, integer constants the way you write signed, positive integers.

Strings

Surround the characters of a string constant with double quotes. For example:

"pqrstuvw"

Subsystem IDs

You can refer to a subsystem ID in either of two ways. First, you can use the name of a TACL variable that has been appropriately initialized. For example, ZEMS^VAL^SSID is the name of the subsystem ID for EMS. For information about how to initialize such a subsystem ID, see The Filter Compiler on page 5-44.

Alternatively, you can represent a subsystem ID as a series of component parts:

organization . subsystem . version

organization

is the name of the organization associated with the subsystem. It includes from one to eight letters, numbers, or dashes, beginning with a letter (HP, for example).

subsystem

is either an integer or a name. Each organization uses integers to represent its subsystems; you can use an integer to specify subsystem. For subsystems for the NonStop server, for example, you can use names such as the following to specify subsystem: EMS, DNS, TMF, EXPAND, FOX, MONITOR, and PATHWAY. You can use org.name for any subsystem that is present in the system template file. For a complete list of NonStop Kernel subsystem names and numbers, see the Operator Messages Manual.

version

is either an integer or a version identifier. An integer represents the subsystem version. This form is defined for all subsystems. A version identifier, such as D20, represents the version of a NonStop Kernel subsystem. This form is defined only for subsystems.
Considerations

- The compiler converts a subsystem ID expressed in the syntax just shown to an internal form. For a description of the internal form of a subsystem ID, see the SPI Programming Manual.

- To compare subsystem IDs, you should ordinarily use the SSID function, which removes the version number from a subsystem ID. Comparisons with version numbers stop working as soon as a new version of the event messages is released. For more information, see SSID Function on page 5-42.

- HP recommends that you use symbolic values rather than integers to represent subsystems for the NonStop server; EMS for subsystem and D20 for version, for example.

Example 1

To express the subsystem ID of subsystem 2, version 51, of the (fictional) Quirks Corporation:

QUIRKS.2.51

Example 2

To express the subsystem ID of the HP EMS subsystem, version D20:

TANDEM.EMS.D20

File Names

The syntax for expressing a file name is:

```
[ \ sys-name . ] $ vol-name [ . subvol-name ] . file-name
```

- **sys-name**
  - if present, is the name of the system (node).

- **vol-name**
  - is the name of the volume.

- **subvol-name**
  - if present, is the name of the subvolume. The filter compiler uses your file defaults if you omit `subvol-name`.

- **file-name**
  - is the file part of the file name.
Considerations

- Do not include spaces in file names. Spaces are included in the syntax box only to increase visibility.
- You can use either uppercase or lowercase characters for file names.
- The compiler converts file names to internal form (24-bytes).
- A file name must include the volume and the file name. The system name and subvolume name are optional.

Examples

These file names are correct:

$SYSTEM.X.Y      \comm.$pubs.any.name

These file names are incorrect:

Y       any.name

Y and any.name do not begin with either a backslash (\) or a dollar sign ($).

Constant Lists

A constant list is a structured constant that is composed of simple constants.

The syntax for a constant list is:

```
( constant [ , constant ] ... )
```

constant

is a number, a quoted string, a subsystem ID, or a file name. Each number in the constant list is assumed to occupy 1-byte and must be in the range 0 through 255.

Considerations

- Constant lists in the filter language are similar to those in TAL but use parentheses, rather than square brackets. Square brackets are special to TACL. (Square brackets trigger TACL macro expansion.)
- The compiler converts a constant list to an internal form that consists of an array of bytes.
- The data type of constant lists is unsigned.

Example

You can create an array of 4-bytes:

```
(3, 4, 95, 8)
```
Bit-Extraction Operator

To extract bits from a specified operand creating an unsigned integer value:

\[ \text{operand} . < \text{bit-1} : \text{bit-2} > \]

\textbf{operand}

is any token value, field, or constant that occupies a byte, word, double word, or quad word.

\textbf{bit-1}

is an integer in the range 0 through 63 that does not exceed the range implied by the data type: (0:7) for bytes, (0:15) for words, and so forth.

\textbf{bit-2}

is an integer in the range 0 through 63, inclusive, and must be greater than or equal to \textit{bit-1}.

\textbf{Considerations}

The range for the byte data type (0:7) differs from TAL conventions.

\textbf{Example}

This example demonstrates a bit-extraction operator extracting bits from a token value:

\texttt{IF XYZ^TKN^INTWORD.<12:15> = 15 THEN PASS 1;}

This example compares the four low-order bits of the full-word integer value with 15 and checks whether the four bits are all on.

\textbf{Comparisons}

The syntax for comparisons:

\[
\begin{align*}
&= \\
&> \\
&>= \\
&<>
\end{align*}
\]

\[
\begin{align*}
&\text{operand} \\
&<>
\end{align*}
\]

\[
\begin{align*}
&< \\
&<=
\end{align*}
\]
Operand

is a token name (qualified or unqualified), a field name (qualified or unqualified), a constant, or a bit-extraction operator applied to any of the elements just mentioned.

The way the filter compares values—token values with token values or token values with constants—depends on the data types involved.

**Associating Token and EMF Data Types**

Table 5-2 lists the EMF (EMS filter) data type used to represent each of the token (and field) types and related information. Y (yes) in the Warning column of Table 5-2 advises you to not use the data type in a comparison. The compiler issues a warning message if you do so. A dash (-) indicates information that depends on the individual token or is not applicable, as with invalid types.

<table>
<thead>
<tr>
<th>Token Type (ZSPI-TDT-)</th>
<th>EMF Type</th>
<th>Size of Element</th>
<th>Number of Elements</th>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Signed</td>
<td>2</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>BYTE</td>
<td>Unsigned</td>
<td>1</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>CHAR</td>
<td>String</td>
<td>1</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>CRTPID</td>
<td>Unsigned</td>
<td>2</td>
<td>4</td>
<td>Y</td>
</tr>
<tr>
<td>DEVICE</td>
<td>Unsigned</td>
<td>2</td>
<td>4</td>
<td>Y</td>
</tr>
<tr>
<td>ENUM</td>
<td>Signed</td>
<td>2</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>ERROR</td>
<td>Unsigned</td>
<td>2</td>
<td>7</td>
<td>Y</td>
</tr>
<tr>
<td>FLT</td>
<td>Unsigned</td>
<td>2</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>FLT2</td>
<td>Unsigned</td>
<td>2</td>
<td>4</td>
<td>Y</td>
</tr>
<tr>
<td>FNAME</td>
<td>File name</td>
<td>24</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>FNAME32</td>
<td>String</td>
<td>1</td>
<td>32</td>
<td>Y</td>
</tr>
<tr>
<td>INT</td>
<td>Signed</td>
<td>2</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>INT2</td>
<td>Signed</td>
<td>4</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>INT4</td>
<td>Signed</td>
<td>8</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>LIST</td>
<td>Illegal</td>
<td>-</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>MAP</td>
<td>Unsigned</td>
<td>2</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>SSCTL</td>
<td>Illegal</td>
<td>-</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>SSID</td>
<td>Unsigned</td>
<td>2</td>
<td>6</td>
<td>N</td>
</tr>
<tr>
<td>SSTBL</td>
<td>Illegal</td>
<td>-</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>STRUCT</td>
<td>Unsigned</td>
<td>2</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>SUBVOL</td>
<td>String</td>
<td>1</td>
<td>16</td>
<td>Y</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Unsigned</td>
<td>8</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>TOKENCODE</td>
<td>Unsigned</td>
<td>4</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>TRANSID</td>
<td>Unsigned</td>
<td>8</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>UINT</td>
<td>Unsigned</td>
<td>3</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>UNDEF</td>
<td>Illegal</td>
<td>-</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>USERNAME</td>
<td>String</td>
<td>1</td>
<td>1</td>
<td>Y</td>
</tr>
</tbody>
</table>
As shown in Table 5-2, the filter language associates many token data types with four EMF data types: signed, unsigned, string, and file name.

For comparison all token values and field values are treated as one of these four types.

The filter language treats many token data types as unsigned values. Because of the internal structure of such data types, it is not meaningful to use most EMF comparison-operators, such as <, to compare values of many token data types. For example, it is not meaningful to say one subsystem ID is less than another. Therefore, the filter language treats such values as unsigned, which lets you use them in equal (=) and not equal (<> ) comparisons.

Comparisons are defined for all EMF data types.

The values that correspond to one or more of the operands in a comparison can be missing. For information on how the filter treats such comparisons, see Comparisons With Missing Values on page 5-25.

File-name comparisons with equal (=) and not equal (<> ) operators are treated as a special case. For details, see Comparing File Names on page 5-23. The compiler issues a warning for comparisons of file names with operators other than equal or not equal, and treats such comparisons as string comparisons, using the internal form of the file names. For more information, see Comparing Strings on page 5-22.

**Comparing Values Within an EMF Data Type**

Within an EMF (EMS filter) data type, you can compare token value with token value and token value with constant. For example, token values of INT, INT2, INT4, BOOLEAN, and ENUM can be compared with each other because they belong to one EMF data type: signed. Within an EMF data type the filter performs conversions automatically.

Certain comparisons within an EMF data type make sense, but others are meaningless. For example, to compare two tokens of type ZSPI\^TYP\^ERROR for equality makes sense in certain contexts. To make a less-than comparison between those tokens or between two tokens that are both unsigned but have different token data types might be meaningless.

**Comparing Values With Different EMF Data Types**

You can compare two tokens only if their EMF (EMS filter) data types agree. For example, you cannot compare a ZSPI\^TYP\^INT type token (signed) with a ZSPI\^TYP\^UINT type token (unsigned). You can compare a ZSPI\^TYP\^INT type token to a ZSPI\^TYP\^ENUM type token because they have the same EMF data type: signed.

In many cases, you can compare a token value to a constant of another EMF data type. The filter language converts the constant, when reasonable, to the EMF type of the token to which it is compared. Conversion of this kind is possible between signed and unsigned values and between string and file-name values. Conversion of this kind
is not possible between a signed (or unsigned) value and a string (or file name) value. These types are considered incompatible.

## Comparing Values With Composite Data Types

The filter language treats certain token types as composites, for purposes of comparing. These composites consist of two or more data-type elements, as defined here. For example, a subsystem ID consists of six unsigned integers.

You can determine the elements by looking at the SIZE OF ELEMENT and NUMBER OF ELEMENTS columns in Table 5-2. For example, the token type ZSPI^TDT^SSID has a SIZE OF ELEMENT of 2 and a NUMBER OF ELEMENTS of 6, indicating that an unsigned integer word is the basic element of the SSID type.

To compare the two composites, the filter examines them element by element until an unequal comparison results. The filter then determines whether those elements are in the relation specified by the comparison operator. The filter compares composites similarly to the way it compares strings. That is, a string can be viewed as an array of bytes.

You can define a new token type, based on basic token types, that will give you an array of tokens. For example, you can include this structure definition in the DDL source file:

```plaintext
?SECTION uvw-ddl-triple
  ! a triplet of signed integers
?TALBOUND 0
DEFINITION uvw-ddl-triple.
    02 int            TYPE zspi-ddl-int OCCURS 3 TIMES.
END
```

And include this TOKEN-TYPE declaration:

```plaintext
TOKEN-TYPE  uvw-typ-triple       VALUE IS zspi-tdt-int
            DEF   IS uvw-ddl-triple.
```

If you have included the definitions just mentioned, you have a user-defined type UVW-DDL-TRIPLE, for the user subsystem UVW. You can use it to create triplets of integers.

The filter compares such entities element by element to determine the value of the composite comparison. In this case, the filter performs a signed comparison of the three elements. For another way to compare arrays, see Comparing Fields on page 5-24.

## Comparing Strings

In string comparisons, the compiler pads the shorter string with blanks before performing a byte-by-byte comparison.
Except for expressions governed by a LITERALLY function, string comparisons ignore character case, which makes these strings equal:

"abc"
"ABC"
"AbC"

For information on making case-sensitive comparisons, see LITERALLY Function on page 5-41.

Comparing Signed Values

No conversion is necessary in a comparison of a ZSPI^TYP^INT token TKN^I with a small integer. For example:

IF TKN^I < 3 THEN PASS;

The value 3 and the token both have INT values.

If TKN^J is of type ZSPI^TYP^INT2, this statement requires a data conversion, which the language makes automatically because the data elements (INT and INT2) are of different sizes:

IF TKN^I < TKN^J THEN PASS;

If the data-element sizes differ, the filter converts each element to an 8-byte signed quantity.

You can compare two values that have a different number of elements ne1 and ne2. In such cases, the filter compares only nemin elements, where nemin is the lesser of ne1 and ne2. For example, when you compare a subsystem ID to the value returned by an SSID function, the values consist of six and five unsigned integer words, respectively, because the SSID function omits the version. The comparison stops after the comparison of the fifth integer.

Comparing Unsigned Values

The comparison of unsigned values is very similar to that of signed values except there is no sign extension.

Comparing File Names

These file-name comparisons are defined only for equal (=) or not equal (<> ) comparisons. In other comparisons, file names are treated as string data. Node numbers get no special treatment.

Nodes (systems) pose a special problem for file-name comparisons. Three nodes, possibly distinct, are involved:

- The node that compiles the filter
- The node that generates the event message
• The node that compares the file names

When writing a file name in a filter, you should normally omit the node name.

The file names are equal:

• If both file names are in local form, the value of the file-name comparison depends on whether the names match.

• If both file names are in network form, the value of the file-name comparison depends both on whether the system numbers match and on whether the rest of the file parts match.

• If one file name is network and the other is local, the value of the file-name comparison depends both on whether the one in network form designates the same system as ZEMS^TKN^SYSTEM and on whether the rest of the file parts match.

The file name comparisons just described apply only to equal (=) or not equal (<>)
comparisons. Other comparisons treat file names as string data, using the 24-character internal representation of the file name.

**Note.** Be careful of the following limitation when comparing file names with seven-character volume names. If $RELEASE is also $21 on the local node, comparing $21 with $RELEASE on node \A is TRUE. However, the same comparison on any other node is FALSE because $RELEASE is too long to put in network form. The comparison cannot be performed properly. The problem occurs only when you compare a seven-character volume name with a logical-device number of a device on a remote node.

**Comparing Fields**

Field values have the same data types as token values. You can compare a field with a constant, a token value, or another field.

In addition to the previous comparisons, you can compare arrays of fields, using the
*range* parameter in the field syntax.

The filter compares each pair of fields in two structured tokens or in a token and a composite constant. Comparison stops as soon as a FALSE comparison occurs. The maximum number of comparisons that can occur is equal to the number of array elements in the array with the lesser number of elements.

For example, suppose A and B are arrays of file names. If the first two file names are equal, the expression A <> B involves only one comparison because A <> B is FALSE on the first comparison.

As another example, you can compare an array of three single-word integer fields to an array of four double-word integer fields. The compiler converts each single- and double-word integer value to an 8-byte value (using sign extension) and then makes at most three (the lesser) comparisons.
Comparisons With Missing Values

The values of one or both operands of a comparison might be missing due to any of these causes:

- The comparison refers to a token that is not in the current event message.
- The comparison refers to an optional parameter that was omitted when the other parameters, if any, were supplied.
- The comparison refers to a field of an extensible structured token and either:
  - The structured token itself is not in the current event message.
  - The token is there, but the field is not defined in this version of the structured token.

If the value of either operand of a comparison is missing, the value of the comparison is FALSE. That is, the comparison itself is FALSE but not necessarily the value of the expression in which the comparison occurs.

These expressions are all FALSE if the tokens TKN^M and TKN^N are not in the event message:

TKN^M = 5
TKN^M <> 5
TKN^M = TKN^N
TKN^M <> TKN^N

With the same assumptions, this expression is TRUE:

\[ \text{NOT ( TKN^M = 5 )} \]

Boolean Expressions

Table 5-3 shows the elements that combine to form Boolean expressions. Some expression elements are defined in the table. Other elements are only mentioned there but are described more fully in a later subsection.

The letters A and B in the table represent Boolean expressions, which are legal combinations of elements in the table.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE, a Boolean expression, is the Boolean value of a successful comparison.</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE, a Boolean expression, is the Boolean value of an unsuccessful comparison.</td>
</tr>
<tr>
<td>Boolean-variable</td>
<td>Boolean-variable, a Boolean expression, has the value given it by an assignment statement or the value FALSE if not set.</td>
</tr>
</tbody>
</table>
Declarations

The FILTER declaration is the declaration for the entire filter, which includes options to declare a filter name, filter parameters, and Boolean variables. Boolean variables are the only variables that you can declare.

Some components of the FILTER declaration are described in separate subsections for quick reference.

Precedence of Operators

The operators following are listed in the order of their evaluation. The compiler evaluates comparisons first, then NOT, and so forth.

- Comparisons ( < <= > >= = <> )
- NOT
- AND
- OR
- Assignment ( := )

Table 5-3. Elements of Boolean Expressions (page 2 of 2) (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>(A AND B) is a Boolean expression that is TRUE if both A and B are TRUE. Otherwise (A AND B) is FALSE. (If A is FALSE, B is not evaluated.)</td>
</tr>
<tr>
<td>OR</td>
<td>(A OR B) is a Boolean expression that is TRUE if A is TRUE, if B is TRUE, or if both A and B are TRUE. Otherwise (A OR B) is FALSE. (If A is TRUE, B is not evaluated.)</td>
</tr>
<tr>
<td>NOT</td>
<td>(NOT A) is a Boolean expression that is TRUE if A is FALSE. Otherwise (NOT A) is FALSE.</td>
</tr>
<tr>
<td>( A )</td>
<td>A Boolean expression surrounded by parentheses is also a Boolean expression.</td>
</tr>
<tr>
<td>comparison</td>
<td>A comparison is a Boolean expression. For details, see Comparisons on page 5-19.</td>
</tr>
<tr>
<td>EMSTEXTMATCH function</td>
<td>EMSTEXTMATCH, a Boolean expression, is a Boolean function. For details, see EMSTEXTMATCH Function on page 5-39.</td>
</tr>
<tr>
<td>LITERALLY function</td>
<td>LITERALLY, a Boolean expression, is a Boolean function. For details, see LITERALLY Function on page 5-41.</td>
</tr>
<tr>
<td>MATCH function</td>
<td>MATCH, a Boolean expression, is a Boolean function. For details, see MATCH Function on page 5-42.</td>
</tr>
<tr>
<td>TOKENPRESENT function</td>
<td>TOKENPRESENT, a Boolean expression, is a Boolean function. For details, see TOKENPRESENT Function on page 5-43.</td>
</tr>
</tbody>
</table>
FILTER Declaration

The syntax for declaring a filter:

```
FILTER filter-name [ ( param [ , param ] ... ) ];

BEGIN [ SSID ( subsystem-id ) ]

[ variable-decls ]

statement [ ; statement ] ...

END
```

**filter-name**

is an identifier.

**param**

is the qualified name of a parameter token optionally followed by the keyword REQUIRED or by the keyword OPTIONAL. For the meaning of these keywords, see Filter Parameters on page 5-29.

**SSID ( subsystem-id )**

if present, gives the subsystem ID that changes the EMF default subsystem ID within the filter. If you omit this parameter, the NULL subsystem ID becomes the EMF default subsystem ID.

**variable-decls**

if present, is the keyword BOOLEAN, followed by a comma-separated list of identifiers.

**statement**

is any filter-language statement.

Considerations

- Each Boolean variable is set to FALSE before each event message is filtered. Therefore, results that must be saved across event messages must be saved in an application, not in a filter.
- You must declare each Boolean variable used.
- You must initialize every symbolic subsystem ID that you use in the filter. For information on initializing subsystem IDs, see The Filter Compiler on page 5-44 and Example 3 on page 5-28.
- You can declare a maximum of 50 Boolean variables.
Example 1
This filter selects event messages local to the \ABCD system:

```
FILTER forward^only^local^events;
BEGIN
  IF ZEMS^TKN^SYSTEM = [#SYSTEMNUMBER \ABCD] THEN PASS
  ELSE FAIL;
END;
```

Example 2
This filter selects event messages that have a ZEMS^TKN^ACTION^NEEDED token set TRUE, no matter which subsystem created the message. The #SET command just before the filter defines the subsystem ID for EMS, namely ZEMS^VAL^SSID, for use in the filter itself.

```
[#SET ZEMS^VAL^SSID
   [ZSPI^VAL^TANDEM].[ZSPI^SSN^ZEMS].[ZEMS^VAL^VERSION] ]
FILTER ftest;
BEGIN SSID ( ZEMS^VAL^SSID ) -- EMS is default subsystem ID
  IF ZEMS^TKN^ACTION^NEEDED = [ZSPI^VAL^TRUE] THEN PASS;
END; -- End of filter specification
```

Without the subsystem ID, the compiler would treat ZEMS^TKN^ACTION^NEEDED as a token of the subsystem that created the event message. That error could give unpredictable false positives.

Example 3
This filter selects certain DNS event messages. The #SET command just before the filter defines the subsystem ID for DNS, namely ZDNS^VAL^SSID, for use in the filter itself.

```
[ #SET ZDNS^VAL^SSID [ZSPI^VAL^TANDEM].[ZSPI^SSN^ZDNS].0 ]
FILTER DNS^NonStop;
BEGIN
  IF ZSPI^TKN^SSID = SSID ( ZDNS^VAL^SSID ) AND
      ( ZEMS^TKN^EVENTNUMBER = [ZDNS^EVT^CHECKPOINT^FAIL] OR
      ZEMS^TKN^EVENTNUMBER = [ZDNS^EVT^CHECKOPEN^FAIL] OR
      ZEMS^TKN^EVENTNUMBER = [ZDNS^EVT^CREATEBACKUP^FAIL] OR
      ZEMS^TKN^EVENTNUMBER = [ZDNS^EVT^BACKUP^DELETED] OR
      ZEMS^TKN^EVENTNUMBER = [ZDNS^EVT^RESYNC^FAIL] OR
      ZEMS^TKN^EVENTNUMBER = [ZDNS^EVT^TRAP] )
    THEN PASS;
END;
```

The TACL #SET command is necessary because subsystem IDs are not automatically initialized when the standard definition files are loaded.
Filter Parameters

Filter parameters let you specify the value of certain tokens when you install the filter, rather than when you compile the filter specification. The values of these parameter tokens cannot be changed while the collector or distributor is examining a series of event messages. Otherwise, filter parameters must be changed by object-oriented SPI command messages like ADD, ALTER, DELETE, or REPLACE. For more information, see Overview of Filter Operation on page 5-8.

Parameter tokens are known by name and cannot duplicate the names of tokens that you refer to in the event message.

The syntax to declare each filter parameter follows. For the larger context in which this declaration occurs, see FILTER Declaration on page 5-27.

```
param
  [ REQUIRED ]
  [ OPTIONAL ]
```

*param* is a token name that is qualified by the subsystem ID of the user subsystem. For details, see Qualified Tokens on page 5-12.

If you omit both the REQUIRED and OPTIONAL keywords, the parameter token is optional.

Parameter tokens are user-defined. When you declare parameters, you can but are not required to qualify the parameter tokens with the subsystem ID of the user subsystem. For details, see Qualified Tokens on page 5-12.

To refer to parameter tokens, use simple (unqualified) token names.

The EMF default subsystem ID has no effect on references to parameter tokens.

Considerations

- If a REQUIRED parameter token is missing, the collector or distributor issues the error ZEMS^ERR^REQ^PARAM: one or more filter parameter tokens are missing.
- In a comparison, a reference to a missing OPTIONAL parameter token is treated similarly to a reference to a missing event-message token. That is, the comparison is always FALSE if the parameter token is missing.
- You can include multiple instances of a token when you specify a set of parameter tokens. Use an index to refer to each instance within the filter.

Example 1

Some typical parameter specifications:
FILTER f ( SSID(myssid, MY^TKN^1) );
FILTER g ( SSID(myssid, MY^TKN^2) OPTIONAL );
FILTER h ( SSID(myssid, MY^TKN^3) REQUIRED );

Example 2
This example shows a reference to the MY^TKN^3 parameter token:

IF MY^TKN^3 > 131 THEN PASS 1;

MY^TKN^3 is qualified by myssid in the declaration but is unqualified in the comparison.

Statements
This subsection describes the filter language statements: the assignment, compound, destination, IF, PASS, and FAIL statements.

Statements are terminated by a semicolon.

Note the placement of semicolons in these IF statements:

IF flag1 THEN
BEGIN
PASS 1;
END;    -- End of first IF statement

IF flag2 THEN
BEGIN
PASS 2;
END     -- No semicolon here
ELSE
BEGIN
PASS 3;
END;    -- End of second IF statement

The second IF statement has no semicolon after the first compound statement because the IF statement does not end there but is followed by an ELSE statement.

EMF is a free-format language. A single statement can be on one or more lines. Two or more statements can be on one line.

Assignment Statements
The assignment statement stores the value of a Boolean expression in a declared variable.

\[
\text{variable} := \text{Boolean-expression}
\]

variable

is any variable that is declared in the filter.
**Boolean-expression**

is any valid Boolean expression.

**Considerations**

- Any variable not explicitly assigned the value TRUE is FALSE because variables are initialized to FALSE each time a new event message is examined.
- You cannot save Boolean values from one event message to the next. Variables are automatically initialized to FALSE each time a new event message is examined.
- Assignment statements can increase filter efficiency if the values saved in Boolean variables avoid redundant tests.

**Example**

In this example, an assignment statement sets the variable cflag TRUE if this is a critical event message:

```
FILTER atest;
BEGIN
  BOOLEAN cflag, vflag;
  cflag := ZEMS^TKN^EMPHASIS <> 0;   -- Critical event
 .
 .
 .
 IF cflag AND NOT vflag THEN PASS;  -- Tests flags
END;               -- End of filter specification
```

**Compound Statement**

The compound statement groups multiple statements together as a single statement. Compound statements can stand alone or serve as THEN or ELSE clauses of an IF statement.

If present, the subsystem ID specifies the EMF (EMS filter) default subsystem ID, which provides for implicit qualification of all unqualified names of event-message tokens within the compound statement. (Parameter tokens are unaffected by the EMF default subsystem ID.)

```
BEGIN [ SSID ( subsystem-id ) ] [ LIST ( ssid-token ) ]
[ statement ; ] ...
END
```

**SSID ( subsystem-id )**

if present, gives the new EMF default subsystem ID.
LIST ( ssid-token )

if present, allows the user to enter the list identifier by the ssid-token.

Considerations

- If you do not specify a subsystem ID on the compound statement, the subsystem ID in ZSPI^TKN^SSID is assumed. In such a compound statement, simple (unqualified) names of tokens represent tokens of the subsystem that generated the event message.

- Explicit qualification of a token name overrides the implicit qualification provided by a compound statement. For details, see Qualified Tokens on page 5-12.

Examples

- In this filter excerpt, the first compound statement leaves the EMF default subsystem ID unchanged. The second compound statement establishes EMS as the EMF default subsystem ID.

  IF flag THEN
  BEGIN
   .
   .
   .
  END;

  BEGIN SSID ( ZEMS^VAL^SSID )
   .
   .
   .
  END;

- This filter filters the events where error lists are present:

  BEGIN LIST (SSID(ZFIL^VAL^SSID,ZSPI^TKN^ERRLIST))
Destination Statement

The destination statement has been added to the filter source to describe a distributor’s routing configuration. Filters with destination statements can only be used in printing distributors. Multiple statements per filter are allowed.

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID n,</td>
<td>Routing ID : required</td>
</tr>
<tr>
<td>TYPE type,</td>
<td>Dest type : required</td>
</tr>
<tr>
<td>NAME fname,</td>
<td>Dest name : required</td>
</tr>
<tr>
<td>FORMAT ON</td>
<td>OFF ,</td>
</tr>
<tr>
<td>RECLEN n,</td>
<td>Desired rec len : optional</td>
</tr>
<tr>
<td>INDENT n,</td>
<td>Indentation : optional</td>
</tr>
<tr>
<td>OBJECT fname,</td>
<td>Program object : optional</td>
</tr>
<tr>
<td>IN fname,</td>
<td>IN file : optional</td>
</tr>
<tr>
<td>OUT fname,</td>
<td>OUT file : optional</td>
</tr>
<tr>
<td>STARTUP &quot;text&quot;,</td>
<td>Startup text : optional</td>
</tr>
<tr>
<td>PRI n,</td>
<td>Priority : optional</td>
</tr>
<tr>
<td>CPU n,</td>
<td>Cpu : optional</td>
</tr>
<tr>
<td>TIMEOUT sec</td>
<td>Timeout : optional</td>
</tr>
</tbody>
</table>

RID is the routing ID. It can be an integer in the range from 1 to 255. It need not be unique within one filter. That is, more than one destination can have the same routing ID. The RID is used in the PASS statement to indicate to which destination the event is to be routed.

NAME is the name of the routing destination (a local or remote name). The EMF (EMS filter) language requires file names to begin either with a dollar sign ($) or a backslash (\) character. If a node name is not given, the distributor's node is used later. To let the distributor's home terminal be specified as a destination, the name $HOME has been reserved. To avoid hard-coded file names, a DEFINE name can be given that is later resolved by the distributor.

TYPE is PROCESS | DISK | PRINTER | PRIMCOL | ALTCOL | CONSOLE

It is supplied for informational purposes only. It makes the filter easier to maintain. No checking is done by either the compiler or the distributor to verify this parameter.

FORMAT indicates if events for this destination should be formatted by EMSTEXT or submitted in binary form. The default is set to ON (formatted).

RECLEN for formatted events, indicates how many columns are to be formatted per line. The largest length accepted is 4096. If not specified, the device's record length is substituted.

INDENT specifies by how many columns lines are to be indented, excepting the first line of the event. This number must be smaller than RECLEN. Default is 36.
OBJECT is the program file name of the destination process. It must be fully qualified. If the node name is not given, the name of the distributor's node is substituted later. Currently, the EMF (EMS filter) language requires file names to begin with either a dollar sign ($) or a backslash (\) character. To avoid hard-coded file names, a DEFINE name can be given that is later resolved by the distributor. The destination type must be of type PROCESS although the compiler does not enforce this.

The parameters IN, OUT, STARTUP, PRI, and CPU are optional but can only be given if OBJECT is specified.

IN is the input file name of the destination process. It must begin with either a dollar sign ($) or a backslash (\) character. If a node name is not given, the distributor's node is used later. To avoid hard-coded file names, a DEFINE name can be given that is later resolved by the distributor. OBJECT must be specified. If omitted, the distributor's IN file is used.

OUT is the output file name of the destination process. It must begin with either a dollar sign ($) or a backslash (\) character. If a node name is not given, the distributor's node is used later. To avoid hard-coded file names, a DEFINE name can be given that is later resolved by the distributor. OBJECT must be specified. If omitted, the distributor's OUT file is used.

STARTUP is the startup sequence that is sent to the destination process. It must be text embedded within quotes. Current maximum length is set to 128 characters. OBJECT must be specified.

PRI is the priority of the destination process. If not specified, the priority of the distributor is inherited. OBJECT must be given.

CPU is the CPU number for the destination process. If not specified or not available, the distributor picks the next available CPU, excluding its own. OBJECT must be given.

TIMEOUT is the maximum time in seconds that this destination process should be allowed to process an event before sending a response to the distributor. The distributor selects the largest timeout value from all current destinations and waits for that amount of time for any destination to respond to a write request.

Considerations
For details, see Section 16, Event Routing.

Example

DESTINATION
RID 1,
TYPE PROCESS,
NAME =_fault_analyzer,
FORMAT OFF,
OBJECT =_ANALYZER_OBJ,
STARTUP 'type x',
CPU 5
DISCARDEVENT Statement

The DISCARDEVENT statement stops filter execution immediately and rejects the current event message in any compiled filter, whether it is the only filter present or any filter in a multiple-filter environment. Unlike the FAIL statement, DISCARDEVENT rejects the message outright and does not pass it on to any other filters.

Example

This filter fragment makes use of a DISCARDEVENT statement and an implicit FAIL statement:

```
IF fflag THEN DISCARDEVENT;
IF qflag THEN PASS;
-- If control reaches this point, an implicit FAIL
-- statement is executed.
END  -- End of filter specification
```

FAIL Statement

The FAIL statement stops filter execution immediately and rejects the current event message if the compiled filter is the only filter present or is the last filter in a multiple-filter environment. In all other cases, the current event message is passed on to the next filter within that collector or distributor. This strategy is different from that of the DISCARDEVENT statement which, upon rejection of the current event message by the filter, rejects the event message without passing it on to the next filter.

Considerations

- If the last executable statement of the filter is executed without executing either a PASS or FAIL statement, the event message is rejected. That is, a FAIL statement is implicitly present as the last statement of the filter.

Example

This filter fragment makes use of both explicit and implicit FAIL statements:

```
FAIL
```
IF fflag THEN FAIL;
IF qflag THEN PASS;
-- If control reaches this point, an implicit FAIL
-- statement is executed.
END  -- End of filter specification

**IF Statement**

The IF statement provides for conditional execution of a statement that follows THEN and a statement that follows ELSE.

```
IF expression THEN statement-1 [ ELSE statement-2 ]
```

- **expression**
  - is any Boolean expression.

- **statement-1**
  - is any statement. **statement-1** is executed only if **expression** is TRUE.

- **statement-2**
  - is any statement, simple or compound. If present, **statement-2** is executed only if **expression** is FALSE.

**Example**

This filter fragment shows IF statements with and without ELSE clauses:

```
.
.
.
IF flag1 THEN PASS 1;
IF flag2 THEN PASS 2 ELSE PASS 3;
END;  -- End of filter specification
```

**PASS Statement**

In a compiled filter loaded into a collector, the PASS statement directs the collector to log the current event message. (The pass value is ignored.) In a compiled filter loaded into a distributor, the PASS statement directs the distributor to print, forward, or return the current event message (and pass value, if any), depending on whether the distributor is a printing, forwarding, or consumer distributor, respectively. The current execution of the filter then terminates.
The filter PASS statement has been enhanced to allow an optional list of routing IDs, in addition to the value that can currently be returned. A routing distributor also returns the PASS value itself to the destination if the event is unformatted, which was previously done for consumer distributors only. In the case of multiple filters, each filter can send a different PASS value to its destinations.

The syntax of the PASS statement:

```
PASS [ n ] [ m ]  |  PASS [ n ] ( m1, m2, ... mN )
```

\( n \)

is any signed integer constant whose value is expressible as a 16-bit signed integer. Include this value to indicate to your application which PASS statement returned a particular event message. The pass value allows you to categorize event messages to prevent redundant computations in your application.

\( m, m1...mN \)

are routing IDs as defined in the DESTINATION statements.

The PASS statement causes a consumer distributor to return the following tokens in its response message to a GETEVENT command message sent by an application:

- **ZEMS^TKN^EVENT** is a type `bytestring` token that contains the entire event message that the PASS statement selected. The distributor always returns this token when a PASS statement is executed.

- **ZEMS^TKN^PASSVAL** is a type `int` token. The distributor returns this token only if an `integer-value` follows the PASS keyword.

**Examples**

This filter fragment shows a PASS statement that returns values 1 or -1, depending on the Boolean variable qflag:

```
.  
.  
.  

IF qflag THEN
   PASS 1
ELSE
   PASS -1;
END;  -- End of filter specification
```
This filter fragment shows a PASS statement that routes the event to destinations 1 and 2 (and no pass value returned):

```plaintext
IF qflag THEN
   PASS ,1,2;
```

**Functions**

The filter language includes Boolean functions DECOMPOSE, DECOMPOSEERROR, EMSTEXTMATCH, FILENAMECOMPARE, LITERALLY, MATCH, TOKENPRESENT, and the SSID function.

**DECOMPOSE Function**

This function returns the parts of a 12-word internal-format file name, a file-name string, or a process handle. The file name can be either a disk file name or a process file name.

### DECOMPOSE (value, p-list)

- **value**
  - is the file name or process handle to be decomposed. The data type for value can be a string or a process handle.

- **p-list**
  - describes the values to be produced by the decomposition. p-list can contain one or more of these values, separated by commas:

  - SYSTEM NAME: System name of process handle or file name
  - SYSTEM NUMBER: System number of process handle or file name
  - DESTINATION NAME: Device, process, or ldev of a file name
  - FIRST QUALIFIER: First qualifier of a file name
  - SECOND QUALIFIER: Second qualifier of a file name
  - CPU PART: CPU number for process handles, unnamed processes
  - SEQNO PART: Sequence number portion
  - PIN PART: Process identification number portion
  - NAME PART: Name portion only
DECOMPOSEERROR Function

This function returns the error resulting from the most recently executed DECOMPOSE function if an error occurred. A 0 is returned if no error occurred.

```
DECOMPOSEERROR
```

Examples Using DECOMPOSE and DECOMPOSEERROR

This statement retrieves all events generated by processes on `\NCN`:

```
IF DECOMPOSE (ZEMS^TKN^PROC^DESC, SYSTEM NAME)="\NCN"
   AND DECOMPOSEERROR = 0 THEN
   PASS;
```

This statement retrieves all events generated by any process named `$null`:

```
IF DECOMPOSE (ZEMS^TKN^PROC^DESC, DESTINATION NAME, NAME PART)='$null'
   AND DECOMPOSEERROR = 0 THEN
   PASS;
```

EMSTEXTMATCH Function

The EMSTEXTMATCH function compares the text of an event message, as formatted by EMSTEXT, against a specified template. EMSTEXTMATCH returns TRUE if the text generated by EMSTEXT matches the template.

```
EMSTEXTMATCH ( value, paramtkn )
```

`value`

is the initial template to be supplied to EMSTEXT. `value` must be either a constant or a parameter token whose contents conform to the header-template-key parameter for EMSTEXT. For more information about the header-template-key parameter, see the EMSTEXT procedure material in Section 15, EMS Procedures.

`paramtkn`

is a parameter token passed to the distributor, containing the template against which the text generated by EMSTEXT is to be compared. The template can contain optional wild-card characters. A question mark (?) matches any single character. An asterisk (*) matches zero or more characters.
Considerations

- Use this function with caution. If possible, filtering should always be done by tokens. Use the EMSTEXTMATCH function only if no other way is possible (for example, for event messages that contain text instead of tokens). EMS is moving toward using tokens instead of text in event messages. Using text increases maintenance and reduces performance.

- When you use EMSTEXTMATCH, you must consider errors that might occur when the distributor calls EMSTEXT to generate the text needed for the pattern match operation. Possibly the distributor will access a different set of event templates than you expected, and therefore the text returned by EMSTEXT will be different than you expected. This would occur if the operating system version (and thus the content of the template file) changed between when the filter was written and when it was executed.

- Two cases in which the EMSTEXTMATCH function returns an error are:
  - When EMSTEXT returns an error
  - When TEXTMATCH fails

Example

If a text item in the event formatted according to init^template^key matches exactly the text string in the parameter token, then pass this event message:

IF EMSTEXTMATCH (init^template^key, zvpt^fltr^eventtext) THEN
  PASS

- A TACL Macro "INSERTCOMMAS" can be used with EMSTEXTMATCH function to avoid error regarding size of the map structure. The macro can be defined in the following way:

  [#DEF InsertCommas MACRO|BODY|[#DELTA /COMMANDS
  ed^InsertCommas/%*%]]
  [#DEF ed^InsertCommas DELTA|BODY|J:<:S $;-DI,$>]

Example

EMSTEXTMATCH can be used in the following way with InsertCommas:

IF EMSTEXTMATCH
  (([InsertCommas[zems^ddl^inittemplate]]),zems^tkn^text) THEN
  PASS;
FILENAMECOMPARE Function

This function compares two values to determine whether they refer to the same object. FILENAMECOMPARE returns TRUE if the values represent the same object.

FILENAMECOMPARE ( value , value )

value

is a token name representing a D-series file name.

This statement passes all events generated by a particular instance of $null:

IF FILENAMECOMPARE (ZEMS^TKN^PROC^DESC, \\ncn.$null*) THEN
PASS;

This test succeeds only if $null is still running with the same sequence number contained in the event message:

IF FILENAMECOMPARE (ZEMS^TKN^PROC^DESC, \\ncn.$null:430*) THEN
PASS;

LITERALLY Function

The LITERALLY function forces case-sensitive evaluation of string or MATCH-function comparisons within a Boolean expression, contrary to normal comparison conventions.

The value of LITERALLY, TRUE, or FALSE depends on the evaluation of its argument: a Boolean expression.

LITERALLY ( Boolean-expression )

Boolean-expression

is any valid Boolean expression.

Example

These Boolean expressions are all TRUE:

"This is IT" = "This is it"

LITERALLY ( "This is IT" = "This is IT")

LITERALLY ( "This is IT" <> "This is it")
MATCH Function

The MATCH function determines whether the value of a specified token matches the pattern specified by a template. MATCH returns TRUE if the value of the token (or field) matches the template. MATCH comparisons are case insensitive as are string comparisons. For more information, see Comparing Strings on page 5-22.

```
MATCH ( token-specifier, template )
```

token-specifier

is the name of any token or field whose value type is string or file name.

template

is a quoted string that contains the characters—and wild-cards asterisk and question mark—to compare with token-specifier. A question mark (?) matches any single character in token-specifier. An asterisk (*) matches zero or more consecutive characters in token-specifier.

Considerations

- The length of the token specifier should always be fewer than 512.
- If token-specifier is a file name, template matches the external form of the file name, as shown in the example. You can use either network or local file names.
- There is no way of representing an asterisk or question mark literally within a template. They are exclusively wild cards.

Example

If the event message contains a ZFIL^TKN^FILENAME token and the token value is a file name of a file on $SYSTEM.SYSTEM, the following PASS statement is executed:

```
IF MATCH (ZFIL^TKN^FILENAME, "$SYSTEM.SYSTEM.*") THEN PASS;
```

SSID Function

The SSID function returns the first two parts (organization and subsystem number) of a subsystem ID. The return value is the internal form of an SSID that has been truncated to exclude the version.

Use this function to ignore the version when comparing subsystem IDs.

The syntax for the SSID function is:

```
SSID ( subsystem-id )
```
subsystem-id

is a subsystem ID, expressed as a name or as a constant.

Consideration

Subsystem IDs are treated in comparisons as the EMF (EMS filter) unsigned values, and unsigned values of unequal length are compared for the length of the shorter value. Therefore, to compare a subsystem ID with the SSID function of another subsystem ID ignores the version in the comparison.

Example

This example shows the use of an SSID function in a comparison:

```
[#SET ZTMF^VAL^SSID
 [ZSPI^VAL^TANDEM].[ZSPI^SSN^ZTMF].[ZTMF^VAL^VERSION] ]
```

FILTER tmf_test;
BEGIN
IF ZSPI^TKN^SSID = SSID ( ZTMF^VAL^SSID ) THEN
    BEGIN
        .
        .
    END;
END;

Statements within the compound statement in the previous example are executed only if TMF generated the event message, regardless of the version of the programmatic interface.

TOKENPRESENT Function

The TOKENPRESENT function determines whether a specified token is currently present and returns TRUE or FALSE.

```
TOKENPRESENT ( argument )
```

argument

is the name of a token or field.

Considerations

- As usual, if the token name is declared as a filter parameter, TOKENPRESENT looks for the token among the set of parameter tokens. Otherwise, TOKENPRESENT looks for the token in the event message.
- You can use TOKENPRESENT to determine whether an OPTIONAL parameter was actually passed.
Example

This filter executes the compound statement only if the application actually passed the MY^TKN^STIME parameter token to the distributor:

```
[ #DEF myssid STRUCT LIKE ZSPI^DDL^SSID; ]
[ #SET myssid [myorg].[mynum].[myver] ]

FILTER ptest( SSID ( myssid, MY^TKN^STIME ) OPTIONAL );
BEGIN
  IF TOKENPRESENT ( MY^TKN^STIME) THEN
    BEGIN
      .
      .
      END;
  END;
```

The Filter Compiler

This subsection describes how to compile a filter specification. It describes the EMF (Event Management Service filter) language compilation command, compiler directives, format of compiler input and output, compiler errors, and compiler completion-codes.

The compiler generates a filter object program, which can be loaded later by operator or program into a collector or distributor.

For an introduction to the filter language and the filter compiler used by compiled filters, see Writing and Compiling Filters on page 5-1. This subsection and The Filter Language on page 5-7 cover these topics in more detail.

Note. Because of the interface of the filter language to TACL, symbolic names in this section are in TACL form, using circumflex (^) symbols rather than hyphens.

Compiler Input and Output

The filter compiler reads language text from an EDIT file that contains a filter specification or accepts statements interactively from a terminal.

TACL is used as a preprocessor of the language source text. Therefore, any text that is input to the compiler (filter statements or compiler directives) is interpreted first by TACL and must therefore be acceptable to TACL. This leads to some restrictions on the usage of characters that are special to TACL as follows.

Follow TACL rules if you use TACL special characters in the way TACL expects, such as using brackets ([]) to request macro expansion. For information on using TACL special characters in other ways, see Escape Character (~) on page 5-9.

TACL expands text enclosed in brackets ([]) before the filter compiler sees it.
Before compiling a filter, you must load the standard definitions for SPI and for any subsystem whose names you use in the filter. Loading standard definitions defines the names for a subsystem: names of tokens, names of values, and most other names that you might need.

Names of subsystem IDs are defined but not initialized by loading the standard definition files. You can place TACL commands at the beginning of a filter (in brackets) to initialize subsystem IDs.

This SET command initializes the subsystem ID for the TMF subsystem:

```
[SET ZTMF^VAL^SSID
  [ZSPI^VAL^TANDEM].[ZSPI^SSN^ZTMF].[ZTMF^VAL^VERSION] ]
```

All standard definition files reside in the $SYSTEM.ZSPIDEF subvolume or can be placed on another volume at the choice of the operations staff at your site. For more information on standard definition files, see the SPI Programming Manual.

You must load the standard definitions of SPI and of each subsystem whose tokens you use. These TACL statements load the standard definitions for SPI and EMS:

```
#PUSH dummy
#LOAD /KEEP 1, LOADED dummy/ $SYSTEM.ZSPIDEF.ZSPITACL
#LOAD /KEEP 1, LOADED dummy/ $SYSTEM.ZSPIDEF.ZEMSTACL
#POP dummy
```

If your site places the standard-definition files on another volume, change $SYSTEM to the other volume name in these statements.

For information on how to create and attach a TACL segment file, see the TACL Reference Manual. It provides a quick way to save and restore standard definitions.

Loading standard definitions defines the names of tokens and many other definitions that you need.

### Compiler Directives

The compiler supports the LIST, NOLIST and SOURCE directives. Each directive must begin with a question mark (?) in the first character position and must be on a line by itself.

#### LIST Directive

The LIST directive causes current source statements to be included in the compiler output file. For details, see also the NOLIST directive.

```
?LIST
```
**NOLIST Directive**

The NOLIST directive causes current source statements to be excluded from the compiler output file. For details, see also the LIST directive.

```plaintext
?NOLIST
```

**SOURCE Directive**

The SOURCE directive causes the compiler to include language text from the specified file.

```plaintext
?SOURCE file-name
```

*file-name*

is a file name of an EDIT file (file code 101) with language text.

**Consideration**

Text to be expanded by TACL (that is, text within brackets) must be wholly contained in one file because TACL expansions for the whole file are performed before any SOURCE directives take effect.

**Compiler Invocation**

The compiler reads a filter specification in an EDIT file and produces a filter object file.

To run the compiler from TACL:

```plaintext
EMF / [ { IN } source ]
     { INV }
[ , { OUT } listing ]
     { OUTV }
[ , run-option ] ...
/ [ object-file ]
```

*source*

is the filter specification. If *source* is a file name, use the IN keyword. If *source* is a TACL variable name, use the INV keyword.

*listing*

is the listing destination. If *listing* is a file name, use the OUT keyword. If *listing* is a TACL variable name, use the OUTV keyword.
run-option

is CPU, LIB, MEM, PRI, STATUS, SWAP, or TERM. For more information, see the RUN command in the TACL Reference Manual.

object-file

is the file name to contain the compiled filter. The default object file name is FOBJECT, on the current volume and subvolume.

Considerations

- If you omit the IN and INV parameters, the compiler prompts the TACL IN file—typically a terminal—interactively for the filter specification.
- If you omit the OUT and OUTV parameters, the compiler sends its listing to the TACL OUT file—typically a terminal.
- Filter object files have file code 845.
- The compiler does not support the NOWAIT option. If you want NOWAIT compilation, start a background TACL process. Use the background TACL, rather than your interactive TACL, to compile the filter.
- While trying to create a new object file, if the compiler finds an old object file with the same name and file code, the new file replaces the old file.
- When creating an object file, if the compiler cannot use the indicated file name, it uses the name ZZEFnnnn (in which nnnn are four digits) and issues a warning message.

Compiler Errors and Warnings

Compiler errors are classified as warnings, errors, or fatal errors:

- Warning messages, which begin with *** Warning ***, describe a program construct that is either incorrect or atypical. They help you identify sources of potential trouble.
- Error messages, which begin with *** Error ***, describe an incorrect construct. After finding an error of this type, the compiler continues to run and to present other error messages if appropriate.
- Fatal-error messages, which begin with *** Fatal ***, describe situations originating with the compiler in which the compiler cannot continue. You must correct the problem and recompile to check that all compilation errors are found.
- Messages that begin with *** EMF:, are fatal-error messages that originate with TACL.
Compiler Warning Messages

This subsection describes warning messages that indicate potential problem areas of your source filter and that begin with *** WARNING ***.

The compiler inserts a circumflex (^) in your listing to mark the item the message describes.

Each of these message text boxes contains the exact text of one warning message. The text following each box describes the cause, effect, and recovery for that message.

<table>
<thead>
<tr>
<th>Warning Message</th>
<th>Cause</th>
<th>Effect</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>*** Warning *** All occurrences of this token will be legal</td>
<td>An index within a parameter declaration is not legal.</td>
<td>None. The compiler ignores it. Within the filter body, you can refer to any parameter token as a token array.</td>
<td>None needed.</td>
</tr>
<tr>
<td>*** Warning *** Bad object code filename, default used</td>
<td>The name of the object file given on the EMF command is invalid.</td>
<td>The compiler attempts to name the file FOBJECT (on the current default volume and subvolume).</td>
<td>None needed.</td>
</tr>
<tr>
<td>*** Warning *** Could not name object file obj-name</td>
<td>The compiler could not name the object file obj-name. Typically, a file with that file name already exists but does not have a file code of 845.</td>
<td>The compiler names the file ZZEFnnnn on the same volume and subvolume.</td>
<td>None needed.</td>
</tr>
<tr>
<td>*** Warning *** Data type coercion necessary; { unsigned } value will be treated as { signed } { string }</td>
<td>The data type of value is not fully supported by the compiler. For example, the programmatic interface has single and double precision floating-point numbers, which the compiler does not support.</td>
<td>value is treated as specified in the message (either unsigned, signed, or string).</td>
<td></td>
</tr>
</tbody>
</table>
Recovery. None needed.

```plaintext
*** Warning *** Failed to rename old-file to new-file
            (error err)
```

Cause. The compiler received error `err` while attempting to rename a file but was able to recover.

Effect. The file is not renamed, but the compiler continues.

Recovery. None needed.

```plaintext
*** Warning *** Invalid compiler command - command
```

Cause. The indicated compiler directive does not exist or is misspelled.

Effect. The command is not executed, but the compiler continues.

Recovery. None needed.

```plaintext
*** Warning *** Invalid file name
```

Cause. The indicated file name contains a syntax error.

Effect. The command referencing the file name does not execute, but the compiler continues.

Recovery. None needed.

```plaintext
*** Warning *** Invalid source file name
```

Cause. The file name in a `?SOURCE` compiler directive is incorrect.

Effect. The directive is not carried out, but the compiler continues.

Recovery. None needed.

```plaintext
*** Warning *** Renamed old-file to new-file
```

Cause. The compiler could not use the specified object filename `old-file`, so it renamed the object file to ZZEFnnnn `new-file`.

Effect. The file is renamed and the compiler continues.

Recovery. None needed.

```plaintext
*** Warning *** This relational operator won't work on file names
```

Cause. The indicated comparison is not defined for file names. You can use only `=` or `<>` in file name comparisons.
**Effect.** The compiler performs the comparison as if the names were string data of the same length.

**Recovery.** None needed.

**Compiler Error Messages**

This subsection describes error messages that represent source-filter errors and that begin with *** ERROR ***.

After presenting one of these errors, the compiler continues to run and to diagnose any further problems.

The compiler inserts a circumflex (^) in your listing to mark the problem the message describes.

Each of these message text boxes contains the exact text of one error message. The text following each box describes the cause, effect, and recovery for that message.

*** Error *** Bad field name: EMF TACL macro, internal error.

**Cause.** An internal error occurred.

**Effect.** The compiler continues to its next task.

**Recovery.** Report this internal error to your service provider.

*** Error *** Bad field name: erroneous information returned from TACL

**Cause.** The compiler received invalid information from TACL.

**Effect.** The compiler continues to its next task.

**Recovery.** Report this internal error to your service provider.

*** Error *** Bad field name: field type is not a legal ZSPI data type

**Cause.** You cannot use the indicated field data type, which is associated with a subcomponent of a structured token.

**Effect.** The compiler continues to its next task.

**Recovery.** Use a legal ZSPI data type.

*** Error *** Bad field name: no such field or index out-of-range
**Cause.** The indicated field name (the name of a subcomponent of a structured token) is undefined or has an inappropriate array index. If the array index is out of range, the circumflex (^) marks the incorrect index. A field without an index has an assumed index of zero, by a TACL rule. If you declare x(1:5) and then refer to x, you receive this error message.

**Effect.** The compiler continues to its next task.

**Recovery.** Verify that you have correctly entered a valid field or index.

```
*** Error *** Bad field name: no such TACL variable or not type STRUCT
```

**Cause.** The indicated variable is undefined or is not a field of type STRUCT.

**Effect.** The compiler continues to its next task.

**Recovery.** Use either a defined variable or a field of a STRUCT.

```
*** Error *** Bad field name: TACL variable must be type STRUCT
```

**Cause.** The indicated TACL variable should be a STRUCT variable rather than a text variable.

**Effect.** The compiler continues to its next task.

**Recovery.** Check that you use a STRUCT variable, not a text variable.

```
*** Error *** Bit extraction is illegal for this data type
```

**Cause.** You cannot perform bit extraction on the indicated token (or constant) because the bit extraction is not defined for this data type. Bit extraction is defined for data types of signed or unsigned integers of 1, 2, 4, or 8-bytes.

**Effect.** The specified bit extraction is not executed.

**Recovery.** Perform bit extraction only on tokens it is defined for, specifically data types of signed or unsigned integers of 1, 2, 4, or 8-bytes.

```
*** Error *** Can't source file, error err on open
```

**Cause.** A file named in a compiler SOURCE directive cannot be opened by the OPEN procedure.

**Effect.** The compiler SOURCE directive is not executed.

**Recovery.** Check that the file is properly secured and is accessible by the current user.

```
*** Error *** Can't source file, file code code
```
Cause. A file named in a compiler SOURCE directive has an invalid file code. The file should be an EDIT file.

Effect. The compiler SOURCE directive is not executed.

Recovery. Check that your compiler SOURCE directive refers to an EDIT file.

*** Error *** Currently not supported: feature

Cause. The indicated feature is not supported by this version of the compiler.

Effect. The unsupported feature is not executed.

Recovery. Either use only the features in your version of the compiler, or update your compiler.

*** Error *** Data types differ for this comparison

Cause. The indicated relational operator compares values with incompatible data types.

Effect. The comparison is not executed.

Recovery. Check that the values for comparison are of compatible data types.

*** Error *** Dup attribute: Duplicate DESTINATION attribute

Cause. The parser detected an incompatibility in the DESTINATION statement.

Effect. The DESTINATION statement is not executed.

Recovery. Check that you do not duplicate attributes within the same DESTINATION statement.

*** Error *** Dup destination: Duplicate DESTINATION name

Cause. The parser detected an incompatibility in the DESTINATION statement.

Effect. The DESTINATION statement is not executed.

Recovery. Check that you do not specify another DESTINATION with the same name.

*** Error *** End of field lies outside any event message

Cause. The size and offset of the indicated field is too large (greater than 4096-bytes) for an event message.

Effect. The command is not executed.

Recovery. Check that fields for event messages contain fewer than 4096-bytes.

*** Error *** End of field lies outside this token's value
**Cause.** The size and offset of the indicated field is too large to be the value of the indicated token. The compiler cannot detect this condition if the token is of variable length and not extensible.

**Effect.** The command is not executed.

**Recovery.** Check that the end of field falls within the token’s value by modifying the field or the token appropriately.

### *** Error *** Expected a Boolean variable

**Cause.** The indicated entity is the name of a token or the name of a subsystem ID rather than the name of a Boolean variable as expected.

**Effect.** The command is not executed.

**Recovery.** Check that Boolean variables are provided to the compiler when needed.

### *** Error *** Expected a subsystem descriptor name

**Cause.** The indicated name is undefined or not of the correct form.

**Effect.** The command is not executed.

**Recovery.** Check that you reference a defined, correct subsystem descriptor name.

### *** Error *** Expected a valid token: bad map (error err on SSNULL)

**Cause.** The indicated token should be extensible, but the map was rejected by the SSNULL procedure with *err*.

**Effect.** The command is not executed.

**Recovery.** For a description of SSNUL, see the *SPI Programming Manual*.

### *** Error *** Expected a valid token: data type is map but none was given

**Cause.** The token data type indicates that the token is an extensible one, but there was no accompanying map structure.

**Effect.** The command is not executed.

**Recovery.** Provide a map structure to accompany any extensible tokens.

### *** Error *** Expected a valid token: map structure is too large

**Cause.** The token is used as an extensible token, but the size of the map information is greater than 4000-bytes.
**Effect.** The command is not executed.

**Recovery.** Check that your map information does not exceed 4000-bytes.

```
*** Error *** Expected a valid token: not a valid SPI data type
```

**Cause.** The data type of the token code is not one of those defined by the programmatic interface.

**Effect.** The command is not executed.

**Recovery.** Use SPI-defined data types for token codes. (For details, see the SPI Programming Manual.)

```
*** Error *** Expected a valid token: variable does not exist
```

**Cause.** The indicated name is unknown to TACL.

**Effect.** The command is not executed.

**Recovery.** Check that the referenced token exists and is entered correctly.

```
*** Error *** Expected a valid token: variable expanded to an incorrect form
```

**Cause.** Although the variable exists, it does not represent a legal token code in the programmatic interface.

**Effect.** The command is not executed.

**Recovery.** Check that you use valid tokens where needed.

```
*** Error *** Expected the name of a token
```

**Cause.** You supplied a non-token identifier where a token name is expected. You get this message if you use a Boolean variable or subsystem descriptor name where a token name is expected.

**Effect.** The command is not executed.

**Recovery.** Use a token name instead of a Boolean variable or subsystem descriptor name.

```
*** Error *** Field access does not begin at a word boundary
```

**Cause.** The field, which requires an even byte offset, has an odd byte offset. All fields that are not byte-oriented must have an even byte offset. TACL performs this alignment automatically for its structures.
Effect. The command is not executed.

Recovery. Check that the field has an even byte offset.

*** Error *** Field names must be less than 255 characters

Cause. The name of the indicated field is longer than 255 characters.

Effect. The command is not executed.

Recovery. Restrict the length of referenced field names to no more than 255 characters.

*** Error *** Format: Indent or reclen not allowed if format off

Cause. The parser detects an incompatibility in the DESTINATION statement.

Effect. The command is not executed.

Recovery. Check that the format is on (if required) for the specified command.

*** Error *** Identifier too long

Cause. This identifier has more than 30 characters.

Effect. The command is not executed.

Recovery. Try to restrict the length of your identifiers to no more than 30 characters to ensure they can be distinguished from each other.

*** Error *** Illegal bit extraction bounds

Cause. The user specified a bit range for a bit extraction that is not allowed for the length of the data type accessed.

Effect. The bit extraction is not executed.

Recovery. Check that you specify a valid bit range for bit extraction.

*** Error *** Indent too big: Indentation too large

Cause. The parser detected an incompatibility in the DESTINATION statement.

Effect. The command is not executed.

Recovery. Check that the indentation specified in the command is valid.

*** Error *** Invalid character

Cause. The indicated character occurs outside a string constant.
Effect. The command is not executed.

Recovery. Check the syntax of your command and remove the indicated character.

```
*** Error *** Invalid syntax - continuing
```

Cause. The indicated statement has an error in syntax.

Effect. The compiler tries to recover by ignoring one or more lexical tokens, but might not be able to do so.

Recovery. Check the statement for syntax errors, and fix any that exist.

```
*** Error *** Invalid syntax - end of input
```

Cause. The last part of the filter specification is missing or incorrect. You might get this message if you press the BREAK key before compilation is complete.

Effect. The command is not executed.

Recovery. Check the syntax of your commands, or determine what otherwise caused the early termination.

```
*** Error *** Must have object: Must specify a program object name OBJECT
```

Cause. The parser detects an incompatibility in the DESTINATION statement.

Effect. The command is not executed.

Recovery. Check that a program object name is specified.

```
*** Error *** Name missing: Must specify a destination name NAME
```

Cause. The parser detects an incompatibility in the DESTINATION statement.

Effect. The command is not executed.

Recovery. Check that a destination name is specified.

```
*** Error *** RID missing: Must specify a routing ID RID
```

Cause. The parser detects an incompatibility in the DESTINATION statement.

Effect. The command is not executed.

Recovery. Check that a routing ID is specified.

```
*** Error *** RID not defined: Routing ID not previously defined
```
**Cause.** The parser detects a routing ID specification in a PASS statement, and the RID was not defined in a previous DESTINATION statement.

**Effect.** The command is not executed.

**Recovery.** Check that the specified routing ID is defined in a previous DESTINATION statement.

*** Error ***  Startup too big: Startup message is too long

**Cause.** The parser detects an incompatibility in the DESTINATION statement.

**Effect.** The command is not executed.

**Recovery.** Check that your startup message is an acceptable length.

*** Error ***  This data type is illegal
(as a MATCH operand)

**Cause.** The data type of the indicated operand is invalid. You can receive this message in two different contexts:

- With the Match function, you used a type other than file name or string.
- You used a token with no associated value, such as ZSPI^TKN^ENDLIST, where a value is needed.

**Effect.** The command is not executed.

**Recovery.** Check that data types in your command are legal.

*** Error ***  This is not a legal subsystem descriptor

**Cause.** The indicated constant should be of the form owner.integer.integer, where owner is 8 characters or less, and each integer is in the range of 0 through 65535.

**Effect.** The command is not executed.

**Recovery.** Check that the constant is of the form owner.integer.integer, where owner is 8 characters or less in length and each integer is in the range of 0 through 65535.

*** Error ***  This number must be between 0 and 255, inclusive

**Cause.** The indicated number is outside the allowable range.

**Effect.** The command is not executed.
Recovery. Check that the indicated number falls in the range from 0 to 255.

*** Error *** This token was not declared with this SSID

Cause. The indicated token was declared to be a parameter token, but was also declared to be associated with a different subsystem descriptor than it is now used with. A parameter token can be used with only one subsystem descriptor.

Effect. The command is not executed.

Recovery. Check that you use only tokens that are declared with the appropriate SSID.

*** Error *** Too many Boolean variables have been declared

Cause. The compiler sets a limit of 50 Boolean variables.

Effect. The command is not executed.

Recovery. Declare no more than 50 Boolean variables to the compiler.

*** Error *** Too many dest stmts: Max number of DESTINATION statements exceeded

Cause. Too many DESTINATION statements were specified.

Effect. The command is not executed.

Recovery. Do not exceed the maximum number allowed for DESTINATION statements.

*** Error *** Too many RIDs: Max number of routing IDs exceeded

Cause. The parser encountered too many RIDs specified in the PASS statement.

Effect. The command is not executed.

Recovery. Do not exceed the maximum number allowed for RIDs.

*** Error *** Type missing: Must specify a destination type TYPE

Cause. The parser detects an incompatibility in the DESTINATION statement.

Effect. The command is not executed.

Recovery. Check that a destination type is specified.

*** Error *** Undeclared Boolean variable
**Cause.** An identifier, assumed to be a Boolean variable, was used but not previously declared. The compiler automatically declares it to avoid duplicate messages.

**Effect.** The command is not executed.

**Recovery.** Check that all used Boolean variables have been declared.

```
*** Error *** Untermminated string
```

**Cause.** A string constant is missing its final double quote, for example “xyz.

**Effect.** The command is not executed.

**Recovery.** Check that to close the double quotes on all string constants.

```
*** Error *** Variable has already been declared
```

**Cause.** The indicated Boolean variable has been declared twice.

**Effect.** The command is not executed.

**Recovery.** Check that you declare each Boolean variable only once.

### Compiler Fatal-Error Messages

This subsection describes fatal-error messages that begin with *** FATAL ***. The compiler cannot recover from the situations described by these messages because compiler execution terminates abnormally (with an ABEND). Any errors that follow the fatal error go undetected in this compilation but are found on recompilation.

Each of these message text boxes contains the exact text of one fatal-error message. The text following each box describes the cause, effect, and recovery for that message.

```
*** Fatal *** A rename failed, the Guardian error is given by err.
```

**Cause.** The compiler received an error on the temporary object file and cannot recover.

**Effect.** The compile terminates.

**Recovery.** Determine and fix the cause of the specified Guardian error.

```
*** Fatal *** Cannot run on this version of Guardian
```

**Cause.** You cannot use the EMF compiler on a version of the operating system earlier than B00 (including TNS 1).

**Effect.** The compiler does not run.
Recovery. Use a more recent version of the operating system (B00 and later).

*** Fatal *** Can't make a new ZZEFnnnn file name

Cause. The compiler cannot successfully create the specified file name.

Effect. The compile terminates.

Recovery. Report this internal error to your service provider.

*** Fatal *** Exceeded available parameter storage space

Cause. Too many parameter tokens are declared, using more than 32000-bytes of storage. The amount of storage needed is four bytes per simple token plus the size of the map for extensible tokens. The whole filter cannot exceed 65535-bytes.

Effect. The compile terminates.

Recovery. Check that your parameter tokens do not exceed the stated storage capacity.

*** Fatal *** Internal error, please report to Tandem Computers

Cause. An unspecified internal error occurred.

Effect. The compiler terminates abnormally.

Recovery. Report this internal error to your HP representative.

*** Fatal *** Problem with temporary object file, error

Cause. The temporary file needed by the compiler either could not be created or, though created, could not be opened. The error number is listed.

Effect. The compile terminates.

Recovery. Check that the temporary object file exists and has proper access privileges.

*** Fatal *** Symbol table is full

Cause. All the extended memory available for storing symbols (65536-bytes) is used.

Effect. The compile terminates.

Recovery. Make space available in the symbol table.

*** Fatal *** TACL communication:
  file error Guardian error
**Cause.** The compiler receives an error while trying to communicate with its TACL process.

**Effect.** The compile terminates.

**Recovery.** Correction depends on the specified Guardian error.

### TACL Fatal-Error Messages

This subsection describes fatal-error messages produced by TACL that begin with *** EMF:. When you have used a construct in your filter specification that causes a fatal error in TACL, TACL does the following:

- Indicates the source location of the error
- Displays one of the error messages that are described in this subsection (all of which are prefixed by *** EMF:)
- Stops the compiler

Because compilation is abruptly terminated, any errors that follow the fatal error message go undetected in this compilation but are diagnosed on recompilation.

Each of these message text boxes contains the exact text of one TACL fatal-error message. The text following each box describes the cause, effect, and recovery for that message.

<table>
<thead>
<tr>
<th>Message Text Box</th>
<th>Cause</th>
<th>Effect</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>*** EMF: Break hit, compile terminated</td>
<td>You pressed the BREAK key. The compiler cannot run nowait.</td>
<td>TACL stops the compiler and returns control to you.</td>
<td>Run again without pressing the BREAK key.</td>
</tr>
<tr>
<td>*** EMF: Compiler stopped abnormally (reason: err)</td>
<td>An internal compiler error occurred.</td>
<td>The compiler terminates abnormally.</td>
<td>Determine the cause of the abnormal stop based on the reason specified.</td>
</tr>
<tr>
<td>*** EMF: Compiler stopped early</td>
<td>An authorized person or process stopped the compiler with a STOP command.</td>
<td>The compiler stops before it has completed its tasks.</td>
<td>Determine the cause of the abnormal stop based on the reason specified.</td>
</tr>
<tr>
<td>*** EMF: Compile unsuccessful</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cause. The compilation failed for some reason other than bad input.

Effect. The compile ends without successful results.

Recovery. Determine what caused the failed compile (possibly a TACL problem).

```plaintext
*** EMF: Unknown compiler request has caused an internal error
```

Cause. An undefined or misspelled TACL macro directive has been processed by TACL; TACL then terminates the compilation abnormally. An error internal to the compiler or to a TACL macro caused the termination.

Effect. The compile terminates abnormally.

Recovery. Save all related information for your service provider.

Completion Codes

After finishing a compilation, the compiler returns a completion code, which it saves in the TACL structure_completion and displays in the TACL OUT file.

A completion code, which is especially useful in conjunction with the batch facility, is not an error code. Whereas an error code tells what is wrong (typically with the source program), a completion code describes what the compiler produced.

**Table 5-4. Compiler Completion Codes**

<table>
<thead>
<tr>
<th>Completion Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful compilation</td>
</tr>
<tr>
<td>1</td>
<td>Warning messages only; listing and object file complete</td>
</tr>
<tr>
<td>2</td>
<td>Errors and warnings; listing complete, no object file</td>
</tr>
<tr>
<td>3</td>
<td>Fatal error; premature termination, incomplete listing, no object file</td>
</tr>
<tr>
<td>4</td>
<td>Compilation not started; no compiler output</td>
</tr>
<tr>
<td>5</td>
<td>Internal compiler error</td>
</tr>
<tr>
<td>6</td>
<td>Compiler was stopped by a TACL STOP command or by a STOP procedure call from a process</td>
</tr>
<tr>
<td>8</td>
<td>Object file named ZZEFnnnn; listing and object file complete</td>
</tr>
</tbody>
</table>

If you receive a completion code of 5 (or any code not listed above), report the code and the information from the ABEND to your service provider.
6 Filter Tables and Burst Filters

This section describes filter tables and burst filters and describes how to construct, add, replace, and delete them from a collector or distributor:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter Tables</td>
<td>6-1</td>
</tr>
<tr>
<td>Burst Filters</td>
<td>6-7</td>
</tr>
</tbody>
</table>

For more information on compiled filters, the EMS filter (EMF) language, and the filter compiler, see Section 5, Compiled Filters. For more information on configuring and implementing BDS in a collector or distributor, see Section 7, Burst Detection and Suppression.

Both collectors and distributors support filter tables and burst filters, which you can use instead of or in addition to compiled filters.

Filter Tables

You can use filter tables to minimize the complexity of EMS filter installation. Because they are faster than compiled filters, they reduce event processing times. You can use filter tables instead of—or, in the case of multiple filters, in conjunction with—standard compiled filters.

Because a compile step is not required, filter tables offer ease of maintenance and improved efficiency. They are particularly useful when you need to filter events by simple criteria such as event number and subsystem ID, and when there you need frequent and fast online update of filter contents.

Filter Table Features and Functions

A filter table contains columns and rows of subsystem owners, subsystem IDs (text or numeric), and event numbers. You can define the table type as PASS or FAIL.

A PASS filter selects events to forward to their appropriate destinations. For collector-based filter tables, these are disk log files; for distributor-based filter tables, they can include such destination devices as printers, operator terminals, log files, or management applications.

A FAIL filter, conversely, selects events to be discarded. If you specify only one PASS filter table, events that are not selected by the filter table are not logged to disk or other destination devices. If you specify only one FAIL filter table, events not selected by the filter are forwarded to their destination device.

The filter table’s source is written to an EDIT file and can be loaded directly into an EMS collector or distributor. The collector or distributor compiles the source file and saves the filter object in a disk file. After an object file has been obtained, it can be loaded into the collector or distributor instead of the source.
The filter table contains three base columns: subsystem owner, subsystem ID, and event number. You can replace the event number column with a standard type or user type column. A hierarchical dependency exists between columns: an owner can have several subsystem IDs, and a subsystem ID can have several event numbers.

You can specify a pass value for each event or group of events. The pass value is returned to the application the same way that consumer distributors and compiled filters currently provide.

Filter table directives allow filtering by other event header tokens (process name, node, CPU, PIN, emphasis, user ID, time).

Filter tables can be added, deleted, or replaced with standard SPI commands sent to a collector or distributor.

Filter tables can co-exist with compiled filters in collectors or distributors. The total number of filters allowed in a collector or distributor is 10.

**Filter Table Format**

A filter table is represented as an EDIT file. When loaded into a collector or distributor, the table is automatically converted to a filter object and saved as a disk file. After the object file is available, it can be loaded into a collector or distributor instead of the EDIT file.

The name of the object file is derived from the source file by prefixing an “O” character (as in “Object”). If the file name is already eight characters long, the last character is discarded.

For example, the format of the EDIT file is:

```plaintext
?COMMENT
EMS FILTER TABLE

?EMPHASIS 1
?USER 165,168

?COMMENT

owner    subsys    event#              passval

?PASS
TANDEM   EMS       1000..1050          10
          1100
TMF       *                   20
TLAM      -3                  30
          513                 31
          139                 32
MYSYS    15        100..500            40
YOURSYS  *                             50
```

The filter table EDIT file is divided into sections. Any blank lines or lines with an exclamation mark in the first column are skipped. Lines with a question mark (?) in the
first column are interpreted as directives. The filter must contain a ?PASS or ?FAIL directive. The ?COMMENT directive skips lines until another directive is encountered.

The owner name in the filter table (up to 8 characters) starts in position 1, the subsystem name or number in position 10, and the event number in position 20. Event number ranges are permitted (a..b), as well as wild cards in the owner, subsystem, or event column.

For instance, if a wild card appears in the event column, all events for the SSID listed in the subsystem column pass (or fail). Numbers need not be in any specific order. Negative event numbers are permitted. Event number 0 is not allowed. An error results if one is specified. Owners, and subsystems within an owner, can appear only once. A duplication results in an error.

The event number column can be replaced by either event standard type or event user type. To do so, include either directive ?STYPE or directive ?UTYPE without a value. The two types are mutually exclusive. The same rules and restrictions as for event numbers apply.

Pass values can be specified in column 40. These are identical to the compiled filter's pass values and are returned to the consumer application. Pass values can be in the range from -32768 to +32767. However, a zero pass value is not permitted. The difference between the largest pass value and the smallest cannot exceed 32767 for a given subsystem. Because pass values increase the size of the filter table, HP recommends that you keep the relative range small and specify one pass value per subsystem where possible. For instance, pass values in the range from 1 through 7 (or 101 through 107) would require 3 bits for each event listed in the table. Pass values are allowed only for PASS filters.

Filter Table Keywords

These filter table directives must precede the actual filter table.

Table 6-1. Filter Table Keywords (page 1 of 2)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?PROCESS name</td>
<td>name is the ID of the process that generated the event. The comparison is not case sensitive, but the match must be for exactly the number of characters given (maximum of six for Cxx–based events) except when the asterisk (<em>) wild-card character appears as a last character (example: $SP</em>).</td>
</tr>
<tr>
<td>?NODE n</td>
<td>n is the node number of the system that generated the event. The value for n must be smaller than 255 for Cxx-based events.</td>
</tr>
<tr>
<td>?CPU n</td>
<td>n is number of the CPU in which the event generator runs. The value for n must be smaller than 16.</td>
</tr>
<tr>
<td>?PIN n</td>
<td>n is the PIN number of the process that generated the event. The value for n must be smaller than 255 for Cxx-based events.</td>
</tr>
</tbody>
</table>
**Table 6-1. Filter Table Keywords** (page 2 of 2) (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?EMPHASIS ( n )</td>
<td>( n ) is either (0) or (1) and signifies the severity of the event. If set to (1), the event is considered critical.</td>
</tr>
<tr>
<td>?USER ( n,m )</td>
<td>( n ) is the group number and ( m ) the user number of the user ID associated with the process that generated the event. Either one must be smaller than 256. The user number can be the asterisk (<em>) wild-card character (example: 255,</em>).</td>
</tr>
<tr>
<td>?START ( time )</td>
<td>( time ) is the start time given in hh:mm (hours:minutes, local civil time). If stop time is not given, the following midnight 00:00 is used. Any event that has a generation time outside the start/stop interval fails a PASS filter. The date portion of the timestamp is not considered; that is, the filter operates on a daily basis and need not be updated. Not available with compiled filters.</td>
</tr>
<tr>
<td>?STOP ( time )</td>
<td>( time ) is the stop time given in hh:mm (hours:minutes, local civil time). Stop time must be larger than start time. If start time is not given, the previous midnight 00:00 is used. Any event that has a generation time outside the start/stop interval fails a PASS filter. The date portion of the timestamp is not considered; that is, the filter operates on a daily basis and need not be updated. Not available with compiled filters.</td>
</tr>
<tr>
<td>?STYPE [ n ]</td>
<td>( n ) is an enumeration for the standard event type. The possible range for ( n ) is defined in the event standards section of the ZEMSDDL file. Zero is not a valid type. If ( n ) is omitted, it is assumed that the event number column in the filter table is to be interpreted as standard event type column.</td>
</tr>
<tr>
<td>?UTYPE [ n ]</td>
<td>( n ) is an enumeration for the user-defined event type. Some possible values for ( n ) are defined in the event standards section of the ZEMSDDL file. Zero is not a valid type. ?STYPE and ?UTYPE are mutually exclusive. If ( n ) is omitted, it is assumed that the event number column in the filter table is to be interpreted as user-defined event type column.</td>
</tr>
</tbody>
</table>

**Considerations for Keywords**

All values are tested for equality with their counterparts in the event header. Generation time is first converted to local civil time.

Other keywords that are processed as filter table directives are ?PASS, ?FAIL, and ?COMMENT. One ?PASS or ?FAIL directive is required. A ?COMMENT directive is terminated when another directive is encountered.

Directives are not allowed in the actual filter table body. You can place comments by using the exclamation mark (!) in the first column or anywhere in column 50 and beyond.
Filter Table Errors

These errors can be returned when the conversion from filter table text file to filter table object fails. The return code for these errors is ZEMS-ERR-FLT-FORM (1019). The detailed error code is returned in ZEMS-TKN-FILTER-ERROR. The row and column locations of the error occurrences are returned in ZEMS-TKN-FAIL-REASON. (The column is in bits <0:5>, and the row is in bits <6:15>). For descriptions of distributor errors related to filter table use, see 1019: ZEMS-ERR-FLT-FORM on page 21-9 and 1020: ZEMS-ERR-FLT-LOAD on page 21-10.

Detailed Error Codes

<table>
<thead>
<tr>
<th>Error Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required param missing</td>
<td>-10</td>
</tr>
<tr>
<td>Too many subsystems</td>
<td>-11</td>
</tr>
<tr>
<td>Open problem</td>
<td>-12</td>
</tr>
<tr>
<td>Editreadinit problem</td>
<td>-13</td>
</tr>
<tr>
<td>Editread problem</td>
<td>-14</td>
</tr>
<tr>
<td>PASS or FAIL missing</td>
<td>-15</td>
</tr>
<tr>
<td>Invalid SSID</td>
<td>-16</td>
</tr>
<tr>
<td>Unknown SSID</td>
<td>-17</td>
</tr>
<tr>
<td>Invalid event number</td>
<td>-18</td>
</tr>
<tr>
<td>Invalid format for range</td>
<td>-19</td>
</tr>
<tr>
<td>Too many event numbers</td>
<td>-20</td>
</tr>
<tr>
<td>Table overflow</td>
<td>-21</td>
</tr>
<tr>
<td>Duplicate owner</td>
<td>-22</td>
</tr>
<tr>
<td>Duplicate subsystem</td>
<td>-23</td>
</tr>
<tr>
<td>Invalid passval</td>
<td>-24</td>
</tr>
<tr>
<td>Passval not allowed</td>
<td>-25</td>
</tr>
<tr>
<td>Invalid column position</td>
<td>-26</td>
</tr>
<tr>
<td>Wildcard placement error</td>
<td>-27</td>
</tr>
<tr>
<td>Subsystem entry required</td>
<td>-28</td>
</tr>
<tr>
<td>Owner entry required</td>
<td>-29</td>
</tr>
<tr>
<td>Event entry required</td>
<td>-30</td>
</tr>
<tr>
<td>Invalid owner</td>
<td>-31</td>
</tr>
<tr>
<td>No filter table entry</td>
<td>-32</td>
</tr>
<tr>
<td>Invalid start time</td>
<td>-40</td>
</tr>
<tr>
<td>Invalid stop time</td>
<td>-41</td>
</tr>
<tr>
<td>Invalid user ID</td>
<td>-42</td>
</tr>
<tr>
<td>Invalid standard type</td>
<td>-43</td>
</tr>
<tr>
<td>Invalid cpu number</td>
<td>-44</td>
</tr>
<tr>
<td>Invalid pin number</td>
<td>-45</td>
</tr>
<tr>
<td>Invalid process ID</td>
<td>-46</td>
</tr>
<tr>
<td>Invalid standard type</td>
<td>-47</td>
</tr>
<tr>
<td>Invalid user defined type</td>
<td>-48</td>
</tr>
<tr>
<td>Invalid emphasis</td>
<td>-49</td>
</tr>
<tr>
<td>Invalid keyword</td>
<td>-50</td>
</tr>
<tr>
<td>Duplicate keyword</td>
<td>-51</td>
</tr>
<tr>
<td>Event type incompatibility</td>
<td>-52</td>
</tr>
<tr>
<td>Invalid burst parameter N</td>
<td>-53</td>
</tr>
<tr>
<td>Invalid burst parameter T1</td>
<td>-54</td>
</tr>
<tr>
<td>Invalid burst parameter T2</td>
<td>-55</td>
</tr>
<tr>
<td>Invalid burst parameter T3</td>
<td>-56</td>
</tr>
<tr>
<td>Invalid burst parameter S</td>
<td>-57</td>
</tr>
<tr>
<td>Invalid burst parameter L</td>
<td>-58</td>
</tr>
</tbody>
</table>
Multiple Filters

Alternate collectors and distributors accept specification of multiple filters in the startup line of the EMSACOLL and EMSDIST program, respectively. Multiple filters are run in sequential order for each event. Assuming PASS filters only, an event is disqualified if all filters return a fail condition. Execution of filters stops with the first PASS condition encountered. Conversely, assuming FAIL filters, execution stops with the first FAIL condition encountered. Events that pass are sent to the next filter.

You can use a combination of PASS and FAIL filters when specific events are to be excluded from or included in a set; for example, to pass all events for subsystem XYZ except event #n. In this example, a FAIL filter would exclude event #n and pass all other events to a following PASS filter that makes the selection by subsystem.

Compiled filters, filter tables, and one burst filter can be mixed. Typically, a filter table precedes the compiled filter for performance reasons. If a burst filter is used, it should follow the other filters.

For information about adding and deleting filters, see Loading Filter Tables on page 6-6. For a summary of procedures available for configuring and loading burst tables and filter tables, see Implementing BDS on page 7-6 and Implementing Specific-Event Suppression on page 7-7.

For descriptions of distributor commands that accept an object token, see Section 17, Distributor Commands and Responses. For descriptions of collector commands that accept object tokens, see Section 19, Collector Commands and Responses

Recovery Actions

When you encounter an error while adding or deleting a set of filters, the collector or distributor reports the filter in error but does not back out the successful operations. Use the STATUS command to determine the current valid set of filters in the collector or distributor.

Loading Filter Tables

You can load a single filter or multiple filters into a collector or distributor in several ways:

- If an alternate collector or distributor is not yet started, use the EMSACOLL or EMSDIST program with the FILTER option to specify one or more filters.
- To load one or more filters into a running primary or alternate collector, use the EMSCCCTRL program with the FILTER option.

Directive SUPPRESS missing = -59
Incompatible directives = -60
Object file purge error 48 = -100
Object file create error = -101
Object file open error = -102
Object file write error = -103
Filter Tables and Burst Filters

- Use the object-based SPI command ADD to load multiple filters or a single filter into a collector or distributor.

Filter names are submitted as multiple object token values. To remove a filter or replace a set of filters with a PASSALL default filter, use the DELETE command. The CONTROL command cannot be used by collectors to add, alter, replace, or delete filters. For distributors, the CONTROL command cannot be used to load multiple filters because it does not provide object tokens to specify filter files; in fact, it deletes all other existing filter files when used to load a single new filter, so it should not be used when more than one filter is required in a distributor. For a description of object-oriented commands (ADD, ALTER, REPLACE, and so on) that you use to load filter tables and other filter types, see the distributor and command information in Section 17, Distributor Commands and Responses and Section 19, Collector Commands and Responses.

For more detailed guidelines on loading filters, see Section 7, Burst Detection and Suppression.

Logical Connection Is AND

The logical connection between the single directives and between directives and actual filter tables is AND. For each directive, only one value can be given. To implement OR conditions for header tokens, more than one filter table can be connected in a cascade. The logical connections between rows in the filter table are OR connections.

Restrictions

In a filter table, the maximum number of events per subsystem, when subtracting the lowest event number from the highest, cannot exceed 4096. The maximum number of subsystem IDs per owner, when subtracting the lowest from the highest, cannot exceed 256.

Burst Filters

Use a burst filter—a type of filter table—to identify the criteria for event burst detection and suppression (BDS) in a collector or distributor. When a burst filter is installed in a collector or distributor, BDS prevents the forwarding of any event burst that exceeds the specified number of events per time duration. In collectors, you can also enable BDS using the SUPPRESS parameter of the EMSACOLL and EMSCCTRL programs. However, a burst filter and the SUPPRESS parameter cannot be used simultaneously to implement BDS for a given collector.

For a detailed description of the BDS feature of EMS, including methods for configuring and implementing BDS in collectors and distributors, see Section 7, Burst Detection and Suppression.

A burst filter, like any other filter, can be added, deleted, and used in conjunction with compiled filters and standard filter tables. Like standard filter tables, you specify burst
Filter Table and Burst Filters

filter directives in an EDIT file. The collector or distributor converts this EDIT file to a filter object in the same manner as for standard filter tables.

A burst filter always acts like a FAIL filter because it suppresses event bursts that conform to the prevailing BDS configuration criteria. When BDS is enabled in a collector, these suppressed events are discarded rather than logged to disk. When BDS is enabled in a distributor, suppressed events are discarded instead of being sent to their normal destination devices.

A burst filter is identified with the directive ??SUPPRESS and can contain only burst configuration parameters. Event header token directives and filter table entries are not accepted. After a burst filter has been loaded into a collector or distributor, the event stream is monitored, with event bursts detected and suppressed on the basis of the burst filter’s configuration parameters.

To disable BDS in a collector or distributor, delete the burst filter, either through the appropriate EMS program (EMSCCTRL for a collector) or by issuing a SPI DELETE command with the burst filter object specified. To change burst parameters, delete the burst filter and then add a new burst filter containing the altered BDS parameters.

**Note.** The distributor STATUS command does not provide any specific information about the burst filter, nor does it provide any statistics counters. The EMSCINFO DETAIL display can provide this information for primary and alternate collectors.

You can install only one burst filter in a collector or distributor. However, you can combine it with filter tables or compiled filters. If another filter precedes the burst filter, only events not selected by the first filter are processed by the burst filter. However, if a burst filter is followed by another filter, bursting events (as defined and suppressed by the burst filter) are not sent to the second filter, and only nonbursting events are passed on.

When an event burst is detected, any subsequent event that matches the burst configuration criteria is suppressed until the burst ends. When the burst is detected, the collector or distributor generates a burst-detected event that describes the bursting event. The burst-detected event notifies operators that EMS has begun suppressing an event and that they should not interpret the absence of further copies of the event as an indication that the underlying problem has been corrected.

When the collector or distributor determines that a particular event burst has ended, it generates a burst-ended event to notify operators that the underlying problem might have been corrected and that normal processing of the event has once again been enabled. The burst-ended event also identifies the number of events that were suppressed.

**Note.** By default, burst-detected and burst-ended events are sent to $0. However, you can specify that the events be sent to an alternate collector and that they be logged in the TMDS or TSM log. For more information, see BDS Parameters on page 7-3.
Burst Filter Directives

Defined in terms of the filter directives used to configure a burst filter, an event burst is the occurrence of N or more similar events during T1 seconds. An event burst is considered to have ended when the bursting event does not occur for T2 seconds. The values of the N, T1, and T2 parameters are supplied by the user.

In addition to the N, T1, and T2 parameters, other burst configuration parameters include:

- Maximum number of bursting events that can be monitored simultaneously (S)
- Length of the subject token value used in event comparison (L)
- Time between periodic checks for burst end conditions (T3)

The timer starts when the first burst is detected and is cancelled when there are no more bursts. If the subject length parameter (L) is set to zero, the subject token ID is still compared, but the value is disregarded in the comparison for an event match. The token ID is also disregarded if L is set to -1.

Determining an acceptable value for the S parameter might require careful fine-tuning. The greater the value of S, the larger the number of simultaneous event bursts that can be detected. However, a high S value increases filter processing time and table size.

All directives must be preceded by a question mark (?).

**Note.** Other table keywords (except ?COMMENT) and filter table entries cannot appear together with these burst filter directives. Only one burst filter can be loaded into a collector or distributor. However, other filter tables and compiled filters can be added. The order of execution is the same as the order in which the filters are added.
### Table 6-2. Burst Filter Directives

<table>
<thead>
<tr>
<th>Filter Directive</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>?SUPPRESS</td>
<td>The filter is a burst filter. This directive is required and must precede all other burst filter configuration directives.</td>
</tr>
<tr>
<td>?N m</td>
<td>The number of similar events repeated within T1 seconds that constitute a burst condition. The variable <em>m</em> must be in the range of 2 through 32,767 events. The default value is 100.</td>
</tr>
<tr>
<td>?T1 time</td>
<td>The time interval during which similar events are counted to determine if a burst has occurred. The variable <em>time</em> is given in seconds and must be in the range of 1 through 3,600 seconds. The default value is 120 seconds.</td>
</tr>
<tr>
<td>?T2 time</td>
<td>The time interval during which a bursting event must occur at least once. Otherwise, the burst is considered to have ended. The variable <em>time</em> is provided in seconds and must be in the range of 1 through 3600 seconds. The default value is 120 seconds.</td>
</tr>
<tr>
<td>?T3 time</td>
<td>The time interval chosen for periodic checking of burst end conditions. The timer starts when the first burst condition is detected and cancels when there are no more burst conditions to monitor. The variable <em>time</em> is given in seconds and must be in the range of 15 through 3600 seconds. The default value is 300 seconds.</td>
</tr>
<tr>
<td>?S m</td>
<td>The number of simultaneous bursts that can be monitored and the size of the burst table. The variable <em>m</em> must be in the range of 2 through 128 bursts. The default value is 6 bursts. The larger the value that is chosen for <em>S</em>, the higher the probability that event bursts will be detected, but at the expense of increased processing time duration and table size.</td>
</tr>
<tr>
<td>?L m</td>
<td>The length, in bytes, of the subject value that is used to determine if events are similar. The variable <em>m</em> must be in the range of -1 through 254 bytes (the default value is 254 bytes). If set to -1, subject token and value are not considered in the determination of similar events. Only the event number and subsystem ID are used in the comparison. If set to zero, the subject token ID is compared, but the value is not. The value chosen for <em>L</em> impacts the size of the burst table.</td>
</tr>
</tbody>
</table>
Burst Filter Example

This example shows EDIT file for a burst filter:

```plaintext
?COMMENT
Burst Detection/Suppression
?SUPPRESS
! Number of event occurrences for a burst
?N 50
! Burst start detection interval (seconds)
?T1 60
! Burst end detection interval (seconds)
?T2 120
! Periodic check for burst end (seconds)
?T3 30
! Maximum number of simultaneous bursts
?S 20
! Subject length to compare
?L 0
```

For examples of the consequences of different BDS configurations, see BDS Configuration Examples on page 7-5.

For a summary of the methods for configuring and enabling BDS from the primary collector, an alternate collector, and a distributor, see Implementing BDS on page 7-6.
Burst Detection and Suppression

This section describes the burst detection and suppression (BDS) and specific-event suppression features of EMS:

For more information about the structure of filter tables and burst filters, see Section 6, Filter Tables and Burst Filters.

Event Suppression Strategies

An event burst occurs when a similar event is issued repeatedly. Unwanted and redundant event messages waste system resources and can prevent other more critical problems from being quickly detected and corrected.

EMS supports two strategies for detecting and suppressing event bursts and specific event types:

Suppressing Event Bursts

Within a specific collector or distributor process, you can detect and suppress all event bursts that exceed a specified number and time duration.

This strategy requires the use of burst filters (or collector program keywords) to enable BDS. For primary and alternate collectors, you can enable BDS programmatically, using either the SUPPRESS or FILTER keyword in the EMSACOLL and EMSCCTRL programs.

For distributors, BDS can be enabled only by using the EMSDIST program or the distributor ADD command to add a burst filter that contains the BDS directives. For EMSDIST syntax guidelines, see EMSDIST—Distributor Program on page 13-34. For information about the ADD command, see ADD Command (ZCOM-CMD-ADD) on page 17-12.

Suppressing All Events of a Specific Type

Within a specific collector or distributor process, detect and suppress all events of a specific type even if they are not repeated (for example, as in an event burst).
This strategy, referred to here as specific-event suppression, requires the use of a filter table or compiled filter that clearly identifies the specific event type to be suppressed.

Pre-log filtration (PLF) refers to the use of one or more filters in a primary or alternate collector to prevent certain events from being sent from the collector to its log file. This includes at most one burst filter to suppress event bursts and one or more filter tables or compiled filters that suppress some or all occurrences of specific event messages. Suppressed events are discarded.

⚠️ Caution. Use caution when using compiled filters or filter tables to totally suppress events that are generated by subsystems that are specific to the NonStop server. Many of these events are monitored by HP and other applications. Total event suppression can result in state information from a particular subsystems for the NonStop server being discarded and becoming nonretrievable. Because events from the subsystems specific to the NonStop server are almost always sent to the primary collector ($0), you must use discretion in using such filters in the primary collector when the suppressed event information might be of value to the user.

### BDS Features

The EMS burst detection and suppression (BDS) feature lets system operators manage burst events in primary collector, alternate collector, and distributor processes. An event burst occurs when a similar event is issued repeatedly. For a definition of similar events, see BDS Parameters on page 7-3.

In most instances, the initial event message in the burst provides sufficient notification to the operator, and subsequent bursts of the same event message waste valuable disk space and system resources. As with PLF, suppressed event bursts are discarded.

In terms of burst filter and SUPPRESS directives, an event burst is the occurrence of N similar events in T1 seconds, where N and T1 represent BDS parameter values that you supply. An event burst is considered to have ended when the bursting event does not occur for T2 seconds, where T2 is also a user-supplied parameter value.

When you apply BDS to an EMS collector (primary or alternate), you can prevent event bursts of a specific size and duration from being written to the EMS log files. When an event burst is detected, the collector suppresses logging of subsequent similar events until the burst has ended. This saves the disk space and the system resources required to write the redundant events to disk. The bursting events discarded by BDS during burst suppression are eliminated from the system and are not available for subsequent distribution or review.

When you apply BDS to an EMS distributor, the distributor discards redundant events and does not pass them on to their destination devices or processes. Instead, these events remain resident in EMS log files.

The monitoring and suppression of events by BDS is transparent to event users and consumers. If the normal event stream does not contain frequent and repetitive events, BDS is inactive from the perspective of event consumers. If you expect and want to
see repetitive events, either disable BDS or alter the BDS parameters so they do not suppress the expected repetitive events.

BDS users should first create burst filters that enable burst detection in one or more of your EMS distributors. This prevents redundant events from reaching destination processes and devices while still making the events available to the EMS log files (events are discarded and eliminated from the system when BDS is applied to an EMS collector).

When you are comfortable with the burst configuration parameter options and the overall function of BDS, you might choose to move burst detection to your collectors (especially alternate collectors) to prevent the logging of repetitive events altogether.

The BDS algorithm is an enhanced version of the BDS facility available in all EMS distributors and collectors.

**BDS Parameters**

Six parameters control the behavior of BDS in EMS collectors and distributors: N, T1, T2, T3, L, and S. You can implement BDS parameters when:

- Starting an alternate collector, use the SUPPRESS keyword parameters in the RUN EMSACOLL command.
- Starting an alternate collector, use the FILTER keyword parameters in the RUN EMSACOLL command to load a burst filter that contains the BDS parameters.
- Configuring a running primary or alternate collector, use the SUPPRESS keyword parameters of the EMSCCTRL program.
- Configuring a running primary or alternate collector, use the FILTER keyword parameters of the EMSACOLL command to load a burst filter that contains the BDS parameters.
- Starting a distributor, use the FILTER keyword parameters in the EMSDIST program to load a burst filter that contains the BDS parameters.

Or create a program that uses the SPI interface to ADD or DELETE a burst filter.

For a detailed description of the methods used to configure and enable BDS in collectors and distributors, see Implementing BDS on page 7-6.

Two or more EMS events are considered similar if they:

- Originate from the same subsystem (that is, have the same SSID)
- Possess the same event number
- Possess the same token code for their first event subject
Collectors convert text generated by a sender-invoked WRITE procedure into text events. Two or more text events are considered similar only if their text is identical.

The BDS parameters in Table 7-1 represent both the directives used to construct a burst filter and the parameter values specified with the SUPPRESS keyword of the EMSACOLL and EMSCCTRL programs. Once you supply these directives, either through a burst filter or an EMSACOLL or EMSCCTRL SUPPRESS command, BDS is enabled, and EMS can monitor the event stream, detecting event bursts on the basis of the N, T1, and L parameter values.

When an event burst is detected, EMS suppresses logging of the bursting event until it determines that the burst has ended. If no value is specified for a given parameter, its default value is automatically applied.

Table 7-1. Burst Detection Parameter Meanings and Default Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>The number of events that constitute a burst</td>
<td>100 events</td>
</tr>
<tr>
<td>T1</td>
<td>The time period in which N events constitute a burst</td>
<td>100 seconds</td>
</tr>
<tr>
<td>T2</td>
<td>The time period during which the occurrence of no events ends a burst</td>
<td>120 seconds</td>
</tr>
<tr>
<td>T3</td>
<td>The time interval between checks for ended bursts</td>
<td>300 seconds</td>
</tr>
<tr>
<td>L</td>
<td>The comparison byte length of the first subject value</td>
<td>254 bytes</td>
</tr>
<tr>
<td>S</td>
<td>The maximum number of simultaneous bursts that can be detected and monitored</td>
<td>6 bursts</td>
</tr>
</tbody>
</table>

Table 7-2. Value Ranges for Burst Detection Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>2 events</td>
<td>32,767 events</td>
</tr>
<tr>
<td>T1</td>
<td>1 second</td>
<td>3600 seconds (1 hour)</td>
</tr>
<tr>
<td>T2</td>
<td>1 second</td>
<td>3600 seconds (1 hour)</td>
</tr>
<tr>
<td>T3</td>
<td>15 seconds</td>
<td>3600 seconds (1 hour)</td>
</tr>
<tr>
<td>L</td>
<td>0 bytes*</td>
<td>254 bytes</td>
</tr>
<tr>
<td>S</td>
<td>2 bursts</td>
<td>128 bursts</td>
</tr>
</tbody>
</table>

* L can also be set to -1, which means that the subject token code is ignored in determining event similarity.

When an event burst is detected, the EMS collector or distributor generates a burst-detected event that describes the bursting event. This description identifies the subsystem that generated the event burst, the event number, and the token code of
the first event subject in the event burst. This burst-detected event alerts operators that
EMS has begun suppressing an event. Operators should not interpret the absence of
further copies of the event as an indication that the underlying problem has been
corrected.

When EMS determines that a burst has ended, it generates a burst-ended event to
notify operators that the underlying problem has been corrected and that normal
processing of the event has again been enabled. In addition to identifying the ended
burst, the burst-ended event also reports the number of redundant events that were
discarded by the EMS collector or distributor.

By default, the burst-detected and burst-ended events are sent to $0. However, you
can have these events sent instead to an alternate collector using a DEFINE:

```
add define =_ems_alternate_log, file $ACO
```

You can also log these events to the TMDS (D-series) or TSM Service Log (G-series):

```
add define =_ems_system_log, file $ZLOG
```

Because the EMS collector and distributor processes are active only when incoming
messages and events are received, they might need to be awakened periodically to
check for burst-ended conditions after an event burst has been detected. The user can
configure this wakeup period, the T3 parameter value.

**BDS Configuration Examples**

These examples illustrate two BDS configurations and their effect on event burst
detection and suppression.

**Example 1**

The primary collector, $0, is configured with BDS enabled, where N=10 events, T1=10
minutes, T2=5 minutes, and T3=1 minute. If a process began issuing the same event
at 30-second intervals beginning at exactly 10:00:

- $0 continues to log the event until 10:05, when the eleventh copy of the bursting
event arrives. Because $0 has detected the event more than ten times in 10
minutes, $0 logs a burst-detected event. It does not log subsequent occurrences of
the event but instead discards them.

- At 10:20:05, the offending process is stopped, so the last event in the burst was
issued at 10:20:00.

- Some time between 10:25 and 10:26 (10:20 + 5 minutes + (0-1 minute)), $0 logs
an event declaring that the event burst has ended (burst-ended event).

**Example 2**

A system (node \ABC) is running SNAX with six lines (named $LINE1 through
$LINE6). Each line is configured with four physical units (PUs), named #PU1 through
#PU4. Each PU is configured with 32 LUs (named #LU:n01 through #LU:n32, where n
represents the PU number 1 through 4). In this configuration, a change of state in one of the lines generates a line event, four PU events, and 128 (4x32) LU events. However, only the line event requires attention. The other events simply waste processing time and disk space. BDS can be used here to eliminate most of the 128 LU events, but probably none of the PU events (because there are only four of them). To accomplish this, you must analyze the event burst that you want to minimize.

Suppose that the 128 LU events are generated over a period of 30 seconds. If $N$ is set to 10 events, $L$ to 15 bytes, $T_1$ to 30 seconds, and $T_2$ to 60 seconds, the LU burst is limited to only 10 events, plus the BDS event-detected event that announces the start of the event burst. $L$ is set to 15 so that the subject of all of the LU events will compare as equal. The subject of the LU event is the name of the LU. This LU name takes the form $\backslash$ABC.$\text{LINE}n.\#LUPll$, where $n$ is the line number, $p$ is the PU number, and $ll$ is the LU number. For any given line, the LU name is the same for the first 15 characters.

The overall assumption made in this example is that meaningful LU events do not occur frequently enough to trigger BDS. Because the BDS configuration parameters assigned to suppress LU events are applied to all events, it must also be assumed that any other meaningful events will not unintentionally trigger BDS.

### Implementing BDS

Using TACL, you can configure and enable BDS several ways, depending on the state of the collector or distributor.

#### Implementing BDS From a Primary Collector

To configure and enable BDS in the primary collector ($0$), use one of these methods:

- While the primary collector is running, use the SUPPRESS ON command of the EMSCTRL program to specify and enable BDS directives for the primary collector. For a detailed description of EMSCTRL command syntax options, see Section 13, EMS Programs.

- If the SUPPRESS function of EMSCTRL has not been specified, use the FILTER keyword of EMSCTRL to specify the name of the burst filter that contains the burst configuration directives.

- While $0$ is running, invoke a program that uses the SPI interface to modify the filter criteria (for example, a program that contains an ADD command with the burst filter as its object).

#### Implementing BDS From an Alternate Collector

To configure and enable BDS in an alternate collector, use one of these methods:

- When the alternate collector is first started, use the SUPPRESS keyword of the EMSACOLL program to specify the BDS configuration directives. For a detailed description of EMSACOLL command syntax options, see Section 13, EMS.
Programs. If you use EMSACOLL SUPPRESS to enable BDS, you cannot use the EMSACOLL FILTER option.

- When the alternate collector is first started, use the FILTER keyword of the EMSACOLL command to add a burst filter that contains the BDS directives. If you use the EMSACOLL FILTER option to load a burst filter to enable BDS, you cannot use the EMSACOLL SUPPRESS option.

- After the alternate collector is running, use the SUPPRESS keyword of EMSCCTRL to specify BDS directives.

- If the EMSCCTRL SUPPRESS function has not been specified, use the EMSCCTRL FILTER command to specify a burst filter that contains the BDS directives. (This command enables PLF.)

- While the alternate collector is running, invoke a program that uses the SPI interface to modify the filter criteria (for example, a program that contains an ADD command with the burst filter as its object).

Implementing BDS From a Distributor

You can configure and enable BDS in a distributor using one of these methods:

- When the distributor is first started, use the FILTER keyword of the EMSDIST program to load a burst filter into that distributor.

- While the distributor is running, invoke a program that uses the SPI interface to modify the filter criteria (for example, a program that contains an ADD command with the burst filter as its object).

Implementing Specific-Event Suppression

Using TACL, specific-event suppression can be configured and enabled using a number of methods. Specific-event suppression causes all occurrences of specific event messages that are received by the collector or distributor to be suppressed. Specific-event suppression can be configured and implemented only by means of a filter table or a compiled filter that identifies the event type to be suppressed and that is added to the collector or distributor process.

⚠️ Caution. Use caution when using compiled filters or filter tables to totally suppress events that are generated by subsystems specific to the NonStop server because many of these events are monitored by HP and other applications. Total event suppression can result in state information from a particular subsystems for the NonStop server being discarded and becoming nonretrievable. Because events from subsystems specific to the NonStop server are almost always sent to the primary collector ($0), you must use discretion in using such filters in the primary collector when the suppressed event information might be of value to the user.
Implementing Specific-Event Suppression From a Primary Collector

To enable specific-event suppression in the primary collector, use one of these methods:

- While the primary collector ($0) is running, use the FILTER keyword of EMSCCTRL to specify a filter table or compiled filter that contains the event-suppressing directives (this command enables PLF).
- While $0 is running, invoke a program that uses the SPI interface to modify the filter criteria (for example, a program that contains an ADD command with the event-suppressing filter table or compiled filter as its object).

Implementing Specific-Event Suppression From an Alternate Collector

To enable specific-event suppression in an alternate collector, use one of these methods:

- When the alternate collector is first started, use the FILTER keyword of EMSACOLL to specify a filter table or compiled filter that contains the event-suppression directives. (This command enables PLF.)
- While the alternate collector is running, use the FILTER keyword of EMSCCTRL to specify a filter table or compiled filter that contains the event-suppression directives. (This command enables PLF.)
- While the alternate collector is running, invoke a program that uses the SPI interface to modify the filter criteria (for example, a program that contains an ADD command with the event-suppressing filter table or compiled filter as its object).

Implementing Specific-Event Suppression From a Distributor

To enable specific-event suppression in a distributor, use one of these methods:

- When the distributor is first started, use the FILTER keyword of the EMSDIST program to load the event-suppressing filter table or compiled filter. For a detailed description of EMSDIST command syntax options, see Section 13, EMS Programs.
- If the distributor is already running, invoke a program that uses the SPI interface to modify the filter criteria (for example, a program that contains an ADD command with the event-suppressing filter table or compiled filter as its object).
Types of Collector Events BDS and PLF Do Not Suppress

Many of the events generated by an EMS collector are not suppressed by either the BDS or pre-log filtration (PLF) facilities. These events concern the state and behavior of the collector. These events are not suppressed for one or more of these reasons:

- The event is crucial to the EMS architecture; for example, a file switch event that each log file must have as the first record in the file.
- Suppression of the event serves no purpose; for example, burst events. If BDS is enabled, you would want to see burst events.
- The event signifies a serious collector error; for example, an alternate collector shutdown.
- The event only occurs once or very infrequently; for example, a system load event.
- The event might be important to the operator; for example, a message discard event.

**Note.** EMS distributors do not exempt any of these events from suppression.
Burst Detection and Suppression

Types of Collector Events BDS and PLF Do Not Suppress
This part of the manual discusses EMS events and describes procedures involving standard events:

- Section 8, Reporting Events
- Section 9, Standard Events
- Section 10, Generating Standard Events
- Section 11, Procedure Calls for Standard Events
Part III: Designing and Implementing Event Reporting
EMS is a general facility for collecting and distributing event information, and it accepts event messages from NonStop Kernel subsystems, such as Pathway and Expand, and from user-written subsystems.

This section describes how your program can produce event messages and send them to EMS for collection and distribution. It also includes guidelines that describe the design of an event-reporting strategy that allows your subsystem to integrate smoothly into an operational environment that includes other user-written subsystems and subsystems for the NonStop server.

This section provides an overview and explanation of the philosophy behind the procedure in Section 10, Generating Standard Events, which documents in detail how to create events for a subsystem following the recommended standard for event generation.

Subsystem refers to a program that creates and reports event messages. EMS treats any program that can provide a subsystem ID as a subsystem and lets that program report event messages. Management application, or application, refers to a program that retrieves event messages, sends command messages, or both.

Designing and implementing an event-reporting strategy involves:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1: Decide What Events to Report</td>
<td>8-2</td>
</tr>
<tr>
<td>Task 2: Decide What to Include in Event Messages</td>
<td>8-5</td>
</tr>
<tr>
<td>Task 3: Create a Data Definition File</td>
<td>8-11</td>
</tr>
<tr>
<td>Task 4: Write the EMS Interface in Your Program</td>
<td>8-12</td>
</tr>
</tbody>
</table>

For a sample program that reports event messages, see Appendix B, Example of Reporting Events.

For details on these and other tasks, see Section 9, Standard Events, Section 10, Generating Standard Events, and Section 11, Procedure Calls for Standard Events.
Task 1: Decide What Events to Report

Determine what events your subsystem can detect, which of those events it should report to EMS, and which of those reported events should be considered critical or action events.

Task 1.1: Decide Which Detectable Events to Report to EMS

In general, your subsystem should produce an event message to report any occurrence that might affect how the system or network is managed or maintained. Be selective, however, to avoid overwhelming management applications and operations staff with messages that are of little help to them.

These guidelines can help you decide what to report for your subsystem:

- Report any significant unexpected occurrence, such as data loss or hardware or software failure.
- Report occurrences necessary for diagnosing a problem, such as state changes, path switches, and takeovers by backup processes, even if expected. For example, problem diagnosis applications might rely entirely on the event message log, and an unreported path switch could cause confusion.
- Report an action-attention event whenever your subsystem is unable to continue without operator intervention. When the operator has taken the appropriate action or the problem is otherwise cleared, report an action-completion event to indicate that operator intervention is no longer required.
- Report events only about the objects closely related to your subsystem. Your event message might mention an object controlled by another subsystem if that object had a supporting role in the event, but the object in your subsystem should be the principal subject of the event message. For example, if your subsystem cannot access a file because a disk volume is unavailable, your subsystem should report its own problem (inability to access a file) and assume that the disk process will report (or already has reported) any exceptional conditions relating to the disk volume.
- Report events of interest to multiple people or applications. Do not report events when the audience is limited and identifiable. For example, if a user supplies a file name and the file is not present, report the error to the user, but do not send an event message to EMS.
- Report events that result from an analysis of statistics (such as an error rate exceeding an established threshold value), but do not create an event message just to report statistics. The preferred way to provide access to statistics is to support a command message (such as STATS) that returns statistics in its response.
- Report each significant occurrence of an event, but not each event when a flurry of similar events occur because of the same basic problem (if your subsystem can
detect this situation). For example, a subsystem that performs retries when a failure occurs should send one event message after exhausting the retry count, rather than send one event message for each retry attempt. You might discover more subtle cases where it would be appropriate to condense redundant events into a single event message or to report only the first occurrence of an event.

- Report each event in only one event message. That is, each event message should be self-contained. If multiple messages are used to report the same event, there is no way to ensure that an application would retrieve all of them in order with no intervening messages.

**Task 1.2: Decide Which Reported Events Are Critical or Action Events**

Determine if any of the events you report to EMS should be treated as critical or action events. ViewPoint recognizes these two special classes of events as being different from normal events.

**Critical Events**

Critical events should be events that indicate significant loss of or damage to your subsystem’s environment.

Critical events are denoted by a special token, called ZEMS-TKN-EMPHASIS. If the emphasis token has the value TRUE, ViewPoint highlights the message in its display.

Choose your critical events carefully. If most events are displayed as critical events, the concept loses its significance.

Designate these event classes as critical:

- Potential or actual loss of data
- Loss of a major subsystem function
- Loss of fault-tolerance capability, such as a redundant resource or a failure-recovery function
- Loss of subsystem integrity (an unrecoverable internal error)

**Action Events**

Action events arise when a subsystem determines that a problem cannot be resolved without operator intervention, such as needing a tape mounted.

Follow these guidelines for choosing action events:

- Report an action event when your subsystem cannot continue its work until and unless the operator takes some action. Subsystems that control mechanical peripherals often have to resort to reporting action events, such as when tapes
need to be mounted or when printers run out of paper, but action events can be appropriate for other subsystems in other circumstances as well.

- Do not report an action event when only a specific user or process can resolve the problem. Assume that EMS will route the message through a filter that selects action events for ViewPoint to display for an operator. If an operator likely would not have the information needed to respond to the action event, an action event is not the appropriate mechanism for resolving the problem.

Action-event messages occur in pairs: an action-attention message informs the operator of what needs to be done, and an action-completion message tells the operator that the action situation has been resolved. A typical action-event scenario might progress as:

1. A subsystem encounters a problem for which operator assistance is required. It creates and sends an action-attention message that describes the situation and includes:
   - An action-needed token with the value TRUE, which indicates that operator action is required
   - An action-ID token, which helps to identify the event
   - As the subject of the event message, a token containing the name of the resource that needs attention
     The action-ID token, the subject token, subsystem ID, and system number are all necessary to specify an action event uniquely.

2. ViewPoint retrieves the event message, through a filter that passes action events, and displays the message for the operator.

3. The operator corrects the problem. The subsystem might be able to detect that the problem was corrected. If it cannot, the operator must inform the subsystem through the operator interface to that subsystem.

4. The subsystem creates an action-completion message, which contains:
   - An action-needed token (with the value FALSE this time)
   - The action-ID token (with the same value as in the corresponding action-attention message)
   - The SUPPRESS-DISPLAY token with a value of TRUE
   - The name of the resource that was serviced as the subject of the event message

5. The subsystem sends the message to EMS.

6. ViewPoint retrieves the action-completion message through its filter. It compares the system number, subsystem ID, action-ID token, and subject with the corresponding tokens in the action-attention messages currently outstanding and dims the matching attention message on the operator display.
Both messages in the action-event pair should include the appropriate action tokens because applications use those tokens to match the action-attention message with the corresponding action-completion message.

Because the operator might overlook a displayed event message or, rarely, an event message might be lost, your subsystem should reissue any action-attention event that has not been remedied within an appropriate period of time. Applications retrieving action events should treat an identical copy of a currently outstanding action-attention message as if it were an action-completion message for the previous issuance of that action-attention message. (That is, the old copy should be made inactive by the new copy so that only one copy of the action event is active at a time.)

**Task 2: Decide What to Include in Event Messages**

Event messages are based on tokens and are built in a standard manner by the EMS routines as described in Section 15, EMS Procedures.

The EMS routines require that many common event-message components be passed as parameters, ensuring that all event messages have at least the basic facts. However, a lot of room remains for subsystem variation in event messages. This subsection describes:

- Information automatically provided for you when the event message is initialized
- Components you are required to include in event messages
- Components you can optionally include in event messages

**General Guidelines**

When designing how your subsystem will report events, remember a few basic principles:

- Because any event message has two possible audiences—applications and operators—it is important that the message convey meaningful information to both audiences. The tokenized information should contain a complete description of the event for applications, and the text should summarize the event for operators.

- The event messages should be self-contained, each fully describing a particular event, for these reasons:
  - Because you cannot ensure that a filter that retrieved a particular event message would retrieve the continuation event message (if there were one), do not depend on a second message to carry additional information about the same event to the same recipient.
  - Because the operator or application might have to take remedial action based only on the information in the message, include everything the recipient needs in order to respond with an appropriate interactive or programmatic command.
Event messages are almost always filtered before reaching an interested party, so ensure that the information is presented in a way that allows many filtering options. Some examples of filtering considerations:

- It is more convenient to filter on a token at the top level than on a token within a list. Present any information that a user might want to include in the filtering criteria as a token at the top level, as opposed to packaging it inside a list.
- If you think a user might want to retrieve a particular class of event messages (for example, all protocol errors of a certain type), consider defining a token to identify the message class to make it easier for the user to write a filter to extract a particular class of messages.
- It is more difficult for the filter programmer to extract information presented as an array of generic tokens than as a set of unique tokens. Avoid using arrays of tokens unless they contain values to which the same semantics apply. For example, if the message contains a starting time and an ending time, define two tokens (such as TKN-START-TIME and TKN-END-TIME) rather than using the same token (such as TKN-TIME) twice.

In general, make things as easy as possible for the recipients of event messages. A particular event-message type is implemented only once but is filtered, retrieved, and displayed many times.

**What Is Provided for You**

This information in the event message is always automatically provided by EMS:

- The length of the final event message (ZSPI-TKN-USEDLEN)
- The maximum version of fields in structures (ZSPI-TKN-MAX-FIELD-VERSION)
- The time at which the event was generated (ZEMS-TKN-GENTIME) if you do not specify a time
- The node, processor number, PIN, process descriptor, and user ID of the reporting process (ZEMS-TKN-NODENUM, ZEMS-TKN-CPU, ZEMS-TKN-PIN, ZEMS-TKN-PROC-DESC, ZEMS-TKN-USERID)

The EMS collector that receives the event message provides the time at which the message was written to the collector’s log file (ZEMS-TKN-LOGTIME).

**What You Must Provide**

Some tokens are required for all EMS messages, and some are required only for certain kinds of messages. For a detailed description of required tokens, see Section 14, EMS Definitions.
What You Must Provide

The ZEMS-TKN-EMPHASIS token is present in every event whether you specify it or not. When not specified, its value defaults to FALSE. Although not required, you should set it to TRUE in critical events to maintain compatibility with subsystems for NonStop server.

Subsystem ID

In order to report event messages, your program must use a subsystem ID: a unique identifier in the standard SPI form. This value is required as a parameter to EMSINIT and is assigned to the ZSPI-TKN-SSID token in the header.

Event Number

Each kind of event message that a subsystem can generate must have an associated event number that is unique within that subsystem. The combination of the subsystem and the event number defines the content and interpretation of the event message. Event numbers are defined as values for the ZEMS-TKN-EVENTNUMBER token. By convention, event numbers have symbolic names of the form subsys-EVT-name.

After extracting the subsystem ID and event number, an application should be able to predict, within certain boundaries, what tokens are present in the message. To help ensure that their programs continue to work from release to release, application and filter programmers should rely only on the presence and meaning of tokens that the subsystem developer has indicated are stable and unconditionally present in every instance of an event message with a given event number.

To reduce the likelihood of a filter or an application becoming obsolete, ensure that programmers writing programs to access event messages produced by your

---

Table 8-1. Summary of Required Tokens

<table>
<thead>
<tr>
<th>Token</th>
<th>Contents</th>
<th>When Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>Subsystem ID</td>
<td>Always</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>Event number</td>
<td>Always</td>
</tr>
<tr>
<td>your subject token</td>
<td>Subject token</td>
<td>Always</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>Critical or not critical</td>
<td>Always (by convention)</td>
</tr>
<tr>
<td>ZSPI-TKN-MANAGER</td>
<td>Name of manager process</td>
<td>When subject is not otherwise unique or when necessary to identify management interface process for subject</td>
</tr>
<tr>
<td>ZEMS-TKN-ACTION-NEEDED (nonshared)</td>
<td>Action attention or action completion event</td>
<td>When event is an action event</td>
</tr>
<tr>
<td>ZEMS-TKN-ACTION-ID (nonshared)</td>
<td>Action event identifier</td>
<td>When event is an action event</td>
</tr>
</tbody>
</table>
subsystem are aware of which tokens in each of your messages are unconditional and stable—that is, tokens that are fundamental to the meaning of that message and unlikely to be changed or deleted in future versions of your software. If you must delete an unconditional token or change its meaning, indicate this by changing the value of the event-number token.

For subsystems for NonStop server, the management programming manual for each subsystem indicates which tokens are unconditional (present in all instances of event messages with that particular event number) and which tokens are not to be assumed to be always present, either because they are conditional (present in only certain instances of that event message) or volatile (subject to change or deletion).

**Subject Token**

The subject token identifies the entity that is most directly involved in the event. The subject of the event could be a hardware component, such as a controller or processor, or a software component, such as a process, a protocol layer, or a file. A subject token should define both the type and the name of the object. For example, a subsystem could have a subject token called BANK-TKN-ATMNAME of type ZSPI-TYP-STRING. The token name indicates that the subject is an automated teller machine (ATM), and the token value contains the name of that ATM in the form of a string.

Because the subject of an event could be virtually any object in the subsystem, and because object types and object names vary so much from subsystem to subsystem, subjects in event messages are identified by a marker token. You provide the appropriate subject token as a parameter to EMSINIT (and EMSADDSUBJECTS if you are adding additional subjects), and the EMS routine inserts two tokens into the buffer: a subject mark (ZEMS-TKN-SUBJECT-MARK) and your subject token. This subject mark is used by EMSGET when the application programmer or filter programmer requests the subject token using the special token ZEMS-TKN-SUBJECT.

These guidelines can help you define the subjects of your event messages:

- One subject token is required in an event message. However, you can include additional subject tokens, when appropriate, to report that multiple objects were involved in the event or that several components might have been the subject of the event, but the specific one is uncertain. Multiple subject tokens are also appropriate when the subject is known by more than one name.

- For the management application or operations staff to control or inquire about the event subject, the subject must be uniquely identifiable. That is, include enough detail about the subject and subsystem that the specific instance of the object can be identified. For most subsystems, the subject name and the subsystem ID uniquely identify the object. For some subsystems, names are unique only within each instance of the subsystem. In this case, include the ZSPI-TKN-MANAGER token to specify which instance of the subsystem the subject belongs to.

- The subject of the event does not need to be an object that can be inquired about through a command message to your subsystem. Sometimes it is appropriate to be more specific in an event message (for example, to identify a certain ATM
component instead of the ATM itself). Sometimes the process producing the event message does not have the same view of the environment as the process that supports the programmatic command-response interface for that subsystem (or the subsystem might not support a command-response interface).

- For action events, identify the resource that requires attention as the subject of the event message. The identified resource should be the same for the action-attention and action-completion messages and should be the only subject in each message.

**Emphasis Token**

The emphasis token (ZEMS-TKN-EMPHASIS) indicates how serious you consider the event you are reporting. A value of TRUE indicates that the message is reporting a critical event, as described in Critical Events on page 8-3. A value of FALSE (the default value) indicates that the event is not critical.

**Manager Token**

Include a manager token (ZSPI-TKN-MANAGER, of type file name) if the subject token might not uniquely identify the subject or if, for any other reason, the application retrieving the message needs more information before it can determine how to access the appropriate object through its command-message interface.

For example, if a subsystem uses file names to identify its event-message subjects, the subject names are unique within the system, and a manager token is not needed to resolve ambiguities. However, Pathway uses subject names that must be interpreted in the context of the particular Pathway subsystem that created the event message, so Pathway uses the manager token to report the process name of the PATHMON process controlling the subject.

In general, the manager token should contain the name of the process to which a command message for that subsystem would be routed. This process might or might not be the process that an application issuing a command message would actually open. For example, for Pathway, the PATHMON process is both the process named in the manager token and the process the application would open. But for EXPAND, the application opens SCP (Subsystem Control Process), which then routes the command to the manager process named in the manager token. This concept of the manager token is compatible with the concept of the manager token in DNS. For more information, see the Distributed Name Service (DNS) Management Operations Manual.

**Action Tokens**

An action event requires two event messages: an action-attention message and an action-completion message. Each message contains a token called ZEMS-TKN-ACTION-NEEDED, which serves two purposes:

- It indicates that the message is an action event.
• Its value distinguishes an action-attention message (the value is TRUE) from an action-completion message (the value is FALSE).

Identify the resource that needs service by making it the subject of the event message. Both messages of the action-event pair must identify the same subject, using the same name and token for that resource.

To distinguish multiple outstanding action requests on the same subject from the same subsystem, use the ZEMS-TKN-ACTION-ID token, which contains an integer chosen by the subsystem to make this action request unique. If the action event is already unique within the subsystem by using the subject name, you must (by convention) still supply ZEMS-TKN-ACTION-ID but assign it the value 0 (zero).

The event numbers for the attention and completion messages need not be the same. Thus, a given type of action-attention event could be cleared by any one of several types of action-completion events, depending on the circumstances.

An application matches an action-completion event with its corresponding action-attention event by comparing the contents of the tokens ZEMS-TKN-SSID, ZEMS-TKN-SYSTEM, ZEMS-TKN-ACTION-ID, and the subject of the event message. The application should ignore the version field of the subsystem ID, however.

What You Can Optionally Provide

In addition to the required information, you should often add other potentially valuable information to an event message to make it as easy as possible for the message recipient to extract information. When deciding what additional information to include, consider:

• Additional information about the subject of the event message. For example, it might be useful to include status information about the subject of the event message.

• Information about the subsystem. For example, an internal program location might help someone using a listing of the subsystem code as an aid in resolving a problem.

• Additional information about a library procedure or another subsystem’s error that was directly involved in the event. You can add this information as an error list within your subsystem’s event message. (See the SPI Programming Manual.)

• The generation time of the event if it is important that it not be allowed to default to the time that EMSINIT was called. (The value for ZEMS-TKN-GENTIME is an optional parameter to the EMSINIT routine.)

Be conservative about the amount of information you add. You can add tokens later, if the need arises, without affecting applications and filters that use the existing tokens.

You can choose the format of the information you decide to include in event messages, but consider these guidelines:
• Avoid packing several items as bit fields within a word because such fields can be
difficult for an application to extract.

• Follow familiar formats for familiar items. An easy way to ensure compatibility with
NonStop Kernel subsystems and with the expectations of application programmers
retrieving your event messages, is to use the appropriate ZSPIDDL-defined types
for your tokens.

• If your subsystem also supports a command-message interface, use the same
tokens in event messages that you have defined for the command interface
wherever the two sets overlap.

• To avoid confusion on the part of the application retrieving event information
reported by your subsystem, do not use the same token to represent values whose
interpretations are different even if their formats are the same. A token should have
the same meaning every time it appears in any event message from a given
subsystem. For example, a token defined to contain the name of a source file
should never be used to contain the name of a destination file.

• File names should appear in remote form only if the file is on a remote node
relative to the process reporting the event (that is, if the file resides on a system
other than the one designated by the value of ZEMS-TKN-SYSTEM provided by
EMSINIT).

• Always try to use names that are independent of the system configuration, if
possible, because the configuration context is not always available to the
application retrieving the event message. For example, do not use logical device
numbers as file names (that is, report $SYSTEM, not $21) because reconfiguring
the system might result in the assignment of different logical device numbers.

Task 3: Create a Data Definition File

Any program can send information to a primary or alternate collector to be written to
the log file associated with that collector. For example, a program can write a simple
text message by doing a WRITE to $0, as described in the Guardian Programmer’s
Guide. However, if you want your program to send SPI-compatible event messages
(event messages containing tokens that aid in filtering and processing the messages
programmatically), your program must use a subsystem ID: a structured value that
uniquely identifies the subsystem.

1. Define the subsystem ID in the Data Definition Language (DDL) source file that
includes all the definitions required for the programmatic interface or interfaces to
your subsystem.

2. Use this file to produce data definition files in the language in which your
subsystem is written and in any languages used to write applications that will
retrieve event messages from your subsystem.
For specific information about how to define a subsystem ID, see the *SPI Programming Manual*. For more information about DDL, see the *Data Definition Language (DDL) Reference Manual*.

For an explanation on how a DDL dictionary (not a source file) is used to create event templates and store other information for a subsystem, see Section 9, *Standard Events*, Section 10, *Generating Standard Events*, and the *DSM Template Services Manual*.

The DDL file must contain definitions for tokens that appear in event messages created by your subsystem and, if your subsystem also supports a command-response interface, for tokens in your command and response messages as well. For event messages, for example, the DDL file would include definitions for these (and other) items in addition to the subsystem ID:

- Symbolic names representing the possible values of the ZEMS-TKN-EVENTNUMBER token
- Tokens that represent the possible subjects of the event messages
- Tokens that contain information describing the event that occurred

**Task 4: Write the EMS Interface in Your Program**

To implement the EMS interface that you have designed, include these steps in your program:

1. Open an EMS collector process: either the primary collector ($0) or an alternate collector.
2. Build an event message for each type of event your subsystem reports.
3. Send the event messages to the collector.
4. Close the collector.

**Example: Writing an EMS Interface Into a Program**

This example shows how to use these four steps to implement the EMS interface to your program.

**Step 1: Open an EMS Collector**

To open an EMS collector process, use the OPEN procedure, passing either $0 or the alternate collector name as the file name. READ/WRITE access mode and SHARED exclusion mode are required but need not be specified, as they are assumed by default. This TAL example illustrates this call:

```
INT COLNAME [0:11] := [ "$0", 11*" " ];
INT FILENUM := -1;
```
CALL OPEN (COLNAME, FILENUM);

In COBOL85, use ENTER COBOL85^SPECIAL^OPEN, rather than the OPEN verb, to open the FD when you are using messages containing tokens. Do not use this statement when you are using ASCII text.

Step 2: Build Event Messages

Because an event message has a specialized header and other unique characteristics, building one requires a different set of procedures than building a command or response message. (Because the two message types are similar, however, it is instructive to learn the way command-response messages are created. For this information, see the *SPI Programming Manual*.)

These EMS routines provide assistance in building event messages:

- **EMSINIT** Initializes the event-message buffer, inserting information such as the subsystem ID, the event number, and the subject of the event.
- **EMSADDTOKENS** Adds up to four tokens to an event-message buffer that has already been initialized by EMSINIT.
- **EMSADDSUBJECT** Inserts an additional subject in an initialized event-message buffer. You must use this procedure, not EMSADDTOKENS, to add more subject tokens to an event message.

Most information to be added to the event message is passed to the EMS routines as parameters, but the routines can derive some information themselves (such as the caller’s system, PIN, and process descriptor).

For a detailed description of the EMS routines, see Section 15, EMS Procedures.

Step 3: Send an Event Message

To send an event message to a primary or alternate collector, use the WRITEREAD procedure. Pass to the WRITEREAD procedure the file number returned from your OPEN call, the event-message buffer you created using the preceding EMS routines, and the length of the buffer. This TAL example illustrates this WRITEREAD call:

```
! INT FILENUM is assumed from OPEN example above
INT .EVENT^BUF [0:MAX^EVENT];
INT EVENT^SIZE;

! Call EMSINIT, EMSADDTOKENS, and so on.
...

! Call SSGETTKN to determine buffer length--
CALL SSGETTKN ( EVENT^BUF,
    ZSPI^TKN^USEDLEN,
    EVENT^SIZE );
```
Example: Writing an EMS Interface Into a Program

! Call WRITEREAD to send event-message buffer to $0--
CALL WRITEREAD ( FILENUM,         ! from OPEN
    EVENT^BUF,       ! event-message buffer
    EVENT^SIZE,      ! from SSGETTKN above
    0 );             ! read count

The read count must be 0 (zero) when sending an event message. WRITEREAD is used instead of WRITE to indicate to the collector that the buffer is an event message and not ASCII text.

In COBOL85, use the WRITE verb (not READ WITH PROMPT, as you might expect) to send the event message to the collector.

**Step 4: Close an EMS Collector**

To close an EMS collector, use the CLOSE procedure. Pass the CLOSE procedure the file number returned by the corresponding OPEN call, as this TAL example shows:

! INT FILENUM is assumed from OPEN example above.

CALL CLOSE ( FILENUM );

For more information about procedure calls, see the *Guardian Procedure Calls Reference Manual*. 
Standard events are a small set of events that subsystems and applications should generate to support operations management and conform with future enhancements to these standard events. The small number of standard events improves many aspects of event management.

This section describes standard events for anyone interested in using events to manage NonStop systems, including:

- Developers who generate EMS events in their subsystems and applications
- Developers who write management applications and EMS filters
- Operators who rely on EMS events to manage their systems

Each company—HP, third parties, and customers—should use this information to determine the management functions and EMS events that its subsystems or applications will support:

- Subsystems that support a management function should, at a minimum, generate the events that are needed to support that function.
- Subsystems that comply with standard events should, at a minimum:
  - Follow the rules defined for all standard events.
  - Provide all the tokens defined for that event.
  - Use the standard EMS template defined for that event to display the event.
- Subsystems that define their own events to support other management functions should follow the rules for creating extensions to standard events.

To understand standard events, you need a basic knowledge of EMS, including the concepts of EMS events, event tokens, how EMS events are collected and logged, and the externals of EMS procedures such as EMSINIT, EMSADDTOKENS, EMSADDSUBJECTS, and EMSGETTKN. This information is provided throughout this manual and in the SPI Programming Manual.
Introduction to Standard Events

This section describes events that subsystems and applications should generate to support operations management on NonStop systems. Detailed content of each event is specified as are the circumstances under which to generate the event, which EMS template to use to display each event, and how programmed operators (management applications) and human operators should use these events.

The set of standard events is derived from requirements of known existing management applications. Subsystem and application developers can add their own tokens to these standard events or define their own events to support additional management functions. To ensure some consistency between the subsystem-defined events and standard events, subsystem and application developers should use the same framework in describing their management functions and follow the same procedures in deriving their event content.

If your management decides to support standard events, adhere to all statements in this section whenever possible.

These terms are used throughout this section:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem application</td>
<td>Any program that generates EMS events.</td>
</tr>
<tr>
<td>Management application</td>
<td>A program that uses EMS events to manage a NonStop system.</td>
</tr>
<tr>
<td>Program operator</td>
<td>A program that performs management tasks usually performed by a human operator. The term is interchangeable with management application. Operator by itself refers to both human and program operators.</td>
</tr>
<tr>
<td>Management function</td>
<td>A task performed by an operator or a function provided by a management application to manage a NonStop system or network. A standard management function is one supported by one or more standard events.</td>
</tr>
<tr>
<td>Standard event</td>
<td>An event that contains all the tokens defined by the standard in this section for that event—all required common tokens and all required event-specific tokens. They could contain tokens defined by the subsystem or application.</td>
</tr>
<tr>
<td>Subsystem-defined event, private event</td>
<td>An event defined by a subsystem or application, which should contain all the common tokens defined for a standard event and any tokens defined by the subsystem or application. Also known as standard-conforming events because they can interoperate with standard events.</td>
</tr>
<tr>
<td>Nonstandard event</td>
<td>Any other event, such as one defined prior to the establishment of standard events. Nonstandard events typically do not have all the required common tokens of standard events.</td>
</tr>
</tbody>
</table>
Requirements for Standard Events

These events are designed to support requirements from known existing management applications running on NonStop systems. These requirements should be satisfied for future enhancements to these standard events. The requirements are:

- Each standard event supports a well-defined management function or operation. Events without a clear purpose should not be defined. This table shows management functions and the type of events that support them:

<table>
<thead>
<tr>
<th>Management Function or Operation</th>
<th>Supporting Standard Event Type</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object state monitoring</td>
<td>Object Unavailable</td>
<td>9-4</td>
</tr>
<tr>
<td></td>
<td>Object Available</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Object Other State Change</td>
<td></td>
</tr>
<tr>
<td>Reactive problem management</td>
<td>Object Unavailable</td>
<td>9-5</td>
</tr>
<tr>
<td>Proactive problem management</td>
<td>Usage Threshold</td>
<td>9-8</td>
</tr>
<tr>
<td></td>
<td>Transient Fault</td>
<td></td>
</tr>
<tr>
<td>Production requests requiring operator attention</td>
<td>Operator Attention Needed</td>
<td>9-9</td>
</tr>
<tr>
<td></td>
<td>Operator Attention Completed</td>
<td></td>
</tr>
</tbody>
</table>

- Standard events support the batch environment.

  Standard events contain a batch job ID token only if the object is part of a batch job (has a job ID). Batchjob ID is a conditional token.

- Standard events drive automation.

  All standard events should contain sufficient information to perform the intended management function. The information must have the same syntax and semantics, whenever possible, across all subsystems and applications. This makes writing management applications easier for the intended management function or operation.

- Standard events avoid generating a deluge of events by a single system condition.

  Subsystems and applications that have many objects that a single system condition could affect should be able to generate one event for all these objects, instead of one event for each object. Standard events let subsystems define a group name that represents all their objects of the same type or let them specify multiple objects in the event.

  A subsystem that has many objects that a single system condition could affect should not generate one event for each object. The potential volume for such events is too high.

- Standard events are presented to the human operator in a consistent manner.

  Information in EMS events is represented by EMS tokens. Tokens are designed for programs, not for human operators. EMS event viewers, such as ViewPoint, format
and display these tokens for the human operator according to the EMS template provided for the event.

You define an EMS template for each standard event so that information is presented consistently to operators. Subsystem and application developers are expected to use these standard templates for their standard events.

With each event, EMS event viewers also display advisory text—additional text which is defined by the subsystems and applications that generate the event, and which contains:

- Probable cause of the condition reported in the event
- Effect on the system from the condition reported in the event
- Recovery actions the operator can take to correct or bypass the condition reported in the event

- Event text for the human operator should be easily translated to other languages.

Standard events should not contain tokens with text for the human operators. All text for human operators should be defined using EMS templates. Standard templates make translation easier because there are far fewer templates to translate.

Object State Monitoring Functions

Monitoring the state changes of objects is often used by problem management applications to help detect, isolate, and sometimes diagnose hardware and software failures. Object-state-monitoring products such as Distributed Systems Management Solutions (DSMS), service tools used by customer engineers, and automated operator products such as Programmable Network Administrator (PNA), rely on events to provide the state change information for objects. These management applications need to know:

- When an object becomes available for use

An object is available when it is supplying the service it was designed to provide at any level of performance, quality, or response time; specifically, when the operator has enabled the object for use and all the underlying services that the object depends on are available for use. Objects from another subsystem or the operating system could supply these services. For example, a SNAX/XF LU needs the SNAX/XF LINE to be available before it can become available, or the object must have enough memory before it can run. Report the Object Available event when the object enters this state.

- When an object becomes unavailable for use

An object is unavailable when it cannot supply the service it was designed to provide at any performance level, quality, or response time; specifically, when the operator has disabled the object for use or the underlying services that the object depends on are not available. Report the Object Unavailable event when the object enters this state.
enters this state. Report this same event every time the operator tries but fails to bring the object back into service.

- When an object needs an operator to change its state

An object is in an intervention-needed state when it needs the operator to change its state. For example, a Pathway terminal needs the intervention of an operator if it is to be taken out of the suspended state, so the suspended state of a Pathway terminal is of interest to an operator. Report the Object Other State Change event when the object enters this state. If the object enters a state that is better described as unavailable, report the Object Unavailable event instead.

- When an object persists in a state longer than expected

An object is in a persistent state when it stays in a state longer than expected and the operator should take notice. For example, if SNAX/XF takes longer than expected to activate a Physical Unit (PU) or an application takes longer than expected to connect to the network, report an Object Other State Change event for the operator to take notice.

There is a subtle difference between persistent and transient states. In general, a state change is persistent if the state remains long enough for management applications to take action. It is transient if the state does not remain long enough for a management application to intervene. Do not report transient object states that do not require operator intervention in events because they do not facilitate problem management, and the potential volume of such events is very high.

There are no restrictions on the design of the internal states of subsystems and applications. Subsystem and application developers are responsible for defining the internal states of their objects, but they should report state changes that conform to the requirements defined here.

Reactive Problem Management Functions

A problem is any incident that results in the loss of a system resource. When the problem has already occurred, reactive problem management involves problem detection through final resolution, including tracking and control:

1. Problem detection and isolation. An operator should be able to determine the loss of availability of a system resource and how to isolate the problem to the failed hardware, firmware, or software component.

   Hardware failures deal with hardware objects such as CPU, memory, controller, peripheral device, fan, or power supply.

   Firmware failures deal with software that is downloaded to devices such as the disk controller or tape controller.

   Software failures deal with the loss of service of any software objects like a process, a network connection, a subdevice, a protocol layer, or any function provided by a subsystem or application.
2. Problem diagnosis. An operator should be able to detect the specific cause of a problem and the action required to resolve it.

3. Problem bypass and recovery. An operator should be able to bypass a problem, if necessary, until the problem can be resolved. The decision whether to bypass or not is a tradeoff between the cost to the enterprise due to the loss of a failed component and the cost in providing the bypass capability.

4. Problem resolution. An operator should be able to initiate the action necessary to repair or replace a failed component.

5. Problem tracking and control. An operator should be able to track a problem from its detection through its final resolution. A database of previous problems helps correlate incidents to their underlying root-cause problems and thereby helps provide a timely recovery. The correlation mechanism should be able to determine whether a problem is new or the recurrence of a known problem (problem rediscovery).

After a problem occurs, many Object Unavailable events are usually reported by subsystems and applications that are affected, directly or indirectly, by the problem. The problem management application has the difficult tasks of isolating the event that contains the actual cause of the problem from events that describe the effects of the problem, and of determining whether the event is reporting a new problem or the recurrence of a known problem. The next subsection describes how to perform these tasks using the Object Unavailable event.

Identifying the Actual Cause of the Problem

The Object Unavailable event contains a token called change reason that indicates why the object went out of service. Possible values for this field are:

- Normally terminated. The object stopped normally.
- Operator initiated. The operator took the object down.
- System initiated and error is within subsystem itself. The object failed because of an internal error.
- System initiated and error is due to failure in underlying service. The object failed because an underlying service on which this object depended failed.
- Unknown. The problem is unknown (avoid using this value).

The first three values describe the actual failure causes, and operators do not have to look for the failure in other subsystems or applications.

For failure in underlying service, the name of the dependent object is in the event. The problem management application likely indicates that the object failed as a result of the failure of another object, possibly an object in another subsystem or application. This event contains information about the type and name of the underlying object that failed.
To find the event with the actual cause of the problem, use the problem management application to search the EMS log for an Object Unavailable event with a subject the same as the underlying object specified in this event. Repeat this search algorithm until it finds an Object Unavailable event that contains the actual cause of the failure. Because related events are generated close in time, examine only events with generation times close to this event; in general, within a few minutes. You can use filters to assist the EMS Distributor in the search.

Event generation time and object names in the Object Unavailable event help locate the event that contains the actual cause of the problem. This is possible only if all subsystems report the unavailability of their objects using the Object Unavailable event, follow the recommended conventions for naming their objects, and do not use unknown as the change reason.

Problem Rediscovery

One aspect of problem tracking and control is to determine if an Object Unavailable event is reporting a new problem or the recurrence of a previously known problem. To help with this task, the Object Unavailable event contains a field called symptom string, which uniquely identifies where in the subsystem or application code the fault manifested itself. This lets management applications quickly isolate the fault to a given piece of software in the system, and to differentiate among problems reported by a subsystem and determine if a similar problem was already reported. If a similar problem was reported, the operator does not waste effort rediagnosing the problem, and can quickly bypass or correct it.

The symptom string—in ASCII form—should contain:

- The release version update of the subsystem, including any version information (ID, date, and so on) that uniquely identifies a piece of software in the system.
- The subsystem module name. A unique name within the subsystem; for example, the name of a procedure where the fault occurred.
- An identifier within the module name; for example, a code statement label that indicates where the fault occurred in the module.

If the fault manifests itself as inconsistent data, logic error, or any errors that a subsystem or application can detect before it aborts its service, the subsystem should construct and report the symptom string in the Object Unavailable event.

If the fault manifests itself in a CPU or system freeze or in a NonStop Kernel trap, the subsystem cannot construct and report the above symptom string. In this case, the operating system constructs the fault code—halt code for CPU or system freeze, trap code for a NonStop Kernel trap—and the code location of the process when the fault occurs. This information helps identify the problem.
Proactive Problem Management Functions

Proactive problem management deals with managing problems that might, but have not yet, occurred. This involves predicting, from received EMS events, whether to take actions to prevent an object from becoming unavailable or performing at less than full capacity. The Object Monitoring Facility (OMF) provides some of these functions.

Transient Faults

Transient faults are faults in the system that were automatically recovered by the system—such as correctable memory error, retryable controller error, line or network resets. These faults, if they persist, could lead to the loss of a system resource. Report the Transient Fault event when the objects encounter the transient faults. To prevent flooding the EMS collector, do not report the Transient Fault event for every encounter if they take place in a very short time interval. Report the Transient Fault event only after every few occurrences. If the transient fault occurs continuously, the subsystem or application should consider the fault permanent and take the object out of service; in this case, it should report an Object Unavailable event.

Use of System Resources

Use level of an object or resource can indicate a gradual degradation in the availability of the object (for example, the use of the communication line is reaching its theoretical limit) or it could signal the impending loss of an object (for example, a critical file is 80 percent full.) In general, any object that is critical to the operation of a subsystem or application should be monitored, and the Usage Threshold event should be reported when the usage level of the object exceeds the configured level.

Usually, subsystems and applications that control critical objects should monitor and report the Usage Threshold events. For certain resources, however, they are better monitored and reported outside the subsystems and applications that control or use them. These resources are usually system-wide resources used by many subsystems and applications.

The resources that subsystems and applications should monitor are:

- Data communication line utilization—specifies the percentage of the theoretical capacity of the line that is currently being used. The subsystem or application that controls the line divides the throughput by the theoretical line speed (both in number of bytes per second). Throughput is obtained by dividing the number of bytes of data sent over a time period by the same time period.

- Internal buffer usage—specifies the percentage of the buffer pool that is currently being used. The currently used space (in bytes or other units) is divided by the total space in the pool.

- Task queue length—specifies the number of requests waiting for service in the subsystem or application. A counter is incremented whenever a request is added to the service queue and decremented whenever a request is removed from the queue and serviced.
These system-wide resources are better monitored outside the subsystems and applications (making an individual subsystem or application unnecessary) by monitor programs such as the Object Monitoring Facility (OMF):

- CPU (percent busy)
- Disk files (percent file full)
- Disk volumes (percent volume full)
- Transaction response time (in units of time)

Subsystems, applications, and monitor programs that generate Usage Threshold events should provide for each object a configured high and a configured low usage level, which dictate when the Usage Threshold event for an object is reported. An operator should be able to configure these levels to control the type of resources and the levels at which their usages are reported.

The configured low level indicates the threshold level to which the object utilization must fall before its usage is reported (for example, measuring free disk space). The configured high level indicates the threshold level that the utilization of an object must reach or exceed before its usage is reported. Usage of an object is first reported when it reaches the configured high level, then when it drops below the configured low level, again when it rises above the configured high level, and so on.

Toggling between the configured high and configured low levels prevents many simultaneous events from being reported if the utilization of the object hovers around one of these levels. This algorithm follows the one defined by ISO in the document ISO/IEC 10165-2 dated August 1991: “Information Technology—Open Systems Interconnection—Structure of Management Information—Part 2: Definition of Management Information.”

Production Requests Requiring Operator Attention

When a subsystem or application cannot continue its work until the operator takes some action, an event message should be issued. Subsystems and applications that control or use mechanical peripherals often have such situations; for example, when tapes need to be mounted or when printers run out of paper.

Use the Operator Attention Needed event to inform the operator of what needs to be done. Use the Operator Attention Completed event to tell the operator that the action situation has been resolved. A typical scenario:

1. A subsystem encounters a problem for which operator assistance is required.
2. The subsystem creates and sends an Operator Attention Needed event message, specifying the service that needs attention and an ID identifying this request.
3. A management application, such as ViewPoint, retrieves the event message and displays the message for the operator.
4. The operator corrects the problem. If the subsystem cannot detect the operator has corrected the problem, it should provide an interface, like the command-response interface, so the operator can notify the subsystem when the problem is corrected.

5. The subsystem creates and sends the Operator Attention Completed event message, specifying the completed service and the same ID as in the original request.

6. The management application finds and dims the display of the outstanding Operator Attention Needed event that matches this Operator Attention Completed event.

Because the operator might overlook a displayed event message or, rarely, an event message might be lost, the subsystem should reissue any Operator Attention Needed event that is not attended to within an appropriate period of time. The operator treats the old copy of the event as inactive so that only one Operator Attention Needed event from the subsystem is active at a time.

Underlying Philosophy of Standard Events

All subsystems and applications should report and display certain conditions in the system and event messages to support the management functions described in Requirements for Standard Events on page 9-3. The conditions being reported are indicated by the names of the standard events:

- Object Available event
- Object Other State Change event
- Object Unavailable event
- Operator Attention Completed event
- Operator Attention Needed event
- Transient Fault event
- Usage Threshold event

An event message describes what happened in a system. The subsystem or application that detects the condition formats and sends the event message to EMS, which stores it in a log file. Management applications (including operators) use these messages to determine what actions to take in managing the system.

An event message includes many kinds of information besides what happened. Information generally found in every event message is:

- Information about the message itself—a value identifying it as an event message (as opposed to a SPI command or response message), a count of how long the message is, and a timestamp indicating when the message was generated.
Information about the subsystem or application reporting the message—a subsystem identifier (SSID), process identifiers, and other relevant information about the creator of the message. Some information is conditional; for example batch job ID. Standard events contain a batch job ID token only if the object is part of a batch job (has a job ID).

The SSID uniquely identifies the subsystem or application that defines and generates the event message. Information in the event message usually requires the SSID for interpretation unless it is standard event information. In that case, the information is unique across all subsystems and applications and does not require the SSID for interpretation.

Information about the subject of the event message—objects selected by subsystems and applications, such as the name or names of objects that the event reports on, that are the subjects of their events.

Information about the condition reported in the event message—an event number that uniquely identifies the message within the subsystem or application and indicates what happened to what object in the subsystem, and an event type that uniquely identifies the type of message in the system and indicates what happened in the subsystem. The event type is either explicitly defined in this section or implicitly assumed by default, and provides a quick indication of the conditions reported in the system.

Information about the actual incident — details about the occurrence and its environment.

Information in an event message is represented by EMS tokens. An EMS token describes the type, length, and value of a piece of information. An EMS template is associated with an event message, but not as tokens in the event message, to provide instructions for the display of the event message.

Certain event messages, like the Object Unavailable event, contain information about their cause, effect, and recovery. This information is also provided in an EMS template.

This section defines only the minimal set of tokens needed in a standard event to support a management function. It standardizes all token use except for the objects in a subsystem and the error or diagnostic information in describing the fault (that is, information specific to a subsystem that cannot be standardized). Standard events—and the standard templates defined with them—provide a minimal but common environment where a management application can make the same assumptions across all subsystems and applications regarding a management function. Standard events do not completely define all possible situations in the system reported in the event. Subsystems and applications should provide additional information in these events that can help operators better understand the subsystem conditions that these events report.

This section discusses some important underlying concepts of standard events. For details of a standard event, see Detailed Description of Standard Events on page 9-17. For details about the display of a standard event, see Description of Standard EMS Templates on page 9-34.
Subsystems determine the objects to report in an event. However, this section recommends some commonly used schemes for naming objects in events. Use one of these schemes to name your objects to minimize the amount of subsystem-specific knowledge a management application needs to learn.

For the requirements and the naming schemes for objects represented by a single group name, see Object Name for a Group of Objects on page 9-16. Group name is particularly important for subsystems that have many objects that could be affected by a single condition in the system. Group name lets the subsystem report a single event for all these objects instead of a single event for each of these objects.

**Event Numbers**

Every event message has an event number. The event number together with the subsystem ID (SSID) uniquely identifies an event message in a NonStop Kernel network, which lets management applications quickly identify the action needed for a particular object in the system. Design the event number to identify the subject type and condition within a subsystem or application. Subject type is the type of the object specified in the event subject, such as an X.25 line. The condition is the situation being reported for the event subject. For example, an object is unavailable.

For example, if an application has two object types: A and B, and both are reporting these conditions: object unavailable, object available, and object crossed usage threshold, the application will have six unique event numbers, one for each of these incidents:

- Event reporting object type A crossed usage threshold
- Event reporting object type A unavailable
- Event reporting object type A available
- Event reporting object type B crossed usage threshold
- Event reporting object type B unavailable
- Event reporting object type B available

An event number is associated with a set of tokens. The tokens that will be present are the tokens specified as unconditional when the event is defined. Using the event number and SSID information lets management applications predict the tokens in an event.

**Event Types**

Each standard event has a unique event type—the standard event type—specified by the Standard Content Type token. The event type identifies the condition, such as object unavailable, and the tokens in the event. Because these tokens are defined here, they are unique across all subsystems and applications and can be interpreted without the specific knowledge of the subsystem or application.
A standard event type lets operators quickly identify the kind of events they need to collect and process in their systems. Unlike event number, which lets management applications take action on a specific object type, event type lets management applications simply and efficiently process a group or class of events, regardless of which subsystem or application the events came from.

For example, if a management application only needs to see events of certain types, you can write a filter that only examines and forwards events of the needed type. Without event type, the filter would have to examine SSID and event number for each event and compare their contents to a list of needed values, a resource intensive and error-prone operation, and you would need to change the filter whenever a new subsystem or application is added to the system.

This section also specifies event types for some commonly reported information, like trace, debug, or diagnostics information for problems in the production environment. The content of these events is defined by the subsystems—the required common tokens are defined here. These event types are specified by the User Content Type token in the event message. This token is different from the Standard Content Type token so these event messages can be migrated to standard events when the content of such an event is standardized. Providing these event types (without the event content) lets operators easily separate these subsystem-defined events from the standard events.

Events that do not have the Standard Content Type and the User Content Type tokens belong to the class of events defined prior to this standard. The content of these events can only be interpreted using the event number and the SSID. Detailed knowledge of the subsystem or application is required.

**Object Name for Event Subject**

Every event message has an event subject—the objects that the event is most concerned with. Any object name included in an event subject:

- Must not change before the object is deleted
- Should uniquely identify the object from any point in the network

If this is not possible, the subsystem or application must identify the entity—for example, the name of a manager process, a directory service or an algorithm—that is needed to resolve the name in the network.

- Should suggest the commands that can be used to control or inquire about the object

Use the object type information associated with an object to suggest the command set that applies to the object. The object type for an object is usually encoded with the object name. Hence, subsystems and applications must ensure a unique object type for an object.

For example, a subsystem could have an object token called BANK-TKN-ATMNAME. The token name BANK-TKN-ATMNAME indicates the event subject is
an automated teller machine (ATM), and the token value contains the actual name of that ATM.

- Should uniquely identify the process, or control point, in the NonStop Kernel network where management applications can send commands to control or inquire about the object.

If the name cannot identify the control point, the subsystem or application must specify the name of a manager process if that is the control point.

The naming scheme for an event subject depends on the type of object. In general, consider the name of an object a handle to be consistent with the naming philosophy of the D-series and later operating systems. The ways to name an object are:

- Object is an operating system object and defined within a single process.

If the object is an operating system object—addressable, and within the control of the NonStop Kernel operating system—and defined within a single process of name node.$process, you must use one of these formats to uniquely name the object within the process:

\node.$process.#subdev.#subdev
\node.$process.#subdev
\node.$process

If the $process name is also the control point where commands for the object can be sent, subsystem and application developers do not need to supply other information in the event to identify the control point. They could identify a different control point, such as the name of a manager process, if necessary.

- Object is an operating system object but not defined within a single process.

If the object is an operating system object but not defined within a single process—such as CPU, bus, backplane, fan, and power supply—name the object using node.anything.

Subsystem and application developers must specify the entity, providing the name of the manager process in the event or describing the algorithm used to resolve the uniqueness of these names or to identify the control point.

- Object is not an operating system object.

If the object is not an operating system object, the object can be named using whatever format best describes the object. The name used must be easily differentiated from one of the preceding form.

Subsystem and application developers must specify the entity, providing the name of the manager process in the event or describing the algorithm that is used to interpret the name.
Object Name for Underlying Object

When an object that a subsystem communicates with directly, but that is defined and controlled in another subsystem, goes out of service, an Object Unavailable event is reported. If the object went out of service because another object that this object depended on has became unavailable, the name of the other object should be in the event message. Management applications use this name to locate the Object Unavailable events reported by other subsystems and applications that contain the actual cause of the problem.

For example, subsystems A, B, and C have objects a1, b1, and c1, respectively; and the availability of c1 depends on the availability of b1, which in turn depends on a1:

```
subsystem C  subsystem B  subsystem A
object c1    object b1    object a1
```

When object a1 becomes unavailable because of some internal error, object b1 is taken out of service because its underlying object a1 is not available. Likewise, object c1 is taken out of service because its underlying object b1 is not available. These Object Unavailable events are reported to EMS:

- Subsystem A: subject=a1 cause=internal error
- Subsystem B: subject=b1 cause=underlying obj failed
  underlying object name=a1
- Subsystem C: subject=c1 cause=underlying obj failed
  underlying object name=b1

The Object Unavailable event for c1 is linked to the Object Unavailable event for b1 because the underlying object name in event c1 is the same as the subject name for event b1. Likewise, event b1 is linked to the Object Unavailable event for a1. Because event a1 contains the actual cause of the problem, further linking of related events is no longer necessary.

For a subsystem or application using the service of an underlying object, the name of the underlying object reported in the Object Unavailable event must be the name used to establish communication between itself and the provider of the underlying object:

- If File System FILE_OPEN_ alone is needed to establish communication with the underlying object, the name used in the FILE_OPEN_ should be the name for the underlying object.
- If File System FILE_OPEN_ plus other mechanisms are needed to establish communication with the underlying object (for example, a multithread server might require an additional internal name to identify the thread of communication), the name used in the FILE_OPEN_ procedure, plus any internal names, should be the name for the underlying object. One or more tokens in the event can represent the internal names.
Object Name for a Group of Objects

The object name in an event can represent a single object or multiple objects. A single name that represents multiple objects is a group name. A group name can be used for any objects in an event. If it is used as an event subject, it can reduce the number of events a subsystem generates. If it is used to replace a list of objects in an event, it can reduce the size of an event message.

A group name in an event must meet these criteria:

- All objects in the group are of the same type.
- The group must include all subordinate objects of a parent that are of the same type.
- The command and control interface must support commands that operate on all group names supplied in the events.
- The event recipient must be able to take the group name from the event and ask the subsystem to return a list of the individual members. The subsystem must be able to answer such a query regardless of the state of any of the objects in the group or of the state of other objects in the subsystem.

An event recipient must be able to tell the difference between a group object name and an individual object name. Differentiation can be done in two ways:

- Token identification, which refers to having group object name tokens that are different from individual object name tokens. For example, the token identifier ZXXX-TKN-LINEGROUP would contain a group name whereas the token identifier ZXXX-TKN-LINE would contain an individual line name.
- Using a name with a wild-card character; for example, onw with an asterisk (*) appended to the end. The presence of the wild-card character indicates that the
name in the token is a group. This style of group naming is consistent with the second level of support for object template naming used by communication subsystems internally at HP.

The wild-card format is allowed because it is an established way for naming a group of objects by certain subsystems. The new token identifier format is allowed because not all objects use the wild-card naming scheme.

Subsystem developers should use one of these schemes to represent their group names. Do not proliferate another scheme as it will complicate the design of management applications. The scheme a subsystem uses should be clearly documented.

Detailed Description of Standard Events

Information in an event message is represented by EMS tokens. Every event message contains a set of tokens called common-standard tokens and another set of tokens called event-specific tokens.

Common-standard tokens are tokens that must be present in all event messages. They are defined in this section. Some common-standard tokens, which are automatically provided by EMS when the subsystem or application uses the procedure provided by EMS (for example, EMSINIT) to build its event buffer, are described in Common-Standard Tokens That EMS Provides on page 9-18. Other common-standard tokens, which must be provided by the subsystem or application when it builds the event buffer, are described in Common-Standard Tokens That Subsystems Provide on page 9-20.

Event-specific tokens are tokens that are present only in the event of that type. For standard events, the event-specific tokens are defined in the subsection where that event is described. For events defined by subsystems (private events), the event-specific tokens are defined in that subsystem or application.

EMS tokens defined in this section are for the NonStop Kernel only. Each token is described by this information:

- The DDL name of the token and the standard values assigned to the token. The DDL name is specified as ZSPI-TKN-??? or ZEMS-TKN-??? and is defined in the standard SPI definition file (ZSPIDDL) or the standard EMS definition file (ZEMSDDL), respectively.

- The DDL name of the token type. The DDL name is specified as ZSPI-TYP-??? and is defined in the standard SPI definition file ZSPIDDL.

- The condition under which the token is to be provided. “U” means the token is an unconditional token and must always be provided. Many unconditional tokens are automatically generated for you, which means you do not create them using procedures. Descriptions of the tokens and guidelines for the token generating procedures indicate if you need to create them.
“C” means the token is a conditional token and must be provided if the specified condition applies; if the token is not provided, the default meaning specified is to be associated with the absence of the token. Some tokens, for example ZEMS-TKN-EMPHASIS and ZEMS-TKN-SUPPRESS-DISPLAY, are always in an event, even if you did not add or generate them. If you do not add or generate them, these tokens receive default values. However, the ZEMS-TKN-BATCHJOB-ID token exists only if the object of the event is a batch job. The ZEMS-TKN-BATCHJOB-ID token is not automatically generated if the object of the event is not a batch job.

- A complete description of the meanings and rules regarding the token:
  - If the token is a structure, the DDL names of the fields is provided
  - If the token is an enumerated type, the DDL names for all the enumerated values are defined
  - If the token is a conditional token, its default meaning is defined in case the token is not specified

### Common-Standard Tokens That EMS Provides

These tokens are provided by EMS automatically when the subsystem or application uses the EMS procedures (EMSINIT, EMS_???_EVT_BLD_ and so on) to format the event message and uses the EMS collector to log the message.

These tokens are listed here for completeness. Unconditional token types are identified with a “U.”

<table>
<thead>
<tr>
<th>Generation Timestamp</th>
<th>ZEMS-TKN-GENTIME (ZSPI-TYP-TIMESTAMP,U)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The Greenwich mean time used by the reporting subsystem when it created the event message. In most cases, it is close to the time the event occurred.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Log Timestamp</th>
<th>ZEMS-TKN-LOGTIME (ZSPI-TYP-TIMESTAMP,U)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The Greenwich mean time the collector used when it wrote the event message to its log file. Event messages are ordered in the log files by log time, not generation time.</td>
</tr>
</tbody>
</table>
## Common-Standard Tokens That EMS Provides

<table>
<thead>
<tr>
<th>Sender</th>
<th>ZEMS-TKN-NODENUM</th>
<th>(ZSPI-TYP-INT2, U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process ID</td>
<td>ZEMS-TKN-CPU</td>
<td>(ZSPI-TYP-UINT, U)</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-PIN</td>
<td>(ZSPI-TYP-UINT, U)</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-PROC-DESC</td>
<td>(ZSPI-TYP-STRING, U)</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-USERID</td>
<td>(ZSPI-TYP-BYTE-PAIR, U)</td>
</tr>
</tbody>
</table>

The Expand system-number, CPU number, PIN, process descriptor, and user ID of the process reporting the event. These tokens reflect the sender only when WRITEREAD(X) is used to send the event to the collector. If the management application only handles events (tokenized) sent by WRITEREAD(X), these tokens can be used to get sender ID data. However, if the management application must also process the HP numbered console messages or events (text, non-tokenized) sent by WRITE(X), then the special EMS tokens, ZEMS-TKN-XSENDERID and/or ZEMS-TKN-XSENDERID-PD must be supplied to the EMSGET or EMSGETTKN procedure.

<table>
<thead>
<tr>
<th>Max field Version</th>
<th>ZSPI-TKN-MAX-FIELD-VERSION</th>
<th>(ZSPI-TYP-VERSION, U)</th>
</tr>
</thead>
</table>

The highest field version among the nonnull fields of the extensible structures added to the buffer with EMSADDTOKENS.

<table>
<thead>
<tr>
<th>Bytes used</th>
<th>ZSPI-TKN-USEDLEN</th>
<th>(ZSPI-TYP-INT, U)</th>
</tr>
</thead>
</table>

The number of bytes actually used in the event message buffer. A subsystem can use this to determine how many bytes to send to a collector through the WRITEREAD procedure.
Common-Standard Tokens That Subsystems Provide

These tokens are defined by HP but provided by subsystems and applications when they create the event message. These tokens are required in all event messages. Unconditional token types are identified with a “U;” conditional token types are identified with a “C.”

<table>
<thead>
<tr>
<th>Subsys ID</th>
<th>ZSPI-TKN-SSID</th>
<th>(type structure,U)</th>
</tr>
</thead>
</table>

A structured token with these fields:

- **Z-OWNER** subsys owner (8-character string)
- **Z-NUMBER** subsys number (16-bit signed integer)
- **Z-VERSION** subsys version ID (16-bit unsigned integer)

SSID identifies the subsystem or application that defines, owns and reports the event. It must be unique within a NonStop Kernel network.

The value of SSID should be **subsys-VAL-SSID**, where **subsys** is the 4-letter acronym assigned to the subsystem or application by the company that owns the subsystem. For HP, it always begins with “Z” followed by the 3-character abbreviated subsystem ID. For customers and third-parties, it can be anything but must not begin with “Z.” For example, ZPWY-VAL-SSID is the SSID definition for the HP Pathway subsystem.

The field **Z-OWNER**, known as subsystem owner, identifies the name of the company or organization that provides the subsystem or application. For subsystems and applications written by HP, this field must be “TANDEM.” For subsystems and applications written by third parties and customers, this field must be registered with HP to guarantee that it is unique throughout a network of NonStop systems. For registration instructions, see **Section 10, Generating Standard Events**.

The field **Z-NUMBER**, known as subsystem number, identifies the subsystem or application within the set of subsystems and applications provided by the owner of the subsystem.

The field **Z-VERSION**, known as subsystem version ID, identifies the software release version update of the subsystem. The value of this field should increase from one release to the next. For subsystems and applications provided by HP, this value should match the three-character release ID (one character and two digits) in the product RVU of the subsystem. For example, if the product RVU for an subsystem for the NonStop server (EMS in this case) is T9632C21^11APR90^..., C21 must also be the value of this field. The left byte is the ASCII character C, and the right byte is the unsigned integer value of 21.
marked by ZEMS-TKN-SUBJECT-MARK  (ZSPI-TYP-MARK,U)

Identifies the objects that are most directly involved in the event. The token following the token ZEMS-TKN-SUBJECT-MARK is the subject name.

The subject of the event could be a hardware component, such as a controller or processor, or a software component, such as a process, a protocol layer, a file, a network connection, a subdevice, a transaction thread, or any named function provided by the subsystem.

An event can contain multiple subjects; that is, more than one object specified in an event. An event can contain a group name as a subject. The group name refers to a collection, or group, of objects. If multiple subjects or a group name is used, all objects must be of the same type and have experienced the same condition. Using multiple objects and group names in the subject minimizes the number of events generated. Subsystem and application developers should decide whether to use single subject, multiple subjects, group object, or any combination of these in their events.

The event subject is provided as a parameter to EMSINIT (and EMSADDSUBJECTS if there is more than one subject.) The EMS routines insert these two tokens in the event message: ZEMS-TKN-SUBJECT-MARK, a subject mark token followed by the subject token. EMSGET uses the subject mark token when the management applications request the subject token using the special token ZEMS-TKN-SUBJECT.

Certain requirements must be met for names in the event subject. Additional criteria must be satisfied if the name is also a group name. For more details, see Underlying Philosophy of Standard Events on page 9-10.
Manager Name
ZEMS-TKN-NAME-MANAGER (ZSPI-TYP-STRING,C)
Identifies the manager process for objects in the event subject. It must be in \node.$process form.

If this token is present, it is the name of the process where commands can be sent to inquire or control the object reported in the event subject.

If this token is not present:
- If the event subject is a process name, the process name is the \node.$process part of the event subject name.
- If the event subject is not a process name, the location where commands can be sent or the event subject names can be interpreted must be documented by the subsystem. The location might be an external command handler where commands for the subject can be sent or an external directory service or algorithm that can be used to interpret the event subject name.

For more details, see Underlying Philosophy of Standard Events on page 9-10 and Object Name for Event Subject on page 9-13.

In general, if this process name is the management interface that accepts SPI commands, the management application must be openable by applications by appending .#ZSPI to the process name. This process might be the process that an application issuing a SPI command would actually open. For example, for PATHWAY, the PATHMON process is both the process named in this token and the process the management application would open. For EXPAND, the application would open SCP (Subsystem Control Process), which then routes the command to the manager process named in this token.

If more than one object is reported in the event subject, all objects must be managed by the same manager process.

Event Number
ZEMS-TKN-EVENTNUMBER (ZSPI-TYP-ENUM,U)
Identifies the event number of the event message. Enumerated values are defined by the subsystem and should have this symbolic or literal name:

\[\text{subsys-EVT-eventname}\]

where subsys is the 4-character subsystem acronym assigned to this subsystem (see ZSPI-TKN-SSID token above) and eventname provides descriptive information for the event number. For example, ZSX1-EVT-PU-NOT-READY is the symbolic event name for the PU not ready event in the SNAX/XF subsystem.

An event number identifies the specific situation being reported. It must uniquely identify the subject type and condition within the reporting subsystem or application. The values are defined by the subsystem or application. For more details, see Event Numbers on page 9-12.
Standard Events

Common-Standard Tokens That Subsystems Provide

**Standard Content Type**

ZEMS-TKN-CONTENT-STANDARD (ZSPI-TYP-ENUM,C)

Indicates the type of standard event. Standard enumerated values are:

- ZEMS-VAL-NULL (default)
- ZEMS-VAL-TRANSIENT-FAULT
- ZEMS-VAL-OBJECT-UNAVAILABLE
- ZEMS-VAL-OBJECT-AVAILABLE
- ZEMS-VAL-OTHER-STATE-CHANGE
- ZEMS-VAL-ATTN-NEEDED
- ZEMS-VAL-ATTN-COMPLETED
- ZEMS-VAL-USAGE-THRESHOLD

If an event is one of the events defined in this standard, this token must be present, and its value must be its event type. If an event is not one of the standard events, this token must not be present, or its value must be EMS-VAL-NULL.

This token can co-exist with a ZEMS-TKN-CONTENT-USER type token.

**User Content Type**

ZEMS-TKN-CONTENT-USER (ZSPI-TYP-ENUM,C)

Identifies the type of a subsystem-defined event. Standard enumerated values are:

- ZEMS-VAL-NULL (default)
- ZEMS-VAL-DATA-TRACE
- ZEMS-VAL-DATA-DEBUG
- ZEMS-VAL-DATA-DIAGNOSTIC

Subsystems can add their own values, but they must be greater than or equal to ZEMS-VAL-MIN-USER-VALUE.

If an event is a subsystem-defined event and contains trace, program debug, or diagnostic information for problems in a production environment, this token must be present to indicate the type of information conveyed by the event. Otherwise this token should not be present, or its value must be ZEMS-VAL-NULL.

This token can coexist with the Standard Content Type ZEMS-TKN-CONTENT-STANDARD token. When both type tokens are present, it indicates the event contains tokens for both the standard subsystem-defined event. This usually means the event was a private event defined by the subsystem initially and then enhanced subsequently—by adding other tokens—to become a standard event.
Critical Indicator

ZEMS-TKN-EMPHASIS  (ZSPI-TYP-BOOLEAN, C)
Indicates the critical nature of the condition reported in the event. Standard values are:

ZSPI-VAL-TRUE   indicates event is critical.
ZSPI-VAL-FALSE  indicates event is not critical (default).

An event is critical if it reports any of these conditions:

- Potential or actual loss of data
- Loss of a major subsystem function
- Loss of fault tolerance, such as loss of a redundant resource or loss of a failure-recovery function
- Loss of subsystem integrity such as an unrecoverable internal error

If an event is critical, this token should have the value ZSPI-VAL-TRUE. Otherwise, it should have the value ZSPI-VAL-FALSE.

Suppress Display

ZEMS-TKN-SUPPRESS-DISPLAY  (ZSPI-TYP-BOOLEAN, C)
Indicates whether the event should be displayed on an operator console like ViewPoint. Standard values are:

ZSPI-VAL-TRUE   indicates not to display
ZSPI-VAL-FALSE  indicates to display          (default)

If an event is not to be displayed on the operator console, this token must be present with the value ZSPI-VAL-TRUE. Otherwise, this token should have the value ZSPI-VAL-FALSE. An event should only be displayed if it is meaningful to the operator. Events that provide trace, debug, diagnostic, status, or historical information are helpful for analyzing subsystem problems by system analysts or developers. Such events are not very useful to the operator, so they should not be displayed.

This token existed prior to standard events; therefore, it is necessary to preserve the semantics of this token here. New management applications should use other criteria, such as the type of event, instead of this token to determine if an event should be displayed. Only ViewPoint (of the HP products) uses this token in determining whether to display. $Z0 and EMSDIST ignore this token.

Batch Job ID

ZEMS-TKN-BATCHJOB-ID  (ZSPI-TYP-UINT, C)
Identifies the ID of the batch job associated with the object or the process that controls the object. Standard events contain a batchjob ID token only if the object is associated with a batch job (has a job ID). Batchjob ID is a conditional token.
Object Available Event Tokens

An Object Available event must be generated for a subsystem or application object whenever the object becomes available.

The subsystem or application (or its manager process) that controls the object is responsible for reporting this event. This event should be reported as soon as the object becomes available, including at system and application startup time.

The subject of this event is the object that became available. If multiple objects became available for the same reason, these objects can be represented by lists or group names.

For more information about the use of this event, see Requirements for Standard Events on page 9-3 and Object State Monitoring Functions on page 9-4.

These event-specific tokens must be provided in addition to the common-standard tokens. Unconditional token types are identified with a “U.”

Current State

ZEMS-TKN-STATE-CURRENT (ZSPI-TYP-ENUM, U)
Indicates the new state of the object. Subsystems and applications should provide the state information specific to their environments. After a state value is reported in the event, its meaning cannot change.

Previous State

ZEMS-TKN-STATE-PREVIOUS (ZSPI-TYP-ENUM, U)
Indicates the previous state of the object. Subsystems and applications should provide the state information specific to their environments. After a state value is reported in the event, its meaning cannot change.

Change Reason

ZEMS-TKN-CHANGE-REASON (ZSPI-TYP-ENUM, U)
Indicates the reason the object became available. Standard enumerated values are:

ZEMS-VAL-OPERATOR-INITIATED
ZEMS-VAL-UNDERLYING-UP
ZEMS-VAL-REASON-UNKNOWN

Subsystems can add their own values, but they must be greater than or equal to ZEMS-VAL-MIN-USER-VALUE.

ZEMS-VAL-OPERATOR-INITIATED indicates that the operator was made available. ZEMS-VAL-UNDERLYING-UP indicates that the object has come up because the underlying object it depends on has become available.

ZEMS-VAL-REASON-UNKNOWN indicates a the subsystem does not indicate a cause for the object's availability or the reason is not listed. Subsystems should avoid using this reason.
Object Other State Change Event Tokens

An Object Other State Change event must be reported for a subsystem or application object whenever the object enters a state, other than object available or unavailable, with these qualities:

- Requires operator intervention before its state can change
- Persists in the same state long enough for the operator to notice

The subsystem or application (or its manager process) that controls the object must report this event as soon as the condition is detected. The subsystem or application must also provide the cause, effect, and recovery text of this event to the operator.

Do not use this event to report:

- Operator attention needed for production requests like mounting a tape. Use the Operator Attention Needed event.
- That an object became unavailable or that an object became available. Use the Object Unavailable and Object Available events, respectively.

The event subject is the object with one of the previous state changes. If multiple objects have the same state change, these objects can be represented by lists or group names.

For more information about the use of this event, see Requirements for Standard Events on page 9-3 and Object State Monitoring Functions on page 9-4.

These event-specific tokens must be provided in addition to the common-standard tokens. Unconditional token types are identified with a "U." These tokens are described in Object Available Event Tokens.

- Current State: ZEMS-TKN-STATE-CURRENT (ZSPI-TYP-ENUM, U)
- Previous State: ZEMS-TKN-STATE-PREVIOUS (ZSPI-TYP-ENUM, U)
- Change Reason: ZEMS-TKN-CHANGE-REASON (ZSPI-TYP-ENUM, U)

Object Unavailable Event Tokens

An Object Unavailable event is to be generated for a subsystem or application object whenever:

- The object becomes unavailable.
- The operator cannot restart the unavailable object.

The subsystem or application (or its manager process) that controls the object must report this event. Subsystems and applications should not generate events for objects that are not under their control. This event should be reported as soon as the unavailability condition of the object is detected. The subsystem or application must also provide the cause, effect, and recovery text of this event to the operator.
The subject of this event is the object that has became unavailable. If multiple objects went out of service for the same reason, these objects can be represented by lists or group names.

For more information about the use of this event, see Object State Monitoring Functions on page 9-4 and Reactive Problem Management Functions on page 9-5.

These event-specific tokens are to be provided in addition to the common-standard tokens. Unconditional token types are identified with a “U.” Conditional token types are identified with a “C.”

<table>
<thead>
<tr>
<th>Current State</th>
<th>ZEMS-TKN-STATE-CURRENT (ZSPI-TYP-ENUM, U)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates the new state of the object. Subsystems and applications should provide the state information specific to their environments. After a state value is reported in the event, its meaning is not allowed to change.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Previous State</th>
<th>ZEMS-TKN-STATE-PREVIOUS (ZSPI-TYP-ENUM, U)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates the previous state of the object. Subsystems and applications should provide the state information specific to their environments. After a state value is reported in the event, its meaning cannot change.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Change Reason</th>
<th>ZEMS-TKN-CHANGE-REASON (ZSPI-TYP-ENUM, U)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates the reason that the object is not available. Standard enumerated values are:</td>
</tr>
<tr>
<td></td>
<td>● ZEMS-VAL-NORM-TERMINATION indicates the object was terminated normally.</td>
</tr>
<tr>
<td></td>
<td>● ZEMS-VAL-OPERATOR-INITIATED indicates the operator has taken the object out of service.</td>
</tr>
<tr>
<td></td>
<td>● ZEMS-VAL-INTERNAL-FAILED indicates an error was encountered inside the subsystem that caused the subsystem to take the object out of service. The error could be an internal error like program logic or invalid data, or it could be an external error like disconnected by remote network services. If external errors are from another subsystem that this object communicates with directly, ZEMS-VAL-UNDERLYING-FAILED should be the change reason value.</td>
</tr>
<tr>
<td></td>
<td>● ZEMS-VAL-UNDERLYING-FAILED indicates that the underlying service this object depended on failed. Underlying service is the service provided by another subsystem or application.</td>
</tr>
<tr>
<td></td>
<td>● ZEMS-VAL-REASON-UNKNOWN indicates the change reason is not known or not listed. Avoid use of this as much as possible because it significantly complicates the operator’s task of isolating the cause of the problem.</td>
</tr>
</tbody>
</table>
Object Unavailable Event Tokens

**Symptom String**

ZEMS-TKN-SYMPOTM-STRING (ZSPI-TYP-STRING,C)

Identifies where in the subsystem code the fault occurred. The information should point to a specific subsystem code location, and the information should remain unchanged for the same symptom from release to release of the program code. The purpose is to use this symptom string to help determine if a problem with similar symptoms has been reported previously.

The string is defined by the subsystem. It should be an ASCII string with this information:

- Release version update of the subsystem including version date and ID that will uniquely identify a given piece of software.
- Module name like the name of a procedure call where the fault occurred. The module name should be unique within the subsystem.
- Some identifier within the module name, like code statement label, where the fault occurred.

Provide this token if the change reason (ZEMS-VAL-INTERNAL-FAILED) indicates the error was encountered inside the subsystem.

**Underlying Object Name**

ZEMS-TKN-UNDERLYING-OBJ-NAME (ZSPI-TYP-STRING,C)

Indicates the name of the underlying object whose failure caused the object reported in the subject of this event to go out of service. It should be a fully qualified file name.

If the change reason is ZEMS-VAL-UNDERLYING-FAILED, this token must be present to indicate the name of the underlying object. Otherwise this token must not be present.

The name of the underlying object is the operating system object name—like process name, file name, or subdevice name—that the subject of this event communicates with. In most cases, this name alone is sufficient to identify uniquely the underlying object name that the subject of this event depended on. If not, the subsystem should provide additional tokens to uniquely identify the underlying object and describe how a management application can isolate the actual cause of the problem in these events.

**Private Tokens**

Additional event-specific tokens should be provided by subsystems that could help the operator diagnose the problem (for example, system procedure error code, network disconnect reason code, error information from underlying subsystem, and so on).
Operator Attention Completed Event Tokens

(This event was known as an action event prior to standard events.)

An Operator Attention Completed event must be reported by a subsystem or application whenever the request from an earlier Operator Attention Needed event has been completed. The subsystem or application that reported the Operator Attention Needed event is responsible for reporting this event.

The subject of this event should be the resource in the original Operator Attention Needed event message. The name must have the same format as in the original request.

For more information about the use of this event, see Requirements for Standard Events on page 9-3 and Production Requests Requiring Operator Attention on page 9-9.

These event-specific tokens must be provided in addition to the common-standard tokens.

<table>
<thead>
<tr>
<th>Action ID</th>
<th>ZEMS-TKN-ACTION-ID (ZSPI-TYP-INT,U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifies the action ID that was specified in the original Operator Attention Needed event.</td>
<td></td>
</tr>
</tbody>
</table>

**Note.** The Action ID token is not shared, which means it must always be qualified by the EMS SSID.

<table>
<thead>
<tr>
<th>Attention Needed</th>
<th>ZEMS-TKN-ACTION-NEEDED (ZSPI-TYP-BOOLEAN,U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicates whether the event is for operator attention or for operator attention not needed. Standard values are:</td>
<td></td>
</tr>
<tr>
<td>ZSPI-VAL-TRUE</td>
<td>indicates attention needed.</td>
</tr>
<tr>
<td>ZSPI-VAL-FALSE</td>
<td>indicates attention not needed.</td>
</tr>
<tr>
<td>Because this is an Operator Attention Completed event, this token must not be present or has value ZSPI-VAL-FALSE.</td>
<td></td>
</tr>
<tr>
<td>This token existed prior to this standard even; therefore, it is necessary to preserve the semantics of this token.</td>
<td></td>
</tr>
</tbody>
</table>

**Note.** The Attention Needed token is not shared, which means it must always be qualified by the EMS SSID.

Operator Attention Needed Event Tokens

(This event was known as an action event prior to standard events.)

An Operator Attention Needed event is to be reported by a subsystem or application whenever it needs an operator to take some action before it can continue its work, such as mount a tape or change paper on a printer. Do not confuse this event with the Other State Change event which requires operator intervention before the state of an object can be changed.
The subsystem or application (or its manager process) that needs the attention of the operator is responsible for reporting this event, and for providing the cause, effect, and recovery text of this event to the operator.

The subject of this event is the resource, such as tape or printer, that needs attention. The name of the resource should be a fully qualified operating system name. Only one resource per event should be specified.

For more information about the use of this event, see Requirements for Standard Events on page 9-3 and Production Requests Requiring Operator Attention on page 9-9.

These event-specific tokens must be provided in addition to the common-standard tokens. Unconditional token types are identified with a “U.” Conditional token types are identified with a “C.”

**Action ID**

ZEMS-TKN-ACTION-ID \(\text{(ZSPI-TYP-INT,U)}\)

Identifies the action ID for this request.

It is used by the management application to associate this request with the Operator Attention Completed event.

**Note.** The Action ID token is not shared, which means it must always be qualified by the EMS SSID.

**Attention Needed**

ZEMS-TKN-ACTION-NEEDED \(\text{(ZSPI-TYP-BOOLEAN,U)}\)

Indicates if the event is for operator attention or for operator attention not needed. Standard values are:

- **ZSPI-VAL-TRUE** indicates attention needed.
- **ZSPI-VAL-FALSE** indicates attention completed.

Because this is an Operator Attention Needed event, this token must be present and of value ZSPI-VAL-TRUE.

This token existed prior to this standard event; therefore, it is necessary to preserve the semantics of this token.

**Note.** The Attention Needed token is not shared, which means it must always be qualified by the EMS SSID.

---

**Transient Fault Event Tokens**

A Transient Fault event must be generated for a subsystem or application object whenever the object encountered a transient fault that could result in a permanent failure in the future. The subsystem or application (or its manager process) that controls the object is responsible for reporting this event.

The subject of this event is the object that experienced the transient fault. If multiple objects experienced the transient fault for the same reason, these objects can be represented by lists or group names.
For more information about the use of this event, see Requirements for Standard Events on page 9-3 and Proactive Problem Management Functions on page 9-8.

These event-specific tokens must be provided in addition to the common-standard tokens. Unconditional token types are identified with a “U.”

Usage Threshold Event Tokens

A Usage Threshold event is reported for a resource whenever the usage of the resource crosses a preconfigured threshold level. Usually, the subsystem that controls the resource (one that can take the resource in or out of service), or the subsystem that uses the resource must report this event. But it could also be the program that monitors the resource that must report this event. For more information about the use of this event, see Proactive Problem Management Functions on page 9-8.

The utilization level of a system resource is intended to supply a measure of how much an object is being used. The level could be specified as a percentage (between zero and one hundred) or as an absolute number. Subsystem developers are free to choose the best way to represent the utilization data of their objects.

If the level is specified as a percentage, zero means the object is absolutely idle over some period of time, and 100 means the object was never idle over the same period of time. The period of time for measurement (sampling time) that is meaningful is determined by the subsystem developer.

For certain objects, the utilization level is more appropriately specified in an absolute number rather than a percentage. For example, transaction response times are measured in units of time, queue lengths in units of requests, storage in units of bytes, and rate in units of bytes per second.

The subject of this event is the object whose usage threshold is being reported. If multiple objects experience the same changes and all have the same configured

Type of Fault

<table>
<thead>
<tr>
<th>ZEMS-TKN-TXFAULT-TYPE</th>
<th>(ZSPI-TYP-ENUM,U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerated values are defined by subsystems, and they must be greater than or equal to ZEMS-VAL-MIN-USER-VALUE.</td>
<td></td>
</tr>
<tr>
<td>Indicates the type of transient fault like correctable memory error, excessive line resets, or recoverable controller errors.</td>
<td></td>
</tr>
</tbody>
</table>

Private Tokens

Additional event-specific tokens should be provided by subsystems and applications that could help the operator diagnose the transient fault.

For example, for correctable memory error, the diagnostic information could be the hardware address of the memory chip that experienced the fault and the error code if known. For retryable controller error, the diagnostic information could be the error data that the controller returns to the I/O process like SNAX/XF or X25AM. For excessive network resets, the diagnostic information could be the error data provided by the network and the excessive reset count.
report_hi and report_low values, these objects can be represented by lists or group names.

These event-specific tokens must be provided in addition to the common-standard tokens. Unconditional token types are identified with a “U;” conditional token types are identified with a “C.”

<table>
<thead>
<tr>
<th>Current UtilizationLevel</th>
<th>ZEMS-TKN-UTIL-LEVEL-CURR (ZSPI-TYP-FLT, U)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Identifies the current utilization level being reported.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Previous UtilizationLevel</th>
<th>ZEMS-TKN-UTIL-LEVEL-PREV (ZSPI-TYP-FLT, C)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Identifies the previous reported utilization level. This token is always required except for the first time the event is reported.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Previous Timestamp</th>
<th>ZEMS-TKN-UTIL-TIME-PREV (ZSPI-TYP-TIMESTAMP, C)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Identifies the time the last utilization level was reported. This token is always required except for the first time the event is reported.</td>
</tr>
</tbody>
</table>
## Standard Events

### Usage Threshold Event Tokens

#### Configured High Level to report

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-UTIL-CONFIG-HI</td>
<td>Identifies the configured usage value that triggers event generation when usage has exceeded this value.</td>
</tr>
</tbody>
</table>

#### Configured Low Level to report

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-UTIL-CONFIG-LOW</td>
<td>Identifies the configured usage value that triggers event generation when usage has fallen below this value.</td>
</tr>
</tbody>
</table>

#### Unit of Measure

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-UTIL-UNIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-PERCENT</td>
<td></td>
<td>Unit in %</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-USEC</td>
<td></td>
<td>Unit in microseconds</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-MSEC</td>
<td></td>
<td>Unit in milliseconds</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-HSEC</td>
<td></td>
<td>Unit in hundredth-of-seconds</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-TSEC</td>
<td></td>
<td>Unit in tenth-of-seconds</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-SEC</td>
<td></td>
<td>Unit in seconds</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-MIN</td>
<td></td>
<td>Unit in minutes</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-HOUR</td>
<td></td>
<td>Unit in hours</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-BPS</td>
<td></td>
<td>Unit in bytes/sec</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-KBPS</td>
<td></td>
<td>Unit in kilobytes/sec</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-MBPS</td>
<td></td>
<td>Unit in megabytes/sec</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-GBPS</td>
<td></td>
<td>Unit in gigabytes/sec</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-TBPS</td>
<td></td>
<td>Unit in terabytes/sec</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-BYTES</td>
<td></td>
<td>Unit in bytes</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-KBYTES</td>
<td></td>
<td>Unit in kilobytes</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-MBYTES</td>
<td></td>
<td>Unit in megabytes</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-GBYTES</td>
<td></td>
<td>Unit in gigabytes</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-TBYTES</td>
<td></td>
<td>Unit in terabytes</td>
</tr>
<tr>
<td>ZEMS-TKN-UTIL-UNIT-COUNTER</td>
<td></td>
<td>Unit in count</td>
</tr>
</tbody>
</table>
Description of Standard EMS Templates

Information in standard events is encoded by tokens, which are designed for programs and not intended for use by human operators. EMS templates are used to provide formatting instructions for the display of these tokens by operator console products like ViewPoint.

This subsection defines a standard EMS template for each standard event. Subsystem and application developers use these templates to display their standard events so that information presented to the operator is consistent and uniform among all standard events.

Subsystem and application developers can customize these templates, as described in Section 10, Generating Standard Events.

Note. Standard templates are for systems running D-series or later RVUs only. The actual EMS templates, using the template compiler language, are described here. For an explanation of the template language, see the DSM Template Services Manual.

Object Available Event Standard Template

MSG:  ZEMS-TKN-CONTENT-STANDARD, ZEMS-VAL-OBJECT-AVAILABLE
       "Object available  <31><32> - <31><33>", event number: <1>
       ", reason: <2>
       ", previous state: <3>
       ", current state: <4>
       "<*IF 21>, manager: <11><*ENDIF>"
       "<*IF 22>, batch ID: <12><*ENDIF>"
       "<*IF 23>, user content: <13><*ENDIF>"

1:ZEMS-TKN-EVENTNUMBER
2:ZEMS-TKN-CHANGE-REASON, ENUM
3:ZEMS-TKN-STATE-PREVIOUS, ENUM
4:ZEMS-TKN-STATE-CURRENT, ENUM
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
13:ZEMS-TKN-CONTENT-USER, ENUM
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)
31:ZEMS-TKN-SUBJECT-MARK
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE
Object Other State Change Event Standard Template

MSG:  ZEMS-TKN-CONTENT-STANDARD, ZEMS-VAL-OTHER-STATE-CHANGE
iT(Other) State Change <31><32> - <31><33>"
", event number: <1>"
", reason: <2>"
", previous state: <3>"
", current state: <4>"
"<IF 21>, manager: <11><ENDIF>"
"<IF 22>, batch ID: <12><ENDIF>"
"<IF 23>, user content: <13><ENDIF>"
1:ZEMS-TKN-EVENTNUMBER
2:ZEMS-TKN-CHANGE-REASON, ENUM
3:ZEMS-TKN-STATE-PREVIOUS ENUM
4:ZEMS-TKN-STATE-CURRENT ENUM
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
13:ZEMS-TKN-CONTENT-USER, ENUM
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)
31:ZEMS-TKN-SUBJECT-MARK
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE

Object Unavailable Event Standard Template

MSG:  ZEMS-TKN-CONTENT-STANDARD, ZEMS-VAL-OBJECT-UNAVAILABLE
iT(Object unavailable <31><32> - <31><33>"
", event number: <1>"
", cause: <2>"
", previous state: <3>"
", current state: <4>"
"<IF 24>, underlying object: <14><ENDIF>"
"<IF 25>, symptom string: <15><ENDIF>"
"<IF 21>, manager: <11><ENDIF>"
"<IF 22>, batch ID: <12><ENDIF>"
"<IF 23>, user content: <13><ENDIF>"
1:ZEMS-TKN-EVENTNUMBER
2:ZEMS-TKN-CHANGE-REASON, ENUM
3:ZEMS-TKN-STATE-PREVIOUS ENUM
4:ZEMS-TKN-STATE-CURRENT ENUM
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
13:ZEMS-TKN-CONTENT-USER, ENUM
14:ZEMS-TKN-UNDERLYING-OBJ-NAME
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)
24:TOKENPRESENT(ZEMS-TKN-UNDERLYING-OBJ-NAME)
31:ZEMS-TKN-SUBJECT-MARK
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE
Operator Attention Completed Event Standard Template

Note. The Action ID token is not shared, which means it must always be qualified by the EMS SSID.

MSG: ZEMS-TKN-CONTENT-STANDARD, ZEMS-VAL-ATTN-COMPLETED
"Operator attention completed for <31><32> - <31><33> thank you"
", event number: <1>"
", action ID: <2>"
"<*IF 21>, manager: <11><*ENDIF>"
"<*IF 22>, batch ID: <12><*ENDIF>"
"<*IF 23>, user content: <13><*ENDIF>"
1:ZEMS-TKN-EVENTNUMBER
2:SSID (ZEMS-VAL-SSID, ZEMS-TKN-ACTION-ID)
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
13:ZEMS-TKN-CONTENT-USER, ENUM
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)
31:ZEMS-TKN-SUBJECT-MARK
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE

Operator Attention Needed Event Standard Template

MSG: ZEMS-TKN-CONTENT-STANDARD, ZEMS-VAL-ATTN-NEEDED
"Operator attention needed for <31><32> - <31><33> please"
", event number: <1>"
", action ID: <2>"
"<*IF 21>, manager: <11><*ENDIF>"
"<*IF 22>, batch ID: <12><*ENDIF>"
"<*IF 23>, user content: <13><*ENDIF>"
1:ZEMS-TKN-EVENTNUMBER
2:SSID (ZEMS-VAL-SSID, ZEMS-TKN-ACTION-ID)
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
13:ZEMS-TKN-CONTENT-USER, ENUM
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)
31:ZEMS-TKN-SUBJECT-MARK
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE
Transient Fault Event Standard Template

MSG: ZEMS-TKN-CONTENT-STANDARD, ZEMS-VAL-TRANSIENT-FAULT
"Transient Fault <31><32> - <31><33>"
", event number: <1>"
", fault type: <2>"
"<*IF 21>, manager: <11><*ENDIF>"
"<*IF 22>, batch ID: <12><*ENDIF>"
"<*IF 23>, user content: <13><*ENDIF>"
1:ZEMS-TKN-EVENTNUMBER
2:ZEMS-TKN-TXFAULT-TYPE ENUM
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
13:ZEMS-TKN-CONTENT-USER, ENUM
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)
31:ZEMS-TKN-SUBJECT-MARK
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE

Usage Threshold Event Standard Template

MSG: ZEMS-TKN-CONTENT-STANDARD, ZEMS-VAL-USAGE-THRESHOLD
"Usage threshold <31><32> - <31><33>"
", event number: <1>"
", usage level (current): <2>"
"<*IF 24>, usage level (last reported): <3><*ENDIF>"
", usage unit: <4>"
", timestamp GMT (current): <5>"
"<*IF 25>, timestamp GMT (last reported): <6><*ENDIF>"
", configured usage level (high): <7>"
", configured usage level (low): <8>"
"<*IF 21>, manager: <11><*ENDIF>"
"<*IF 22>, batch ID: <12><*ENDIF>"
"<*IF 23>, user content: <13><*ENDIF>"
1:ZEMS-TKN-EVENTNUMBER
2:ZEMS-TKN-UTIL-LEVEL-CURR, FORMATDATA ("F16.3")
3:ZEMS-TKN-UTIL-LEVEL-PREV, FORMATDATA ("F16.3")
4:ZEMS-TKN-UTIL-UNIT, ENUM
5:ZEMS-TKN-GENTIME, TIME
6:ZEMS-TKN-UTIL-TIME-PREV, TIME
7:ZEMS-TKN-UTIL-CONFIG-HI, FORMATDATA ("F16.3")
8:ZEMS-TKN-UTIL-CONFIG-LOW, FORMATDATA ("F16.3")
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
13:ZEMS-TKN-CONTENT-USER, ENUM
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)
24:TOKENPRESENT(ZEMS-TKN-UTIL-LEVEL-PREV)
25:TOKENPRESENT(ZEMS-TKN-UTIL-TIME-PREV)
31:ZEMS-TKN-SUBJECT-MARK
Extensions to Standard Events

Subsystem developers can make these extensions:

- Enhance the meaning of standard events
  Subsystem developers should provide extra tokens to a standard event if these tokens help describe the conditions reported by the subsystem. Subsystem developers could, but are not required to, extend the standard template to display these tokens.

- Enhance the meaning of standard management functions
  Subsystem developers could define their own EMS events and templates to enrich a management function defined in this section.

- Support management functions not in this section
  Subsystem developers could specify their own EMS events and templates to support management functions, like job accounting, that are not defined in this section.

The tokens, events, and management functions defined by subsystems should not impact the existing management applications and should not require other subsystems or applications to change. The additions should only provide value-added functions to those defined here.

General Guidelines

- The tokens added to a standard event must not change the meaning of the standard event.

- The content of subsystem-defined events should be derived using the procedures described in Section 10, Generating Standard Events. They must support a well-defined management function and have the same assumptions as the standard events described in this section.

- If the tokens, events, and management functions defined by subsystems or applications are of general interest and could benefit other subsystems or applications, submit your additions for incorporation in this standard by sending requests through your HP account analysts.
10 Generating Standard Events

This section describes how to generate standard events and provides an example of a fictitious subsystem to help subsystem developers design, define, build, and release event messages:

Tasks 2, 3 and 6 should be used iteratively as you analyze and design your event messages. You might want to go through these tasks a couple of times to optimize and complete the design of your events.

For a description of standard events and how to use them, see Section 9, Standard Events.

For a description of procedure calls that help subsystems and applications format and generate standard events, see Section 11, Procedure Calls for Standard Events.

To help subsystem and application developers generate standard events, these are provided:

- The DDL definitions for the tokens of the standard events
- The EMS templates for the standard events
- The callable procedures to encode the standard events, to encode the common-standard tokens of private events, and to determine if a usage threshold event should be generated
Task 1. Determine Your Subsystem ID and Acronym

1. Register your company name with HP.
   The company name is the Z-OWNER field (8-character) of your SSID.

2. Request a subsystem number and name from your subsystem number keeper.
   The subsystem number is the Z-NUMBER field (16-bit signed integer) of your SSID.
   The number and name are assigned by the company that owns the subsystem and must be unique among all the subsystems owned by that company. The company should have someone designated as the keeper and assigner of all subsystem numbers and names.

3. Request a subsystem acronym from your subsystem acronym keeper.
   The subsystem acronym is a four-character ID assigned to your subsystem. You will use the acronym as a prefix for the names of all your tokens and as a prefix for the names of all your event definition files that you create.
   NonStop Kernel subsystem acronyms are, by convention, the letter “Z” followed by a three-letter subsystem abbreviation. For example, the HP Clock subsystem acronym is ZCLK. To ensure your subsystem acronym is unique, start your acronym with a letter other than “Z.”
   The acronym is assigned by the company that owns the subsystem and must be unique among all subsystems owned by that company. The company should have someone designated as the keeper and assigner of all subsystem acronyms.

Task 2. Analyze Your Subsystem Environment

Manageability is the focus on easing the burden of running a system. Like quality assurance, it must be an integral part of your subsystem design, not an afterthought. When you design or add functions to a subsystem, consider how to:

- Configure, control, or inquire about the functions in your subsystem
- Quickly recover from failures
- Anticipate problems (such as performance degradation) to prevent potential loss of subsystem service

Quick recovery from failures means minimizing the amount of time your subsystem or application is unavailable, and is necessary for providing a continuous availability environment where a 7 x 24 business function can be implemented on NonStop systems.
Provide automated recovery for your functions whenever possible. When you cannot, provide specific information about:

- The problem so that management applications can be written to quickly recover from failures
- Potential problems so that actions can be taken to prevent severe degradation in your subsystem service

Section 9, Standard Events, standardizes the information—EMS events and tokens—and the conditions under which to report this information to management applications. Thoroughly analyze your subsystem and its environment to determine the objects that should report these events and their error and diagnostic information. The more thorough your analysis, the more manageable your subsystem will be.

Task 2.1: Identify Types of Objects to Manage in Your Subsystem

1. Identify all objects that your subsystem controls, provides, or uses:
   - A hardware component (ATM machine, CPU, disk, controller, bus, channel, or modem) that your subsystem controls
   - A software component (process, file, transaction, database, or piece of microcode) that your subsystem controls or uses
   - A service (cash withdrawal function in an ATM, or reliable transport service) that your subsystem or application provides
   - A resource (tape, disk, or buffer pool) that your subsystem uses

2. Determine which objects to manage:
   - Think about the people and programs that will use your subsystem.
   - Think in terms of the services and the service level objectives—such as availability and performance—that your subsystem provides.
   - Objects needed by operators and management applications to help your subsystem maintain these service levels when one or more of your functions become unavailable (planned or unplanned) or when performance degrades.

3. Do not overlook:
   - Functions performed by your subsystem, especially those externalized by your programmatic or interactive user interfaces
   - Functions performed or provided by any utilities your subsystem provides
   - Services and resources your subsystem uses (Often overlooked when designing the management interface, they might be critical to the operation of your subsystem.)
Task 2.2: Identify the Characteristics of Your Objects

1. Define the names of each of your objects.

   Every object must be addressable within the network; that is, the object must have a name. A name is necessary because the object must be accessible separately from other objects in the subsystem. Both program and human interfaces should be able to determine the existence and name of any object. For requirements and possible name formats, see Underlying Philosophy of Standard Events on page 9-10.

2. Define the possible states of each of your objects.

   The behavior of an object can be completely represented by its states and the transition between these states.

   a. Define the states that describe the normal behavior of each object.

      Every object undergoes a set of states that defines the way it normally behaves and carries out its functions. The nature of these states is defined by the object. For example:

      - Up, down, and suspended are states usually associated with the behavior of a terminal
      - Disconnected, connecting, connected, disconnecting, restart, reset, and so on are states usually associated with a network connection

   b. Define states that describe system conditions which cause your object to be in a state—other than available and unavailable—that:

      - Could persist for a long time so an operator should take notice
      - Needs operator action to change
      - Transient faults could be detected in
      - Could let usage levels of the object severely degrade subsystem performance

      For example, if your object is a network connection that could take longer than expected to connect, define a new state in addition to the connecting state to indicate this condition. Section 9, Standard Events lists the set of conditions to consider to make your objects more manageable.

3. Determine the management data needed to manage your objects.

   Design management data to use the asynchronous management interfaces and the command and control interfaces.

Task 2.3: Identify the State Transitions of Your Objects

A state transition diagram defines the behavior of an object. To construct the diagram:
1. Define the valid states of your object.
2. Define the valid transitions between these states.
3. Define the possible conditions (induced internally or externally to your subsystem) that cause each transition.
4. Define the actions for operators or your subsystem to take after the transition (for example, report an EMS event, or restart the object).

The semantics of your object states reflect your object, and will vary between objects and subsystems. For uniformity in object states among subsystems, standard events define three object states:

<table>
<thead>
<tr>
<th>Object State</th>
<th>Standard Event Object Is Reported In</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available</td>
<td>Object Available</td>
</tr>
<tr>
<td>Unavailable</td>
<td>Object Unavailable</td>
</tr>
<tr>
<td>Other (persistent, need operator to change)</td>
<td>Object Other State Change</td>
</tr>
</tbody>
</table>

The event type of the standard event identifies the reported operational state.

Identify these operational states from your object states.

For an illustration of state transition diagrams and the relationship between operational states and object states, see Example of a Fictitious NonStop Kernel Subsystem on page 10-31.

For a description of the conditions you should consider in addition to the normal states of your object, see Task 3. Generate Standard Events for Your Subsystem on page 10-6.

**Task 2.4: Identify the Events for Your Subsystem**

One action in a state transition might be to generate an EMS event message indicating a change in your object that merits the attention of operators or management applications.

**When to Generate Events**

- Generate a standard event if the condition that caused the state change is one of the conditions defined in Section 9, Standard Events.
- Define your own events if the condition is not one of the standard conditions (described later).

**When to Not Generate Events**

When your object changes state and reports an EMS event message, the operators or management applications will probably issue commands to direct the recovery of your object or inquire more about the state of your object.
Task 2.5: Design Your Asynchronous Management Interface

Define the asynchronous management interfaces for objects that need to record or report their changes immediately, like state changes, as a result of encountering certain situations in the system.

<table>
<thead>
<tr>
<th>Type of Information</th>
<th>Asynchronous Management Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMS, TMDS or TSM</td>
<td>Object state change information</td>
</tr>
<tr>
<td>MEASURE</td>
<td>Performance and workload data</td>
</tr>
<tr>
<td>Audit services</td>
<td>Security data</td>
</tr>
</tbody>
</table>

Task 2.6: Design Your Command and Control Management Interfaces

Command and control management interfaces let:

- You configure, control, or inquire about your objects
- Operators and management applications change how your subsystem operates

They are integral to the manageability of your subsystem and must be provided. They include command interpreters for operators (for example, SCF, TMFCOM, PATHCOM) and programmatic interfaces for management applications (for example, SPI).

Task 3. Generate Standard Events for Your Subsystem

Determine the standard events and the event subjects for events that your subsystem should generate, and customize these standard events to your environment.

Task 3.1: Determine the Operational States of Your Objects and Subsystem Functions

1. Generate an event for states that you can detect (as defined in Object State Monitoring Functions on page 9-4):

<table>
<thead>
<tr>
<th>Operational State</th>
<th>Event Type to Generate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unavailable</td>
<td>Object Unavailable</td>
</tr>
</tbody>
</table>
**Operational State** | **Event Type to Generate**
---|---
Available | Object Available
Persistent | Object Other State Change
Need operator to change state | Object Other State Change

2. Generate an event for any functions to be performed in your subsystem:

<table>
<thead>
<tr>
<th>Function</th>
<th><strong>Event Type to Generate</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor resource usage for each object</td>
<td>Usage Threshold</td>
</tr>
<tr>
<td>Operator mounts tapes, fills printer, and so on</td>
<td>Operator Attention Needed, Operator Attention Completed</td>
</tr>
<tr>
<td>Detect recoverable errors (controller error, network reset, memory error, and so on)</td>
<td>Transient Fault</td>
</tr>
</tbody>
</table>
Task 3.2: Customize Your Standard Events

As needed for your subsystem, you can customize your standard events:

<table>
<thead>
<tr>
<th>For event type...</th>
<th>Customize as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Unavailable</td>
<td>Identify which errors make the subject unavailable and where they originate (use ZEMS-TKN-FAILURE-CAUSE values).</td>
</tr>
<tr>
<td></td>
<td>If the subject is unavailable because of errors in your subsystem (ZEMS-TKN-FAILURE-CAUSE = ZEMS-VAL-INTERNAL-FAILED):</td>
</tr>
<tr>
<td></td>
<td>● Define tokens describing error and diagnostic information that caused the subject to be unavailable.</td>
</tr>
<tr>
<td></td>
<td>● If errors warrant, define the format of a symptom string to include in the event message (ZEMS-TKN-SYMPTOM-STRING token).</td>
</tr>
<tr>
<td></td>
<td>If the subject is unavailable because the underlying subsystem failed (ZEMS-TKN-FAILURE-CAUSE = ZEMS-VAL-UNDERLYING-FAILED):</td>
</tr>
<tr>
<td></td>
<td>● Define tokens describing error or diagnostic information returned by the underlying subsystem.</td>
</tr>
<tr>
<td></td>
<td>● Provide the name in the ZEMS-TKN-UNDERLYING-OBJ-NAME token. If different than the name the underlying subsystem uses in its event subject for this error:</td>
</tr>
<tr>
<td></td>
<td>1. Provide information to help management applications locate the events.</td>
</tr>
<tr>
<td></td>
<td>2. Design your interface so management applications do not need additional subsystem-specific information.</td>
</tr>
<tr>
<td></td>
<td>3. Define tokens for the additional name and describe how to locate the events reported by the underlying subsystem.</td>
</tr>
<tr>
<td>Object Available, Object Other State Change</td>
<td>1. Define previous states (for ZEMS-TKN-STATE-PREVIOUS) and current states (for ZEMS-TKN-STATE-CURRENT) for all subjects that generate these types.</td>
</tr>
<tr>
<td></td>
<td>2. Define change reasons (for ZEMS-TKN-CHANGE-REASON) for all valid state transitions of these subjects.</td>
</tr>
</tbody>
</table>
Task 4. Define Private Event Types for Your Subsystem

To use EMS events to support management functions not currently defined in Section 9, Standard Events, use this task to determine whether EMS events should be used and, if so, the content of these events. This task ensures some consistency between events you defined and standard events.

If you define management functions that could benefit others, forward your results from this analysis to HP for possible inclusion in the standard events.

For help converting your existing events, see Task 5, Migrate Existing Events on page 10-13. You are not required to define the management functions for your existing...
events, but it could provide a better focus for your existing events and might help identify enhancements to those events.

**Task 4.1: Determine Management Functions and If EMS Is the Appropriate Platform**

Management functions in an operations environment can be grouped into three areas, largely consistent with the:
- ISO Open Systems Interconnection (OSI) Architecture
- IBM SystemView Architecture
- HP System Management Strategy

Different mechanisms are available for reporting system data:

<table>
<thead>
<tr>
<th>Report Mechanism</th>
<th>Reported Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMS</td>
<td>Discrete system situations (hardware/software failures)</td>
</tr>
<tr>
<td>MEASURE</td>
<td>System information in aggregate form</td>
</tr>
</tbody>
</table>

Determine which management area your management functions belong in and whether EMS is the appropriate platform:

- **Problem Management (or Reactive Problem Management)**
  - **Definition**: Detection of a system problem and managing that problem through its final resolution.
  - **Software Providing Function**: TMDS or TSM (for support hardware failures)
  - **EMS Support?**: All functions

- **Production Management**
  - **Definition**: Management of the production environment. Functions include data integrity and availability, as well as storage, printer, and batch management.
  - **Software Providing Function**: The Disk Process, Tape Management System, Spooler, and NetBatch
  - **EMS Support?**: Production requests that require operator attention
Task 4.2: Specify Which Operations Are Defined for Your Management Functions

Specify in detail the operations or tasks—for example, problem detection and isolation for the reactive problem management function—that a user or program needs to perform for each of your management functions.

Task 4.3: Specify the System Data Needed to Automate These Operations

Define the data items (for example, operating system data items) needed to support the operations defined for your management function:

- Define data items in general terms, not specific to one instrumentation platform like EMS or Measure.
- Specify the type and semantics of the data items.

Example: Defining System Data for Reactive Problem Management

This example illustrates the process of determining the operating system data to support the operations in reactive problem management.
Data for Problem Detection and Isolation

The operator needs to know that an object is no longer providing the services it was designed to provide (for example, a CPU or controller failed, or a software process ended abnormally). For timely detection and isolation, report the problem immediately on detection within the subsystem, including at least:

- The type of object that failed
- The instance name of the object that failed
- Process ID of the failing process (if it is a software process)

For example:

- CPU #15 failed. The failed object type is CPU, and the instance name of the CPU is #15 of system \nodeA.
- Software process $SRVR failed. The failed object type is PROCESS, and the instance name is $SRVR of system \nodeA.

This data is sufficient for the operator to detect that a problem exists and to identify the affected objects.

Data for Problem Diagnosis

To identify the actual cause of the problem, provide:

- Time stamp of when the failure occurred
- Diagnostic data to help diagnose the problem
- Source of error:
  - If the failure is from the reporting subsystem, provide the cause of the problem.
  - If the failure is from a service the reporting subsystem uses, the cause of the problem is elsewhere. Provide the name of the underlying service so operators can find the events reported by that service for this problem.

For example, note if a Pathway terminal went out of service because of an internal error or because the SNAX LU it drives went out of service.

Data for Problem Bypass and Recovery

To perform problem bypass and recovery, the operator does not need additional data. One way to effect bypass and recovery is to have a configuration database that contains a list of all objects (actives and spares) and a list of failed components to replace. (HP does not provide these tools.) The operator can use this information to reconfigure a running system by substituting failed components with spares. At this level, no additional data is needed for this function.
Data for Problem Resolution
To foster problem resolution, provide:

- Recommended action—the subsystem's best guess at how to correct the problem. This sometimes fixes the problem, but not always. It is difficult for a subsystem to determine the correct recovery action without having all the problem information.

- Manager process name—for the failed object. If there is no manager process, it is the name of the process that provides the command and control interface to control the failed object.

Data for Problem Tracking and Control
To foster problem tracking and control, provide:

- Version of the subsystem that contains the problem

- A unique character string identifying the place in the subsystem code that had the problem

Task 4.4: Specify the EMS Event Types for Your System Data
The system data items you have identified are the tokens in an EMS event.

- If several conditions are reported using a single EMS event, consider defining multiple EMS events with each event reporting only one significant condition. The resulting events are more meaningful. For example, Section 9, Standard Events, defines an Object Available, Object Unavailable, and Other State Change events instead of one general event to report object state changes.

- Define enumerated values (if they do not already exist) for these event types. One of these values will be stored in the ZEMS-TKN-CONTENT-USER token.

Task 5. Migrate Existing Events
If your subsystem currently generates nonstandard events, read Migration Rules and then do one of:

- Incorporating Old Events Into New Events on page 10-14

- Revising Old Events on page 10-14

If you do not have existing nonstandard events to migrate, continue with Task 6. Design Your Event Messages on page 10-15.

Migration Rules
Do not change an EMS event if the changes will force existing management applications to change.

You cannot:
Incorporating Old Events Into New Events

If the event you are converting from (old event) has the same meaning (that is, both events report the same condition for the same subject in type and format) as a standard event you already defined (new event):

1. If the event number (and the literal for the event number) of your new event is different from the number of your old event, change the new event number (and its literal) to the old event number. You must not change an existing event number or its literal.

2. Add to the new event those tokens present in the old event but not present in the new event.

3. Use the template of the new event to display your event. Add to the template of your new event those tokens present in the template of your old event but not present in the new event template.

Revising Old Events

If your old event does not have the same meaning as any of the new events, standardize your old event:
1. Add to your old event any common-standard tokens defined in Section 9, Standard Events, that are not already in your old event.

2. Classify your old event by assigning an event type to the event:
   - Use the event types defined for the ZEMS-TKN-CONTENT-STANDARD or ZEMS-TKN-CONTENT-USER tokens whenever possible.
   - If you define your own event types, you must define corresponding enumerated values for the ZEMS-TKN-CONTENT-USER token.
   - Use as few event types as possible to make event filtering easier and more efficient for operators.

3. Add to your template those common-standard tokens that you have added to your event.
   If your template style is different from that of a standard event, consider changing your template to look like a standard event. This makes it easier for operators to scan the event message for information.

**Task 6. Design Your Event Messages**

Determine your event messages from the event types and event subjects you have identified. This task is for the design of a single event. Repeat it for each event message you need to design.

Normally, you define an event message—with a unique event number—for each subject and event type combination. The two situations in which you might assign an event number differently:

- Your event message contains many possible recovery actions. In this case, define multiple event messages per event type and subject combination so each message specifies a well-defined recovery action.
- If your event is enhanced from an existing event, you must keep the same event number and its literal.

1. Designate the event number.
   
   Every event message must have an event number, unique within your subsystem SSID. Define all your event numbers using literals of ss-EVT-eventname, such as ZX25-EVT-LINE-DOWN, where eventname indicates the subject (X.25 line) and condition (down) reported in the event.

2. Determine if your event is reporting a critical or noncritical condition.
   
   For guidelines for the ZEMS-TKN-EMPHASIS token, see Common-Standard Tokens That Subsystems Provide on page 9-20.

3. Determine whether to display your event on the operator console.
For guidelines for the ZEMS-TKN-SUPPRESS-DISPLAY token, see Common-Standard Tokens That Subsystems Provide on page 9-20.

4. Determine whether the manager process is relevant to the event. It is relevant only if your event contains the ZEMS-TKN-NAME-MANAGER token.

5. Define whether the event is standard or private:
   - A standard event is an event defined in Section 9, Standard Events. It must have a non-null ZEMS-TKN-CONTENT-STANDARD token value.
   - A private event is an event defined by you or another subsystem that is not a standard event. The ZEMS-TKN-CONTENT-STANDARD token must have a null value, and ZEMS-TKN-CONTENT-USER must have a non-null value.

6. Define unconditional and conditional tokens in the event:
   - Unconditional tokens are always present in an event (for example, the ZEMS-TKN-EVENTNUMBER token).
   - Conditional tokens are present only under appropriate conditions. There is a specific meaning if a conditional token is not present in the event. For example, ZEMS-TKN-BATCHJOB-ID is a conditional token. Its absence means no job ID is associated with the object.

7. Define whether the event has internal (volatile) tokens.
   You can change internal tokens in an event message without notice. They are used only by the subsystem.

8. Define the event message text to display for the event.
   You must have an EMS template for your event. The template tells the EMSTEXT program how to display your event message in operator console products like Web ViewPoint. EMSTEXT uses the template you provided for an event before using the standard template that EMS provides for each standard event.
   - For a standard event, use the standard template display. Do not provide a different display unless:
     ○ The event has more than one subject.
     ○ The subject is not a printable ASCII string.
     ○ The event has private tokens that you think operators should see.
   - For a private (nonstandard) event, describe how to display the event.
   The display of all event messages should look similar. Follow the standard template format in Task 9. Create and Build Your EMS Templates on page 10-26.

9. Define cause, effect, and recovery procedures for the event. Describe:
   - What causes the event to be generated
   - The effect the reported condition has on your subsystem, system, and network
Generating Standard Events

Task 7. Write Your Event External Specification

How the operator must diagnose and resolve the reported problem (For example, take a trace, and then take the line down and restart it.)

The cause, effect, and recovery information tells operators what to do with your events. Use as much detail as possible.

You must include this information in Task 7. Write Your Event External Specification on page 10-17. Specify this information explicitly in the event detail database in one of two ways:

- Define EMS templates that contain the cause, effect, and recovery procedures of your events; install these templates in the event detail database EVENTTD using the TEMPLI program. For more information, see Support Note “S91056: ViewPoint Expanded Event Detail Support in C21.”

- Customize the text displayed by the event detail screen by building a key-sequenced file, EVENTCX, in the system load subvolume. For more information, see the ViewPoint Manual.

Task 7. Write Your Event External Specification

Write your Event External Specification (ES) to provide specific information about your event to the users. The section in the ES to update is indicated by {...}.

Task 7.1: Prepare the Subsystem Event External Specification Template

1. In the NonStop Technical Library, copy and paste the text under Event External Specification Sample File on page C-1 into a .txt file in any text editor.

2. Transfer the .txt file to your subvolume.

3. Rename the .txt file in your subvolume to sssses (for example, ZSAMES), where ssses is the subsystem acronym.

Task 7.2: Describe Your Subsystem Environment

Describe how your subsystem generates EMS events. Include information about your subsystem that could help programs or people use your events:

- What your subsystem does and where more details can be found.

- What processes are defined in your subsystem and which ones are responsible for generating events.

- Whether your subsystem uses code from another subsystem and, if so, whether this code generates EMS events on your behalf. If so, briefly describe what this code does and which subsystem it comes from.
Task 7.3: List the Standard Management Functions Your Subsystem Supports

- Whether your subsystem interacts with processes from another subsystem either in the generation of your events or in the management of your subsystem. If so, describe these processes.

- Whether your subsystem depends on tokens and events defined by another subsystem. If so, briefly describe the subsystems and EMS files that you need (for example, ZSPIDDL, ZEMSDDL, and ZCOMDDL).

- Describe the command and control interface a user can use to inquire about or control an object reported in your events, and where to find more details.

- Whether your subsystem has a manager process. If so, whether it is reported in an event. Provide this information about the manager:
  - Its function, and whether it resolves object names or is the focal point for commands to control and inquire about an object.
  - What kind of process it is (a process pair, a persistent process, or neither), whether you can have more than one manager process and, if so, what the implications are.
  - Its name. Whether it is a well-known (fixed) name or if it changes depending on the event.

- Whether you are generating events for EMS only or if you are also generating events for TMDS or TSM.

- Whether you send events to the primary collector or to an alternate collector.
  
  If possible, your subsystem should let the destination collector be configurable for better flexibility.

### Task 7.3: List the Standard Management Functions Your Subsystem Supports

<table>
<thead>
<tr>
<th>Standard Event Types You Can Generate</th>
<th>Corresponding Standard Management Functions to Check in Your ES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Available</td>
<td>Object state monitoring</td>
</tr>
<tr>
<td>Object Unavailable</td>
<td>Reactive problem management</td>
</tr>
<tr>
<td>Object Other State Change</td>
<td>Object state monitoring</td>
</tr>
<tr>
<td>Transient Fault</td>
<td>Proactive problem management</td>
</tr>
<tr>
<td>Usage Threshold</td>
<td>Proactive problem management</td>
</tr>
<tr>
<td>Operator Attention Needed</td>
<td>Production requests requiring operator</td>
</tr>
<tr>
<td>Operator Attention Completed</td>
<td>Production requests requiring operator</td>
</tr>
</tbody>
</table>
Task 7.4: Describe the Private Management Functions You Support

If your EMS events support nonstandard management functions, list them in your ES {3.2}. Create a new subsection {3.2.x} to describe each listed management function. At a minimum, provide:

- The management function and what operations it performs
- Event types designed to support this function
- How management applications use your events to perform the operations

For examples, see Section 9, Standard Events.

Task 7.5: Define Your Event Subjects

1. Define the DDL names for your event subjects.
2. List these names and those from other subsystems that you use in your ES.
   - If name is a string or fixed structure, use ss-TKN-subjectname.
   - If name is represented by an extensible structured token, use ss-MAP-subjectname.

Define your event subject as a printable ASCII string whenever possible. Avoid using fixed or extensible structures. You cannot use the standard templates if your subject is a fixed or extensible structure.

If your subject is a group name, you must specify operator instructions for obtaining member names from the group name. For example, to obtain the member names from group name ZSAM-TKN-GROUP1, issue the SPI command CMD1 ... GROUP1 to the manager process specified in the event.

Task 7.6: Define the Event Types You Support

List the standard event types you support in your ES.

Specify the DDL names ss-VAL-typename for the private event types you defined and add them to your standard list.

Task 7.7: Define Your Event Numbers and Their CDMT Attributes

Define the DDL names ss-EVT-eventname for your event numbers. List them in your ES under their event subjects with their CDMT attributes—critical, display, manager process, and event type.
Task 7.8: Enumerate Your Private Values to Standard EMS Tokens

If you have private values for these tokens, define their values $ss$-VAL-$valuename$ and list them under the token in your ES:

- ZEMS-TKN-CHANGE-REASON
- ZEMS-TKN-CONTENT-USER
- ZEMS-TKN-STATE-CURRENT
- ZEMS-TKN-STATE-PREVIOUS
- ZEMS-TKN-TXFAULT-TYPE
- ZEMS-TKN-UTIL-UNIT

Task 7.9: List Private Tokens From Other Subsystems

If you use tokens defined by other subsystems—other than SPI or EMS—create a new subsection (4.3.x) for each subsystem and list their tokens there. List these tokens under the headings “Common ... tokens” or “Event-specific ... tokens.”

Task 7.10: List Private Tokens From Your Subsystem

If you specify private tokens, list them in your ES with this explanation:

- Name of the token—for example, ZSAM-TKN-$tokenname$
- Type of token—for example, ZSPI-TYP-STRING
- Kind of token—conditional (C) or unconditional (U)
- List of possible values with brief description
- If conditional, under what conditions the token is provided and what the default meaning is if the token is not provided

Task 7.11: Describe the Details of Each Event Message

1. Create a new subsection {5.x} in your ES to describe each of your event messages. Order the subsections alphabetically by the symbolic event names.

2. Describe your event messages by using one of the templates provided in the ES:
   - Unconditional tokens, conditional tokens, internal tokens
     - List the private tokens you defined.
     - List the standard tokens only if the token contains more specific values than those defined for the token in ES (3.0) or if you want to provide your own event-message text and you need to reference that token in your description.
   - Event message text
Describe the message text for the display of your event if different from the one provided by the standard template. You will need to provide your EMS template to describe this event text later.

- **Cause description, effect description, recovery procedures**

  Describe the cause, effect, and recovery procedures for the reported condition. This information is intended for the operator. **Make the information as complete, detailed, and specific as possible.**

  - Cause descriptions tell the operator what caused the system to generate the event message. Do not confuse cause with effect.

  - Effect descriptions explain what effect this event has on the system. If a message has no effect, enter “None.” Do not describe recovery actions here.

  - Recovery procedures help the operator solve the problem indicated in the event message. If the recovery process involves several steps, write them in sequential order.

  If operators might not be able to recover without additional help, tell the operators to contact their service provider and specify what information they should be ready to provide. Do not merely give the instruction “Contact your service provider.”

  If your event message only reports status, you might not need recovery information. In this case, enter “Informative message only; no corrective action is needed.”

- **Other considerations**

  Describe information specific to this event like usage to correlate Object Unavailable events from other subsystems.
Task 8. Create and Build Your DDL Definitions

Create a DDL file for the DDL definitions for your events. Each subtask corresponds to a section of the same name in the sample file from which you create your definitions.

Name the language output files:

- ssC for C
- ssCOB for COBOL
- ssPAS for PASCAL(*)
- ssTACL for TACL
- ssTAL for TAL

(*) optional language output

You must provide two types of clauses when creating and building your DDL Definitions file:

- An 89 enumeration clause associates a name with a specified or default enumeration value and optionally specifies a display string for the value.
- An AS clause specifies a display string for a value in a level 89 enumeration clause or, at the definition level, specifies a default display string for a field of type ENUM. When an AS clause is part of a level 89 clause, AS specifies the display string for a field that has the value associated with that level 89 item.

For more information about the 89 enumeration clause and the AS clause, see the Data Definition Language (DDL) Reference Manual.

Task 8.1: Prepare the DDL Definitions File Template

1. In the NonStop Technical Library, copy and paste the text in DDL Definitions Sample File on page C-20 into a .txt file in any text editor.
2. Transfer the .txt file to your subvolume.
3. Rename the .txt file in your subvolume to ssssDDL (for example, ZSAMDDL), where ssss is the subsystem acronym.

Task 8.2: Specify Subsystem Product Information

Specify the name, number, version, and release date of your product. Change this section whenever you change the version or release date of your product.

Task 8.3: Define Your Subsystem ID (SSID)

Define your SSID as specified in the Subsystem Definition Table of your ES. The version field of your SSID should reflect the release identifier of your subsystem. The field is a 16-bit unsigned integer and must increase in value from one release to the next.

The syntax and semantics of the SSID version field are defined by each company.
Task 8.4: Define Your External SSID

Specify the company name and the subsystem acronym assigned to your subsystem; for example, TANDEM.PWY.0 for the HP Pathway subsystem.

Task 8.5: Specify Event Number Definitions ZEMS-TKN-EVENTNUMBER

Provide an 89 enumeration clause that briefly describes the type for each of your event types; for example, X25 line down.

Make the enumeration clause 40 characters or fewer with no AS clause specifying the default display for this enumerated token. Use AS clauses in the individual level 89 items. This information is displayed by the standard template.

This specifies the values for literals of the event numbers defined in the Event Numbers Definition Table of your ES.

Task 8.6: Specify Private Event Type Definitions ZEMS-TKN-CONTENT-USER

If you added your own event types in the Event Types Definition Table of your ES, enumerate their values:

1. Start your values from ZEMS-VAL-MIN-USER-VALUE (1024).
2. Provide an 89 enumeration clause that briefly describes the type for each of your event types; for example, usage data for network connection time in minutes.

Make the enumeration clause 40 characters or fewer with no AS clause specifying the default display for this enumerated token. Use AS clauses in the individual level 89 items. This information is displayed by the standard template.

Task 8.7: Specify Private Enumerations for Standard Tokens

If you added your own values to any of these standard tokens, as defined in your ES Standard EMS Tokens Definition Table, enumerate their values:

- ZEMS-TKN-STATE-CURRENT
- ZEMS-TKN-STATE-PREVIOUS
- ZEMS-TKN-CHANGE-REASON
- ZEMS-TKN-TXFAULT-TYPE
- ZEMS-TKN-UTIL-UNIT

1. Start your values (each token) from ZEMS-VAL-MIN-USER-VALUE (1024).
2. Provide an 89 enumeration clause that briefly describes the type for each of your event types, for example, network connecting.
Make the enumeration clause 40 characters or fewer with no AS clause specifying the default display for this enumerated token. Use AS clauses in the individual level 89 items. This information is displayed by the standard template.

**Task 8.8: Specify Private Enumerations of Other Subsystems’ Tokens**

If you added your own values to any enumerated tokens provided by another subsystem, as defined in Tokens from Other Subsystems of your ES, list your values:

1. Start your values as required by that subsystem.
2. If you defined the token with 89 enumeration clauses, do the same with your values.

     Provide an 89 enumeration clause that briefly describes the type for each of your event types. Make the enumeration clause 40 characters or fewer with no AS clause specifying the default display for this enumerated token. Use AS clauses in the individual level 89 items. This information is displayed by the standard template. Do not specify a default display string for another subsystem’s tokens.

**Task 8.9: Specify Private Enumerations of Your Subsystem's Tokens**

If you defined your own tokens—listed in Private Tokens of this Subsystem of your ES—and they are defined as enumerations, specify their values:

1. Specify values for your enumerations.
2. Provide an 89 enumeration clause (40 characters or fewer) that briefly describes the value. This simplifies the design (or customization) of your template for these tokens. Use the AS clause to specify a default display string because these are your own tokens.

**Task 8.10: Define Event Subjects**

Define the event subjects listed in the Event Subjects Definition Table of your ES. Define only subjects that you specify, not subjects from other subsystems.

Whenever possible, define your subject as a printable ASCII string so the standard templates can display it. Otherwise you need to customize the standard templates.

Event subject is like any token. The event subject name listed in your ES is the TOKEN-CODE literal.

**Task 8.11: Define Private Tokens**

If you defined your own tokens in Private Subsystem Tokens Definitions Table of your ES (or defined your own subject tokens), define these tokens.
Determine whether your token is a simple or extensible structured token:

- A simple token contains a single field or multiple fields in a fixed structure. A simple token cannot be extended. Extension is done by adding other tokens. Use simple tokens whenever possible.

- An extensible structured (MAP) token is a structure that lets new fields be added in subsequent versions of the subsystem.

**Defining a Simple Token**

Define your simple tokens using the DDL TOKEN-CODE statement:

- Token number (required). Define your token numbers:
  - Name your token number as `ss-TNM-tokenname`. For example, if your token name is ZSAM-TKN-mytoken1, your token number should be called ZSAM-TNM-mytoken1.
  - Assign a unique value—from 1 through 9998—to each of your token number names.
  - Define your token numbers in one place—in token-number definitions.

- Token type (required). If your token is an enumerated token or fixed structure, define your own token type (for example, ZSAM-TYP-tp-diagcode); otherwise use a standard SPI token type (for example, ZSPI-TYP-STRING or ZSPI-TYP-INT).

Specifying your own token type lets you provide a DDL DEF for your token. The DDL DEF lets you define the “89 enumeration clause” for your enumerated token or the fields for your structure token. Define your token type with the DDL TOKEN-TYPE statement:

  - Token-data-type. You must build your token type from one or more of the standard SPI token data types like ZSPI-TDT-ENUM.
  - DDL DEF. Specify the DDL definition that defines the structure and length of your token.

Standard SPI token types and standard SPI token data types are defined in the ZSPIDDL file and described in the *SPI Programming Manual*.

- SSID clause. Specify your external SSID unless your token is shared with other subsystems.

- HEADING clause. Provide a heading that briefly describes your token. This lets programs like INSPECT display more descriptive information for your tokens.

**Defining a MAP Token**

Define your extensible structured token using the DDL TOKEN-MAP statement:

- Token-number. See **Defining a Simple Token**.
Task 8.12: Compile Your DDL Definitions

Follow the established build procedure of your company.

Task 9. Create and Build Your EMS Templates

You must provide an EMS template file to display the events of your subsystem which, at a minimum, describes the SSID, name, and version of your subsystem.

Each step corresponds to a section in the sample EMS template file.

If you provide the cause, effect, and recovery of an event in templates, do not define those templates in this file. You should have already defined those templates in another template file in Task 6. Design Your Event Messages on page 10-15.

1. Prepare the EMS template file sample file.
   a. In the NonStop Technical Library, copy and paste the text under EMS Templates Sample File on page C-30, into a .txt file in any text editor.
   b. Transfer the .txt file to your subvolume.
   c. Rename the .txt file in your subvolume to SsssTMPL (for example, SSAMTMPL), where Ssss is the subsystem acronym, using an S at the beginning. (ZsssTMPL is the template object.)

2. Provide a description (the SSID, SSNAME, and VERSION) of your subsystem.

3. If you defined your own enumerations with 89 enumeration clauses to any of these standard EMS tokens:
   - ZEMS-TKN-CONTENT-USER
   - ZEMS-TKN-CHANGE-REASON
   - ZEMS-TKN-STATE-CURRENT
   - ZEMS-TKN-TXFAULT-TYPE
   - ZEMS-TKN-STATE-PREVIOUS
   - ZEMS-TKN-UTIL-UNIT

If you must use MAP tokens, for more information on how they work, see the SPI Programming Manual.
assign the DDL definitions of your enumerations to these tokens so the standard
templates can display them.

4. If you defined your own enumerations with 89 enumeration clauses to tokens from
other subsystems, assign the DDL definitions of your enumerations to those tokens
so they can be incorporated into the display by templates defined by you or by
those other subsystems.

5. If you defined displays for your standard events different from those of the standard
templates, define templates for these displays:
   a. Copy the standard template from EMS (defined in ZSPIDEF.SEMSTMPL) that
describes your standard event to your template file.
   b. Modify the template that is by standard type to a template by event-number by
      replacing the statement:
      MSG: ZEMS-TKN-CONTENT-STANDARD, ZEMS-VAL-eventtype-name
      with
      MSG: ZEMS-TKN-EVENTNUMBER,ss-VAL-eventnumber-name
   c. Modify the display of any tokens in your template. To display additional tokens,
      add the tokens to the end of your template if possible. Do not delete any
tokens from the standard template.

Note. You must provide one event-number template for each standard event for which
you want to customize the display.

6. If you defined your own private events, you must provide one event-number
template for each of your private events. Follow one of the standard templates in
designing the template for your private event so the display for all your event
messages looks similar to the operators. Use Step 5 to create the templates for
your private events.

7. Compile your EMS templates. Follow the established build procedure in your
company.

Task 10. Code and Test Your Event Generation

Encode and send your events to an EMS collector, and test your event generation
code.

Task 10.1: Define the EMS Collector for Your Events

In general, EMS events that are for operations management (all standard events) and
EMS events intended for a large audience to the primary collector ($0); otherwise,
send your EMS events to an alternate collector. If you send events to an alternate
collector, let the operator specify the collector name.
Use the FILE_OPEN_ procedure to open the primary or alternate collector process.

Task 10.2: Build Your Event Message Buffer

1. If you are generating a usage threshold event:
   a. For each resource whose usage is to be reported, allocate a control block dedicated for that resource. Initialize the control block (only once) using the EMS_UTCB_INIT_ procedure.
   b. Call EMS_USAGE_THRESHOLD_CHK_ every time the usage level of a resource changes. This returns a flag signifying whether to generate the usage threshold event. For more information, see Standard Usage Threshold Event on page 11-31.

2. Call the EMS_COMMON_TOKENS_EVT_BLD_ procedure (only once per event message) to initialize your event message buffer with all the required common-standard tokens and specify your first event subject.
   This automatically provides all common-standard tokens provided by EMS. For more information, see EMS_COMMON_TOKENS_EVT_BLD_ Procedure on page 11-44.

3. If your event has multiple subjects, call the EMSADDSUBJECT or EMSADDSUBJECTMAP procedure one or more times to add the additional subjects to your event message buffer. For more information, see EMSADDSUBJECT and EMSADDSUBJECTMAP Procedures on page 15-8.

4. If your event has private tokens, call EMSADDTOKENS or EMSADDTOKENMAPS one or more times to add your private tokens to your event message buffer. For more information, see EMSADDTOKENS and EMSADDTOKENMAPS Procedures on page 15-10. You can also call these procedures to override the default value for these header tokens:

<table>
<thead>
<tr>
<th>Required Common Token</th>
<th>Default Value</th>
<th>(page 1 of 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-NAME-MANAGER (change for private buffers only)</td>
<td>Not present</td>
<td>unless specified by the corresponding optional parameter in the EMS_COMMON_TOKENS_EVT_BLD_ procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZEMS-TKN-NAME-MANAGER can only be added by EMSADDTOKENS; it cannot be changed by this procedure</td>
</tr>
<tr>
<td>ZEMS-TKN-CONTENT-STANDARD (change for private buffers only)</td>
<td>ZEMS-VAL-NULL</td>
<td></td>
</tr>
</tbody>
</table>
Task 10.3: Send Your Event Message Buffer to the EMS Collector

After building your event message buffer, send it to the EMS collector:

1. Call the SPI SSGETTKN procedure to obtain the length of your event message buffer, which is saved in ZSPI-TKN-USEDLEN. For details, see the SPI Programming Manual.

2. Call the WRITEREAD procedure to send the event message buffer to the open collector. Use the number of bytes to write from the SSGETTKN procedure.

   Design your code to handle these errors returned from the WRITEREAD call:
   - Error 33—EMS collector has no buffer space
     This is a temporary condition. Wait a second and then reissue the WRITEREAD.
   - Error 60—wrong ID
     The EMS collector has been restarted since you opened it. Close the collector, reopen it, and issue the WRITEREAD again.

3. If needed, call the FILE_CLOSE_ procedure to close the EMS collector process after each event message.

   If you have many event messages to send, leave the collector open to avoid opening and closing the collector for each event message. The alternate collector can handle a finite, but large, number of openers, so you might not want to keep the alternate collector open for a long period of time. The primary collector can handle any number of openers.

Task 10.4: Compile Your EMS Code

Include, in order, the definitions of:

- The standard SPI tokens
- The standard EMS tokens
- The tokens of other subsystems you depended on (if any)
The tokens you defined for your subsystem

Include only the definitions in the programming language of your code. For information on how to include a definition file in your programming language, see the SPI Programming Manual. Get all HP provided tokens from the ZSPIDEF subvolume.

<table>
<thead>
<tr>
<th>Programming Language of Definition</th>
<th>Standard SPI Tokens</th>
<th>Standard EMS Tokens</th>
<th>Standard COMM SUBSYS Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAL</td>
<td>ZSPITAL</td>
<td>ZEMSTAL</td>
<td>ZCOMTAL</td>
</tr>
<tr>
<td>TACL</td>
<td>ZSPITACL</td>
<td>ZEMSTACL</td>
<td>ZCOMTACL</td>
</tr>
<tr>
<td>COBOL85</td>
<td>ZSPICOB</td>
<td>ZEMSCOB</td>
<td>ZCOMCOB</td>
</tr>
<tr>
<td>C</td>
<td>ZSPIC</td>
<td>ZEMSC</td>
<td>ZCOMC</td>
</tr>
</tbody>
</table>

### Task 10.5: Test Your EMS Code

You must test your EMS code to verify:

- An event is generated only under the expected conditions and the event generated is of the appropriate type.
- All required tokens in an event message are present.
- All optional tokens in an event message are present only under the appropriate conditions.
- All tokens in an event are correctly encoded.
- All event messages are logged to the appropriate collectors.
- There are no unexpected or unknown tokens in the event message.

There are several ways to test the content of an event, including:

- Send the event to the EMS collector and write an EMS distributor to read the event from the log for verification.
- Use INSPECT to set a breakpoint in your program just before the event is sent to the collector and display the event buffer using the DISPLAY event-buffer TYPE EMS command.

The primary collector is available at system generation, so you can immediately log events to it. Run EMSCINFO to determine the location and name of the current log file.

If you report events to an alternate collector, you need to run EMSACOLL to start up an alternate collector process. Specify the log file associated with this collector when you start the program.

You should be able to use the default attributes of the primary and alternate collectors for testing. If you need to change the defaults, run EMSCCTRL.

For details about EMSCINFO, EMSACOLL, and EMSCCTRL, see Section 13, EMS Programs.
Task 11. Release and Distribute Your EMS Files

Release and distribute these EMS files you created (including your code modules that format and generate the events):

<table>
<thead>
<tr>
<th>Event/Token Definition</th>
<th>Formatting Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssssDDL</td>
<td>ssssTMPL</td>
</tr>
<tr>
<td>ssssC</td>
<td>SidTMPL</td>
</tr>
<tr>
<td>ssssCOB</td>
<td></td>
</tr>
<tr>
<td>ssssTACL</td>
<td></td>
</tr>
<tr>
<td>ssssTAL</td>
<td></td>
</tr>
</tbody>
</table>

where ssss is your subsystem acronym and id is your three-character ID of the subsystem.

Follow the software release and distribution procedures established in your organization.

Example of a Fictitious NonStop Kernel Subsystem

This example illustrates the design of EMS events for a fictitious NonStop Kernel subsystem. The behavior of each object is described with state transition diagrams.

The events designed for this subsystem are described in the DDL Definitions Sample File on page C-20 and the EMS Templates Sample File on page C-30.

Overview of Fictitious Subsystem

- Name: SAMPLER
- Acronym: ZSAM
- Number: 99
Subsystem ZSAM provides network independent end-to-end reliable transport service for applications through the X.25 public and private data networks. Access to the X.25 network service is provided by another subsystem.

A local application attaches to a remote application using a dedicated end-to-end transport connection. It accesses the transport service through one of the subdevices configured for this subsystem. A maximum of 250 subdevices—which means a maximum of 250 active transport connections or 250 active application attaches—are supported at one time.

When a local application first attaches to the subsystem, its user profile—kept on a user profile tape—is verified before access is granted. When a local application detaches from the remote application, its usage information—maintained by this subsystem—is written to a special EMS log (alternate collector).

The subsystem is implemented as a set of persistent processes. The manager process is a NonStop process pair whose name is configured at system startup. The manager process is the focal point for operators to control or inquire about objects in the subsystem.

**Objects to Be Managed**

Evaluation of ZSAM (Task 2.1) indicates that ZSAM should monitor the availability and quality of:

- Subdevice slots
- Local applications
- Transport connections
- Underlying X.25 network service
- User profile tape
Event Subjects

The corresponding subject names to the specified objects types are:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Event Subject Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subdevice slot</td>
<td>(Not analyzed in this example)</td>
</tr>
<tr>
<td>Local application</td>
<td>(Not analyzed in this example)</td>
</tr>
<tr>
<td>Transport connection</td>
<td>ZSAM-TKN-subj-tp</td>
</tr>
<tr>
<td>Underlying X.25 network service</td>
<td>ZSAM-TKN-subj-netx25</td>
</tr>
<tr>
<td>User profile tape</td>
<td>ZSAM-TKN-subj-tape</td>
</tr>
</tbody>
</table>

Event Types

ZSAM supports all standard management functions and generates all standard events that support them. ZSAM also generates private events containing network usage information of the local application when the application detaches from ZSAM. Management applications use these events for billing applications based on network usage. The event type is ZSAM-VAL-data-usage. The data items for this event type are:

- Network connect time (in minutes)
- Number of data bytes sent to remote application
- Number of data bytes received from remote application
- Number of X.25 data packets sent for this application
- Number of X.25 data packets received for this application

State Transition Diagram and Events for ZSAM-TKN-subj-netx25
### Subsystem States

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Current State</th>
<th>EMS State</th>
<th>Standard Event to Be Generated</th>
<th>ZSAM Event Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net down</td>
<td>- &gt; net up</td>
<td>A</td>
<td>Obj avail</td>
<td>ZSAM-EVT-netx25-up</td>
</tr>
<tr>
<td>Net up</td>
<td>- &gt; net reset</td>
<td>A</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Net reset</td>
<td>- &gt; x reset</td>
<td>A</td>
<td>Tran fault</td>
<td>ZSAM-EVT-netx25-dataloss</td>
</tr>
<tr>
<td>Net reset</td>
<td>- &gt; net up</td>
<td>A</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Net up</td>
<td>- &gt; net down</td>
<td>U</td>
<td>Obj unavail</td>
<td>ZSAM-EVT-netx25-down</td>
</tr>
</tbody>
</table>

(Not shown as a state, but while in “net up” state, the req queue for net svc is checked whenever a request is put on or taken off the queue. A usage threshold event ZSAM-EVT-util-req is sent when the req queue crossed the threshold.)

### State Transition

<table>
<thead>
<tr>
<th>State Transition</th>
<th>Reason</th>
<th>Standard Token Values</th>
</tr>
</thead>
</table>
| 1                | Received NET READY indication from underlying network service | ZEMS-TKN-STATE-STATE-PREVIOUS (ZSAM-VAL-): netx25-down  
ZEMS-TKN-STATE-STATE-CURRENT (ZSAM-VAL-): netx25-up  
ZEMS-TKN-CHANGE-REASON (ZSAM-VAL-): cr-netx25-online |
| 2                | Received NET NOT READY indication from underlying network service | State change event not generated (only Object Unavailable event) |
| 3                | Received RESET WITH DATA LOSS indication from underlying network service | State change event not generated |
| 4                | Number of resets with data loss > N1 times (reset count) | State change event not generated (only Transient Fault event) ZEMS-TKN-TXFAULT-TYPE = ZSAM-VAL-tf-resets-net |
State Transition Diagram and Events for ZSAM-TKN-subj-tp

<table>
<thead>
<tr>
<th>Subsystem States</th>
<th>Previous</th>
<th>Current</th>
<th>EMS State</th>
<th>Standard Event to Be Generated</th>
<th>ZSAM Event Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>disconnected</td>
<td>-&gt; connecting</td>
<td>U</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>connecting</td>
<td>-&gt; xconn</td>
<td>U</td>
<td>obj other state</td>
<td>ZSAM-EVT-tp-x-conntime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt; connected</td>
<td>A</td>
<td>change</td>
<td>ZSAM-EVT-tp-connected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt; disconnected</td>
<td>U</td>
<td>obj avail</td>
<td>ZSAM-EVT-tp-disconnected</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>obj unavail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xconn</td>
<td>-&gt; connected</td>
<td>A</td>
<td>obj avail</td>
<td>ZSAM-EVT-tp-connected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt; disconnected</td>
<td>U</td>
<td>obj unavail</td>
<td>ZSAM-EVT-tp-disconnected</td>
<td></td>
</tr>
<tr>
<td>connected</td>
<td>-&gt; reset</td>
<td>A</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt; disconnecting</td>
<td>U</td>
<td>obj unavail</td>
<td>ZSAM-EVT-tp-disconnected</td>
<td></td>
</tr>
<tr>
<td>reset</td>
<td>-&gt; xreset</td>
<td>A</td>
<td>tran fault</td>
<td>ZSAM-EVT-tp-x-reset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt; connected</td>
<td>A</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>xreset</td>
<td>-&gt; disconnecting</td>
<td>U</td>
<td>obj. unavail</td>
<td>ZSAM-EVT-tp-disconnected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt; connected</td>
<td>A</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>disconnecting</td>
<td>-&gt; xdisc</td>
<td>U</td>
<td>usage data</td>
<td>ZSAM-EVT-tp-data-usage</td>
<td></td>
</tr>
<tr>
<td>xdisc</td>
<td>-&gt; disconnected</td>
<td>U</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>
### State Transition Diagram and Events for ZSAM-TKN-subj-tape

The tape can be in one of these states: mounted or unmounted. The Operator Attention Needed standard event (event number ZSAM-EVT-tape-mount-needed) is generated to request that an operator mount the user profile tape. The Operator Attention Completed standard event (event number ZSAM-EVT-tape-mount-done) is generated to indicate the tape is mounted and operator attention is no longer needed.

### Private Tokens and Templates for Events

<table>
<thead>
<tr>
<th>Event-name (ZSAM-EVT-)</th>
<th>Event-type (ZEMS-VAL-)</th>
<th>Private tokens (ZSAM-TKN-)</th>
<th>Private Templates? (page 1 of 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tape-mount-needed</td>
<td>attn-needed</td>
<td>--</td>
<td>No</td>
</tr>
<tr>
<td>tape-mount-done</td>
<td>attn-completed</td>
<td>--</td>
<td>No</td>
</tr>
<tr>
<td>netx25-down</td>
<td>object-unavailable</td>
<td>netx25-error</td>
<td>Yes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pdnx25-error</td>
<td></td>
</tr>
</tbody>
</table>

* An event-number template is provided for each of these events to customize the event message text (display).
** An event-number template is provided for the event message text of this private event.
<table>
<thead>
<tr>
<th>Event-name</th>
<th>Event-type</th>
<th>Private tokens</th>
<th>Private Templates?</th>
</tr>
</thead>
<tbody>
<tr>
<td>netx25-up</td>
<td>object-available</td>
<td>--</td>
<td>No</td>
</tr>
<tr>
<td>netx25-dataloss</td>
<td>transient fault</td>
<td>--</td>
<td>No</td>
</tr>
<tr>
<td>netx25-util-req</td>
<td>usage threshold</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>tp-disconnected</td>
<td>object-unavailable</td>
<td>tp-diagcode</td>
<td>Yes*</td>
</tr>
<tr>
<td>tp-connected</td>
<td>object-available</td>
<td>--</td>
<td>No</td>
</tr>
<tr>
<td>tp-x-conntime</td>
<td>other-state-change</td>
<td>--</td>
<td>No</td>
</tr>
<tr>
<td>tp-x-reset</td>
<td>transient-fault</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>tp-data-usage</td>
<td>data-usage</td>
<td>data-usage</td>
<td>Yes**</td>
</tr>
</tbody>
</table>

* An event-number template is provided for each of these events to customize the event message text (display).

** An event-number template is provided for the event message text of this private event.
11

Procedure Calls for Standard Events

This section describes procedure calls that help subsystems and applications format standard events:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to Procedure Calls</td>
<td>11-2</td>
</tr>
<tr>
<td>EMS_TRANSIENTFAULT_EVT_BLD_Procedure</td>
<td>11-2</td>
</tr>
<tr>
<td>EMS_OBJ_UNAVAIL_EVT_BLD_Procedure</td>
<td>11-7</td>
</tr>
<tr>
<td>EMS_OBJ_AVAIL_EVT_BLD_Procedure</td>
<td>11-13</td>
</tr>
<tr>
<td>EMS_OTHER_STATE_CHANGE_EVT_BLD_Procedure</td>
<td>11-18</td>
</tr>
<tr>
<td>EMS_OPER_ATTN_NEEDED_EVT_BLD_Procedure</td>
<td>11-23</td>
</tr>
<tr>
<td>EMS_OPER_ATTN_COMPED_EVT_BLD_Procedure</td>
<td>11-27</td>
</tr>
<tr>
<td>Standard Usage Threshold Event</td>
<td>11-31</td>
</tr>
<tr>
<td>EMS_COMMON_TOKENS_EVT_BLD_Procedure</td>
<td>11-44</td>
</tr>
<tr>
<td>Procedure for Sending Events</td>
<td>11-48</td>
</tr>
<tr>
<td>Program Example</td>
<td>11-48</td>
</tr>
</tbody>
</table>

For a description of standard events and how to use them, see Section 9, Standard Events.

For a description of how to generate standard events, see Section 10, Generating Standard Events.

Appendix C, Standard Event Sample Files, provides three samples:

- A data definition language (DDL) file
- An EMS template
- A subsystem event external specification.

You can copy the text from these sample files for use on your system.
Introduction to Procedure Calls

To generate a standard event, your subsystem must call the corresponding event-building procedure followed by a send procedure. One event-building procedure is provided for each of the standard event types.

To generate common tokens only, use the EMS_COMMON_TOKENS_EVT_BLD_ procedure call.

After calling an event-building procedure, you can append other tokens to the standard event by calling the EMS procedures EMSADDTOKENS, EMSADDTOKENMAPS, EMSADDSUBJECT, or EMSADDSUBJECTMAP.

The event-building procedures call EMSINIT internally, so users should not call EMSINIT or EMSINITMAP when using the event-building procedures to generate the standard events.

These procedures are callable from TAL, COBOL85, and C programs. The procedure definitions are in the system file EXTDECS. The literals used in the procedure descriptions are included in the files ZEMSDDL, ZEMSTACL, ZEMSC, ZEMSTAL, and ZEMSCOB.

This section describes the event-building procedures. For the definition of required tokens and their meanings, see Section 9, Standard Events.

Note. No range checking is done on the procedure parameter values except for current state and previous state. Also, the value for BOOLEAN is ZSPI-VAL-FALSE (0) for FALSE and is otherwise TRUE.

EMS_TRANSIENT_FAULT_EVT_BLD_ Procedure

The EMS_TRANSIENT_FAULT_EVT_BLD_ procedure formats the standard transient fault event. The event reports the object that received the transient fault.
The syntax of EMS_TRANSIENT_FAULT_EVT_BLD__ is:

```c
{ status := } EMS_TRANSIENT_FAULT_EVT_BLD__( buffer ! i,o
{ CALL } , maxbuflen ! i,
, SSID ! i
, Event number ! i
, Transient fault type ! i
, Subject token ID ! i
, Subject value ! i
, [ Subject length ] ! i
, [ Subject SSID ] ! i
, [ buflen ] ! o
, [ Critical indicator ] ! i
, [ Manager name:Manager name length ] ! i:i
, [ Job ID ] ! i
);}
```

**status** returned value

INT: value

returns one of these SPI error numbers:

- 0 No error
- 1 Invalid buffer format
- 2 Illegal parameter value
- 3 Missing parameter
- 4 Illegal parameter address
- 5 Buffer full
- 6 Invalid checksum
- 7 Internal error
- 8 Token not found
- 9 Illegal token code or map
- 10 Invalid subsystem ID
- 11 Operation not supported
- 12 Insufficient stack space

**buffer** input, output

INT .EXT:ref:*

is the event-message buffer provided by the caller.

**maxbuflen** input

INT: value

is the length, in bytes, of the buffer, and cannot be greater than ZEMS-VAL-EVT-BUFLLEN (4024).

**SSID** input

INT .EXT:ref:6

is the ID of the subsystem that owns the object that received the transient fault.
**Event number**  
**input**  
**INT : value**  
is a number, specific to this subsystem, that identifies this event message.

**Transient fault type**  
**input**  
**INT : value**  
is the type of the transient fault. The value is defined by the caller and should be greater than or equal to ZEMS-VAL-MIN-USER-VALUE (1024).

**Subject token ID**  
**input**  
**INT (32) : value**  
is the token code of the subject of this event message.

**Subject value**  
**input**  
**STRING . EXT : ref : * **  
is the value of the subject to be added to the event message.

**Subject length**  
**input**  
**INT : value**  
if present, is the length, in bytes, of the Subject value. This parameter is ignored unless the Subject token ID is defined as a variable-length token. For more information, see Considerations on page 11-6.

**Subject SSID**  
**input**  
**INT . EXT : ref : 6**  
if present, is the ID of the subsystem to which the Subject token ID belongs. If not present or the value is all zeros, the default SSID (specified in SSID) is used. In this case, the Subject SSID is not added to the event buffer.

**buflen**  
**output**  
**INT : value**  
if present, is the length, in bytes, of the formatted buffer. If the buffer cannot be built, zero is returned.

**Critical Indicator**  
**input**  
**BOOLEAN : value**  
if present, is a Boolean variable that indicates the critical nature of the problem reported in the event. The literal ZSPI-VAL-FALSE (0) indicates the problem is not critical. Otherwise, the problem is critical.
If not present, the default is ZSPI-VAL-FALSE.

Manager name  input
  STRING .EXT:ref:*  
  if present, is the name of the manager process of the object that received the transient fault.
  If present and Manager name length is not zero, Manager name is added to the event buffer. Otherwise, Manager name is not added to the event buffer.

Manager name length  input
  INT:value  
  if present, is the length, in bytes, of the Manager name.
  If not present or the value is zero, Manager name is not added to the event buffer.

Job ID  input
  INT:value  
  if present, is the job ID (for use by the NetBatch facility) that associates with the object experiencing the transient fault.
  If Job ID is zero or not present, see Considerations on page 11-6.

Condition Code Settings
  The condition code has no meaning following a call to EMS_TRANSIENT_FAULT_EVT_BLD_.

Considerations

- To generate the common tokens:
  
  **SSID:** from SSID.

  **Event subject token code:** from Subject token ID

  **Event subject value:** from Subject value

  **Event subject SSID:** from Subject SSID

  **Manager Name** (zems-tkn-name-manager): from Manager name.

  **Event Number** (zems-tkn-eventnumber): from Event number

  **Standard Content type** (zems-tkn-content-standard): ZEMS-VAL-TRANSIENT-FAULT.

  **User Content type** (zems-tkn-content-user): ZEMS-VAL-NULL.

  **Critical Indicator** (zems-tkn-emphasis): from Critical indicator.

  **Suppress display** (zems-tkn-suppress-display): ZSPI-VAL-FALSE.

  **Batch job ID** (zems-tkn-batchjob-id): from Job ID.

  The values for **User Content type** and **Suppress display** can be changed by calling EMSADDTOKENS.

- To generate the additional tokens:

  **Transient fault type** (zems-tkn-txfault-type): from Transient fault type.

- If **Job ID** is not present or the value is zero, the procedure calls system procedure PROCESS_GETINFO_ to get the job ID. If the job ID obtained is not zero, the job ID is added to the event buffer. Otherwise, the job ID is not added to the event buffer.

  The nonzero job ID obtained from calling PROCESS_GETINFO_ is the NetBatch job ID associated with the process that calls this EMS event-building procedure.

  If the subject reported in this event is managed by a process different from the process that calls this EMS event-building procedure, the caller should provide the **Job ID**, which is the NetBatch job ID associated with the process that manages the subject.

- If the **Subject token ID** has a variable-length value, choose one of these ways to specify the length:

  - If the **Subject length** is present, the **Subject length** is the length, in bytes, of the **Subject value**.

  - If the **Subject length** is not present, place the subject length in the two bytes that immediately precede the **Subject value**. Make the **Subject
value point to the extra two bytes. Do not count the two-byte length field as part of the subject length.

- The subject of this event is the object that received the transient fault. The Subject value must be fully qualified so that it can be uniquely identified from any point in a network; for example, \nodename$.volume.subvolume.filename.

Use EMSADDSUBJECT or EMSADDSUBJECTMAP to specify additional subjects.

- Use EMSADDTOKENS or EMSADDTOKENMAPS to specify any additional information that is useful in describing or diagnosing the transient fault.

### EMS_OBJ_UNAVAIL_EVT_BLD_ Procedure

The EMS_OBJ_UNAVAIL_EVT_BLD_ procedure formats the standard object unavailable event. The event reports the object that became unavailable or the object that the operator cannot restart.

The syntax of EMS_OBJ_UNAVAIL_EVT_BLD_ is:

```c
{ status := } EMS_OBJ_UNAVAIL_EVT_BLD_ ( buffer !i,o 
{ CALL } , maxbuflen !i 
, SSID !i 
, Event number !i 
, Current state !i 
, Previous state !i 
, Change reason !i 
, Subject token ID !i 
, Subject value !i 
, [ Subject length ] !i 
, [ Subject SSID ] !i 
, [ Underlying object name:Underlying Object name length ] !i:i 
, [ Symptom string:Symptom string length ] !i:i 
, [ buflen ] !o 
, [ Critical indicator ] !i 
, [ Manager name:Manager name length ] !i:i 
, [ Job ID ] !i 
);
```

**status returned value**

INT:value

returns one of these SPI error numbers:

- 0 No error
- -1 Invalid buffer format
-2 Illegal parameter value
-3 Missing parameter
-4 Illegal parameter address
-5 Buffer full
-6 Invalid checksum
-7 Internal error
-8 Token not found
-9 Illegal token code or map
-10 Invalid subsystem ID
-11 Operation not supported
-12 Insufficient stack space

buffer input, output
INT .EXT:ref:*
is the event-message buffer provided by the caller.

maxbuflen input
INT:value
is the length, in bytes, of the buffer, and cannot be greater than ZEMS-VAL-EVT-BUFLEN (4024).

SSID input
INT .EXT:ref:6
is the ID of the subsystem that owns the object that is unavailable.

Event number input
INT:value
is a number, specific to this subsystem, that identifies this event message.

Current state input
INT:value
is the current state of the object that became unavailable.

Previous state input
INT:value
is the previous state of the object that became unavailable.

Change reason input
INT:value
is the reason why the object is unavailable. Standard values are:

ZEMS-VAL-NORM-TERMINATION
ZEMS-VAL-OPERATOR-INITIATED
ZEMS-VAL-UNDERLYING-FAILED
ZEMS-VAL-INTERNAL-FAILED
ZEMS-VAL-REASON-UNKNOWN

Users can also enter their own defined values.

Subject token ID input
INT(32):value
is the token code of the subject of this event message.

Subject value input
STRING .EXT:ref:*
is the value of the subject to be added to the event message.

Subject length input
INT:value
if present, is the length, in bytes, of the Subject value. This parameter is ignored unless the Subject token ID is defined as a variable-length token. For more information, see Considerations on page 11-11.

Subject SSID input
INT .EXT:ref:6
if present, is the ID of the subsystem to which the Subject token ID belongs.
If not present or the value is all zeros, the default SSID (specified in SSID) is assumed. In this case, the Subject SSID is not added to the event buffer.

Underlying object name input
STRING .EXT:ref:* 
if present, is the name of the lower-level object whose failure caused the object reported in the subject of this event to go out of service. This parameter is used only if the Change reason parameter has value ZEMS-VAL-UNDERLYING-FAILED. Otherwise, the parameter is ignored.

If present and Underlying object name length is not zero, Underlying object name is added to the event buffer. Otherwise, Underlying object name is not added to the event buffer.

Underlying object name length input
INT:value
if present, is the length, in bytes, of the underlying object name.
If not present or the value is zero, Underlying object name is not added to the event buffer.
**Symptom string**

**Input**

`STRING .EXT:ref:*`

If present, is a string that uniquely identifies the code location where the failure was found.

If present and **Symptom string length** is not zero, **Symptom string** is added to the event buffer. Otherwise, **Symptom string** is not added to the event buffer.

**Symptom string length**

**Input**

`INT:value`

If present, is the length, in bytes, of the symptom string.

If not present or the value is zero, **Symptom string** is not added to the event buffer.

**buflen**

**Output**

`INT:value`

If present, is the length, in bytes, of the formatted buffer. If the buffer cannot be built, zero is returned.

**Critical Indicator**

**Input**

`BOOLEAN:value`

If present, is a Boolean variable that indicates the critical nature of the problem reported in the event. The literal ZSPI-VAL-FALSE (0) indicates the problem is not critical. Otherwise, the problem is critical.

If not present, the default is ZSPI-VAL-FALSE.

**Manager name**

**Input**

`STRING .EXT:ref:*`

If present, is the name of the manager process for the object that is unavailable.

If present and **Manager name length** is not zero, **Manager name** is added to the event buffer. Otherwise, **Manager name** is not added to the event buffer.

**Manager name length**

**Input**

`INT:value`

If present, is the length, in bytes, of the **Manager name**.

If not present or the value is zero, **Manager name** is not added to the event buffer.

**Job ID**

**Input**

`INT:value`
if present, is the job ID (for use by the NetBatch facility) that associates with the object that became unavailable.

If Job ID is zero or not present, see Considerations on page 11-11.

Condition Code Settings
The condition code has no meaning following a call to EMS_OBJ_UNAVAIL_EVT_BLD.

Considerations
- To generate the common tokens:
  SSID: from SSID.
  Event subject token code: from Subject token ID
  Event subject value: from Subject value
  Event subject SSID: from Subject SSID
  Manager Name (zems-tkn-name-manager): from Manager name. Conditional.
  Event Number (zems-tkn-eventnumber): from Event number
  Standard Content type (zems-tkn-content-standard): ZEMS-VAL-OBJECT-UNAVAILABLE.
  User Content type (zems-tkn-content-user): ZEMS-VAL-NULL.
  Critical Indicator (zems-tkn-emphasis): from Critical indicator.
  Suppress display (zems-tkn-suppress-display): ZSPI-VAL-FALSE.
  Batch job ID (zems-tkn-batchjob-id): from Job ID. Conditional.

  The values for User Content type and Suppress display can be changed by calling EMSADDTOKENs.

- To generate the additional tokens:
  Current state (zems-tkn-state-current): from Current state.
  Previous state (zems-tkn-state-previous): from Previous state.
  Change reason (zems-tkn-change-reason): from Change reason.
  Underlying object name (zems-tkn-underlying-obj-name): from Underlying object name. Conditional.
If the Subject token ID has a variable-length value, choose one of these ways to specify the length:

- If the Subject length is present, the Subject length is the length, in bytes, of the Subject value.
- If the Subject length is not present, place the subject length in the two bytes that immediately precede the Subject value. Make the Subject value point to the extra two bytes. Do not count the two-byte length field as part of the subject length.

If Job ID is not present or the value is zero, the procedure calls system procedure PROCESS_GETINFO_ to get the job ID. If the job ID obtained is nonzero, the job ID is added to the event buffer. Otherwise, the job ID is not added to the event buffer.

The nonzero job ID obtained from calling PROCESS_GETINFO_ is the NetBatch job ID associated with the process that calls this EMS event-building procedure.

If the subject reported in this event is managed by a process different from the process that calls this EMS event-building procedure, the caller should provide the Job ID, which is the NetBatch job ID associated with the process that manages the subject.

The subject of this event is the object that became unavailable. The Subject value must be fully qualified so that it can be uniquely identified from any point in a network. An example is `\nodename.$volume.subvolume.filename`.

Use EMSADDSUBJECT or EMSADDSUBJECTMAP to specify additional subjects.

Use EMSADDTOKENS or EMSADDTOKENMAPS to specify any additional information that is useful in describing or diagnosing the failure.
EMS_OBJ_AVAIL_EVT_BLD_ Procedure

The EMS_OBJ_AVAIL_EVT_BLD_ procedure formats the standard object available event. The event reports the object that became available.

The syntax of EMS_OBJ_AVAIL_EVT_BLD_ is:

```assembly
{ status := } EMS_OBJ_AVAIL_EVT_BLD_ ( buffer ! i,o
{ CALL } , max buflen ! i
, SSID ! i
, Event number ! i
, Current state ! i
, Previous state ! i
, Change reason ! i
, Subject token ID ! i
, Subject value ! i
, [ Subject length ] ! i
, [ Subject SSID ] ! i
, [ buflen ] ! o
, [ Critical indicator ] ! i
, [ Manager name:Manager name length ] ! i:i
, [ Job ID ] ! i
);
```

`status` returned value

INT: value

returns one of these SPI error numbers:

- 0 No error
- -1 Invalid buffer format
- -2 Illegal parameter value
- -3 Missing parameter
- -4 Illegal parameter address
- -5 Buffer full
- -6 Invalid checksum
- -7 Internal error
- -8 Token not found
- -9 Illegal token code or map
- -10 Invalid subsystem ID
- -11 Operation not supported
- -12 Insufficient stack space

`buffer` input, output

INT .EXT:ref:*

is the event-message buffer provided by the caller.
maxbuflen  
  INT: value
  is the length, in bytes, of the buffer, and cannot be greater than ZEMS-VAL-EVT-BUFLEN (4024).

SSID
  INT .EXT: ref: 6
  is the ID of the subsystem that owns the object that is available.

Event number
  INT: value
  is a number, specific to this subsystem, that identifies this event message.

Current state
  INT: value
  is the current state of the object that became available.

Previous state
  INT: value
  is the previous state of the object that became available.

Change reason
  INT: value
  is the reason for the state change. Standard values for this token are:

  ZEMS-VAL-OPERATOR-INITIATED
  ZEMS-VAL-UNDERLYING-UP
  ZEMS-VAL-REASON-UNKNOWN
  the caller can specify its own value, but it must be greater than or equal to
  ZEMS-VAL-MIN-USER-VALUE (1024).

Subject token ID
  INT (32): value
  is the token code of the subject of this event message.

Subject value
  STRING .EXT: ref: *
  is the value of the subject to be added to the event message.
**Subject length**

*input*

**INT**:value

if present, is the length, in bytes, of the Subject value. This parameter is ignored unless the Subject token ID is defined as a variable-length token. For more information, see [Considerations](#) on page 11-16.

**Subject SSID**

*input*

**INT .EXT**:ref:6

if present, is the ID of the subsystem to which the Subject token ID belongs. If not present or the value is all zeros, the default SSID (specified in SSID) is assumed. In this case, the Subject SSID is not added to the event buffer.

**buflen**

*output*

**INT**:value

if present, is the length, in bytes, of the formatted buffer. If the buffer cannot be built, zero is returned.

**Critical Indicator**

*input*

**BOOLEAN**:value

if present, is a Boolean variable that indicates the critical nature of the problem reported in the event. The literal ZSPI-VAL-FALSE (0) indicates the problem is not critical. Otherwise, the problem is critical.

If not present, the default is ZSPI-VAL-FALSE.

**Manager name**

*input*

**STRING .EXT**:ref:*

if present, is the name of the manager process for the object that became available.

If present and Manager name length is not zero, Manager name is added to the event buffer. Otherwise, Manager name is not added to the event buffer.

**Manager name length**

*input*

**INT**:value

if present, is the length, in bytes, of the Manager name.

If not present or the value is zero, Manager name is not added to the event buffer.
If Job ID is zero or not present, see Considerations on page 11-16.

**Condition Code Settings**

The condition code has no meaning following a call to EMS_OBJ_AVAIL_EVT_BLD_.

**Considerations**

- To generate the common tokens:
  - SSID: from SSID.
  - Event subject token code: from Subject token ID
  - Event subject value: from Subject value
  - Event subject SSID: from Subject SSID. Conditional.
  - Manager Name (zems-tkn-name-manager): from Manager name. Conditional.
  - Event Number (zems-tkn-eventnumber): from Event number
  - Standard Content type (zems-tkn-content-standard): ZEMS-VAL-OBJECT-AVAILABLE.
  - User Content type (zems-tkn-content-user): ZEMS-VAL-NULL.
  - Critical Indicator (zems-tkn-emphasis): from Critical indicator.
  - Suppress display (zems-tkn-suppress-display): ZSPI-VAL-FALSE.
  - Batch job ID (zems-tkn-batchjob-id): from Job ID. Conditional.

The values for User Content type and Suppress display can be changed by calling EMSADDTOKENS.

- To generate the additional tokens:
  - Current state (zems-tkn-state-current): from Current state.
  - Previous state (zems-tkn-state-previous): from Previous state.
  - Change reason (zems-tkn-change-reason): from Change reason.
• If the Subject token ID has a variable-length value, choose one of these ways to specify the length:
  • If the Subject length is present, the Subject length is the length, in bytes, of the Subject value.
  • If the Subject length is not present, place the subject length in the two bytes that immediately precede the Subject value. Make the Subject value point to the extra two bytes. Do not count the two-byte length field as part of the subject length.

• If Job ID is not present or the value is zero, the procedure calls system procedure PROCESS_GETINFO_ to get the job ID. If the job ID obtained is nonzero, the job ID is added to the event buffer. Otherwise, the job ID is not added to the event buffer.

The nonzero job ID obtained from calling PROCESS_GETINFO_ is the NetBatch job ID associated with the process that calls this EMS event-building procedure.

If the subject reported in this event is managed by a process different from the process that calls this EMS event-building procedure, the caller should provide the Job ID, which is the NetBatch job ID associated with the process that manages the subject.

• The subject of this event is the object that became available. The Subject value must be fully qualified so that it can be uniquely identified from any point in a network. An example is \nodename.$volume.subvolume.filename. Use EMSADDSUBJECT or EMSADDSUBJECTMAP to specify additional subjects.
The EMS_OTHER_STATE_CHANGE_EVT_BLD_ procedure formats the standard other state change event. The event reports the object that has a state change other than object available or unavailable, which:

- Requires operator intervention before its state can change
- Persists in the same state long enough that the operator should take notice

The syntax of EMS_OTHER_STATE_CHANGE_EVT_BLD_ is:

```
{ status := } EMS_OTHER_STATE_CHANGE_EVT_BLD_( buffer !i,o
{ CALL } , maxbuflen
, SSID !i
, Event number !i
, Current state !i
, Previous state !i
, Change reason !i
, Subject token ID !i
, Subject value !i
, [ Subject length ] !i
, [ Subject SSID ] !i
, [ buflen ] !o
, [ Critical indicator ] !i
, [ Manager name:Manager name length ] !i:i
, [ Job ID ] !i
);
```

`status` returned value

`INT:value`

returns one of these SPI error numbers:

- 0 No error
- -1 Invalid buffer format
- -2 Illegal parameter value
- -3 Missing parameter
- -4 Illegal parameter address
- -5 Buffer full
- -6 Invalid checksum
- -7 Internal error
- -8 Token not found
- -9 Illegal token code or map
- -10 Invalid subsystem ID
- -11 Operation not supported
- -12 Insufficient stack space
buffer input, output
   INT .EXT:ref:* is the event-message buffer provided by the caller.

maxbuflen input
   INT:value is the length, in bytes, of the buffer, and cannot be greater than ZEMS-VAL-EVT-BUFLEN (4024).

SSID input
   INT .EXT:ref:6 is the ID of the subsystem that owns the object that had the state change.

Event number input
   INT:value is a number, specific to this subsystem, that identifies this event message.

Current state input
   INT:value is the current state of the object that had the state change.

Previous state input
   INT:value is the previous state of the object that had the state change.

Change reason input
   INT:value is the reason for the state change. Standard values for this token are:
   ZEMS-VAL-OPERATOR-INITIATED
   ZEMS-VAL-UNDERLYING-UP
   ZEMS-VAL-REASON-UNKNOWN
   The caller can specify its own value, but it must be greater than or equal to ZEMS-VAL-MIN-USER-VALUE (1024).

Subject token ID input
   INT(32):value is the token code of the subject of this event message.
Subject value input
STRING .EXT:ref:* is the value of the subject to be added to the event message.

Subject length input
INT:value if present, is the length, in bytes, of the Subject value. This parameter is ignored unless the Subject token ID is defined as a variable-length token. For more information, see Considerations on page 11-21.

Subject SSID input
INT .EXT:ref:6 if present, is the ID of the subsystem to which the Subject token ID belongs. If not present or the value is all zeros, the default SSID (specified in SSID) is assumed. In this case, the Subject SSID is not added to the event buffer.

buflen output
INT:value if present, is the length, in bytes, of the formatted buffer. If the buffer cannot be built, zero is returned.

Critical Indicator input
BOOLEAN:value if present, is a Boolean variable that indicates the critical nature of the problem reported in the event. The literal ZSPI-VAL-FALSE (0) indicates the problem is not critical. Otherwise, the problem is critical.

If not present, the default is ZSPI-VAL-FALSE.

Manager name input
STRING .EXT:ref:* if present, is the name of the manager process for the object that had the state change.

If present and Manager name length is not zero, Manager name is added to the event buffer. Otherwise, Manager name is not added to the event buffer.

Manager name length input
INT:value if present, is the length, in bytes, of the Manager name.

If not present or the value is zero, Manager name is not added to the event buffer.
Condition Code Settings

The condition code has no meaning following a call to EMS_OTHER_STATE_CHANGE_EVT_BLD_.

Considerations

- To generate the common tokens:
  
  SSID: from SSID.

  Event subject token code: from Subject token ID

  Event subject value: from Subject value

  Event subject SSID: from Subject SSID. Conditional.

  Manager Name (zems-tkn-name-manager): from Manager name. Conditional.

  Event Number (zems-tkn-eventnumber): from Event number

  Standard Content type (zems-tkn-content-standard): ZEMS-VAL-OTHER-STATE-CHANGE.

  User Content type (zems-tkn-content-user): ZEMS-VAL-NUL

  Critical Indicator (zems-tkn-emphasis): from Critical indicator.

  Suppress display (zems-tkn-suppress-display): ZSPI-VAL-FALSE.

  Batch job ID (zems-tkn-batchjob-id): from Job ID. Conditional.

  The values for User Content type and Suppress display can be changed by calling EMSADDTOKENS.

- To generate the additional tokens:

  Current state (zems-tkn-state-current): from Current state.

  Previous state (zems-tkn-state-previous): from Previous state.

  Change reason (zems-tkn-change-reason): from Change reason.
If the **Subject token ID** has a variable-length value, choose one of these ways to specify the length:

- If the **Subject length** is present, the **Subject length** is the length, in bytes, of the **Subject value**.

- If the **Subject length** is not present, place the subject length in the two bytes that immediately precede the **Subject value**. Make the **Subject value** point to the extra two bytes. Do not count the two-byte length field as part of the subject length.

If **Job ID** is not present or the value is zero, the procedure calls system procedure PROCESS_GETINFO_ to get the job ID. If the job ID obtained is nonzero, the job ID is added to the event buffer. Otherwise, the job ID is not added to the event buffer.

The nonzero job ID obtained from calling PROCESS_GETINFO_ is the NetBatch job ID associated with the process that calls this EMS event-building procedure. If the subject reported in this event is managed by a process different from the process that calls this EMS event-building procedure, the caller should provide the **Job ID**, which is the NetBatch job ID associated with the process that manages the subject.

The subject of this event is the object that had the state change. The **Subject value** must be fully qualified so that it can be uniquely identified from any point in a network. An example is `\nodename.$volume.subvolume.filename`.

Use EMSADDSUBJECT or EMSADDSUBJECTMAP to specify additional subjects.
**EMS_OPER_ATTN_NEEDED_EVT_BLD_ Procedure**

The EMS_OPER_ATTN_NEEDED_EVT_BLD_ procedure formats the standard operator attention needed event. The event reports the object and the needed operator action, such as mounting a tape or changing paper in a printer.

The syntax of EMS_OPER_ATTN_NEEDED_EVT_BLD_ is:

```
{ status := } EMS_OPER_ATTN_NEEDED_EVT_BLD_( buffer !i,o 
{ CALL } , max buflen !i 
, SSID !i 
, Event number !i 
, Action ID !i 
, Subject token ID !i 
, Subject value !i 
, [ Subject length ] !i 
, [ Subject SSID ] !i 
, [ buflen ] !o 
, [ Critical indicator ] !i 
, [ Manager name:Manager name length ] !i:i 
, [ Job ID ] !i 
); 
```

*status* returned value

INT:value

returns one of these SPI error numbers:

- 0 No error
- -1 Invalid buffer format
- -2 Illegal parameter value
- -3 Missing parameter
- -4 Illegal parameter address
- -5 Buffer full
- -6 Invalid checksum
- -7 Internal error
- -8 Token not found
- -9 Illegal token code or map
- -10 Invalid subsystem ID
- -11 Operation not supported
- -12 Insufficient stack space

*buffer* input, output

INT .EXT:ref:*

is the event-message buffer provided by the caller.
maxbuflen input
INT:value
is the length, in bytes, of the buffer, and cannot be greater than
ZEMS-VAL-EVT-BUFLEN (4024).

SSID input
INT .EXT:ref:6
is the ID of the subsystem that owns the object that needs operator attention.

Event number input
INT:value
is a number, specific to this subsystem, that identifies this event message.

Action ID input
INT:value
is an integer value used by event consumers to match operator attention
completed events with operator attention needed event pairs. For every generated
operator attention needed event, there must also be an operator attention
completed event.

Subject token ID input
INT(32):value
is the token code of the subject of this event message.

Subject value input
STRING .EXT:ref:*
is the value of the subject to be added to the event message.

Subject length input
INT:value
if present, is the length, in bytes, of the Subject value. This parameter is
ignored unless the Subject token ID is defined as a variable-length token. For
more information, see Considerations on page 11-26.

Subject SSID input
INT .EXT:ref:6
if present, is the ID of the subsystem to which the Subject token ID belongs.
If not present or the value is all zeros, the default SSID (specified in SSID) is
assumed. In this case, the Subject SSID is not added to the event buffer.
Procedure Calls for Standard Events

EMS_OPER_ATTN_NEEDED_EVT_BLD_
Procedure

buflen output

INT:value
if present, is the length, in bytes, of the formatted buffer. If the buffer cannot be built, zero is returned.

Critical Indicator input

BOOLEAN:value
if present, is a Boolean variable that indicates the critical nature of the problem reported in the event. The literal ZSPI-VAL-FALSE (0) indicates the problem is not critical. Otherwise, the problem is critical.

If not present, the default is ZSPI-VAL-FALSE.

Manager name input

STRING .EXT:ref:* 
if present, is the name of the manager process for the object that needs operator attention.

If present and Manager name length is not zero, Manager name is added to the event buffer. Otherwise, Manager name is not added to the event buffer.

Manager name length input

INT:value
if present, is the length, in bytes, of the Manager name.

If not present or the value is zero, Manager name is not added to the event buffer.

Job ID input

INT:value
if present, is the job ID (for use by the NetBatch facility) that associates with the object that needs operator attention.

If Job ID is zero or not present, see Considerations on page 11-26.
Condition Code Settings

The condition code has no meaning following a call to EMS_OPER_ATTN_NEEDED_EVT_BLD_.

Considerations

- To generate the common tokens:

  SSID: from SSID.
  
  Event subject token code: from Subject token ID
  Event subject value: from Subject value
  Event subject SSID: from Subject SSID. Conditional.
  
  Manager Name (zems-tkn-name-manager): from Manager name. Conditional.
  
  Event Number (zems-tkn-eventnumber): from Event number
  
  Standard Content type (zems-tkn-content-standard): ZEMS-VAL-ATTN-NEEDED.
  
  User Content type (zems-tkn-content-user): ZEMS-VAL-NULL.
  
  Critical Indicator (zems-tkn-emphasis): from Critical indicator.
  
  Suppress display (zems-tkn-suppress-display): ZSPI-VAL-FALSE.
  
  Batch job ID (zems-tkn-batchjob-id): from Job ID. Conditional.
  
  The values for User Content type and Suppress display can be changed by calling EMSADDTOKENS.

- To generate the additional tokens:

  Action ID (zems-tkn-action-id): from Action ID.
  
  Attention needed (zems-tkn-action-needed): ZSPI-VAL-TRUE.

- If the Subject token ID has a variable-length value, choose one of these ways to specify the length.
  
  - If the Subject length is present, the Subject length is the length, in bytes, of the Subject value.
  
  - If the Subject length is not present, place the subject length in the two bytes that immediately precede the Subject value. Make the Subject value point to the extra two bytes. Do not count the two-byte length field as part of the subject length.
If Job ID is not present or the value is zero, the procedure calls system procedure PROCESS_GETINFO_ to get the job ID. If the job ID obtained is nonzero, the job ID is added to the event buffer. Otherwise, the job ID is not added to the event buffer.

The nonzero job ID obtained from calling PROCESS_GETINFO_ is the NetBatch job ID associated with the process that calls this EMS event-building procedure.

If the subject reported in this event is managed by a process different from the process that calls this EMS event-building procedure, the caller should provide the Job ID, which is the NetBatch job ID associated with the process that manages the subject.

The subject of this event is the object that needs operator attention. The Subject value must be fully qualified so that it can be uniquely identified from any point in a network. An example is \nodename.$volume.subvolume.filename.

Use EMSADDSUBJECT or EMSADDSUBJECTMAP to specify additional subjects.

EMS_OPER_ATTN_COMPED_EVT_BLD_ Procedure

The EMS_OPER_ATTN_COMPED_EVT_BLD_ procedure formats the standard operator attention completed event. The event reports the object that no longer needs operator attention.

The syntax of EMS_OPER_ATTN_COMPED_EVT_BLD_ is:

```c
{ status := } EMS_OPER_ATTN_COMPED_EVT_BLD_( buffer
!i,o
{ CALL } , maxbuflen
!i , SSID
!i , Event number
!i , Action ID
!i , Subject token ID
!i , Subject value
!i , [ Subject length ]
!i , [ Subject SSID ]
!i , [ buflen ]
!o , [ Critical indicator ]
!i , [ Manager name:Manager name length ] !i:i
, [ Job ID ]
!i )
```

status returned value

INT:value

returns one of these SPI error numbers:

0 No error
-1 Invalid buffer format
-2 Illegal parameter value
-3 Missing parameter
-4 Illegal parameter address
-5 Buffer full
-6 Invalid checksum
-7 Internal error
-8 Token not found
-9 Illegal token code or map
-10 Invalid subsystem ID
-11 Operation not supported
-12 Insufficient stack space

buffer input, output
INT .EXT:ref:* is the event-message buffer provided by the caller.

maxbuflen input
INT :value is the length, in bytes, of the buffer and cannot be greater than ZEMS-VAL-EVT-BUFLEN (4024).

SSID input
INT .EXT:ref:6 is the ID of the subsystem that owns the object that no longer needs operator attention.

Event number input
INT :value is a number, specific to this subsystem, that identifies this event message.

Action ID input
INT :value is the action ID that was specified in the original operator attention needed event.

Subject token ID input
INT(32) :value is the token code of the subject of this event message.

Subject value input
STRING .EXT:ref:* is the value of the subject to be added to the event message.
Subject length input
   INT:value
   if present, is the length, in bytes, of the Subject value. This parameter is ignored unless the Subject token ID is defined as a variable-length token. For more information, see Considerations on page 11-30.

Subject SSID input
   INT .EXT:ref:6
   if present, is the ID of the subsystem to which the Subject token ID belongs.
   If not present or the value is all zeros, the default SSID (specified in SSID) is assumed. In this case, the Subject SSID is not added to the event buffer.

buflen output
   INT:value
   if present, is the length, in bytes, of the formatted buffer. If the buffer cannot be built, zero is returned.

Critical Indicator input
   BOOLEAN:value
   if present, is a Boolean variable that indicates the critical nature of the problem reported in the event. The literal ZSPI-VAL-FALSE (0) indicates the problem is not critical. Otherwise, the problem is critical.
   If not present, the default is ZSPI-VAL-FALSE.

Manager name input
   STRING .EXT:ref:* 
   if present, is the name of the manager process for the object that no longer needs operator attention.
   If present and Manager name length is not zero, Manager name is added to the event buffer. Otherwise, Manager name is not added to the event buffer.

Manager name length input
   INT:value
   if present, is the length, in bytes, of the Manager name.
   If not present or the value is zero, Manager name is not added to the event buffer.

Job ID input
   INT:value
if present, is the job ID (for use by the NetBatch facility) that associates with the object that no longer needs operator attention.

If \textit{Job ID} is zero or not present, see Considerations on page 11-30.

\section*{Condition Code Settings}

The condition code has no meaning following a call to \texttt{EMS\_OPER\_ATTN\_COMPED\_EVT\_BLD}.

\section*{Considerations}

\begin{itemize}
  \item To generate the common tokens:
    \begin{itemize}
      \item \texttt{SSID}: from \texttt{SSID}.
      \item Event subject token code: from \texttt{Subject token ID}
      \item Event subject value: from \texttt{Subject value}
      \item Event subject SSID: from \texttt{Subject SSID}. Conditional.
      \item Manager Name (\texttt{zems-tkn-name-manager}): from \texttt{Manager name}. Conditional.
      \item Event Number (\texttt{zems-tkn-eventnumber}): from \texttt{Event number}
      \item Standard Content type (\texttt{zems-tkn-content-standard}): \texttt{ZEMS-VAL-ATTN-COMPLETED}.
      \item User Content type (\texttt{zems-tkn-content-user}): \texttt{ZEMS-VAL-NULL}.
      \item Critical Indicator (\texttt{zems-tkn-emphasis}): from \texttt{Critical indicator}.
      \item Suppress display (\texttt{zems-tkn-suppress-display}): \texttt{ZSPI-VAL-FALSE}.
      \item Batch job ID (\texttt{zems-tkn-batchjob-id}): from \texttt{Job ID}. Conditional.
    \end{itemize}

    The values for User Content type and Suppress display can be changed by calling \texttt{EMSADDTOKENS}.

  \item To generate the additional tokens:
    \begin{itemize}
      \item Action ID (\texttt{zems-tkn-action-id}): from \texttt{Action ID}.
      \item Attention needed (\texttt{zems-tkn-action-needed}): \texttt{ZSPI-VAL-FALSE}.
    \end{itemize}

  \item If the \texttt{Subject token ID} has a variable-length value, choose one of these ways to specify the length:
    \begin{itemize}
      \item If the \texttt{Subject length} is present, the \texttt{Subject length} is the length, in bytes, of the \texttt{Subject value}.
      \item If the \texttt{Subject length} is not present, place the subject length in the two bytes that immediately precede the \texttt{Subject value}. Make the \texttt{Subject value} point to the extra two bytes. Do not count the two-byte length field as part of the subject length.
    \end{itemize}
\end{itemize}
If Job ID is not present or the value is zero, the procedure calls system procedure PROCESS_GETINFO_ to get the job ID. If the job ID obtained is nonzero, the job ID is added to the event buffer. Otherwise, the job ID is not added to the event buffer.

The nonzero job ID obtained from calling PROCESS_GETINFO_ is the NetBatch job ID associated with the process that calls this EMS event-building procedure.

If the subject reported in this event is managed by a process different from the process that calls this EMS event-building procedure, the caller should provide the Job ID, which is the NetBatch job ID associated with the process that manages the subject.

The subject of this event is the object that no longer needs operator attention. The Subject value must be fully qualified so that it can be uniquely identified from any point in a network. An example is \nodename.$volume.subvolume.filename.

Use EMSADDSUBJECT or EMSADDSUBJECTMAP to specify additional subjects.

Standard Usage Threshold Event

This subsection describes:

- The algorithm and procedures for deciding whether to generate the standard usage threshold event
- The procedure to build the standard usage threshold event
- A programming example

The Algorithm

The algorithm for usage threshold event generation was taken from ISO/IEC 10165-2, (Information Technology—Open Systems Interconnection—Structure of Management Information—Part 2: Definition of Management Information) section 9.3.2 Gauge Threshold. This subsection is a slightly modified version of that text.

This algorithm is a general mechanism for generating one or more events from changes in any value of a usage variable. A hysteresis mechanism is provided to avoid the repeated triggering of event notifications when the usage makes small oscillations around a threshold value. This capability is provided by specifying threshold values in pairs, one a high threshold value and the other a low threshold value. The difference between threshold values is the hysteresis interval.

These four variables are needed to calculate when to send an event:
Procedure Calls for Standard Events

Constraints

The Configured_High_Level value must be greater than or equal to the Configured_Low_Level value.

Initially, both the Configured_High_Level_switch and the Configured_Low_Level_switch are set to true.

Behavior

An event is generated when the usage variable reaches or exceeds the Configured_High_Level value and the Configured_High_Level_switch is true. Concurrent with generating the event, the Configured_High_Level_switch is set to false and the Configured_Low_Level_switch is set to true. The setting of the switches guarantees that subsequent crossings of the Configured_High_Level level cause no further event generation unless the usage value becomes equal to or less than the Configured_Low_Level threshold value.

An event is also generated when the usage variable falls to, or below, the Configured_Low_Level value and the Configured_Low_Level_switch is true. Concurrent with generating the event, the Configured_Low_Level_switch is set to false and Configured_High_Level_switch is set to true. The setting of the switches guarantees that subsequent crossings of the Configured_Low_Level level cause no further event generation unless the usage value becomes equal to or greater than the Configured_High_Level threshold value.

Implementation

To implement the algorithm, three procedures are provided:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMS.UTCB.INIT</td>
<td>11-33</td>
</tr>
<tr>
<td>EMS.USAGE_THRESHOLD_CHK</td>
<td>11-35</td>
</tr>
<tr>
<td>EMS.USAGE_THRESHOLD_EVT_BLD</td>
<td>11-36</td>
</tr>
</tbody>
</table>

EMS Manual—426909-005
11-32
A usage threshold control block is allocated by the caller for each object whose utilization level is to be monitored. The usage threshold control block is used by the three procedures to keep information about the Configured_High_Level, Configured_Low_Level, Configured_High_Level_switch, Configured_Low_Level_switch, Current utilization level, Previous utilization level, Previous timestamp, and Internal status. The caller should not modify the content of this usage threshold control block.

For a programming example, see Usage Threshold Programming Example on page 11-41.

These steps explain how the procedures are used to decide whether to report the standard usage threshold event:

Step 1  EMS_UTCB_INIT_ is called to provide the usage threshold control block to later be used by EMS_USAGE_THRESHOLD_CHK_ and EMS_USAGE_THRESHOLD_EVT_BLD_.
    Call this procedure once for each object whose utilization level is to be reported.

Step 2  EMS_USAGE_THRESHOLD_CHK_ is called to check if the current utilization level needs to be reported.
    If EMS_USAGE_THRESHOLD_CHK_ indicates the event should be reported, call EMS_USAGE_THRESHOLD_EVT_BLD_ to build the standard usage threshold event and call the system procedure WRITEREAD to report the event. Otherwise, do not report the current utilization level.

Step 3  To check whether to report the next current utilization level, go to Step 2.

EMS_UTCB_INIT_

EMS_UTCB_INIT_ is called to provide a usage threshold control block to be used by EMS_USAGE_THRESHOLD_CHK_ and EMS_USAGE_THRESHOLD_EVT_BLD_ later.

The syntax of EMS_UTCB_INIT_ is:

```
{ status := } EMS_UTCB_INIT_ ( { CALL      }
  Usage Threshold Control Block            ! i,o
  , Configured High Level                 ! i
  , Configured Low Level                  ! i
  , Utilization level unit                ! i
);  
```

status            returned value

INT:value

returns one of these SPI error numbers:

- 0  No error
- -2  Illegal parameter value
- -3  Missing parameter
Used Threshold Control Block input, output

INT .EXT: ref: 20

is the control block used by subsequent calls to
EMS_USAGE_THRESHOLD_CHK_ and
EMS_USAGE_THRESHOLD_EVT_BLD_.

Configured High Level input

FLOAT :value

is the configured high usage level used in calculating whether a Usage Threshold event should be generated.

Configured Low Level input

FLOAT :value

is the configured low usage level used in calculating whether a Usage Threshold event should be generated.

Utilization level unit input

INT : value

is the utilization level unit used. Standard values are:

- ZEMS-VAL-UTIL-UNIT-PERCENT Unit in %
- ZEMS-VAL-UTIL-UNIT-USEC Unit in micro-seconds
- ZEMS-VAL-UTIL-UNIT-MSEC Unit in milliseconds
- ZEMS-VAL-UTIL-UNIT-HSEC Unit in hundredth-of-seconds
- ZEMS-VAL-UTIL-UNIT-TSEC Unit in tenth-of-seconds
- ZEMS-VAL-UTIL-UNIT-SEC Unit in seconds
- ZEMS-VAL-UTIL-UNIT-MIN Unit in minutes
- ZEMS-VAL-UTIL-UNIT-HOUR Unit in hours
- ZEMS-VAL-UTIL-UNIT-BPS Unit in bytes/sec
- ZEMS-VAL-UTIL-UNIT-KBPS Unit in kilobytes/sec
- ZEMS-VAL-UTIL-UNIT-MBPS Unit in megabytes/sec
- ZEMS-VAL-UTIL-UNIT-GBPS Unit in gigabytes/sec
- ZEMS-VAL-UTIL-UNIT-TBPS Unit in terabytes/sec
- ZEMS-VAL-UTIL-UNIT-BYTES Unit in bytes
- ZEMS-VAL-UTIL-UNIT-KBYTES Unit in kilobytes
- ZEMS-VAL-UTIL-UNIT-MBYTES Unit in megabytes
- ZEMS-VAL-UTIL-UNIT-GBYTES Unit in gigabytes
- ZEMS-VAL-UTIL-UNIT-TBYTES Unit in terabytes
- ZEMS-VAL-UTIL-UNIT-COUNTER Unit in count

Condition Code Settings

The condition code has no meaning following a call to EMS_UTCB_INIT_.


Considerations

The caller should provide enough space (20 words) for Usage Threshold Control Block, and the content should not be modified by the caller.

**EMS_USAGE_THRESHOLD_CHK_**

The EMS_USAGE_THRESHOLD_CHK_ procedure returns whether a standard usage threshold event should be generated.

The syntax of EMS_USAGE_THRESHOLD_CHK_ is:

```plaintext
{ status := } := EMS_USAGE_THRESHOLD_CHK_(
{ CALL      } Current utilization level   ! i
, Usage Threshold Control Block ! i,o
, report event  ! o
);
```

- **status** returned value
  - INT:value
    - returns one of these SPI error numbers:
      - 0  No error
      - -1 Invalid buffer format
      - -3 Missing parameter
      - -4 Illegal parameter address

- **Current utilization level** input
  - FLOAT:value
    - is the current utilization level of the object.

- **Usage Threshold Control Block** input, output
  - INT .EXT:ref:20
    - is the control block used by subsequent calls to EMS_USAGE_THRESHOLD_CHK_ and EMS_USAGE_THRESHOLD_EVT_BLD_.

- **report event** output
  - INT .EXT:ref:1
    - returns ZSPI-VAL-TRUE if the standard usage threshold event should be generated. Returns ZSPI-VAL-FALSE otherwise.

**Condition Code Settings**

The condition code has no meaning following a call to EMS_USAGE_THRESHOLD_CHK_.

---

**EMS Manual—426909-005**

11-35
Considerations

If EMS_USAGE_THRESHOLD_CHK_ returns 0 and report event returns ZSPI-VAL-TRUE, the user should call EMS_USAGE_THRESHOLD_EVT_BLD_ to generate the standard usage threshold event. Otherwise, the user should not generate the standard usage threshold event.

The caller should not modify the content of the Usage Threshold Control Block.

EMS_USAGE_THRESHOLD_EVT_BLD_

The EMS_USAGE_THRESHOLD_EVT_BLD_ procedure formats the standard usage threshold event. The event reports the object whose usage has crossed the configured threshold value.

The syntax of EMS_USAGE_THRESHOLD_EVT_BLD_ is:

```c
{ status := }  EMS_USAGE_THRESHOLD_EVT_BLD_( buffer
!i,o
{ CALL }    , maxbuflen
!i,
, SSID
!i,
, Event number
!i,
, Usage Threshold Control Block
!i,o
, Subject token ID
!i,
, Subject value
!i,
, [ Subject length ]
!i,
, [ Subject SSID ]
!i,
, [ buflen ]
!o,
, [ Critical indicator ]
!i,
, [ Manager name:Manager name length ]
!i:i,
, [ Job ID ]
!i
);  
```

status returned value

INT:value

returns one of these SPI error numbers:

- 0  No error
- -1 Invalid buffer format
- -2 Illegal parameter value
- -3 Missing parameter
- -4 Illegal parameter address
- -5 Buffer full
- -6 Invalid checksum
- -7 Internal error
- -8 Token not found
- -9 Illegal token code or map
- -10 Invalid subsystem ID
- -11 Operation not supported
- -12 Insufficient stack space
Procedure Calls for Standard Events

EMS_USAGE_THRESHOLD_EVT_BLD_

 buffer input, output
    INT .EXT:ref:*  
    is the event-message buffer provided by the caller.

 maxbuflen input
    INT:value  
    is the length, in bytes, of the buffer and cannot be greater than ZEMS-VAL-EVT-BUFLEN (4024).

 SSID input
    INT .EXT:ref:6  
    is the ID of the subsystem that owns the object whose usage is being reported.

 Event number input
    INT:value  
    is a number, specific to this subsystem, that identifies this event message.

 Usage Threshold Control Block input, output
    INT .EXT:ref:20  
    is the control block initialized by EMS_UTCB_INIT_, and EMS_USAGE_THRESHOLD_CHK_ has been called to indicate a standard Usage Threshold event should be reported.

 Subject token ID input
    INT(32):value  
    is the token code of the subject of this event message.

 Subject value input
    STRING .EXT:ref:*  
    is the value of the subject to be added to the event message.

 Subject length input
    INT:value  
    if present, is the length, in bytes, of the Subject value. This parameter is ignored unless the Subject token ID is defined as a variable-length token. For more information, see Considerations on page 11-39.
Subject SSID  input
   INT .EXT:ref:6
   if present, is the ID of the subsystem to which the Subject token ID belongs.
   If not present or the value is all zeros, the default SSID (specified in SSID) is assumed. In this case, the Subject SSID is not added to the event buffer.

buflen  output
   INT:value
   if present, is the length, in bytes, of the formatted buffer. If the buffer cannot be built, zero is returned.

Critical Indicator  input
   BOOLEAN:value
   if present, is a Boolean variable that indicates the critical nature of the problem reported in the event. The literal ZSPI-VAL-FALSE (0) indicates the problem is not critical. Otherwise, the problem is critical.
   If not present, the default is ZSPI-VAL-FALSE.

Manager name  input
   STRING .EXT:ref:*
   if present, is the name of the manager process of the object whose usage is being reported.
   If present and Manager name length is not zero, Manager name is added to the event buffer. Otherwise, Manager name is not added to the event buffer.

Manager name length  input
   INT:value
   if present, is the length, in bytes, of the Manager name.
   If not present or the value is zero, Manager name is not added to the event buffer.

Job ID  input
   INT:value
   if present, is the job ID (for use by the NetBatch facility) that associates with the object whose usage is being reported.
   If Job ID is zero or not present, see Considerations on page 11-39.
Condition Code Settings

The condition code has no meaning following a call to EMS_USAGE_THRESHOLD_EVT_BLD_.

Considerations

- To generate the common tokens:
  
  SSID: from SSID.
  
  Event subject token code: from Subject token ID
  
  Event subject value: from Subject value
  
  Event subject SSID: from Subject SSID. Conditional.
  
  Manager Name (zems-tkn-name-manager): from Manager name. Conditional.
  
  Event Number (zems-tkn-eventnumber): from Event number
  
  Standard Content type (zems-tkn-content-standard): ZEMS-VAL-USAGE-THRESHOLD.
  
  User Content type (zems-tkn-content-user): ZEMS-VAL-NULL.
  
  Critical Indicator (zems-tkn-emphasis): from Critical indicator.
  
  Suppress display (zems-tkn-suppress-display): ZSPI-VAL-FALSE.
  
  Batch job ID (zems-tkn-batchjob-id): from Job ID. Conditional.
  
  The values for User Content type and Suppress display can be changed by calling EMSADDTOKENS.

- To generate the additional tokens:
  
  Current utilization level (zems-tkn-util-level-curr): from Usage Threshold Control Block.
  
  Previous utilization level (zems-tkn-util-level-prev): from Usage Threshold Control Block. This token is not added if the event is the first time being reported.
  
  Previous timestamp (zems-tkn-util-time-prev): from Usage Threshold Control Block. This token is not added if the event is the first time being reported.
  
  Configured High level (zems-tkn-util-config-hi): from Usage Threshold Control Block.
  
  Configured Low level (zems-tkn-util-config-low): from Usage Threshold Control Block.
If Job ID is not present or the value is zero, the procedure calls system procedure PROCESS_GETINFO_ to get the job ID. If the job ID obtained is nonzero, the job ID is added to the event buffer. Otherwise, the job ID is not added to the event buffer.

The nonzero job ID obtained from calling PROCESS_GETINFO_ is the NetBatch job ID associated with the process that calls this EMS event-building procedure.

If the subject reported in this event is managed by a process different from the process that calls this EMS event-building procedure, the caller should provide the Job ID, which is the NetBatch job ID associated with the process that manages the subject.

If the Subject token ID has a variable-length value, choose one of these ways to specify the length.

- If the Subject length is present, the Subject length is the length, in bytes, of the Subject value.
- If the Subject length is not present, place the subject length in the two bytes that immediately precede the Subject value. Make the Subject value point to the extra two bytes. Do not count the two-byte length field as part of the subject length.

The subject of this event is the object whose usage is being reported. The Subject value must be fully qualified so that it can be uniquely identified from any point in a network. An example is nodename.$volume.subvolume.filename.

Use EMSADDSUBJECT or EMSADDSUBJECTMAP to specify additional subjects.

The caller should not modify the content of the Usage Threshold Control Block.
Usage Threshold Programming Example

This C example illustrates how the standard usage threshold event is reported:

```c
/*
 * External declarations
 */
#pragma RUNNABLE
#pragma NOXMEM

#include <stdlib.h>    nolist
#include <string.h>    nolist
#include <stdio.h>     nolist
#include <tal.h>       nolist
#include <CEXTDECS(EMS_UTCB_INIT_, EMS_USAGE_THRESHOLD_CHK_)> nolist
#include <CEXTDECS(EMS_USAGE_THRESHOLD_EVT_BLD_, FILE_OPEN_)> nolist
#include <CEXTDECS(SSGETTKN, WRITEREAD, EMSADDTOKENS)> nolist

#include "$dsv.Zspidef.ZSPIC" nolist
#include "$dsv.Zspidef.ZEMSC" nolist
#include "$dsv.Zspidef.ZCOMC" nolist

#pragma PAGE "Globals and defines"
#include "MYAPC" /* Includes DDL generated from MYAPDDL */
#include "myerror" /* handling error messages */

/*
 * Define values
 */
#define NULL             0
#define THRESHOLD_LEN    20
#define LINE_NAME_LEN    25
#define TOKEN_LENGTH     15

#pragma PAGE "main program code"
int main(void)
{
    /*
     * Declare ssid using the typedef from the DDL output
     */
    myap_val_ssid_def  myap_ssid;

    /*
     * Global declarations and variables
     */
    short buffer[ZEMS_VAL_EVT_BUFLEN];
    short buf_len = 0;
    char coll_name[] = "$mycol";
    short coll_num;
    short usage_threshold_cb[THRESHOLD_LEN];
```
float config_high_level = 0;
float config_low_level = 0;
short util_unit = ZEMS_VAL_UTIL_UNIT_PERCENT;
float current_util_level = 0;
short result;
short myap_evt_usage_threshold = 108;
char line_name[LINENAME_LEN] = "value of the subject ";
short zcom_ssid[6];
char token_value[TOKEN_LENGTH] = "token value ";
short status;
short coll_name_len = sizeof(coll_name) - 1;

strncpy(myap_ssid.u_z_filler.z_filler, MYAP_VAL_OWNER, 8);
myap_ssid.z_number = MYAP_SSN_MYAP;
myap_ssid.z_version = MYAP_VAL_VERSION;

/*
 * Open the EMS collector
 */
FILE_OPEN_ (coll_name, coll_name_len, &coll_num);

printf(" coll_num = %d\n", coll_num);

/*
 * Initialize usage Threshold Control block
 */
if (status = EMS_UTCB_INIT_ (usage_threshold_cb  /* return Usage Threshold
   , config_high_level /* configured high level */
   , config_low_level /* configured low level */
   , util_unit))         /* utilization unit */
handle_error_utcb_init(status);

/*
 * Check to see if the current utilization level exceeds the
 * threshold and needs to be reported.
 */
status = EMS_USAGE_THRESHOLD_CHK_ (current_util_level /* current utilization level */
   , usage_threshold_cb /* usage threshold block */
   , &result); /* return zspi_val_true if
the event */
   /* should reported. */

if ((status == 0) && (result == ZSPI_VAL_TRUE))
{
   if (status = EMS_USAGE_THRESHOLD_EVT_BLD_ (buffer /* event buffer */
/* event buffer
length */
   , ZEMS_VAL_EVT_BUFLEN /* event buffer */
   , (short*)&myap_ssid /* ssid */
   , myap_evt_usage_threshold /* event number */
Procedure Calls for Standard Events

Usage Threshold Programming Example

```c
/*
 , usage_threshold_cb   /* usage threshold control block */
 , ZCOM_TKN_SUBJ_LINE   /* subject token id */
 , line_name            /* subject value */
 , LINE_NAME_LEN        /* subject length */
 , zcom_ssid            /* subject ssid */
 , buf_len              /* return actual buffer length */
 , ZSPI_VAL_TRUE ))     /* return actual buffer length */
 handle_error_usage_threshold(status);

/* Add additional tokens if needed */
if ( status = EMSADDTOKENS ( buffer
   ,/* ssid */
   ,MYAP_TKN_PARAM_CPU  /* token_code */
   ,token_value         /* token value */
   ,TOKEN_LENGTH ))
 handle_error_add_token(status);

/* Get the size of the event buffer */
if (status = SSGETTKN ( buffer
   ,ZSPI_TKN_USEDLEN   /* retrieve used buff length */
   ,(char *)buf_len))  /* count */
 handle_error_get_token(status);

/* Send the event to the collector */
if (!WRITEREAD (coll_num         /* return value from OPEN */
   ,buffer           /* event-message buffer */
   ,*buf_len          /* from SSGETTKN above */
   ,0 )))            /* read count */
 handle_error_write_read();
} /* close if statement */
else
 { if (status != 0)
   handle_error();
 } /* close else statement */
return 0;
```
The EMS_COMMON_TOKENS_EVT_BLD_ procedure builds the common tokens required in all subsystem-defined events.

The syntax of EMS_COMMON_TOKENS_EVT_BLD_ is:

```
{ status := } EMS_COMMON_TOKENS_EVT_BLD_( buffer !i,o
{ CALL } , maxbuflen !i
, SSD !i
, Event number !i
, User Content type !i
, Subject token ID !i
, Subject value !i
, [ Subject length ] !i
, [ Subject SSID ] !i
, [ buflen ] !o
, [ Critical indicator ] !i
, [ Manager name:Manager name length ] !i:i

, [ Job ID ] !i
);
```

`status` returned value

INT:value

returns one of these SPI error numbers:

- 0 No error
- 1 Invalid buffer format
- 2 Illegal parameter value
- 3 Missing parameter
- 4 Illegal parameter address
- 5 Buffer full
- 6 Invalid checksum
- 7 Internal error
- 8 Token not found
- 9 Illegal token code or map
- 10 Invalid subsystem ID
- 11 Operation not supported
- 12 Insufficient stack space

`buffer` input, output

INT .EXT:ref:*

is the event-message buffer provided by the caller.

`maxbuflen` input

INT:value
is the length, in bytes, of the buffer. It cannot be greater than ZEMS-VAL-EVT-BUFLEN (4024).

SSID input
INT .EXT:ref:6
is the ID of the subsystem originating the event.

Event number input
INT:value
is a number, specific to this subsystem, that identifies this event message.

User Content type input
INT:value
is a number that identifies the type of a subsystem-defined event.
Standard enumerated values are:
ZEMS-VAL-NULL
ZEMS-VAL-DATA-TRACE
ZEMS-VAL-DATA-DEBUG
ZEMS-VAL-DATA-DIAGNOSTIC
Subsystems can define their own values, but they must be greater or equal to ZEMS-VAL-MIN-USER-VALUE (1024).

Subject token ID input
INT(32):value
is the token code of the subject of this event message.

Subject value input
STRING .EXT:ref:*
is the value of the subject to be added to the event message.

Subject length input
INT:value
if present, is the length, in bytes, of the Subject value. This parameter is ignored unless the Subject token ID is defined as a variable-length token. For more information, see Considerations on page 11-47.

Subject SSID input
INT .EXT:ref:6
if present, is the ID of the subsystem to which the Subject token ID belongs.
If not present or the value is all zeros, the default SSID (specified in SSID) is used. In this case, the Subject SSID is not added to the event buffer.

buflen output
INT:value
if present, is the length, in bytes, of the formatted buffer. If the buffer cannot be built, zero is returned.

Critical Indicator input
BOOLEAN:value
if present, is a Boolean variable that indicates the critical nature of the problem reported in the event. The literal ZSPI-VAL-FALSE (0) indicates the problem is not critical. Otherwise, the problem is critical.

If not present, the default is ZSPI-VAL-FALSE.

Manager name input
STRING .EXT:ref:*
if present, is the name of the manager process of the object that received the event.

If present and Manager name length is not zero, Manager name is added to the event buffer. Otherwise, Manager name is not added to the event buffer.

Manager name length input
INT:value
if present, is the length, in bytes, of the Manager name.

If not present or the value is zero, Manager name is not added to the event buffer.

Job ID input
INT:value
if present, is the job ID (for use by the NetBatch facility) that associates with the object experiencing the event.

If Job ID is zero or not present, see Considerations on page 11-47.
Condition Code Settings

The condition code has no meaning following a call to
EMS_COMMON_TOKENS_EVT_BLD_.

Considerations

- To generate the common tokens:

  SSID: from SSID
  Event subject token code: from Subject token ID
  Event subject value: from Subject value
  Event subject SSID: from Subject SSID
  Manager Name (zems-tkn-name-manager): from Manager name. Conditional.
  Event Number (zems-tkn-eventnumber): from Event number
  Standard Content type (zems-tkn-content-standard): ZEMS-VAL-NULL.
  User Content type (zems-tkn-content-user): from User Content type.
  Critical Indicator (zems-tkn-emphasis): from Critical indicator.
  Suppress display (zems-tkn-suppress-display): ZSPI-VAL-FALSE.
  Batch job ID (zems-tkn-batchjob-id): from Job ID. Conditional.

  The values for User Content type and Suppress display can be changed by calling EMSADDTOKENS.

- If Job ID is not present or the value is zero, the procedure calls system procedure PROCESS_GETINFO_ to get the job ID. If the job ID obtained is nonzero, the job ID is added to the event buffer. Otherwise, the job ID is not added to the event buffer.

  The nonzero job ID obtained from calling PROCESS_GETINFO_ is the NetBatch job ID associated with the process that calls this EMS event-building procedure.

  If the subject reported in this event is managed by a process different from the process that calls this EMS event-building procedure, the caller should provide the Job ID, which is the NetBatch job ID associated with the process that manages the subject.
• If the Subject token ID has a variable length, choose one of these ways to specify the length.
  • If the Subject length is present, the Subject length is the length, in bytes, of the Subject value.
  • If the Subject length is not present, place the subject length in the two bytes that immediately precede the Subject value. Make the Subject value point to the extra two bytes. Do not count the two-byte length field as part of the subject length.

• The subject of this event is the object that received the event. The Subject value must be fully qualified so that it can be uniquely identified from any point in a network. An example is `nodename.$volume.subvolume.filename`.

  Use EMSADDSUBJECT or EMSADDSUBJECTMAP to specify additional subjects.

• Use EMSADDTOKENS or EMSADDTOKENMAPS to specify the additional information that is useful in the subsystem-defined event.

Procedure for Sending Events

To send the formatted event buffer to the configured collector (primary or alternate), use the file system procedure WRITEREAD(X).

Program Example

This C example illustrates the use of an EMS_???_EVT_BLD_ call to build an event buffer and send it to the collector.

Assumption: These handling-error functions exist in the myerror file:

• handle_error_unavail_event(short)
• handle_error_add_token(short)
• handle_error_get_token(short)
• handle_error_write_read()

/
* External declarations
*/
#pragma RUNNABLE
#pragma NOXMEM
#include <stdlib.h>     nolist
#include <string.h>     nolist
#include <stdio.h>      nolist
#include <tal.h>        nolist
#include <CEXTDECS(EMS_OBJ_UNAVAIL_EVT_BLD_)>   nolist

EMS Manual—426909-005
11-48
Program Example

```c
#include <CEXTDECS(EMSADDTOKENS, FILE_OPEN_)>  nolist
#include <CEXTDECS(SSGETTKN, WRITEREAD)>       nolist
#include "$dsv.Zspidef.ZSPIC"  nolist
#include "$dsv.Zspidef.ZEMSC"  nolist
#include "$dsv.Zspidef.ZCOMC"  nolist
#include "MYAPC"            /* generate from myapddl */
#include "myerror"          /* handling error messages */

#include "Globals and defines"
/*
 * Define values
 */
#define NULL          0
#define SYMPTOM_LEN   20
#define SUBJECT_LEN   15
#define TOKEN_LENGTH  15

#pragma PAGE "main program code"
int main(void)
{
    /*
    * Declare the ssid's using the typedefs from the DDL output
    */
zems_val_ssid_def   zems_val_ssid;
myap_val_ssid_def   myap_ssid;

    /*
    * Global declarations and variables -- just an example --
    */
short event_buf[ZEMS_VAL_EVT_BUFLEN];
short buf_len = 0;
short current_state = ZEMS_VAL_MIN_USER_VALUE + 1;
short previous_state = current_state;
char coll_name[] = "$mycol";
char symptom[SYMPTOM_LEN] = "object unavailable ";
char subject_value[SUBJECT_LEN] = "line name ";
short coll_num = 0;
char token_value[TOKEN_LENGTH] = "token value ";
short status;
short myap_evt_obj_unavail = 109;
short coll_name_len = sizeof(coll_name) - 1;
strncpy(myap_ssid.u_z_filler.z_filler, MYAP_VAL_OWNER, 8);
myap_ssid.z_number = MYAP_SSN_MYAP;
myap_ssid.z_version = MYAP_VAL_VERSION;

    /*
    * Open the EMS collector
    */
FILE_OPEN_ (coll_name, coll_name_len, &coll_num);
printf(" sizeof coll_name = %d\n", coll_name_len);
    /*
    * Display error message and exit if collector is unsuccessfully opened.
    * Otherwise build an event and send to the collector
    */
    if (coll_num == -1)       /* return value coll_num from OPEN */
    {
        printf (" Unsuccessfully open %s collector\n", coll_name);
        printf (" coll_num = %d \n", coll_num);
    }
else
```

EMS Manual—426909-005  
11-49
Program Example

```c
/* Build the event buffer by calling EMS_OBJ_UNAVAIL_EVT_BLD_ */
if ( status = EMS_OBJ_UNAVAIL_EVT_BLD_ ( event_buf
    ,ZEMS_VAL_EVT_BUFLEN
    ,(short *)&myap_ssid /*subsystem identifier ssid */
    ,myap_evt_obj_unavail /* event number */
    ,current_state
    ,previous_state
    ,ZEMS_VALOPERATOR_INITIATED /* change reason */
    ,ZCOM_TKN_SUBJ_LINE /* subject token id */
    ,subject_value /* value of the subject to be added */
    ,SUBJECT_LEN
    ,/* subject ssid */
    ,/* underlying-object-name */
    ,/* underlying-object-name-length*/
    ,symptom /* symptom */
    ,SYMPTOM_LEN /* symptom string length */
    ,&buf_len /* return actual buffer length */
    ,ZSPI_VAL_FALSE ))}
    /* critical_indicator */
    handle_error_unavail_event(status); /* call handle error function */

/* Add additional tokens if needed */
if ( status = EMSADDTOKENS ( event_buf
    ,MYAP_TKN_PARAM_CPU /* token code */
    ,token_value /* value of the token */
    ,TOKEN_LENGTH ))
    handle_error_add_token(status);

/* Get the size of the event buffer */
if ( status = SSGETTKN ( event_buf
    ,ZSPI_TKN_USEDLEN /* retrieve used buff length */
    ,(char *)&buf_len))
    handle_error_get_token(status);

/* Send the event to the collector */
if (!(WRITEREAD (coll_num /* return value from OPEN */
    ,event_buf /* event-message buffer */
    ,buf_len /* the return value from SSGETTKN above */
    ,0 )) /* read count */
    handle_error_write_read();
) /* close else statement */
return 0;
) /* close main */
Part IV: Configuring and Maintaining EMS

This part of the manual provides procedures and supplemental information for the configuration and routine use of EMS:

- Section 12, Configuring EMS
- Section 13, EMS Programs
- Section 14, EMS Definitions
- Section 15, EMS Procedures
- Section 16, Event Routing
12 Configuring EMS

This section describes the major components of EMS and their characteristics. This information will help you make decisions about configuring EMS components to meet your network and system management requirements.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Attributes of EMS Components</td>
<td>Some EMS components characteristics are fixed; you cannot change them. Others are variable. You can let them assume default values, or you can define their values when the component is created or while it is running.</td>
<td>12-1</td>
</tr>
<tr>
<td>Configuration Issues</td>
<td>You can tailor your component characteristics to your particular system performance and resource requirements.</td>
<td>12-16</td>
</tr>
<tr>
<td>Installation and System Generation Considerations</td>
<td>You make some decisions regarding EMS component characteristics during the system generation and installation processes.</td>
<td>12-21</td>
</tr>
<tr>
<td>Log File Operation</td>
<td>Managing log files is an important aspect of managing an EMS environment.</td>
<td>12-23</td>
</tr>
</tbody>
</table>

Basic Attributes of EMS Components

The first step to prepare for an EMS configuration is to learn about characteristics of the EMS components that are fixed and to determine which characteristics you can modify to tailor an EMS configuration to your needs.

The major EMS components used to store and process event messages are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Collectors</td>
<td>12-2</td>
</tr>
<tr>
<td>Alternate Collectors</td>
<td>12-2</td>
</tr>
<tr>
<td>Log Files</td>
<td>12-3</td>
</tr>
<tr>
<td>Consumer, Forwarding, and Printing Distributors</td>
<td>12-12</td>
</tr>
<tr>
<td>Compatibility Distributor ($Z0)</td>
<td>12-14</td>
</tr>
</tbody>
</table>
Primary Collectors

Each system must have one, and only one, primary collector process, named $0. The primary collector process is configured during system generation and is started when the system is loaded. It always runs as a process pair. At any time, you can switch the primary and backup processes of the primary collector’s pair.

The primary collector can respond to a maximum of 127 concurrent #ZSPI opens and unlimited event sources. Any of these requesters can issue these opens:

- A distributor accessing the current collector log, as opposed to a specific log file
- A compatibility distributor
- An EMS utility program that accesses the collector (EMSCCTRL or EMSCINFO)
- A management application using the collector’s command-response interface

Alternate Collectors

An alternate collector is similar to a primary collector in the way it handles basic functions such as event-message and log-file management. The difference is that an alternate collector must be explicitly started and stopped.

Alternate collectors have these basic characteristics:

- You can start as many alternate collectors on your system as the requirements of your system dictate.
● An alternate collector can run as a process pair, or it can run without a backup.

● Each alternate collector has a unique name, defined when the collector is started.

● An alternate collector is started by a TACL RUN command or by another process.

● The attributes associated with each alternate collector govern only the operation of that collector.

● Attribute information can be specified as parameters in the TACL command and passed to the alternate collector as STARTUP text. Some attributes directly control the operation of the collector. Others control the operation of the log files for each collector.

● You can change the values of some attributes after the collector is started, either by issuing the SPI CONTROL command or by running the EMSCCTRL program.

● Each alternate collector can be opened by a maximum of 1873 event message issuers and a maximum of 127 management programs (#ZSPI openers).

### Log Files

Log files of the primary and alternate collectors are identical in structure. These log files are disk files with the file code 843. Each collector opens its log files with READ/WRITE access and PROTECTED exclusion mode. In addition to log files, each collector defines configuration files (named ZZEVCONF) that provide a place to hold log file tracking information.

**Note.** The LOGPREFIX keyword of the EMSACOLL program lets you rename the third character in the prefix pattern of an alternate collector’s ZZEVCONF context file and associated log files. For details, see **EMSACOLL—Alternate Collector Program** on page 13-2.

All tasks associated with log switches are performed when the collector is idle (that is, when there are no events to log). These tasks include purging and deallocating the old log file and preallocating the next log file. Although this feature does not contribute to a higher maximum sustained logging rate, it streamlines event logging and improves the collector’s ability to handle bursts during log switches.

You can significantly improve the peak logging rate if events are blocked. For more information about the BLOCKING attribute, see **BLOCKING** on page 12-6.

### Primary Collector Log Files

When a system is loaded, the primary collector starts entering event messages into log files on the system volume. In addition to log files, the collector also uses two fixed files on the system volume (the disk volume from which the system was loaded) to retain log file and collector attributes. The system uses these files to restart logging after system loads or problems with the normal logging volume.

When a system is loaded from the $SYSTEM.SYSnn subvolume, these files are available for use by the primary collector:
- $SYSTEM.SYSTEM.ZZEVCONF as a context file. The context data in this file is used after a system load or log file failure.

- $SYSTEM.ZLOG$nn.ZZEV0000 as a log file. This file receives the first event messages logged after the system comes up.

- $SYSTEM.ZLOG$nn.ZZEVCONF as a context file. A configuration file (ZZEVCONF) is always present on the subvolume that contains the log files. For a more complete description of the ZZEVCONF files, see Log File Operation on page 12-23.

You can switch logging to another subvolume by altering the primary collector LOGSUBVOL attribute, using the SPI CONTROL command or the EMSCCTRL program. If you select a new subvolume, a new ZZEVCONF file is created if one does not already exist. New log files are created by the primary collector, as needed, on the new subvolume.

Figure 12-2 shows the location of the primary collector context and log files on the system volume, as defined by the system at system load.

**Figure 12-2. ZZEVCONF and Log Files on the System Volume**

---

**Alternate Collector Log Files**

To choose the log subvolume an alternate collector is to use, include the subvolume name as a parameter (LOGSUBVOL) on the TACL RUN command when you start the collector.
If you do not specify the log subvolume, the alternate collector looks for a default subvolume (DEFAULTSUBVOL) specification on the TA CL RUN command. If the collector finds a default subvolume specification, it uses this as the log subvolume. If it does not find a default subvolume specification, it uses the value = _DEFAULTS.

Log subvolumes cannot be shared. The log subvolume containing the log files of one collector cannot contain log files for any other collector. In addition to log files, there is a ZZEVCONF file on each log subvolume to record collector context. Each ZZEVCONF file provides a place to hold collector log file configuration data through system loads or any termination of collector execution.

**Note.** The LOGPREFIX keyword of the EMSACOLL program lets you rename the third character in the prefix pattern of an alternate collector’s ZZEVCONF context file and associated log files. For details, see EMSACOLL—Alternate Collector Program on page 13-2.

Figure 12-3 shows the log subvolumes used by two alternate collectors on a volume specified by the user.

**Figure 12-3. ZZEVCONF and Log Files on a User-Specified Volume**

---

**Log File and Collector Attributes**

A number of log file and collector attributes are associated with each primary and alternate collector. The values assigned to these attributes control the operation of the collector.

- Primary collector attributes have default values assigned by the collector when the collector is started at system load.
Alternate collector attributes have default values assigned by the collector when you start the collector, unless you include a value for an attribute as a RUN command parameter.

You can change the value of most log file and collector attributes after a collector is running by using the SPI CONTROL command or the EMSCCTRL program.

### Table 12-1. Log File Attributes and Their Default Values

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Primary Collector Default Values</th>
<th>Alternate Collector Default Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKUP</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>BLOCKING</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>DEFAULTSUBVOL</td>
<td>Not applicable</td>
<td>LOGSUBVOL</td>
</tr>
<tr>
<td>DISCACCESSID</td>
<td>Super ID</td>
<td>Logon ID of user starting collector</td>
</tr>
<tr>
<td>EOFREFRESH</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>LOGSUBVOL</td>
<td>$SYSTEM.ZLOG_\text{nn}$</td>
<td>DEFAULTSUBVOL (or =_DEFAULTS)</td>
</tr>
<tr>
<td>MAXFILE</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>NEXTOFILE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>POOLPAGES</td>
<td>Not applicable</td>
<td>64 pages</td>
</tr>
<tr>
<td>PRIMARYEXTENT</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>PROTECTION</td>
<td>COOO</td>
<td>COOO</td>
</tr>
<tr>
<td>REPLYAFTERWRITE</td>
<td>Not applicable</td>
<td>FALSE</td>
</tr>
<tr>
<td>ROTATEFILES</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>SECONDARYEXTENT</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>WRITEHTRUCACHE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

### BACKUP (Alternate Collector only)

This attribute determines whether the alternate collector runs as a process pair. The value of this attribute must be a valid CPU number on the node where the alternate collector is running. It must not be the same CPU number as the alternate collector’s primary CPU. The value of this attribute is not saved in the ZZEVCONF file. If this attribute is not specified as a RUN command parameter, there is no backup alternate collector, and the collector does not run as a process pair. You cannot change this attribute once the alternate collector has started.

### BLOCKING

The collector writes messages to log files. The collector either writes messages one at a time or writes a number of messages to a buffer and then writes the entire group of messages to the log file. The process of writing a group of messages to a buffer is known as blocking. Blocking significantly increases the speed at which messages are
written to the log file. Blocking of events also reduces the number of message system write requests.

Setting this attribute to ON makes the collector block event messages before writing them to the log file. Default value for the primary and alternate collectors is ON. The EMSCCTRL program can dynamically change the event blocking mode.

**DEFAULTSUBVOL (Alternate Collector only)**

This attribute specifies the subvolume used when the LOGSUBVOL is not specified or becomes inaccessible. This attribute is not saved in the ZZEVCONF file. If you do not specify a value, the attribute assumes the value of LOGSUBVOL. If the LOGSUBVOL becomes inaccessible, disk logging stops.

**DISCACCESSID**

This attribute specifies the user ID that a collector uses to access log files. The default value of this attribute for the primary collector is SUPER.SUPER. To change it, use the EMSCCTRL program or the SPI CONTROL command to change it. The value of this attribute for an alternate collector is determined by the logon ID of the user starting the alternate collector. You cannot change this attribute except by restarting the alternate collector. The value of this attribute is not saved in the ZZEVCONF file.

**EOFREFRESH**

This attribute determines whether the file label is written to disk each time data is written to the log file (each time the end-of-file is extended). The default value for this attribute is TRUE. This attribute is saved in the ZZEVCONF file. To change the value of this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.

**LOGSUBVOL**

This attribute defines the volume and subvolume a collector uses to write its log files. Each primary or alternate collector must have its own log subvolume. Attempts to share a subvolume result in accessibility errors because each collector must have write/protected access to the ZZEVCONF file and the current ZZEVnnnn file in the subvolume. The LOGSUBVOL attribute has a default value of $\$SYSTEM.ZLOGnnn for the primary collector and DEFAULTSUBVOL (or =DEFAULTS) for the alternate collector. To change the value of this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.

---

**Note.** The LOGPREFIX keyword of the EMSACOLL program provides the option of renaming the third character in the prefix pattern of an alternate collector’s ZZEVCONF context file and associated log files. For details, see EMSACOLL—Alternate Collector Program on page 13-2.

**MAXFILE**

This attribute defines the maximum number of files the collector is allowed to retain in the log subvolume. The ZZEVnnnn file names in the subvolume always rotate in sequence from ZZEV0000 through ZZEV9999. However, only MAXFILE files are kept
by the collector. For instance, if MAXFILE is 4 (the default), the files at a given time might be ZZEV0007, ZZEV0008, ZZEV0009, and ZZEV0010. File ZZEV0010 would be the active file.

Allowable values for MAXFILE are 2 through 1000. This attribute value is saved in ZZEVECONF. To change the value of this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.

Under certain conditions, the number of log files in the log subvolume might appear in the output of the FILEINFO command, at MAXFILES + 1. Conditions causing MAXFILES + 1 start with the no log switch delay feature, in which the steps associated with purging the old log file and preallocating the next are executed as low priority subtasks within the primary collector, scheduled when there is no event activity (no events to be logged). When a log switch occurs, the collector schedules a task to purge the old log file and then preallocate the next. If you execute a FILEINFO command in the log subvolume you see one extra file: the old log waiting to be purged, or the next log that has been preallocated (depending on the timing). At this point, the number of files in the log subvolume has increased by one to MAXFILES + 1.

**NEXTLOGFILE**

When you use the CONTROL command to set the value of this attribute to TRUE, the collector immediately closes its current log file, creates the next sequential log file, opens that file, sends a message to the log file indicating that the log switch has occurred, and then continues logging to the new file. The collector then sets the value of NEXTLOGFILE to FALSE. If setting NEXTLOGFILE to TRUE causes MAXFILE to be exceeded, the value of ROTATEFILES determines the action taken. (For more information about the MAXFILE-exceeded condition, see **ROTATEFILES** on page 12-9.) To change the value of this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.

**POOLPAGES (Alternate Collector only)**

This attribute determines the number of memory pages (1 page = 2048 bytes) the alternate collector uses for event buffering. Values in the range 20 through 128 are allowed. This attribute is not saved in the ZZEVECONF file. If POOLPAGES is not specified as a RUN command parameter, 64 pages is used. The POOLPAGES attribute cannot be changed once the collector is running.

**PRIMARYEXTENT**

This attribute establishes the value for the primary extent size (number of pages) used when a primary or alternate collector creates a log file. Even-numbered values in the range 2 through 65534 are allowed. This attribute setting is saved in the ZZEVECONF file. This attribute has a default value of 20. To change the value of this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.
PROTECTION

This attribute defines the file protection (read, write, execute, and purge) assigned to disk files when they are created or renamed. This attribute is saved in the ZZEVCNFG file. It has a default value of COOO. To change this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.

REPLYAFTERWRITE (Alternate Collector only)

This attribute determines when the alternate collector replies to event issuers. If REPLYAFTERWRITE is TRUE, the collector does not reply to event issuers until the event is successfully written to disk. If the event cannot be written to disk because logging has stopped, the collector returns a file management error to the event issuer. This mode of operation ensures event issuers that data has been written to disk.

If REPLYAFTERWRITE is FALSE and the alternate collector has a current backup process, the alternate collector replies to event issuers as soon as the event is successfully checkpointed.

ROTAFILE

This attribute determines the action taken when a log-full condition occurs and MAXFILE has been reached. When a log file becomes full, a collector normally creates the next sequentially named file using the file names ZZEV0000 through ZZEV9999. However, when the maximum number of files (as determined by MAXFILE) already exists, ROTATEFILES determines the action to be taken. The primary and alternate collectors handle this condition slightly differently:

- If ROTATEFILES is TRUE (set to ON), the primary collector purges the oldest log file in the log subvolume and creates a new one. If ROTATEFILES is FALSE (set to OFF), the collector stops logging.

- If ROTATEFILES is TRUE, the alternate collector checks the oldest log file in the log subvolume. If the oldest file has the proper extent sizes, the collector renames the file, giving it the name of the next sequential log file. If this file does not have the proper extent sizes, the collector purges the file and creates a new file with the next sequential file name.

If ROTATEFILES is FALSE and the oldest file is not empty, the alternate collector stops logging. If ROTATEFILES is FALSE and the oldest file is empty, the alternate collector checks to determine whether the oldest file has the proper extent sizes. If the extent sizes are correct, the collector renames the file. If the extent sizes are not correct, the collector purges the file and creates a new log file.

The default value for ROTATEFILES is TRUE for both primary and alternate collectors. The setting of this attribute is saved in the ZZEVCNFG file. To change the value of this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.
SECONDARYEXTENT

This attribute establishes the value for the secondary extent size (number of pages) to be used when a log file is created by a primary or alternate collector. Even-numbered values in the range 2 through 65534 are allowed. The setting of this attribute is saved in the ZZEVCONF file. SECONDARYEXTENT has a default value of 100. To change the value of this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.

WRITETHRUCACHE

This attribute determines whether each disk record should be written directly to disk or should be placed in the disk-process cache buffer and written to disk later. The value of this attribute is saved in the ZZEVCONF file. This attribute has a default value of FALSE. To change the value of this attribute, issue the SPI CONTROL command or run the EMSCCTRL program.

If REPLYAFTERWRITE is FALSE and the alternate collector does not have a current backup process, the alternate collector replies immediately to event issuers.

The setting of REPLYAFTERWRITE is not saved in the ZZEVCONF file. If you do not specify a REPLYAFTERWRITE parameter on the RUN command, a value of FALSE is used. You cannot change the value of this attribute once the alternate collector is running.

Changing Attribute Values

You can change collector attributes programmatically through the CONTROL command or interactively through the EMSCCTRL utility program. EMSCCTRL and CONTROL sometimes use different attribute names to refer to the same collector attribute. For a list of the names accepted by the EMSCCTRL program and the CONTROL command, see Table 13-3 on page 13-17.

For a description of the EMSCCTRL program, see EMSCCTRL—Control Collector Utility on page 13-9. For a description of the programmatic CONTROL command, see CONTROL Command (ZEMS-CMD-CONTROL) on page 19-34.

If you change the values of primary collector attributes, you can adversely affect the operation of the entire system. To help prevent problems caused by incorrect attribute values, the collector accepts CONTROL command messages only from processes with SUPER group privileges. However, processes with SUPER group privileges or processes with the logon ID that started the collector can change alternate collector attributes.

Collector Process Attributes

Two collector process attributes are associated with each primary and alternate collector: FILTER and SUPPRESS. You can assign or alter values for the BDS and PLF process attributes by:
- Running primary or alternate collector—dynamically using the FILTER (PLF) and SUPPRESS (BDS) keyword parameters of the EMSCCTRL program

- New alternate collector—using the SUPPRESS and FILTER keyword parameters of the EMSACOLL program

- Using the appropriate collector command (ZCOM-CMD-ADD, ZCOM-CMD-DELETE, ZEMS-CMD-REPLACE) to add, delete, or replace a pre-log filter or burst filter

**FILTER**

This attribute identifies the current state of pre-log filtration (PLF) for the collector. Pre-log filtration is the use of collector filters to detect and discard specified event messages or event bursts that would otherwise be sent to and stored in the collector’s log files. The filters can be a combination of compiled filters, filter tables, and one burst filter. You can add filters to a collector by using:

- The FILTER keyword of the EMSCCTRL program for primary or alternate collectors
- The EMSACOLL program for alternate collectors only
- The appropriate collector SPI command (ADD, REPLACE, or DELETE)

When filters are present and the FILTER attribute is set to ON, PLF is enabled for that collector; OFF indicates that PLF is disabled. RESET disables PLF and causes the collector to delete any previously entered filters. For a detailed description of the FILTER keyword, see EMSCCTRL—Control Collector Utility on page 13-9.

**SUPPRESS**

This attribute sets BDS configuration values for the collector and the state of BDS (ON or OFF). If SUPPRESS is set to ON, the BDS configuration values supplied from the EMSCCTRL program for primary and alternate collectors or the EMSACOLL program for alternate collectors specify the number and time duration of repetitive similar events that are suppressed from reaching the collector’s log files. If SUPPRESS is set to OFF, the BDS configuration values can instead be implemented by means of a burst filter specified in the FILTER keyword of EMSCCTRL or EMSACOLL, or installed with an SPI ADD or REPLACE command (see FILTER on page 12-11).

When SUPPRESS is set to ON, BDS is enabled for that collector. When SUPPRESS is set to OFF, BDS is disabled unless a burst filter is active. SUPPRESS RESET disables BDS and causes the collector to adopt the default BDS configuration values described in Section 7, Burst Detection and Suppression. For a description of the manner in which this information is stored in the ZEMS-EVT-FILESWITCH event message, see BDS and PLF Information in the FILESWITCH Event on page 12-24.
Consumer, Forwarding, and Printing Distributors

Event-message distributors let you choose event messages from one or more log files by using a filter and route these messages to an appropriate destination.

<table>
<thead>
<tr>
<th>Distributor Type</th>
<th>Message Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer</td>
<td>Application program</td>
</tr>
<tr>
<td>Forwarding</td>
<td>Collector (usually on a remote node)</td>
</tr>
<tr>
<td>Printing</td>
<td>Disk file, device, or process</td>
</tr>
</tbody>
</table>

Using Distributors

Using consumer, forwarding, and printing distributors is optional. You can start and stop these distributors either programmatically or interactively at any time. Any number of each type of distributor can be running, subject to the limit of the maximum number of opens for each type of collector. For a description of these limits, see Primary Collectors on page 12-2 and Alternate Collectors on page 12-2.

Several application programs (up to a maximum of 15) can have a distributor open at the same time to get status and version information. However, only one of these application programs can retrieve event messages and control distributor and logging attributes. The way these distributors start and stop depends on distributor type.

Each consumer distributor serves a single controlling (main) application program. The main application typically starts its own consumer distributor, which it opens to retrieve event messages. After the main application and any other openers have closed it, a consumer distributor automatically stops, unless the AUTOSTOP attribute is specified.

Printing and forwarding distributors, on the other hand, do not necessarily depend on a particular application. Operators and network applications start these distributors as needed. A network application can open one of these distributors, change distributor and logging attributes, and close the distributor, as appropriate. When retrieving event messages from a log file specified by the LOGFILE keyword or ZEMS-TKN-LOGTIME token, the distributor stops automatically at the end of file. Otherwise, printing and forwarding distributors continue to run until stopped by an operator or a network application.

The printing distributor allows distribution of events to selected destinations. The destinations are determined in the filter on an event basis, and a list of routing IDs is passed by the filter interpreter to the distributor event processing logic. All potential destinations are defined as profiles in the filter source and are stored in the filter object. The term routing distributor in this manual describes a printing distributor with a filter that contains destination profiles. For more information about event routing, see Section 16, Event Routing.
Starting a Distributor

You control whether a distributor runs as a process pair by specifying or omitting the BACKUP option when you issue the TACL command to start the distributor.

When you start a distributor, you must specify the distributor type. You can also provide several additional items of information. The distributor startup parameters are:

- Distributor type. You must select a printing, forwarding, or consumer distributor by including the TYPE keyword: TYPE PRINTING (or TYPE P), TYPE FORWARDING (TYPE F), or TYPE CONSUMER (TYPE C).
  - For routing distributors (printing distributors with a filter containing destination profiles): the filters can contain destination profiles to direct events to selected destinations. If the startup line contains a TEXTOUT parameter, specification of a filter with destination profiles results in an error. If multiple filters are used with no destination profiles, the distributor does not process any subsequent filters after encountering a filter with a PASS condition.
  - Consumer and forwarding distributors can be used with multiple filters, but not with destination profiles.
  - For printing distributors only: the TEXTOUT option designates the destinations (processes, devices, or disk files) to which the distributor sends the event-message text. You can specify up to ten destinations.
  - For forwarding distributors only: the TARGET option designates the collector to which the distributor is to forward its event messages. You can specify only one collector. It can be a primary or alternate collector on a local or remote system.
  - For consumer distributors only: the management application that opens the distributor’s programmatic interface is its only destination.

- Source of event messages. To designate the source of the event messages the distributor is to process, specify:
  - COLLECTOR, if you want the distributor to access the current log-file set of the collector or collectors you have named. You can specify up to ten collectors.
  - LOGFILE, if you want the distributor to access the log file you have named. This feature is typically used for archived log files. You can specify only one log file.
  - Neither COLLECTOR nor LOGFILE, if you intend to supply a collector name or log-file name later, through the distributor’s programmatic interface.

Specifying both COLLECTOR and LOGFILE is not allowed.

- Generation time of the first message. Specify a value for the TIME option to indicate that you want the distributor to start processing the log files at the first event message whose generation time is the same as or later than the time you
specified. The value can include both the date and time or just the time. The time value is assumed to represent local civil time.

If you do not specify a value for the TIME option, the effect depends on whether you have specified COLLECTOR or LOGFILE (that is, whether you are accessing the current collector log or a saved log file). If you specified COLLECTOR, event-message processing starts with the next incoming event message. If you specified LOGFILE, event-message processing starts with the first event message in the log file.

- The filter used to select or detect messages. You can specify the file name of a filter object file, filter table EDIT file, or burst filter EDIT file to indicate which filter you want the distributor to use to select or detect event messages. Omitting the FILTER parameter indicates that you intend to supply the filter through the distributor’s programmatic interface or, for consumer and printing distributors, to let the filter default to pass all event messages. (The default filter does not apply to forwarding distributors. You must specify a filter.)

Not all option combinations are allowed. For example, you cannot initiate a printing distributor and request that it send event messages to a collector on another node.

**Compatibility Distributor ($Z0)**

One compatibility distributor, named $Z0, is allowed on each system. It can be configured during system generation and is initiated at system load. In that case, it runs as a process pair. To change which process is the primary process, use the SWITCH option of EMSCCTRL.

$Z0 can always be stopped by the CDISTSTOP option of EMSCCTRL. The utility EMSCCTRL controls the operation of the primary collector, and $Z0 indirectly as well, through a programmatic interface between $0 and EMSCCTRL. $Z0 can only be started by a system load. After it is stopped, it cannot be restarted without bringing the system down. For more information about EMSCCTRL, see EMSCCTRL—Control Collector Utility on page 13-9.

⚠️ **Caution.** In RVUs prior to G06.24, the command EMSCCTRL $0 CDISTSTOP should only be used when both the CPUs on which the process Compatibility Distributor ($Z0) is configured to run (for example, CPU 0,1) are up. If the command is issued when only CPU 0 is up, the $Z0 process on CPU 0 will stop. The backup process will reappear once CPU 1 comes up. If at any instance after that, CPU 0 goes down, attempts to reload it will cause CPU halt %010005.

⚠️ **Caution.** The command EMSCCTRL $0 CDISTSTOP should not be issued when any of the CPUs on which $Z0 is configured to run is being reloaded.
$Z0 writes event messages to a console in DSM display format. For D-series RVUs, you can use PUP LISTDEV and PUP PRIMARY; of the PUP CONSOLE commands, you can only use the CONSOLE device command.

**Note.** Block mode applications cannot be started on the current $0 CONSOLE device. Attempting to start a block mode application, such as ViewSys or TEDIT, results in an error 549. Conversely, if a block mode application is running on a device, that device cannot be used as the $0 CONSOLE device. Attempting to switch the CONSOLE device using the EMSCCTRL $0 TEXTOUT command (or the PUP CONSOLE command on a D-series RVU) results in a terminal error of 12 (indicated in the EMSCINFO display and in the EMS logs).

If you must run block mode applications on the CONSOLE device, use one of these methods:

- Use another device for either the block mode application or the console. If the designated console device is actually a 6530 emulation window on a desktop computer (for example, a PC, Macintosh, or UNIX workstation), open another 6530 window.

- Encapsulate the invocation of a block mode application in a TACL macro that first issues an EMSCCTRL $0 TEXTOUT $0 (or PUP CONSOLE $0 on a D-series RVU), and later issues EMSCCTRL $0 TEXTOUT [#MYTERM] (or PUP CONSOLE [#MYTERM] on a D-series RVU) after the block mode application stops.

The compatibility distributor has a selection criterion, CRITICAL-ONLY, which lets you suppress the printing of selected messages on the operator console. You can use this feature to control the volume of output on the operator console. This is especially useful during system load. When CRITICAL-ONLY is ON, the messages that are suppressed are console messages numbered 6 (LDEV UP), 141 (CLIP DOWNLOADED), and 150 (CSS ACTIVATE PATH).

When CRITICAL-ONLY is OFF, the compatibility distributor displays all messages.

**Other EMS Components**

Other EMS components have fewer configuration requirements and options. The EMS filter (EMF) compiler, the distributor object file, and the EMS utility programs should be available in an easily accessible location. For a description of how the template files must be installed, see [Installation and System Generation Considerations](#) on page 12-21. The EMS definition files should be available in an easily accessible location with the definition files for all other NonStop Kernel subsystems.
Configuration Issues

One of the difficulties in configuring systems and networks is that each decision requires a balance between two system characteristics. For example, higher performance must often be weighed against its cost in resource consumption or in potential loss of data integrity. While not showing you how to avoid such trade-offs, this subsection describes some of the implications of decisions you might make in configuring EMS components.

Task Requirements

The most important configuration decision is what tasks the configuration should support. EMS supports evolutionary decision-making in this respect, so you can add or change functionality as needed. You can start with just the primary collector and an initial log-file strategy. (See Log File Operation on page 12-23.)

You can add consumer, forwarding, and printing distributors as you need them. For each distributor, you must decide what type is appropriate, what filter is required, and whether to initiate the distributor interactively or programmatically. You can easily change from one alternative to another. For example, you might run a printing distributor interactively with a special filter until you decide the filter is correct. When the filter is working, you can write a management application to handle the event messages that the filter passes; when the application is working, you can have it initiate its own consumer distributor programmatically and load the filter.

If messages produced during a system load need to be displayed, add the compatibility distributor to your system configuration.

Networking Considerations

When your EMS tasks include managing multiple systems in a network, there are some additional considerations:

- How do you see the network, from a management point of view? Is each system relatively autonomous? Or do you want to monitor all systems from a central system (a network control node)? The network-control-node configuration naturally implies the use of one or more forwarding distributors.

To reduce message traffic between nodes, the forwarding distributor transfers event messages in blocks when it queues more than one message while a previous message is in transit. Forwarded messages are logged by the collector at the target node (the remote collector) and are then available for processing by all distributors monitoring the remote collector’s log file.

If you send only a few messages to another node and choose not to use blocking, and if the only destination for those messages is a single management application (so having all distributors process the messages is wasteful), have a local consumer distributor retrieve those messages and return them to the application on the remote node. With this configuration, messages might be sent less
efficiently, but they are logged on only one node (the local node), and they need not be processed by all the other distributors at the remote node.

On the other hand, a remote management application communicating with a local consumer distributor must detect and recover from network errors, whereas a forwarding distributor transmitting event messages to the remote node would assume that burden. And, if you later decide to switch from a local consumer distributor to a forwarding distributor, the remote management application would have to be changed to open a consumer distributor on its own node.

- How much operator support is available at each location? If there are operators at each location, some event messages—such as those reporting action events—should probably not be forwarded. If there are locations without operator support, you should have most, if not all, event messages forwarded.

If some of your nodes have limited resources to devote to EMS, arrange the configuration so that the collectors on the nodes with few resources log their event messages directly to files on a remote node (without running through a forwarding distributor). This is particularly effective for systems linked by FOX because of the efficiency and reliability of the link. For systems linked by other types of communications lines, forwarding distributors are better, as they transmit event messages in blocks (for efficiency) and they recover automatically from link failures. (Because collectors begin logging on the local system when the remote system becomes unavailable, a collector must be explicitly redirected to the remote log file when the link is re-established.)

Whatever decisions you make about the type of messages to be forwarded, you should always analyze the requirements carefully and to use effective (highly selective) filters, where possible, to reduce network traffic. Also, ensure that the filters used by your forwarding distributors do not let event messages from a remote node be forwarded back to that node (directly or indirectly). Running event messages in circles around your network is not an efficient use of resources.

Logging Integrity

EMS provides reliable event logging. With tasks associated with log switches being performed when the collector is idle, and with a higher rate of message logging achieved by setting message blocking, the collector has a greater ability to handle event bursts and minimize the possibility of losing events.

Still, a few exceptional conditions can result in lost event messages. You can avoid or reduce the frequency of these conditions by certain configuration decisions, usually at the cost of reducing a collector’s peak logging rate or of dedicating additional resources to the collector.

For a description of how message delivery can continue after logging fails, see Delivery Integrity on page 12-19.

A collector can fail to log some event messages under these circumstances:
● The CPU in which the event message was generated fails before the message can be delivered to the appropriate collector. (EMS cannot prevent this.)

● A collector's log-file space is inadequate. This occurs when a combination of these attribute settings causes log files to fill up too fast:
  ○ ROTATEFILES is FALSE.
  ○ MAXFILE is too small.
  ○ PRIMARYEXTENT, SECONDARYEXTENT, or both are too small.

To avoid this problem, take one or more of these actions:
  ○ Let a collector overwrite existing log files, by setting ROTATEFILES to TRUE.
  ○ Dedicate more disk space to a collector, by increasing MAXFILE, PRIMARYEXTENTS, or SECONDARYEXTENTS.
  ○ Archive and remove filled files at a faster rate than they are currently being removed.

● Both CPUs in which the disk process is running fail simultaneously, and the disk cache or the end-of-file pointer—or both—are lost. To reduce the impact of such CPU failures, set EOFREFRESH and WRITETHRUCACHE to TRUE. This fix does impact peak logging performance. EOFREFRESH has greater potential benefit, in terms of the number of messages that might otherwise be lost, than does WRITETHRUCACHE, and EOFREFRESH costs less in terms of performance. That is why collectors use default values of TRUE for EOFREFRESH and FALSE for WRITETHRUCACHE.

● The rate at which a collector can log event messages is exceeded. This occurs when some combination of these circumstances flood the collector with event messages:
  ○ A collector is slowed down by user-requested disk overhead when EOFREFRESH or WRITETHRUCACHE, or both, are TRUE, and BLOCKING is OFF.
  ○ There is significant contention for the disk volume to which a collector is logging.
  ○ One or more event-message sources (subsystems or forwarding distributors) are generating event messages at an exceedingly high rate.
  ○ A collector’s queuing space is insufficient. To avoid this problem, take one or more of these actions:
    ● Reduce the disk overhead, by setting EOFREFRESH and WRITETHRUCACHE to FALSE and by logging to a disk volume for which there is less contention. (Setting EOFREFRESH and WRITETHRUCACHE to TRUE reduces the risk of losing event messages because of simultaneous CPU failures in the disk-process CPUs. However, setting these attributes to TRUE can actually cause messages to be lost if the
collector cannot then keep up with the peak logging demand. Setting BLOCKING to ON increases the maximum sustained logging rate of the primary and alternate collectors by a factor of two.)

- Verify that subsystems are generating only the event messages they are supposed to generate (for example, check that no subsystem is in a loop).
- Verify that the filters used by forwarding distributors are selecting only the desired event messages.
- At the next system load, enlarge the buffer for the primary collector by increasing the number of pages of resident extended memory that the collector can use. (For a description of how to enlarge the buffer size, see Installation and System Generation Considerations on page 12-21.)
- The next time the alternate collector is started, enlarge the buffer for the collector by increasing the value of the POOLPAGES attribute.

The status information returned by a collector—in response to a STATUS command message or to the interactive EMSCINFO program—includes the current collector attribute settings, as well as statistics indicating the number of event messages received, logged, and discarded due to flooding.

**Delivery Integrity**

Event-message delivery can continue in many cases where logging is impossible; for example, where log-file space is inadequate. In such cases, a collector queues messages in memory and delivers them directly to the appropriate distributors. Appropriate distributors are the compatibility distributor and other distributors (consumer, forwarding, or printing) that retrieve event messages as the collector receives them.

After log-file access is restored, a collector logs the queued event messages and restores message delivery from direct delivery to normal delivery. All messages are logged if logging resumes before queue space is exhausted. Otherwise, the collector must discard some of the newest messages.

**Performance**

Collector performance is determined primarily by the log-file considerations in Logging Integrity on page 12-17. For all other EMS processes (and, to a lesser extent, for collectors as well), the most significant performance issue is filter effectiveness. Filters that select the exact set of event messages required (no fewer and no more) let all event-message collectors, distributors, and their destinations function more efficiently. For guidelines on creating efficient filters, see The Filter Language on page 5-7, and Section 6, Filter Tables and Burst Filters. Blocking is also significant because it can double the rate of message logging.
Reliability

Almost all EMS processes run as process pairs, reducing the possibility that a single-component failure can cause a process to fail. A primary collector always runs as a process pair; an alternate collector can run as a process pair; a compatibility distributor runs as a process pair if it is initiated by a system load or by a RUN command with the BACKUP parameter included. A consumer, printing, or forwarding distributor runs as a process pair if you include the BACKUP parameter in the RUN command. The EMF (EMS filter) language compiler and the various interactive utility programs do not run as process pairs.

Resources

Several logging options control the amount of disk space a collector can use. To conserve disk space, set MAXFILE, PRIMARYEXTENTS, or SECONDARYEXTENTS to a low value. But a smaller log fills faster, requiring you to archive and purge the log files more rapidly, or to set ROTATEFILES to TRUE and let the collector overwrite older event messages.

Using filters in collectors can also reduce the amount of disk space needed to store events.

To reduce the number of EMS processes, use the multiple-source and (in some cases) multiple-destination features of the distributors. For example, if you need two copies of the event-message text, you can run two printing distributors or, as long as it is acceptable for both destinations to receive identical output, you can run one printing distributor to two destinations.

For a description of strategies for reducing consumption of network resources and for logging to remote nodes when the logging resources on the local node are restricted, see Networking Considerations on page 12-16.

Security

Two log file attributes relate to security:

- PROTECTION specifies the read-write-execute-purge file security assigned to each log file when it is created.
- DISCACCESSID specifies the user ID that a collector uses when accessing its log files.

Similar to security for a file, you should set the PROTECTION attribute according to your requirements for protecting confidential information, allowing access to appropriate users while preventing accidental purging of important files.

The primary collector DISCACCESSID attribute defaults to SUPER.SUPER. If a SUPER.SUPER user does not have access to files on a node where you want to place log files, you must change DISCACCESSID to a user ID that does have access to files on that node. For security reasons, the value of DISCACCESSID is not returned when collector status is requested, and the value is not saved across system loads.
Installation and System Generation Considerations

Only a few EMS components have operating characteristics you must consider during system generation: the primary collector, the compatibility distributor (optionally), and the template files. Other components such as code files and definition files only need to be copied to a convenient location. For more information about EMS system generation, see the System Generation Manual.

The Primary Collector

Exactly one primary collector process is required on each system (node). The primary collector is always installed during the SYSGEN phase of the INSTALL process, and is always named $0. The INSTALL process includes the file $ISV.ZEMS.OPCOLL in the SYSGEN auxiliary file, CONFAUX.

During system generation, a buffer is allocated to $0 for queuing event messages on their way to the collector log and for holding filters, including one burst filter.

G-Series RVU Considerations

In systems running a G-series RVU of the NonStop Kernel, you cannot configure the number of pages (XPOOLPAGES) that $0 uses for buffer space. $0 allocates the maximum number of pages, 128, so you do not need to reconfigure XPOOLPAGES.

D-Series RVU Considerations

In systems running a D-series RVU of the NonStop Kernel, this buffer has 50 pages of resident extended memory, which is big enough for most systems but might be too small for some. For an explanation why the collector buffer could prove too small, and for some remedies, see Logging Integrity on page 12-17. An obvious remedy—the only one that directly involves the primary collector—is to enlarge the buffer, but you can only do this during SYSGEN. If you can foresee a need for more than 50 pages in the buffer, enlarge it when you install the collector.

Once the primary collector is running, to find out if the buffer is too small, look at the EMSCINFO status display. (See Example 13-1 on page 13-18.) The Buffer Failures field (right side, lower center) shows the number of event messages that $0 could not accept because the buffer was full. To enlarge it requires another system generation.

To change the buffer size, change the value of the XPOOLPAGES parameter in the SYSTEM_PROCESS_MODIFIERS paragraph of the configuration file. This optional paragraph can include the XPOOLPAGES parameter and EMSFLAGS. (See The Compatibility Distributor ($Z0) on page 12-22.) The paragraph looks like:

```
SYSTEM_PROCESS_MODIFIERS:
    $0 XPOOLPAGES n,
    EMSFLAGS b;
```
If you omit the XPOOLPAGES parameter or the entire paragraph from the configuration file, the primary collector buffer has 50 pages of resident extended memory. If the file includes XPOOLPAGES, you can set $n$ to any decimal numeral from 35 through 128; if you set $n$ outside that range, the buffer has 128 pages. After system generation, the value of $n$ does not change until someone changes it and another system generation takes place.

**The Compatibility Distributor ($Z0$)**

Use the compatibility distributor, $Z0$, to ensure that event messages generated during system load (before any other distributors are running) are displayed.

When configured, $Z0$ starts automatically during system load. To configure the compatibility distributor, include BUILD_Z0_PROCESS in the ALLPROCESSORS section of your SYSGEN specification. (For details, see the System Generation Manual.) The INSTALL program includes the compatibility-distributor object file, $ISV.ZEMS.OCDIST, in the SYSGEN auxiliary file, CONFAUX.

The compatibility distributor receives its configuration information from the primary collector.

The compatibility distributor uses a selection criterion, which you can set to CRITICAL-ONLY ON or CRITICAL-ONLY OFF. (For an explanation of these parameters, see Compatibility Distributor ($Z0$) on page 2-10.)

You can select the default value for CRITICAL-ONLY when the system is generated. This default value is used each time the compatibility distributor is started. If you do not specify a value for selection criterion, the system generation process sets it to CRITICAL-ONLY ON.

**G-Series RVU Considerations**

To define the selection criterion during system generation, for systems running a G-series RVU of the NonStop Kernel, use the EMSCCTRL commands.

**D-Series RVU Considerations**

To define the selection criterion during system generation in systems running a D-series RVU of the NonStop Kernel, use the EMSCCTRL commands or set the value of the EMSFLAGS parameter in the SYSTEM_PROCESS_MODIFIERS paragraph of the configuration file. This optional paragraph can include the EMSFLAGS parameter and XPOOLPAGES. (See The Primary Collector on page 12-21.)

```
SYSTEM_PROCESS_MODIFIERS:
  $0 XPOOLPAGES n,
  EMSFLAGS b;
```
If you omit the EMSFLAGS parameter or the entire paragraph from the configuration file, the default selection criterion for $Z0 is CRITICAL-ONLY ON. If the SYSTEM_PROCESS_MODIFIERS paragraph includes EMSFLAGS, you can set $b$ as:

0   CRITICAL-ONLY OFF  BLOCKING OFF  
2   CRITICAL-ONLY ON   BLOCKING OFF  
4   CRITICAL-ONLY OFF  BLOCKING ON   
6*  CRITICAL-ONLY ON   BLOCKING ON   

* default value for EMSFLAGS

After system generation, the value of $b$ does not change until you reset it and another system generation takes place. You can use the EMSCCTRL program to change the BLOCKING attribute or selection criterion.

$b$ represents the low-order bits in a word that includes internal flags for EMS. Set EMSFLAGS to the values of 0, 2, 4, or 6 (1, 3, and 5 are no longer used).

Template Files

The template files are installed automatically by the INSTALL process. But first, include this entry in the ALLPROCESSORS paragraph of the SYSGEN configuration file:

```
FORMATTER_TEMPLATE_FILES TANDEM^FORMATTER^TEMPLATE^FILES;
```

TANDEM^FORMATTER^TEMPLATE^FILES is a macro defined in the CONFAUX file created by the INSTALL process. For more information about template files, see the DSM Template Services Manual.

Log File Operation

A successful strategy for managing log files is critical to the successful operation of EMS. This subsection describes how collectors create and access log files and how they retain log-file and collector attributes through logging interruptions. This information—along with that under Log Files on page 12-3, and Logging Integrity on page 12-17—can help you choose an appropriate logging strategy for your installation.

Collector Context

The EMS collectors make log-file management easier by using default values for attributes and by maintaining context information in a file on disk.

Primary Collector Context

The primary collector’s overall context is kept in a file named $SYSTEM.SYSTEM.ZZEVCONF$. The ZZEVCONF context files, on $SYSTEM.SYSTEM$ and on individual log subvolumes, contain a ZEMS-EVT-FILESWITCH event message that includes the collector’s status at the time it last switched log files. This status is kept in a token called ZEMS-MAP-COL-STATUS.
The primary collector must have a default subvolume in which to log event messages when other subvolume information is not available or has not yet been provided. The default subvolume is $SYSTEM.ZLOG.nn.

The default subvolume is used to initialize the LOGSUBVOL attribute at system load; it is also used when the current log subvolume becomes inaccessible because of a disk problem. The default subvolume cannot be changed except by system loading from a different system subvolume.

**Alternate Collector Context**

Each alternate collector maintains a context file in the subvolume where its log files are kept. The name of this file is ZZEVCONF. If the log subvolume specified when an alternate collector is started does not contain a file named ZZEVCONF, EMSCCTRL creates this file and starts logging in the file ZZEV0000. If the file ZZEVCONF exists in the specified log subvolume, and if it has a code of 843 (EMS log), the current log file name specified in ZZEVCONF is used to determine the next log file. If the ZZEVCONF file does not have a file code of 843, logging cannot start.

If a valid ZZEVCONF file exists when an alternate collector is started, the file is used to determine the values of any attributes not specified as startup parameters.

**BDS and PLF Information in the FILESWITCH Event**

The ZZEVCONF context files contain a ZEMS-EVT-FI LE SWITCH event message that includes the collector’s status at the time it last switched log files. This event message includes status information about the collector’s pre-log filtration (PLF) and burst detection and suppression (BDS) configurations and states. The states (ON or OFF) of both PLF and BDS for the collector are contained in the ZEMS-MAP-COL-STATUS token of ZEMS-EVT-FI LE SWITCH. BDS configuration values are contained in the ZEMS-TKN-BDS-INFO token. The name of each PLF filter (if any exist) is contained in its own ZEMS-TKN-XFILTERFILE token.

Unlike other fields in the FILESWITCH event, the current BDS and PLF states reflect the state of the collector when the FILESWITCH event message was written, not necessarily the correct BDS and PLF states for the log subvolume.

When the collector switches to an existing log subvolume, it assumes the log file attributes that were recorded in the ZZEVCONF FILESWITCH event of that subvolume. For example, if the EOFREFRESH field in the ZEMS-MAP-COL-STATUS token of the event was set to ON, the collector runs with EOFREFRESH ON, regardless of the state of EOFREFRESH before the log subvolume change.

However, BDS and PLF are considered process attributes rather than log file attributes. Consequently, the states of both BDS and PLF remain constant in response to subvolume switches.

**Note.** The LOGPREFIX keyword of the EMSACOLL program lets you rename the third character in the prefix pattern of an alternate collector’s ZZEVCONF context file and associated log files. For details, see EMSACOLL—Alternate Collector Program on page 13-2.
See the FILTER (PLF) and SUPPRESS (BDS) process attribute descriptions in Collector Process Attributes on page 12-10.

Primary Collector Logging After a System Load

When primary collector logging starts after a system load, the collector must determine where to start logging event messages. It continues logging in the previous log file, if one existed, or creates a new log file if it determines no previous logging took place. The primary collector must also determine collector attributes. It uses previously defined attributes if it can determine what they are. If it cannot, it uses default values.

Determining Primary Collector Log-File Names

The first log-file name used at system load is:

- If a context file (ZZEVCONF) exists in the default subvolume and if the first record in that file is a ZEMS-EVT-FILESWITCH event message, the collector assumes that logging was previously in progress.

  The previous log file is located by retrieving its name from the ZEMS-MAP-COL-STATUS token in the FILESWITCH event message. If that file name corresponds to the file name in the $SYSTEM.SYSTEM.ZZEVCONF file, logging resumes in that log file. If the file names do not agree, the integer portion of the file name (the \textit{nnnn} part of ZZEV\textit{nnnn}) is incremented by one, and logging is started in a new file with that name.

- If no files exist in the default subvolume, or if a ZZEVCONF file exists but the first record in the file is not a ZEMS-EVT-FILESWITCH event message, the collector assumes that logging has not previously been in progress, and it creates $SYSTEM.ZLOG_{nn}.ZZEV0000 as its first log file.
Determining Primary Collector Log-File Attributes

When the primary collector starts logging after a system load and has determined where the initial log file is to be located, it sets its file configuration attributes as:

- If a valid ZZEVCONF file exists on the log subvolume, all values are taken from the ZEMS-MAP-COL-STATUS token, except DISCACCESSID, which is not stored in the COL-STATUS token and always defaults to super ID after a system load.

- If there is no valid ZZEVCONF file on the log subvolume, but there is a valid $SYSTEM.SYSTEM.ZZEVCONF, all values (except DISCACCESSID) are taken from that file instead.

- If there is no valid ZZEVCONF file in either location, the collector creates the first log file with these attribute settings:

  PRIMARYEXTENTS = 20
  SECONDARYEXTENTS = 100
  WRITETHRUCACHE = FALSE
  EOFREFRESH = TRUE
ROTATEFILES = TRUE
MAXFILE = 4
PROTECTION = COOO
DISCACCESSID = SUPER.SUPER
BLOCKING = ON

These attributes can be changed programmatically by sending a CONTROL command message, or interactively by using the EMSCCTRL utility program. The maximum number of extents is always 16.

Logging After a Subvolume Switch

If you change the log subvolume (LOGSUBVOL) for either a primary or alternate collector, the log-file name is selected as:

- If a context file (ZZEVCONF) exists in the new subvolume and if the first record in that file is a ZEMS-EVT-FILESWITCH event message, the collector assumes that logging was previously in progress in this subvolume.

  The previous log file is located by retrieving its name from the ZEMS-MAP-COL-STATUS token in the FILESWITCH event message. The integer portion of the file name (the \textit{nnnn} part of \texttt{ZZEVnnnn}) is incremented by one, and logging is started in a new file with that name.

- If no files exist in the new subvolume, or if a ZZEVCONF file exists but the first record in the file is not a ZEMS-EVT-FILESWITCH event message, the collector assumes that logging has not previously been in progress in this subvolume, and it creates ZZEV0000 as its first log file.

The attribute values used to create log files in the new subvolume are established as:

- Any attributes specified in the CONTROL command message or the EMSCCTRL parameters are always used.

- If a ZZEVCONF file exists in the new subvolume, attributes not specified in the CONTROL command message or EMSCCTRL parameters are taken from that file. DISCACCESSID is not saved in the ZZEVCONF file, so this possibility does not apply to DISCACCESSID.

- If no ZZEVCONF file exists in the new subvolume, attributes not specified in the CONTROL command message or EMSCCTRL parameters keep their current values. DISCACCESSID always keeps its current value across a subvolume switch unless it is explicitly specified in the CONTROL command message or as an EMSCCTRL parameter.

Changing subvolumes might unexpectedly change the settings of one or more attributes if a ZZEVCONF file exists in the new subvolume and contains different attribute settings than those previously in effect. For this reason, explicitly specify any attributes that you are concerned might have undesirable settings in the ZZEVCONF file.
Switching Log Files

All tasks associated with log switches are performed when the collector is idle (that is, when there are no events to log). When a log file is full, or when you request a log-file switch by setting the NEXTLOGFILE option, the collector:

- Determines the name of the next log file. The name of the next file is derived by adding one to the integer portion of the file name of the current log file. If the current log file is ZZEV9999, the name of the next log file is ZZEV0000.

- Determines whether the creation of a new log file causes MAXFILE to be exceeded. Before opening the new log file, the collector checks for a file called ZZEV*mmmm*, where *mmmm* is calculated by subtracting the value of MAXFILE from the integer portion of the new log-file name. For example, if the new name is ZZEV0043 and MAXFILE is 5, the collector checks if the file ZZEV0038 is present. If MAXFILE is exceeded, the action taken depends on the value of ROTATEFILES and on the type of collector (primary or alternate) controlling the log file. For more information about how each collector handles the MAXFILE exceeded condition, see ROTATEFILES on page 12-9.

If the log file to be purged is open, the action depends on the collector type.

The primary collector purges the data in the file, deallocates its extents, and sends the opener an end-of-file indication on the next or current operation, if any. When this happens, an empty file is left on disk. Alternate collectors leave open log files as they are. The data is not purged, and the file’s extents are not deallocated.

If logging stops, the collector continues to check for the presence of the lower-numbered file (ZZEV*mmmm*) at five-second intervals. When the file is removed, the collector continues with the file-switch sequence.

- Checks for a preallocated log file. Using the File Utility Program (FUP)—or the ALLOCATE feature, for alternate collector log files—you can preallocate a collector’s log files to ensure that the collector does not run out of disk space. If a file with the new log-file name already exists and has the proper attributes for a log file, the collector begins logging in this preallocated file.

- Creates the new log file. If a file with the new log-file name does not exist, the collector creates the file, carrying over its current attribute settings, with this exception: if the file switch was initiated by a CONTROL command message or EMSCCTRL parameter that set NEXTLOGFILE and some other attributes, the attributes set with NEXTLOGFILE override the current attribute settings.

- Generates a ZEMS-EVT-FIILESWITCH event message, logs it, and notifies any distributors waiting for the next event message.

- Updates the ZZEVCONF files on the log subvolume (and on $SYSTEM.SYSTEM, for the primary collector) to include the new FILESITCH event message.

Because of the overhead required to switch log files, do not set NEXTLOGFILE frequently. This attribute is provided to let you move a collector out of a log file when
you need that specific file (for example, when you have a scheduled log-file archival routine or when a serious problem is detected and you want to isolate that log file).
This section describes the programs you can run to change the configuration of or display status information about various EMS components:

Table 13-1. Programs That Start EMS Components

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMSACOLL</td>
<td>Object program for the alternate collector</td>
</tr>
<tr>
<td>EMSDIST</td>
<td>Object program for a consumer, forwarding, or printing distributor process</td>
</tr>
</tbody>
</table>

To start and stop these EMS programs, use the TACL RUN and STOP commands.

Several utility programs help you monitor and manage components of EMS.

Table 13-2. EMS Utility Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMSCCTRL</td>
<td>Controls the operation of the primary and alternate collectors and the attributes of collector log files; also controls the compatibility distributor</td>
</tr>
<tr>
<td>EMSCINFO</td>
<td>Displays current collector operational information and event-message statistics</td>
</tr>
<tr>
<td>EMSDISTINFO</td>
<td>Displays current distributor information and filter statistics</td>
</tr>
</tbody>
</table>

Each utility interprets the requests that you provide through run options and command-line options, performs the requested action, and terminates execution.

These utilities provide many of the capabilities available to you through collector and distributor command messages. You can run EMSCINFO to determine the status of a collector or the compatibility distributor, and EMSDISTINFO to determine the status of a consumer, printing, or forwarding distributor. Also, you can use EMSCCTRL to control the operation of the collectors and their log files, and the compatibility distributor. But you must use command messages to control the operation of a consumer, printing, or forwarding distributor; no utility program performs this function.

To understand collector attributes and how they are interrelated, see the corresponding command messages in Section 17, Distributor Commands and Responses, or Section 19, Collector Commands and Responses. Collector attributes are described in terms of RUN-command options in this section and in terms of token fields in Section 19, Collector Commands and Responses, but the underlying operations are the same.
EMSACOLL—Alternate Collector Program

EMSACOLL is the object program for the alternate collector. The alternate collector provides an alternative to the primary collection point made available by the primary collector ($0).

You can start an alternate collector by using the TACL RUN command. You can pass attribute information to the collector as startup text to define the operational characteristics of the collector.

Use the EMSACOLL RUN command line for starting an alternate collector to:

- Assign values to the various burst detection and suppression (BDS) configuration parameters
- Specify one or more filters for pre-log filtration (PLF)
- Specify a different prefix pattern for log files and the alternate collector’s ZZEVCONF configuration file

For a complete description of alternate collector attributes, see Alternate Collector Log Files on page 12-4.

The format for the alternate collector RUN command is:

```
EMSACOLL [ / run-options / ] [ altcol-options ]
```

**run-options**

specifies the valid options for the TACL RUN command, separated by commas, as described in the TACL Reference Manual. These options are of importance:

- **NAME [ process-name ]**

  specifies the name of the alternate collector process. If process-name is omitted, the system assigns a name beginning with X, Y, or Z.

- **NOWAIT**

  if specified, returns control to the TACL.

- **CPU pri-cpu**

  is the CPU number of the primary process.

- **PRI priority**

  assigns the run-time priority of the alternate collector process. This value must be in the range 1 through 199.

**altcol-options**

are any of these alternate collector options, separated by commas:

- **EMSACOLL**
- **/ run-options /**
- **altcol-options**
SUPPRESS [{ ( suppression-parameters ) }]

lets the default values for the burst detection and suppression parameters be overridden with user-specified values. Using the SUPPRESS keyword causes the alternate collector to start with BDS enabled. The absence of SUPPRESS causes BDS to be disabled at startup unless a burst filter is specified in the FILTER keyword.

EMSACOLL’s SUPPRESS and FILTER keywords cannot be specified simultaneously when starting an alternate collector. If the SUPPRESS keyword is used without any SUPPRESS parameters, BDS is enabled and its default parameter values are used. For a detailed description about each BDS parameter and the values, including default values, that can be used with the SUPPRESS keyword, see Section 7, Burst Detection and Suppression.

If the FILTER keyword’s parameters specify a burst filter and SUPPRESS is used, the alternate collector issues an Illegal Syntax error message in its OUT file and performs an abnormal end (ABEND).

suppression-parameters has the format

suppression-parameter [, ... ]

where each suppression-parameter is of the form

parameter [ = ] value

where parameter is one of the BDS parameters (N, T1, T2, T3, L, S) and value is the value you supply for that parameter. The BDS suppression parameters are supplied in a list that is separated by commas.

The values for T1, T2, and T3 are expressed in seconds. The value for N is expressed in events. The value for L is expressed in bytes. The value for S is expressed in burst events. For details about these and other BDS configuration values, see Section 7, Burst Detection and Suppression.

FILTER { filter-name | (filter-name [, filter-name ]... ) } selects the filter or filters to be used with the pre-log filtration (PLF) feature. Each filter-name variable is the name of a filter file. If more than one filter is needed for the alternate collector, these filter names must be specified in parentheses in a comma-separated list. If the specified filters are correct and usable, using the FILTER parameter enables PLF. If FILTER is not used, the alternate collector starts with PLF disabled, and all events are logged to disk. If one of the filters specified is a burst filter, BDS is automatically enabled.

The named filter can be a compiled filter, a filter table, or a burst filter. If a burst filter is named and the EMSACOLL SUPPRESS parameter is also enabled, the alternate collector issues an Illegal Syntax error message in its OUT file and performs an abnormal end (ABEND). No more than one burst filter can be specified.
If any of the filters is bad or incorrectly identified, the alternate collector generates an appropriate error message and terminates. The error message specifies the first bad filter encountered. A bad filter is one that is not in the correct format or cannot be accessed.

**LOGPREFIX char**

lets the user specify a different prefix pattern for an alternate collector’s log files and the configuration file. The default pattern for the context file is ZZEVCONF, and the default pattern for the log files is ZZEVnnnn. The char variable can be any alphanumeric character (0 through 9; A through Z) and modifies the third character of the prefix pattern. For example, if the char variable is Y, the context file is named ZZyVCONF and the log files are named ZZyVnnnn.

**BACKUP backup-cpu**

specifies the CPU number for the backup process. If this option is not specified, the alternate collector does not run as a process pair.

**BLOCKING { ON | OFF }**

assigns the value of the BLOCKING attribute.

**LOGSUBVOL subvol-name**

identifies the subvolume that should be used for the log files. If this option is not specified, the subvolume referred to by DEFAULTSUBVOL is used. If neither option is specified, the value in =_DEFAULTS is used.

**DEFAULTSUBVOL subvol-name**

identifies the subvolume that should be used if the log subvolume becomes inaccessible. If this option is not specified, the default volume is the log subvolume. If DEFAULTSUBVOL is not specified and the log subvolume becomes inaccessible, the alternate collector stops logging.

**ROTATEFILES { ON | OFF }**

tells the collector what action to take if the creation of a new log file causes the number of log files to exceed the MAXFILE limit by assigning the value of the ROTATEFILES attribute. The primary and alternate collectors take different actions when this condition occurs.

If ROTATEFILES is ON (TRUE):

- The primary collector purges the oldest file.
- The alternate collector checks the oldest file.
  
  If this file is open to another process or is not a valid log file, the alternate collector creates the next file in the sequence and continues logging.
If the oldest file is a valid file and not open to another process, the alternate collector either renames the file as the next in the sequence if it has the correct number of extents, or it purges and creates a new file if it does not have the correct number of extents.

If ROTATEFILES is OFF (FALSE):

- The primary collector sends the event message ZEMS-EVT-LOGGING-STOPPED (521) and stops logging.
- The alternate collector checks the oldest file.
  
  If the file is not valid, the alternate collector creates the next file in the sequence.
  
  If the file is valid and not empty, logging stops and the collector generates the ZEMS-EVT-LOGGING-STOPPED (521) event message. If the file is valid and empty, the collector checks the extent size. If the extent size is correct, the collector renames and reuses the file. If the size is not correct, the collector purges the file and creates the next file in the sequence.

**MAXFILE nnnn**

determines the maximum number of files the alternate collector retains in the log subvolume. nnnn must be an integer in the range 2 through 1000.

**EXT { ext | ( pri , sec ) }**

selects the values (in pages) for the PRIMARYEXTENT and SECONDARYEXTENT attributes. If ext is specified, that value is used for both attributes. Otherwise, pri specifies the value of PRIMARYEXTENT and sec specifies the value of SECONDARYEXTENT. ext, pri, and sec must be even integers in the range 2 through 65534.

**BUFFERED { ON | OFF }**

selects the value of the WRITETHRUCACHE attribute. BUFFERED ON sets the attribute to FALSE. BUFFERED OFF sets it to TRUE.

**REFRESH { ON | OFF }**

selects the value of the EOFREFRESH attribute.

**SECURITY { rwep | "rwep" }**

selects the value of the PROTECTION attribute and specifies the file security that the collector is to use when creating a log file. For the security string for the FUP SECURE command, see the File Utility Program (FUP) Reference Manual.

Because distributors must have read-access to their log files, the value of SECURITY limits which people (or processes) can use distributors to retrieve event messages. The system supplies “COOO” if you have not specified
SECURITY since the system load—which makes super-group membership a requirement for event-message retrieval.

REPLYAFTERWRITE { ON | OFF}
selects the value of the REPLYAFTERWRITE attribute.

ALLOCATE
causes the alternate collector to attempt to create and allocate MAXFILE files in the log subvolume. It allocates and creates files in the range ZZEV\text{x x x x} through ZZEV\text{yyyy}, where the values of \text{x} and \text{y} are four-digit numbers whose values are dependent on the values of MAXFILE and the current file number (\text{cfn}).

\[
\text{xxxx} = (\text{cfn} - \text{MAXFILE} + 10001) \mod 10000
\]
\[
\text{yyyy} = \text{cfn}
\]

In no case does ALLOCATE change the current file number.

Examples:
When \text{cfn} = 2 and \text{MAXFILE} = 4, the files created are ZZEV9999, ZZEV0000, ZZEV0001, and ZZEV0002.

When \text{cfn} + 6 and \text{MAXFILE} = 3, the files created are ZZEV0004, ZZEV0005, and ZZEV0006.

If ALLOCATE is specified and the alternate collector can start running, but cannot fully allocate all of the log files or alternate key files in the specified LOGSUBVOL, the collector issues this message to its OUT file:

\text{<process name> : WARNING - Unable to Allocate File <file name>, Error: <file error #>}

The alternate collector continues to run after issuing this message.

POOLPAGES \text{pages}
determines the value of the POOLPAGES attribute. This attribute can have values in the range 20 through 128.

**Startup Error Messages**

If the alternate collector detects an error, it issues a message to its OUT file and terminates abnormally. The general format of an error message is:

\[
\text{process name : message text}
\]

message text
can have one of these values:
Duplicate attribute Specification
   The named attribute has been specified more than once in the startup text.

Extended Segment Allocation Error: file management error #
   The primary alternate collector process cannot allocate its extended data segment.

Illegal BACKUP CPU
   The CPU specified as the backup does not exist or is the same CPU in which the primary process is running.

Illegal Character - character
   The specified illegal character appeared in the startup text.

Illegal Default Subvolume
   The DEFAULTSUBVOL was illegally specified as $SYSTEM.SYSTEM or $SYSTEM.ZLOGnn.

Illegal Extent Specification
   The specified extent size is represented by an odd integer or is greater than 65534.

Illegal Log Subvolume
   The LOGSUBVOL was illegally specified as or defaulted to $SYSTEM.SYSTEM or $SYSTEM.ZLOGnn.

Illegal MAXFILE
   The value specified for MAXFILE is less than 2 or greater than 1000.

Illegal Number - number
   The indicated number appears in the startup text and cannot be represented as a 32-bit integer.

Illegal POOLPAGES
   The value specified for POOLPAGES is less than 20 or greater than 128.

Illegal Security Specification
   The value specified for security is invalid.

Illegal Subvolume - token
   The indicated token appears in the startup text where the collector expects a valid volume or subvolume name.
Illegal Syntax - token
The indicated token cannot legally appear where it does in the startup text.

Startup Warning Messages

If a warning condition is detected, the alternate collector issues a message on the home terminal and continues to run. It does not terminate abnormally as it does with startup error messages. The possible message text values are:

**Warning - Unable to Start Logging—Error:**

file error #, File: file name
The alternate collector attempted to start logging in the LOGSUBVOL and DEFAULTSUBVOL, but both attempts failed.

**Warning - Unable to Allocate File file name, Error:**

file management error number #
The alternate collector generates this message when ALLOCATE is specified as a startup parameter and the alternate collector is unable to allocate MAXFILE files in the log subvolume.

file name
names the first file on which allocation failed.
EMSCCTRL—Control Collector Utility

The EMSCCTRL utility lets you interactively manage the operation of the primary and alternate collectors and the operation of the compatibility distributor.

EMSCCTRL supports a single input line from the TACL RUN command line. By supplying one or more parameters, you select the operational characteristics of the collector. EMSCCTRL translates the human-readable RUN parameters into a CONTROL command (or DELETE FILTER or ADD FILTER command), sends the command to the specified collector, and displays the response from that collector.

For information about the CONTROL, DELETE FILTER, and ADD FILTER command messages that performs similar operations, see Collector Command Descriptions on page 19-29.

You must have super-group privileges to use EMSCCTRL to manage the primary collector. To manage an alternate collector, you must have super-group privileges or be the user who started the alternate collector.

For a complete description of collector attributes, see Primary Collector Log Files on page 12-3 and Alternate Collector Log Files on page 12-4.

The format of the EMSCCTRL RUN command is:

```
EMSCCTRL [/ run-options /] [ HELP ]
[ col-name col-options ]
[ acol-name STOP ]
```

If you omit one of the options that follows the collector name on a particular EMSCCTRL run, the corresponding attribute remains unchanged.

**run-options**

are any of the run options described in the TACL Reference Manual. Separate the option names by commas.

**HELP**

displays a brief syntax description of EMSCCTRL. Omitting the collector name and all collector options is equivalent to HELP.

**col-name**

is the name of the primary collector ($0) or alternate collector on the local system. Use $0, not $Z0, to control either the primary collector or the compatibility distributor associated with the primary collector.

**acol-name**

is the name of an alternate collector on the local system. If acol-name STOP is specified, a shutdown of the named alternate collector is initiated.
col-options

are one or more of these collector options, separated by commas:

ALLOCATE (alternate collector only)

if present, makes the alternate collector attempt to create or allocate MAXFILE files in the current log subvolume. The range of the log file numbers created or allocated is the same as that described in EMSACOLL—Alternate Collector Program on page 13-2.

If ALLOCATE is specified with LOGSUBVOL, the allocation takes place in the new subvolume. Similarly, if ALLOCATE is specified with NEXTLOGFILE, the next log file is opened prior to allocation.

BLOCKING { ON | OFF  }

if ON, causes the collector to block event messages before writing them to the log file. Default value for the primary and alternate collectors is ON.

BUFFERED { ON | OFF  }

if ON, directs the disk process to buffer records in the disk cache buffer, rather than writing them immediately to disk. For more information, see the ZCOL-WRITETHRUCACHE field in CONTROL Command (ZEMS-CMD-CONTROL) on page 19-34.

CDISTMODE { selection-criterion }

specifies the selection criterion used by the compatibility distributor to determine which messages to print.

The compatibility distributor, $Z0, receives all event messages, including pre-EMS messages, in EMS token form. The selection criterion determines the way $Z0 handles each message. selection-criterion can have these values:

selection-criterion

CRITICAL-ONLY ON

$Z0 suppresses the display of some messages on the operator console. This decreases the number of noncritical messages displayed on the console, particularly during a coldload. The messages suppressed are messages with console message numbers 6 (LDEV UP), 141 (CLIP DOWNLOADED), and 150 (CSS ACTIVATE PATH). CRITICAL-ONLY ON is the default selection criterion.
CRITICAL-ONLY OFF

$Z0 displays all messages.

Note. Use the CDISTMODE option only with the primary collector.

CDISTSTOP

stops the compatibility distributor.

Note. Use the CDISTSTOP option only with the primary collector.

For more information about the CDISTSTOP option, see Installation and System Generation Considerations on page 12-21 and Compatibility Distributor ($Z0) on page 12-14.

CDISTUSER usergroup.username

specifies the user ID that the compatibility distributor uses to access the CONSOLE. Usually this option is only used to access CONSOLE devices on remote systems.

Note. Use the CDISTUSER option only with the primary collector.

EXT { ext 
{ ( priext , secext ) }

provides two ways to specify the extents that the collector gives to new log files. EMSCCTRL interprets a list of two values as the values to give the primary and secondary extents. The parentheses in the list alternative are required. EMSCCTRL interprets one value ext—with no parentheses—as the value to give to both primary and secondary extents.

FILTER { filter-name | (filter-name [, filter-name ] ...) 
| ON | OFF | RESET

selects the filters used for pre-log filtration, or enables or disables PLF. The filter-name variable is the name of a filter file. If you want more than one filter, each filter must be specified within parentheses as a comma-separated list. Specifying the FILTER parameter with at least one filter name (assuming the specified filter is correct and usable) enables PLF, if it is not already enabled. You can change the filters while PLF is enabled. The new filters specified replace any previously entered filters in the collector.

Specifying FILTER OFF turns off PLF. Any filters that were previously entered are still retained by the collector, but PLF is disabled. If PLF is already disabled, FILTER OFF has no effect on the collector.

Specifying FILTER ON enables PLF if the collector has retained previously entered filters; otherwise, EMSCCTRL does not enable PLF and instead
generates a “No Operation” message on its OUT file. If PLF is already enabled, FILTER ON has no effect on the collector.

Specifying FILTER RESET disables PLF and causes the collector to delete any previously entered filters. If PLF is already disabled and the collector has no retained filters, FILTER RESET has no effect on the collector.

The filter specified in the filter-name variable can be a compiled filter, a filter table, or a single burst filter. If a burst filter is specified and the SUPPRESS keyword is also specified, EMSCCTRL generates an error message in its OUT file. Only one burst filter can be specified for a collector. Specifying a burst filter when BDS is enabled causes the BDS configuration parameter values to be replaced by those contained in the burst filter.

If a burst filter is deleted and BDS is subsequently enabled, the burst filter configuration parameters from the deleted burst filter are still active for that collector.

If BDS is specified using the SUPPRESS keyword, only SUPPRESS ON/OFF can enable or disable BDS.

LOGSUBVOL [ subvol ]

if subvol is present, it gives the name of the volume and subvolume in which a collector will create log files. If subvol is omitted, this option directs the collector to use the user’s current volume and subvolume.

The default value of the LOGSUBVOL parameter for the primary collector, which is established at system load, is:

sysvol.ZLOGnn

sysvol name of the disk volume from which the system was loaded—typically $SYSTEM.

nn number in the name of the subvolume from which the system was loaded: SYSnn.

If a new file is needed when ROTATEFILES is ON and MAXFILE log files already exist in the log subvolume, the primary collector purges the log file containing the messages logged earliest, and creates a new file with the next name in the sequence (files are named ZZEV0000 through ZZEV9999).

For example, suppose that the MAXFILE value is four, that all four files are in use, that a new file is needed, and that the current log file is named ZZEV0034. The collector purges ZZEV0031, creates ZZEV0035, and resumes logging event messages.
LOGUSERID usergroup.username

specifies the user ID that the primary collector uses when it accesses its log files.

The primary collector uses the super ID to access log files if you have never changed its user ID. If you change the user ID of the primary collector, use LOGUSERID to change the owner and/or security of the ZZEVCONF and log files so the new user ID has read/write/purge access to these files. If the new user ID does not have read/write/purge access to these files, $0 reports error 48 the next time it needs to open these files, and stops logging to disk. To fix this, use LOGUSERID or change the owner and/or security of the log files.

The alternate collector does not support the EMSCCTRL LOGUSERID option. If specified for the alternate collector, EMSCCTRL responds with this error message:

Control Command Failed, RetCode Error = 1005,
   Invalid token value

MAXFILE nnnn

limits the number of log files that can exist at one time in the volume and subvolume specified by LOGSUBVOL. nnnn must be in the range 2 to 1000 (decimal); leading zeros are not required. The default maximum is 4 if you omit the parameter value or the entire parameter.

Note. You can reset MAXFILE to a value less than the actual number of log files existing when the MAXFILE command is issued. The primary and alternate collectors use the new value of MAXFILE to determine the action taken when the current log file fills or a NEXTLOGFILE command is issued. Both collectors assume that only MAXFILE files exist, ignoring the older files.

For example, if ROTATEFILES is ON and a NEXTLOGFILE command is issued, the primary collector purges a single file (currentfile + 1 - MAXFILE). It then creates a new file (currentfile + 1). The remaining older files, if any, remain in the system until you purge them. To illustrate, assume that MAXFILE is reset to 2 and there are three existing log files; file4, file5, and file6. When a NEXTLOGFILE command is issued, file5 is purged, and file7 is created (assuming ROTATEFILES is ON). file4 remains untouched.

NEXTLOGFILE

if present, tells the collector to close the current log file, create a new one, and open it. The collector then generates a fileswitch event message and resumes event-message logging with the new file.
REFRESH { ON | OFF }

if ON, tells a primary or alternate collector to update the end-of-file pointer on disk for each block written. This is safer than REFRESH OFF, less efficient because it involves more writes to disk.

ROTATEFILES { ON | OFF }

tells the collector what action to take if the creation of a new log file causes the number of log files to exceed the MAXFILE limit by assigning the value of the ROTATEFILES attribute. The primary and alternate collectors take different actions when this condition occurs. The ROTATEFILES attribute is ON if you have never explicitly set the ROTATEFILES option.

If ROTATEFILES is ON (TRUE):

- The primary collector purges the oldest file.
- The alternate collector checks the oldest file.

  If this file is open to another process or is not a valid log file, the alternate collector creates the next file in the sequence and continues logging.

  If the oldest file is a valid file and not open to another process, the alternate collector either renames the file as the next in the sequence if it has the correct number of extents, or purges and creates a new file if it does not have the correct number of extents.

If ROTATEFILES is OFF (FALSE):

- The primary collector sends the event message ZEMS-EVT-LOGGING-STopped (521) and stops logging.
- The alternate collector checks the oldest file.

  If the file is not valid, the alternate collector creates the next file in the sequence.

  If the file is valid and not empty, logging stops and the collector generates the ZEMS-EVT-LOGGING-STopped (521) event message. If the file is valid and empty, the collector checks the extent size. If the extent size is correct, the collector renames and reuses the file. If the size is not correct, the collector purges the file and creates the next file in the sequence.

SECURITY { rwep | “rwep” }

specifies the file security that the collector is to use when creating a log file. See the security string for the FUP SECURE command in the File Utility Program (FUP) Reference Manual.

Because distributors must have read-access to their log files, the value of SECURITY limits which people (or processes) can use distributors to
retrieve event messages. If you have not specified SECURITY after a
coldload, or if the collector does not find a ZZEVCONF file in the default or
system subvolume, the system supplies "COOO" to make super-group
membership a requirement for event-message retrieval.

SUPPRESS { ( suppression-parameters ) | ON | OFF | RESET }
specifies the burst detection and suppression (BDS) configuration
parameter values and enables BDS if it is not already enabled for the
collector. You can change the BDS configuration parameters while BDS is
enabled.

If the EMSCCTRL FILTER keyword specifies a burst filter or a burst filter
was previously added to the collector, and EMSCCTRL SUPPRESS is
specified, the EMSCCTRL generates an error message in its OUT file.

suppression-parameters has the format

suppression-parameter [ , ... ]

and where each suppression-parameter is of the form

parameter [ = ] value

where parameter is one of the BDS parameters (N, T1, T2, T3, L, S) and
value is the value you supply for that parameter. The BDS suppression
parameters are supplied in a comma-separated list.

The values for T1, T2, and T3 are expressed in seconds. The value for N is
expressed in events. The value for L is expressed in bytes. The value for S
is expressed in burst events. For details about these and other BDS
configuration values, see Section 7, Burst Detection and Suppression.

Specifying SUPPRESS OFF disables BDS for the collector. The BDS
suppression parameters in the collector, while disabled, are not altered. If
BDS is already disabled, SUPPRESS OFF has no effect on the collector.

Specifying SUPPRESS ON enables BDS for the collector. The collector
uses the last specified suppression parameters. If BDS is already enabled,
SUPPRESS ON has no effect on the collector.

Specifying SUPPRESS RESET disables BDS and causes the collector to
adopt the default BDS configuration parameter values for its own
suppression values. For a detailed description of these default values, see
Section 7, Burst Detection and Suppression.

If BDS is specified by means of a burst filter, either FILTER ON/OFF or
SUPPRESS ON/OFF can be used to enable or disable BDS.
SWITCH { COLL cpu }
{ CDIST cpu }

designates which process of a process pair is to become the primary process. You specify the process by its CPU number, which must designate the processor in which the backup process is currently running. (You specify cpu so two attempts to switch primary processes out of CPU 7, for example, do not cancel each other. The first attempt succeeds; the second attempt gets an error but is otherwise ignored.)

**Note.** Use the SWITCH CDIST option only with the primary collector.

TEXTOUT [ console ]

specifies a console device (CONSOLE), in place of the default console selected at system load, to which $Z0 routes event messages. The EMS CONTROL command, described in Section 19, Collector Commands and Responses, (or the PUP CONSOLE command on a D-series RVU) can make the same change.

To find out the current or default console device for $Z0, use the EMSCINFO—Collector Information Utility on page 13-18.

**Note.** Use the TEXTOUT option only with the primary collector.

**Note.** Block mode applications cannot be started on the current $0 CONSOLE device. Attempting to start a block mode application, such as ViewSys or TEDIT, results in an error 549. Conversely, if a block mode application is running on a device, that device cannot be the $0 CONSOLE device. Attempting to switch the CONSOLE device using the EMSCCTRL $0 TEXTOUT command (or the PUP CONSOLE command on a D-series RVU) results in terminal error of 12 (indicated in the EMSCINFO display and in the EMS logs).

To run block mode applications on the CONSOLE device, do one of:

- Use another device for either the block mode application or the console. If the designated console device is actually a 6530 emulation window on a desktop computer (for example, a PC, Macintosh, or UNIX workstation), open another 6530 window.
- Encapsulate the invocation of a block mode application in a TACL macro that first issues an EMSCCTRL $0 TEXTOUT $0 (or PUP CONSOLE $0 on a D-series RVU) and later issues EMSCCTRL $0 TEXTOUT $0 [#MYTERM] (or the PUP CONSOLE [#MYTERM] command on a D-series RVU) after the block mode application stops.
Referencing Collector Attributes

Table 13-3. Referencing Collector Attributes

<table>
<thead>
<tr>
<th>EMSCTRL Option</th>
<th>ZEMS-MAP-(A)COL-CONTROL Command Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCH COLL</td>
<td>ZCOL-PRIMARYCPU</td>
</tr>
<tr>
<td>LOGSUBVOL</td>
<td>ZCOL-LOGSUBVOL</td>
</tr>
<tr>
<td>NEXTLOGFILE</td>
<td>ZCOL-NEXTLOGFILE</td>
</tr>
<tr>
<td>ROTATEFILES</td>
<td>ZCOL-ROTATEFILES</td>
</tr>
<tr>
<td>MAXFILE</td>
<td>ZCOL-MAXFILENNNN</td>
</tr>
<tr>
<td>EXT</td>
<td>ZCOL-PRIMARYEXTENT</td>
</tr>
<tr>
<td>EXT</td>
<td>ZCOL-SECONDARYEXTENT</td>
</tr>
<tr>
<td>BLOCKING</td>
<td>ZCOL-EVENTBLOCKING</td>
</tr>
<tr>
<td>BUFFERED</td>
<td>ZCOL-WRITETHRUCAKE</td>
</tr>
<tr>
<td>REFRESH</td>
<td>ZCOL-EOFREFRESH</td>
</tr>
<tr>
<td>LOGUSERID</td>
<td>ZCOL-DISCACCESSID</td>
</tr>
<tr>
<td>SECURITY</td>
<td>ZCOL-PROTECTION</td>
</tr>
<tr>
<td>ALLOCATE</td>
<td>ZCOL-ALLOCATE</td>
</tr>
<tr>
<td>FILTER (ON/OFF)</td>
<td>ZCOL-PRELOGFILTER</td>
</tr>
<tr>
<td>SUPPRESS (ON/OFF)</td>
<td>ZCOL-BURSTSUPDETECT</td>
</tr>
</tbody>
</table>

Most EMSCTRL options are related to their corresponding token fields in a straightforward way. For example, a ROTATEFILES option set ON corresponds to a ZCOL-ROTATEFILES field that is TRUE. However, a ZCOL-WRITETHRUCAKE token that is TRUE is equivalent to a BUFFERED option set OFF. For more information, see CONTROL Command (ZEMS-CMD-CONTROL) on page 19-34.
EMSCINFO—Collector Information Utility

The EMSCINFO utility returns status information about a primary or alternate collector. For a description of the command message that provides the same information, see STATUS Command (ZEMS-CMD-STATUS) on page 19-53.

To change these attributes, see EMSCCTRL—Control Collector Utility on page 13-9.

When the DETAIL parameter option is present in the EMSCINFO command syntax, a number of BDS and PLF parameters and counters can be viewed. The DETAIL parameter option lets the user view this additional information while maintaining manageable screen size.

```
EMSCINFO [<collector name>] [, DETAIL]
```

EMSCINFO Display Examples (No DETAIL Parameter)

Example 13-1 shows the primary collector status display produced by EMSCINFO in response to the command “EMSCINFO $0,” without the DETAIL option specified.

Definition of Terms on page 13-19, describes the terms in Example 13-1.

Example 13-1. EMSCINFO Primary Collector Display (No DETAIL Parameter)

```
EMSCINFO - T9631D31 - (03APR95)
Collector = $0, Priority = 201
Primary CPU = 1, Backup CPU = 0
Disk Logging Active
Current Log File = $SYSTEM.ZLOG06.ZZEV0045
Current Record = $000005.050003
Default Log File = $SYSTEM.ZLOG06.ZZEV0045
Primary Extent = 20, Secondary Extent = 100
ROTATEFILES = ON, MAXFILE = 4
REFRESH = ON, BUFFERED = ON
SECURITY = NUUU, BLOCKING = ON
Events Received = 707516, Events Logged = 705697
Events Discarded = 1909
OPENS Received = 241, CLOSES Received = 233
FILE Switches = 45, Unrcv Disk Errors = 0
Invalid Events = 0, Buffer Failures = 1909
SUPPRESS = OFF, FILTER = ON
Compatibility Distributor ($Z0)
Printing Mode = CRITICAL-ONLY
Primary CPU = 0, Backup CPU = 1
Console Name = $0
Default Console Name = $OSP
```
Example 13-2 shows the alternate collector status display produced by EMSCINFO in response to the command “EMSCINFO $ACOL,” without the DETAIL option specified.

The EMSCINFO display for an alternate collector differs from the display for the primary collector in these ways:

- The values of POOLPAGES and REPLYAFTERWRITE are displayed.
- No compatibility distributor information is displayed.

### Example 13-2. EMSCINFO Alternate Collector Display (No DETAIL Parameter)

```
EMSCINFO - T9631D31 - (03APR95)
Collector = $ACOL, Priority = 140
Primary CPU = 7
Current Logging Error = 012
Current Log File = $PUBS.LDHMISC.ZZEV0000
*** Switching to next file ***
Default Log File = $PUBS.LDHMISC.ZZEV0000
Primary Extent = 20, Secondary Extent = 100
ROTATEFILES = ON, MAXFILE = 4
REFRESH = ON, BUFFERED = ON
SECURITY = COOO, POOLPAGES = 64
REPLYAFTERWRITE = OFF BLOCKING = ON
Events Received = 0, Events Logged = 0
Events Discarded = 0
OPENS Received = 2, CLOSES Received = 1
FILE Switches = 0, Unrcv Disk Errors = 1
Invalid Events = 0, Buffer Failures = 0
SUPPRESS = OFF, FILTER = ON
```

### Definition of Terms

The terms in these two examples are described in the order they appear in each figure:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collector</td>
<td>Name of the collector.</td>
</tr>
<tr>
<td>Priority</td>
<td>Current execution priority of the collector.</td>
</tr>
<tr>
<td>Primary CPU</td>
<td>Number of the CPU in which the primary process for the collector is running.</td>
</tr>
<tr>
<td>Backup CPU</td>
<td>Number of the CPU in which the backup process for the collector is running.</td>
</tr>
<tr>
<td>Disk Logging</td>
<td>Displayed if there is no current logging error condition.</td>
</tr>
<tr>
<td>Active</td>
<td>Displays the error number of a logging error encountered during a logging operation, if such a condition exists. Logging errors are either disk failures or operational errors. For a description of operational errors, see <a href="#">Unrcv Disk Errors</a> on page 13-21.</td>
</tr>
<tr>
<td>Current Logging Error = error</td>
<td></td>
</tr>
<tr>
<td>Current Log File</td>
<td>File name of the current log file.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Current Record</td>
<td>Record number (as received from the FILEINFO procedure) of the last record written to the log file if the collector is not switching log files.</td>
</tr>
<tr>
<td>*** Switching to next file ***</td>
<td>Displayed in place of the current record if the collector is in the process of switching log files.</td>
</tr>
<tr>
<td>Default Log File</td>
<td>File name of the last file in use in the default subvolume.</td>
</tr>
<tr>
<td>Primary Extent</td>
<td>Size of the primary extent used for log-file creation.</td>
</tr>
<tr>
<td>Secondary Extent</td>
<td>Size of the secondary extent used for log-file creation.</td>
</tr>
<tr>
<td>ROTATEFILES</td>
<td>Value of the ROTATEFILES attribute (ON or OFF).</td>
</tr>
<tr>
<td>MAXFILE</td>
<td>Current maximum number of log files that the collector can create in this logging subvolume.</td>
</tr>
<tr>
<td>REFRESH</td>
<td>Value of the REFRESH attribute (ON or OFF); see EMSCCTRL—Control Collector Utility on page 13-9.</td>
</tr>
<tr>
<td>BUFFERED</td>
<td>Value of the BUFFERED attribute (ON or OFF); see EMSCCTRL—Control Collector Utility on page 13-9.</td>
</tr>
<tr>
<td>SECURITY</td>
<td>Value of the SECURITY attribute; for security information, see the FUP SECURE command in the File Utility Program (FUP) Reference Manual.</td>
</tr>
<tr>
<td>BLOCKING</td>
<td>Value of the BLOCKING attribute (ON or OFF); see EMSCCTRL—Control Collector Utility on page 13-9.</td>
</tr>
<tr>
<td>Events Received</td>
<td>Number of events the collector has received since the last system load.</td>
</tr>
<tr>
<td>Events Logged</td>
<td>Number of event messages that the collector has logged to the disk log files since the last system load.</td>
</tr>
<tr>
<td>Events Discarded</td>
<td>Total number of event messages that the collector has discarded because of event-message flooding or format errors.</td>
</tr>
<tr>
<td>OPENS Received</td>
<td>Number of times (since the last system load) that the collector has been opened to receive command messages—that is, calls to the OPEN procedure with collector and subdevice $0.#ZSPI (in the case of a primary collector).</td>
</tr>
<tr>
<td>Closes Received</td>
<td>Number of times since the last system load that the collector and subdevice $0.#ZSPI (for the primary collector) has been closed by calls to the CLOSE procedure.</td>
</tr>
<tr>
<td>FILE Switches</td>
<td>Total number of times (since the last system load) that the collector has switched log files. This includes both requested switches and switches triggered by a log file becoming full.</td>
</tr>
</tbody>
</table>
## Definition of Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unrcv Disk Errors</td>
<td>Total number of unrecoverable disk errors returned to the collector by the disk process since the last system load. Operational errors are excluded from this total. These are operational errors: 010 MAXFILE files already exist 043 Unable to obtain disc space for file extent 044 Disc directory or DCT is full</td>
</tr>
<tr>
<td>Invalid Events</td>
<td>Total number of event messages that the collector discarded because of event-message format errors.</td>
</tr>
<tr>
<td>SUPPRESS</td>
<td>Value of the SUPPRESS attribute (ON or OFF). If ON, burst detection and suppression configuration data can be displayed from EMSCINFO when the DETAIL option is specified. For a description of the BDS information that is displayed when the DETAIL option is specified, see [EMSCINFO Display Examples (With DETAIL Option)] on page 13-22.</td>
</tr>
<tr>
<td>FILTER</td>
<td>Value of the FILTER attribute (ON or OFF). If ON, the current pre-log filtration configuration data is displayed from EMSCINFO when the DETAIL option is specified. For a description of the PLF information that is displayed when the DETAIL option is specified, see [EMSCINFO Display Examples (With DETAIL Option)] on page 13-22.</td>
</tr>
<tr>
<td>Buffer Failures</td>
<td>Total number of event messages that the collector discarded because of insufficient memory-pool capacity.</td>
</tr>
<tr>
<td>Compatibility Distributor ($Z0)</td>
<td>Process name of the compatibility distributor.</td>
</tr>
<tr>
<td>Printing Mode</td>
<td>Means that $Z0 is running. Printing mode indicates whether all or only critical-only messages are displayed.</td>
</tr>
<tr>
<td>CRITICAL ONLY/ALL</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>In place of Printing Mode, means that $Z0 is not running. In this case all the lines that follow (“Primary CPU,” “Console Name,” and so forth) do not appear.</td>
</tr>
<tr>
<td>STOPPED</td>
<td></td>
</tr>
<tr>
<td>Primary CPU</td>
<td>Number of the CPU in which the primary process for the compatibility distributor is running.</td>
</tr>
<tr>
<td>Backup CPU</td>
<td>Number of the CPU in which the backup process for the compatibility distributor is running.</td>
</tr>
</tbody>
</table>
EMSCINFO Display Examples (With DETAIL Option)

Example 13-3 is an example of the EMSCINFO display for a primary collector when the DETAIL option is included in the EMSCINFO command line. It shows both the primary collector’s current BDS configuration data information and the PLF data information. In this example, SUPPRESS (BDS) is set to ON, and FILTER (pre-log filtration) is set to OFF.
Example 13-3. EMSCINFO Detail Display (SUPPRESS ON, FILTER OFF)

EMSCINFO - T9631D31 - (03APR95)
Collector = $0, Priority = 201
Primary CPU = 1, Backup CPU = 0
Disk Logging Active
Current Log File = $SYSTEM.ZLOG06.ZZEV0045
Current Record = %000005.050003
Default Log File = $SYSTEM.ZLOG06.ZZEV0045
Primary Extent = 20, Secondary Extent = 100
ROrATEOFILeS = ON, MAXFILE = 4
REFRESH = ON, BUFFERED = ON
SECURITY = NUUU, BLOCKING = ON
Events Received = 707516, Events Logged = 705697
Events Discarded = 1909
OPENS Received = 241, CLOSES Received = 233
FILE Switches = 45, Unrcv Disk Errors = 0
Invalid Events = 0, Buffer Failures = 1909

Burst Detection/Suppression Configuration Data
SUPPRESS = ON
Number of Events in a burst, N = 6
Time (secs) in which a burst is detected, T1 = 100
Time (secs) in which no events ends a burst, T2 = 120
Maximum time (secs) between burst end checks, T3 = 300
Comparison byte length of 1st subject value, L = 4
Maximum simultaneous bursts detectable, S = 6

Current event bursts = 2

-------- Event -------- -------------- Subject -----------
Events
Number SubsystemID   Token Code   Length SubsystemID
Discarded
06666  TANDEM.SPI.D30 (001,255,00100) 004  TANDEM.EMS.0  00027
SubjValue = "Eve"
06002  TANDEM.SPI.D30 (001,255,00100) 004  TANDEM.EMS.0  00016
SubjValue = "Eve"

Pre-Log Filtration Data
FILTER = OFF, Number of Filters = 0

Compatibility Distributor ($Z0)
Printing Mode = CRITICAL-ONLY
Primary CPU = 0, Backup CPU = 1
Console Name = $0
Default Console Name = $OSP
The EMSCINFO detail display example in Example 13-3 provides this information about burst detection and suppression (BDS):

- The SUPPRESS function is set to ON for the primary collector through the SUPPRESS keyword of the EMSCCTRL command (SUPPRESS ON). Therefore, burst detection and suppression is currently enabled for the primary collector.
- The current BDS configuration values and their meanings are listed.
- The current number of event bursts (two) is shown.
- The event number and subsystem ID for each reported event burst is shown.
- Information about the first subject in each burst event is provided (token code, length in bytes, subsystem ID, and subject value).
- The number of events discarded with each event burst is shown.

The EMSCINFO detail display example in Example 13-3 provides this information about pre-log filtration (PLF):

- The FILTER function is set to OFF for the primary collector through the FILTER keyword of the EMSCCTRL command (FILTER OFF). Therefore, pre-log filtration in the primary collector is currently disabled.
- No filters were specified using the FILTER keyword of the EMSCCTRL command. With no filters present in the primary collector, pre-log filtration is disabled.

Example 13-4 is another example of the EMSCINFO display for a primary collector when the DETAIL option is included in the EMSCINFO command line. It shows both the primary collector’s current BDS configuration data information and the PLF data information where both SUPPRESS and FILTER are set to ON. However, no burst filter is present.
Example 13-4. EMSCINFO Detail Display (SUPPRESS ON, FILTER ON)

EMSCINFO - T9631D31 - (03APR95)
Collector = $0, Priority = 201
Primary CPU = 1, Backup CPU = 0
Disk Logging Active
Current Log File = $SYSTEM.ZLOG06.ZZEV0045
Current Record = $000005.050003
Default Log File = $SYSTEM.ZLOG06.ZZEV0045
Primary Extent = 20, Secondary Extent = 100
ROTATEFILES = ON, MAXFILE = 4
REFRESH = ON, BUFFERED = ON
SECURITY = NUUU, BLOCKING = ON
Events Received = 707516, Events Logged = 705697
Events Discarded = 1909
OPENS Received = 241, CLOSES Received = 233
FILE Switches = 45, Unrcv Disk Errors = 0
Invalid Events = 0, Buffer Failures = 1909

Burst Detection/Suppression Configuration Data
SUPPRESS = ON
Number of Events in a burst, N = 6
Time (secs) in which a burst is detected, T1 = 100
Time (secs) in which no events ends a burst, T2 = 120
Maximum time (secs) between burst end checks, T3 = 300
Comparison byte length of 1st subject value, L = 4
Maximum simultaneous bursts detectable, S = 6

Current event bursts = 2

-------- Event -------- ------------------ Subject ------------------
Events
Number SubsystemID Token Code Length SubsystemID Discarded
06666 TANDEM.SPI.D30 (001,255,00100) 004 TANDEM.EMS.0 00027
SubjValue = "Eve"
06002 TANDEM.SPI.D30 (001,255,00100) 004 TANDEM.EMS.0 00016
SubjValue = "Eve"

Pre-Log Filtration Data
FILTER = ON, Number of Filters = 2
Filter File Name = \DSMDEV.$DATA1.COMPFILT1
Filter Type = COMPILED
Filter File Name = \DSMDEV.$DATA2.JBZEMF.FTABPASS
Filter Type = TABLE, PASS

Compatibility Distributor ($Z0)
Printing Mode = CRITICAL-ONLY
Primary CPU = 0, Backup CPU = 1
Console Name = $0
Default Console Name = $OSP
The EMSCINFO detail display example in Example 13-4 provides this information about burst detection and suppression (BDS):

- The SUPPRESS function is set to ON for the primary collector through the SUPPRESS keyword of the EMSCCTRL command (SUPPRESS ON). Therefore, burst detection and suppression for the primary collector was enabled using the SUPPRESS ON command.
- The last-recorded BDS configuration values and their meanings are listed.
- The current number of event bursts (two) is shown.
- The event number and subsystem ID for each reported event burst is shown.
- Information about the first subject in each burst event is provided (token code, length in bytes, subsystem ID, and subject value).
- The number of events discarded with each event burst is shown.

The EMSCINFO detail display example in Example 13-4 provides this information about pre-log filtration (PLF):

- The FILTER function is set to ON for the primary collector using the FILTER keyword of the EMSCCTRL command (FILTER ON). Therefore, pre-log filtration in the primary collector is currently enabled.
- The number of filters present is 2.
- The first filter’s file name and filter type (COMPILED) are shown.
- The second filter’s file name and filter type (TABLE, PASS) are shown.

Example 13-5 is another example of the EMSCINFO display for a primary collector when the DETAIL option is included in the EMSCINFO command line. It shows both the primary collector’s current BDS configuration data information and the PLF data information. In this example, SUPPRESS is set to OFF and FILTER is set to ON. In this example, a burst filter is present.
Example 13-5. EMSCINFO Detail Display (SUPPRESS OFF, FILTER ON)

EMSCINFO - T9631D31 - (03APR95)
Collector = $0, Priority = 201
Primary CPU = 1, Backup CPU = 0
Disk Logging Active
Current Log File = $SYSTEM.ZLOG06.ZZEV0045
Current Record = %000005.050003
Default Log File = $SYSTEM.ZLOG06.ZZEV0045
Primary Extent = 20, Secondary Extent = 100
ROTATEFILES = ON, MAXFILE = 4
REFRESH = ON, BUFFERED = ON
SECURITY = NUUU, BLOCKING = ON
Events Received = 707516, Events Logged = 705697
Events Discarded = 1909
OPENS Received = 241, CLOSES Received = 233
FILE Switches = 45, Unrcv Disk Errors = 0
Invalid Events = 0, Buffer Failures = 1909

Burst Detection/Suppression Configuration Data
SUPPRESS = OFF
Pre-Log Filtration Data
FILTER = ON, Number of Filters = 3
Filter File Name = DSMDEV.$DATA1.COMPFILT1
Filter Type = COMPILED
Filter File Name = DSMDEV.$DATA2.JBZEMF.FTABPASS
Filter Type = TABLE, PASS
Filter File Name = DSMDEV.$DATA3.BURSTFIL
Filter Type = SUPPRESS

Number of Events in a burst, N = 100
Time (secs) in which a burst is detected, T1 = 450
Time (secs) in which no events ends a burst, T2 = 45
Maximum time (secs) between burst end checks, T3 = 45
Comparison byte length of 1st subject value, = ON
Maximum simultaneous bursts detectable, S = 8

Current events bursts = 1

-------- Event -------- ----------------- Subject -----------------
Events
Number SubsystemID Token Code Length SubsystemID
Discarded
06666 TANDEM.SPI.D30 (000,000,00000) 000 0.0.0 00010

Compatibility Distributor ($Z0)
Printing Mode = CRITICAL-ONLY
Primary CPU = 0, Backup CPU = 1
Console Name = $0
Default Console Name = $OSP
The EMSCINFO detail display example in Example 13-5 provides this information about burst detection and suppression (BDS):

- The SUPPRESS function is set to OFF for the primary collector using the SUPPRESS keyword of the EMSCCTRL command (SUPPRESS OFF). Therefore, burst detection and suppression for the primary collector has not been enabled using the SUPPRESS ON command.
- The collector’s last-recorded BDS configuration values and their meanings are not listed because a burst filter is installed. The BDS configuration values are displayed by the burst filter name.
- The event number and subsystem ID information (shown in Example 13-3 and Example 13-4, where SUPPRESS = ON) are not shown.
- Information about the first subject in each burst event (shown in Example 13-3 and Example 13-4, where SUPPRESS = ON) is not shown.
- The number of events discarded with each event burst is not shown.

The EMSCINFO detail display example in Example 13-5 provides this information about pre-log filtration (PLF):

- The FILTER function is set to ON for the primary collector using the FILTER keyword of the EMSCCTRL command (FILTER ON). Therefore, pre-log filtration in the primary collector is currently enabled.
- The number of filters present is 3.
- The first filter’s file name and filter type (COMPiled) are shown.
- The second filter’s file name and filter type (TABLE, PASS) are shown.
- The third filter’s file name and filter type (SUPPRESS) are shown. Because the third filter is a burst filter, burst detection and suppression for the primary collector is enabled (through the EMSCCTRL command’s FILTER keyword, not EMSCCTRL SUPPRESS ON). SUPPRESS cannot be set to ON when a burst filter is specified in the collector because the burst filter is already performing the BDS function.
- The current BDS configuration values and their meanings are listed. The meaning for L is “Ignore subject tokencode and value,” and its value is “ON.” Therefore, the L parameter in the burst filter (the comparison byte length of the first subject value in the token) was set at -1, which is permitted.
- The current number of event bursts detected by the PLF burst filter (1) is shown.
- The event number and subsystem ID for the single event burst are shown.
- Information about the first subject in the one detected event burst is provided (token code, length in bytes, subsystem ID, subject value). However, because the burst filter’s L parameter was set to -1, these values are reported as zeros.
- The number of events discarded with the event burst (10) is shown.
# EMSDINFO—Distributor Information Utility

The EMSDINFO utility returns status information about a specified distributor. Example 13-6 shows the format used by EMSDINFO.

```
EMSDINFO distributor-name
```

## Example 13-6. EMSDINFO Display Format

```
EMSDINFO

EM - T9632D21 - (08SEP95)  Distributor Info/Status
COPYRIGHT TANDEM COMPUTERS INCORPORATED 1991

VERSION                   T9632D31 - 15JUL95
NAME                      $KMZD
TYPE                      PRINTING

PRIMARY CPU               6
BACKUP CPU                NOT CONFIGURED
PRIORITY                  140
ANCESTOR                  $SD13

FILTERS
....... NAME              PASS^EMS
....... OBJECT FILE       $NSM.FILTER.PASSEMS
....... TYPE              COMPILED
....... SIZE              76

EVENT PROCESSING          ACTIVATED
EOFDELAY IN SEC/100       50
SEQUENTIAL BLOCKING       ON

CURRENT POSITION          September 8, 1995 16:05:10
LAST POSITION SET         September 8, 1995 16:05:10

SOURCE COLLECTORS
....... NAME               $0.#ZSPI
....... STATE              FETCHING EVENT
....... IO STATUS          EVENT WAIT STATUS (COLLECTOR)
....... EVENT LOGGING      IN PROGRESS
....... COLLECTOR LOG      \COMM.$SYSTEM.ZLOG02.ZZEV0001
....... CURRENT LOG        \COMM.$SYSTEM.ZLOG02.ZZEV0001
....... RECORD ADDRESS     BLOCK 166   RECORD 28
....... EVENTS FILTERED    1
....... EVENTS TOTAL       1

TEXTOUT DESTINATIONS
....... NAME               $T1.#S1
....... TYPE              TERMINAL
....... RECLEN             80
....... STATE              IDLE
....... EVENTS FILTERED    1
....... EVENTS TOTAL       1
....... FILTER CHANGES     0
```
To learn about command messages that provide distributor status information, see Distributor Command Descriptions on page 17-12.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description (page 1 of 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION</td>
<td>HP version number with the release date.</td>
</tr>
<tr>
<td>NAME</td>
<td>Name of the distributor process.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Type of distributor: CONSUMER, FORWARDING, or PRINTING</td>
</tr>
<tr>
<td>PRIMARY CPU</td>
<td>Number of the CPU in which the primary process of the distributor is currently running.</td>
</tr>
<tr>
<td>BACKUP CPU</td>
<td>Number of the CPU in which the backup process of the distributor is currently running; otherwise, NOT CONFIGURED is displayed.</td>
</tr>
<tr>
<td>ANCESTER</td>
<td>Name of the process that started the distributor.</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>Current execution priority of the distributor.</td>
</tr>
<tr>
<td>FILTERS</td>
<td>Occurs for each filter that is present in the distributor and is accompanied by this information:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>Name of a filter, as declared in the filter specification. If no filter is installed, the display depends on distributor type.</td>
</tr>
<tr>
<td></td>
<td>- For consumer and printing distributors, *** DEFAULT: PASS-ALL is displayed, which indicates that all event messages pass the default filter.</td>
</tr>
<tr>
<td></td>
<td>- For forwarding distributors, *** DEFAULT: PASS-NONE is displayed, which indicates that the distributor will forward no event messages until a filter is specified.</td>
</tr>
<tr>
<td>OBJECT FILE</td>
<td>Name of the object filter file if the filter is a compiled filter and the name of an edit file if the filter is a filter table or burst filter.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Filter type. The filter type can be any one of: COMPILED, COMPILED/PARAMETERS, TABLE/PASS, TABLE/FAIL, TABLE/PASS EXTENDED, TABLE/FAILED EXTENDED, or SUPPRESS.</td>
</tr>
<tr>
<td></td>
<td>An extended filter table contains a column for pass values and/or event header token directives. COMPILED/PARAMETERS indicates that parameter values have been submitted for the compiled filter. SUPPRESS identifies a burst filter.</td>
</tr>
<tr>
<td>SIZE</td>
<td>Size of the filter in bytes.</td>
</tr>
<tr>
<td>TARGET COLLECTOR</td>
<td>Occurs only if this is a forwarding distributor and is accompanied by these terms:</td>
</tr>
<tr>
<td>NAME</td>
<td>Name of a collector process that will receive forwarded event messages.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STATE</td>
<td>One of these collector states: DISCONNECTED, WRITE PENDING, IDLE, or RETRYING ON ERROR.</td>
</tr>
<tr>
<td>LOGFILE</td>
<td>If present, name of the log file that is currently the sole source of event messages examined by this distributor.</td>
</tr>
<tr>
<td>EOFDELAY</td>
<td>Amount of time delay (in seconds/100) after detection of end of file (EOF) before reading the log again. If an EOF is received again, the message is sent to the collector to ask for the next event. Having a delay after end of file (EOF) detection and before reading the log again improves distributor efficiency. This delay reduces interprocess communication with the collector, and results in lower CPU usage. This delay significantly decreases log file access when combined with sequential block buffering at medium to high event rates (5 or more events per second). At lower event rates, the delay improves efficiency when used without sequential buffering. The delay should be at least equal to, and preferably larger than, the average event interval. The default is 500 ms. To select the delay feature, include the parameter DELAY sec/100 in the distributor command line, or use the SPI CONTROL command with the token ZEMS^TKN^SET^EOFDELAY. You can choose a delay value from zero (0) through 9999; selecting a delay of zero turns the delay function off. If a delay is not specified in the command line, the distributor selects 500 ms.</td>
</tr>
<tr>
<td>SEQUENTIAL BLOCKING</td>
<td>Indicates if sequential block buffering has been selected (ON) or deselected (OFF) for all log files except the current log file associated with a collector.</td>
</tr>
<tr>
<td>CURRENT POSITION</td>
<td>Log time (in local civil time) of the event message last examined by the distributor.</td>
</tr>
<tr>
<td>LAST POSITION SET</td>
<td>Log time (in local civil time) set by the last positioning command or by the TIME option on distributor startup. This line specifies the first event message with a log time stamp at or after the given time. The line occurs only if such a log time was given. (You must use the distributor control command to give positioning commands after distributor startup; see Section 15, EMS Procedures.)</td>
</tr>
<tr>
<td>SOURCE COLLECTORS</td>
<td>Precedes the description of zero or more source collectors. These are the collectors whose log files serve as the source of event messages for the distributor to examine. The same sequence of terms occurs once for each source collector:</td>
</tr>
<tr>
<td>NAME</td>
<td>The process name of a source collector.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STATE</td>
<td>One of these distributor states, which are associated with the named source collector: UNUSED, IDLE, FETCHING EVENTS, PERFORMING STATUS, POSITIONING, MAKING LOGNAME LIST, or DELAYING AFTER EOF.</td>
</tr>
<tr>
<td>IO STATUS</td>
<td>One of these distributor I/O status alternatives, which are associated with the named source collector: NO IO, READING LOG, NORMAL STATUS (COLLECTOR), or EVENT WAIT STATUS (COLLECTOR)</td>
</tr>
<tr>
<td>EVENT LOGGING</td>
<td>Either IN PROGRESS or STOPPED which mean, respectively, that logging is proceeding normally or that logging is stopped because of a problem.</td>
</tr>
<tr>
<td>COLLECTOR LOG</td>
<td>Name of a collector log file; the collector is currently writing event messages to this file.</td>
</tr>
<tr>
<td>CURRENT LOG</td>
<td>Name of the log file containing the last event message examined by the distributor.</td>
</tr>
<tr>
<td>RECORD ADDRESS</td>
<td>Record address of the last event message examined by the distributor and consists of:</td>
</tr>
<tr>
<td>BLOCK</td>
<td>Current block number.</td>
</tr>
<tr>
<td>RECORD</td>
<td>Record number within the block.</td>
</tr>
<tr>
<td>EVENTS FILTERED</td>
<td>Number of event messages from this collector that have passed the filter.</td>
</tr>
<tr>
<td>EVENTS TOTAL</td>
<td>Number of event messages received from this collector.</td>
</tr>
</tbody>
</table>

**Note.** Even if a filter passes all event messages, EVENTS FILTERED can be less than EVENTS TOTAL because the filter always ignores certain event messages that are intended only for the compatibility distributor.

<table>
<thead>
<tr>
<th>TEXTOUT DESTINATIONS</th>
<th>Occurs only if this is a printing distributor. The same sequence of terms occurs once for each TEXTOUT destination:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Name of the TEXTOUT file.</td>
</tr>
<tr>
<td>TYPE</td>
<td>One of: TERMINAL, PROCESS, DISKFILE, PRINTER, or a device number.</td>
</tr>
<tr>
<td>RECLEN</td>
<td>Record length of the TEXTOUT destination in bytes.</td>
</tr>
<tr>
<td>STATE</td>
<td>One of:</td>
</tr>
<tr>
<td>REMOVED</td>
<td>(after error)</td>
</tr>
<tr>
<td>IDLE</td>
<td></td>
</tr>
<tr>
<td>WAITING</td>
<td>(on error)</td>
</tr>
<tr>
<td>WAITING</td>
<td>(timed out)</td>
</tr>
<tr>
<td>PRINTING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LAST ERROR</td>
<td>Distributor error number that was last returned to a process</td>
</tr>
<tr>
<td></td>
<td>communicating with the distributor programatically. This line occurs only</td>
</tr>
<tr>
<td></td>
<td>if such an error occurred.</td>
</tr>
<tr>
<td>EVENTS FILTERED</td>
<td>Number of event messages from all sources that have passed the filter</td>
</tr>
<tr>
<td></td>
<td>since the last filter load.</td>
</tr>
<tr>
<td>EVENTS TOTAL</td>
<td>Total number of event messages from all sources that have</td>
</tr>
<tr>
<td></td>
<td>been processed by the filter since the last filter load.</td>
</tr>
<tr>
<td>FILTER CHANGES</td>
<td>Number of times that the filter has been changed since this</td>
</tr>
<tr>
<td></td>
<td>distributor process has been started.</td>
</tr>
</tbody>
</table>
EMSDIST—Distributor Program

The EMSDIST program is the object program for a printing, forwarding, or consumer distributor, any of which you can start with a TACL RUN command.

To access log-file messages, the person or process that runs EMSDIST must have read access to the log files that EMSDIST accesses. Super-group privileges are required if the collector creates its log files with the protection string COOO, which is the system default. For more information, see the SECURITY collector-option in EMSCTRL—Control Collector Utility on page 13-9, or the ZCOL-PROTECTION field in CONTROL Command (ZEMS-CMD-CONTROL) on page 17-17.

If you use a NEWPROCESS procedure call to start EMSDIST, use the minimum number of distributor options on the NEWPROCESS call, because startup error messages are not designed for programmatic access. For example, you can start a consumer distributor and supply only NAME or NAME process-name on the NEWPROCESS call. You can then use command messages to provide the additional information that the distributor requires.

```
EMSDIST [ / run-options / ] dist-options
```

If you enter EMSDIST with no parameters, the EMSDIST program displays a help screen with the EMSDIST syntax.

Critical events are marked with an asterisk (*) character that is placed two columns before the subsystem ID in the output of a printing distributor.

**run-options**

are any of the run options valid for the RUN command in TACL, separated by commas, as described in the TACL Reference Manual. These options are of particular importance for this command:

**NAME [ process-name ]**

gives the name of the distributor process. You must supply the NAME [process-name] information.

**NOWAIT**

returns a TACL prompt to you while the distributor continues to run.

**CPU cpu**

gives the number of the processor where the primary process of the distributor is to execute.

**OUT filename**

specifies the file name to send event messages to. You must use this option, or distributor error messages are not sent anywhere. If no file name is specified,
error messages are sent to the home terminal. If a DEFINE is not specified, all
distributor generated events except ZEMS^EVT^BURST^START and
ZEMS^EVT^BURST^END are written to $0. The burst start and burst end
messages are written to the distributor’s OUT file, as specified in the EMSDIST
run option, OUT. If the EMSDIST run option, OUT, is specified but does not
match the hometerm, burst start and burst end messages are written to the
OUT file as a text error list. If the EMSDIST run option, OUT, is specified
without a value, burst start and burst end messages are written to the
hometerm of the distributor as a text error list. If the value of the EMSDIST run
options OUT and TERM are the same, and neither the =_ems_system_log or
=_ems_alternate_log defines exist, burst start and burst end messages are not
written.

PRI priority
gives the execution priority of the primary distributor process.

dist-options
is any one of these distributor options, separated by commas:

BACKUP  cpu
gives the number of the processor in which the backup process of a distributor
is to execute. BACKUP * directs the distributor to search for an available CPU
and use it. The search begins with the CPU number next higher than that of
the primary CPU.

<table>
<thead>
<tr>
<th>TYPE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSUMER</td>
<td>C</td>
</tr>
<tr>
<td>FORWARDING</td>
<td>F</td>
</tr>
<tr>
<td>PRINTING</td>
<td>P</td>
</tr>
</tbody>
</table>

permanently determines the type of distributor to be started. This parameter is
required.

COLLECTOR  { name
            { ( name [ , name ] ... ) }
gives the name (or a list of names) of collectors whose log files serve as the
source of event messages that the distributor examines. The parentheses are
required if you include a list of names.
The COLLECTOR option excludes use of the LOGFILE option.

name can be the name:

$0
\system-name.$0
alternate-collector-name
\system-name.alternate-collector-name.
You can include a maximum of ten COLLECTOR names.
As the distributor examines event messages, the event messages from the log files of every collector name in the COLLECTOR option are merged by event-message generation time.

**LOGFILE name**

is the name of a log file to serve as the unique source of event messages for this distributor. The LOGFILE option excludes use of the COLLECTOR option.

**FILTER**

\{ \text{file-name} \}
\{ ( \text{file-name} \ [ , \text{file-name} ] ... ) \}

is the name or list of names of the filters to be installed in the distributor. Multiple filters are executed in sequential order for each event. A filter can be a compiled filter, a filter table, or a burst filter. A maximum of ten filters, including one burst filter, is allowed. An EDIT file is accepted for a filter table or a burst filter. When a burst filter is specified, BDS is activated when the distributor starts. If you omit the FILTER parameter when starting up a printing or consumer distributor, a default filter is used. For more information about default filters, see Section 17, Distributor Commands and Responses.

If you omit the FILTER option, distributor operation depends on the type of distributor:

- Consumer or printing distributors pass all event messages.
- Forwarding distributors wait for a filter to be loaded.

For a description of the CONTROL command message that loads, changes, and resets filters, see Distributor Command Descriptions on page 17-12.

**TARGET collector-name**

(for a forwarding distributor only) gives the name of the primary or alternate collector to which event messages are forwarded.

**TEXTOUT**

\{ \text{name} \}
\{ ( \text{name} \ [ , \text{name} ] ... ) \}

(for a printing distributor only) gives the name (or a list of names) of devices, processes, and disk files that receive display-formatted event messages. The parentheses in the list alternative are required.

You can include a maximum of ten TEXTOUT names.

A printing distributor can be configured to determine its home terminal and use it as a TEXTOUT device by specifying TEXTOUT $home. This allows configuration of a dynamic telnet session that automatically starts up a printing distributor that determines its own destination.

If the specified filename does not exist, the printing distributor creates an entry-sequenced file with a record length of 80, a default extent of 20 pages, and a
maxextent of 16 pages. A page contains 2048 bytes. Therefore, the created file can only store 655,530 bytes, which can fill up quickly.

**WAIT n**

overrides the two minute default timeout for all textout destinations specified to a printing distributor. \( n \) is given in seconds. If WAIT is specified, the distributor retries every \( n \) seconds, even if there is only one destination. If the textout is a routing destination, the event is skipped. If a file system error occurs, and WAIT is specified, the distributor does not retry and skips to the next line. This situation can occur with certain unprintable characters. If WAIT is not specified, the error is retried indefinitely.

**TIME \{ date \} \{ [ date ] time \}**

specifies the generation time of the first event message to be examined by the distributor. This is a positioning parameter; only event messages with a generation time that is greater than or equal to \( date-time \) are examined.

If you give only the \( date \) part of the TIME option, 0:00:00 is the default \( time \) value. If you give only the \( time \) part of the TIME option, the current date is the default \( date \) value.

If you omit the TIME option, the default value depends on whether you specified a COLLECTOR or a LOGFILE option:

- If you used a COLLECTOR option, the distributor is positioned to the event message next written to the collector log file.
- If you used a LOGFILE option, the distributor is positioned to the first event message in the log file.

**date**

The \( date \) parameter can have the forms:

\[ year - month# - day \]
\[ month day year \]
\[ day month year \]

\( year \) is a four-digit integer
\( month \) is a three-letter abbreviation for the month (JAN, FEB, and so on)
\( month# \) is the number of the month, one or two digits
\( day \) is a one-digit or two-digit integer

You can separate the parts of the first form of a date with hyphens, and with spaces for the other two forms.

For example, you can specify a date as 1999-12-31, DEC 31 1999, or 31 DEC 1999.
time

The time parameter has the form:

\[ \text{hour : minute [ : second ]} \]

- hour is a one-digit or two-digit integer
- minute is a one-digit or two-digit integer
- second is a two-digit integer

For example, you can specify the time of day as 11:05:59 or 23:05.

The TIME option specifies values in local civil time.

STOP \{ date \}  
\{ [ date ] time | EOF \}

is a positioning parameter used with the TIME option to specify a range of values to be examined by the distributor. The TIME option specifies the start date and time. The STOP option specifies the ending date and time. The distributor automatically stops when it reaches the end of the range.

date

The date parameter can have the forms:

\[ \text{year - month# - day} \]
\[ \text{month day year} \]
\[ \text{day month year} \]

- year is a four-digit integer
- month is a three-letter abbreviation for the month, (JAN, FEB, and so on)
- month# is the number of the month, one or two digits
- day is a one-digit or two-digit integer

You can separate the parts of the first form of a date with hyphens, and with spaces for the other two forms.

For example, you can specify a date as 1999-12-31, DEC 31 1999, or 31 DEC 1999.

time

The time parameter has the form:

\[ \text{hour : minute [ : second ]} \]

- hour is a one-digit or two-digit integer
- minute is a one-digit or two-digit integer
- second is a two-digit integer
For example, you can specify the time of day as 11:05:59 or 23:05.

If the TIME option is omitted and the STOP time is smaller than the most current event, the distributor terminates.

The STOP feature is available only as an option for the EMSDIST program, and not as a programmatic option. It is available exclusively for the printing and forwarding distributors, and not for the consumer distributor.

To terminate a printing or forwarding distributor in collector mode after an EOF is encountered, specify STOP EOF instead of STOP time. This assumes that a TIME option was also given. To terminate a printing or forwarding distributor if either a specified time or an EOF is encountered, specify both STOP time and STOP EOF. In this case, STOP time must be specified first.

DELAY \{ time \}

delays by time after detection of end of file (EOF) before reading the log again. If an EOF is received again, the message is sent to the collector to ask for the next event. time is given in seconds/100.

A delay after end of file (EOF) detection and before reading the log again improves distributor efficiency. This delay reduces interprocess communication with the collector, and results in lower CPU usage. This delay significantly decreases log file access when combined with sequential block buffering at medium to high event rates (5 or more events per second). At lower event rates, the delay improves efficiency when used without sequential buffering. The delay should be at least equal to, and preferably larger than, the average event interval. The default is 500 ms.

To select the delay feature, include the parameter DELAY sec/100 in the distributor command line, or use the SPI CONTROL command with the token ZEMS^TKN^SET^EOFDELAY. You can choose a delay value from zero (0) through 9999; selecting a delay of zero turns the delay function off. If a delay is not specified in the command line, the distributor selects 500 ms.

SBUF \{ ON | OFF \}

selects or deselects sequential block buffering for all log files except the current log file associated with a collector. The default is ON.

AUTOSTOP \{ time \}

specifies how long a consumer distributor stays alive after the last opener has closed it. time is given in seconds.

If time is omitted or set to zero, the distributor terminates when it receives a close message from the last opener. When set to any value greater than zero, termination is delayed by that value (maximum is 9999 seconds). If it is set to -1, the delay is assumed to be infinite. When the distributor is opened again by any process before the timeout occurs, the timeout is canceled.
The distributor resends the last event that the consumer has received; that is, it positions by the context that it recorded last.

**INDENT** $n$

overrides the default indentation (36) for a printing distributor. $n$ is the number of indentation spaces. $n$ must not be larger than the record size of the destination.

**GMT ON**

converts the local time back to GMT (Greenwich mean time) for printing distributors. Time is displayed in GMT instead of LCT (local civil time).

**DUMP ON**

causes the printing distributor output to be in a labeled token dump format instead of EMSTEXT format. Numerical values are displayed in hexadecimal.

## Startup Error Messages

If a distributor process encounters errors while it is starting up, the distributor generates one or more of the startup error messages listed here or in Section 21, Distributor Errors. (Errors in Section 21, Distributor Errors, can occur at other times as well.) Whether you use a RUN command or a NEWPROCESS procedure call for startup, the distributor displays any startup error messages at the appropriate terminal, or places them in an OUT file.

Each startup error message, unless specifically designated as a warning, causes the distributor to terminate execution immediately.

**AUTOSTOP only allowed for consumer distributor**

**Cause.** The AUTOSTOP option was called for a printing or forwarding distributor, but is only allowed for consumer distributors.

**Effect.** The distributor fails to start up.

**Recovery.** Restart without AUTOSTOP called for any nonconsumer distributors.

**Can’t open target collector** collector-name, error error-number

{ Aborting }
{ Continuing... distributor will retry }

**Cause.** The targeted collector could not be opened by the distributor due to the specified error.

**Effect.** If the specified error is FENOSUCHSYS or FENETDOWN, the distributor tries to open the collector again. Otherwise, the distributor fails to start up.
Recovery. If the distributor fails to start up, fix the specified problem with the collector.

| Can’t open textout textout-name, error error-number |

Cause. The targeted textout destination cannot be opened by the distributor.
Effect. The distributor fails to start up.
Recovery. Fix the problem with the specified textout.

| Comma missing |

Cause. A comma required by syntax has been omitted.
Effect. The distributor fails to startup.
Recovery. Check your command syntax, and that all required commas are present.

Date/time format error
{ Expecting: [H]H:MM[:SS] }
{ Expecting: MMM [D]D YYYY [[H]H:MM[:SS]] }
{ Expecting: [D]D MMM YYYY [[H]H:MM[:SS]] }

Cause. The time or date is not formatted properly.
Effect. The distributor fails to start up.
Recovery. Check that the time and date are formatted correctly. Do not use either brackets or braces in the date or time.

| DESTINATION allocation problem. Filter file: file-name |

Cause. The specified destination for a routing distributor cannot be accessed.
Effect. The distributor fails to start up.
Recovery. Check the specified destination and filter file.

| DUMP/WAIT/INDENT/GMT only allowed for printing distributor |

Cause. One of the stated options was specified for a forwarding or consumer distributor, which is not allowed.
Effect. The distributor fails to start up.
Recovery. Only use DUMP, WAIT, INDENT, or GMT with a printing distributor.

| Duplicate burst filter: filter-name |

Cause. More than one burst filter is specified, which is not allowed.
**Impact.** The distributor fails to start up.

**Recovery.** Specify only one burst filter.

**Cause.** Two filters with the same name are specified.

**Effect.** The distributor fails to start up.

**Recovery.** Check that each filter has a unique name.

**Cause.** The same key word has been detected twice.

**Effect.** The distributor fails to start up.

**Recovery.** Eliminate the duplicate keyword.

**Cause.** Two textout destinations with the same name are specified.

**Effect.** The distributor fails to start up.

**Recovery.** Check that each textout has a unique name.

**Cause.** A clause has been left empty that requires specific input.

**Effect.** The distributor fails to start up.

**Recovery.** Check the syntax of your command, making sure that you have filled all clauses as required.

**Cause.** The distributor is not compatible with the operating system you are running.

**Effect.** The distributor fails to start up.

**Recovery.** Use a recent version of the distributor that is compatible with your operating system.

**Cause.** The EOF attribute is specified for TIME, but is only allowed to be specified for the STOP option.

**Effect.** The distributor fails to start up.
Recovery. Specify STOP EOF.

| Filterfile problem. Error: error-number, File: file-name |

Cause. The specified filterfile cannot be accessed.

Effect. The distributor fails to start up.

Recovery. Check the specified filterfile error.

| Filterfile read error: error-number, File: file-name |

Cause. The specified filterfile cannot be read.

Effect. The distributor fails to start up.

Recovery. Check the specified filterfile error.

| Filter allocation problem. File: file-name |

Cause. Resources for the specified filter cannot be allocated.

Effect. The distributor fails to start up.

Recovery. Reduce the number of filters.

| Filter format invalid. Error: error-number, File: file-name |

Cause. The format of the specified filter is invalid. The error and file are specified in the message.

Effect. The distributor fails to start up.

Recovery. Check that the filter is valid and correctly referenced in your command.

| Filter requires parameters; must use SPI |

Cause. The specified filter requires SPI parameters that were not provided.

Effect. The distributor fails to start up.

Recovery. Filters that require SPI parameters cannot be specified in the RUN command. Filters of this type can only be added using a SPI command.

| Filter table conversion error: error-number, File: file-name |
| { Detail error: detail-error-number } |
| { Line# in table: line-number } |
| { Column in table: column-number } |

Cause. The specified filter table source file cannot be converted because of the indicated error.
Effect. The distributor fails to start up.

Recovery. Correct the specified syntax or range error.

Filter version mismatch

Cause. The distributor does not support the version of the specified file.

Effect. The distributor fails to start up.

Recovery. Recompile the filter or use a distributor of a matching version.

Incompatible event destination for this type of distributor

Cause. The destination specified is not compatible with this type of distributor.

Effect. The distributor fails to start up.

Recovery. Check that the specified destination and distributor type are compatible. For example, TEXTOUT can only be specified for a printing distributor. TARGET can only be specified for a forwarding distributor.

Invalid autostop time

Cause. An autostop value is specified outside of the valid range from -1 through 9999.

Effect. The distributor fails to start up.

Recovery. Specify a valid autostop time.

Invalid backup cpu number

Cause. A backup CPU number is specified outside of the valid range from 0 through 15.

Effect. The distributor fails to start up.

Recovery. Specify a valid backup CPU number.

Invalid collector retry time

Cause. A collector retry value is specified outside of the valid range from 1 through 9999.

Effect. The distributor fails to start up.

Recovery. Specify a valid collector retry value.

Invalid EndOfFile delay

Cause. A EOF delay value is specified outside of the valid range from 0 through 9999.
Effect. The distributor fails to start up.

Recovery. Specify a valid EOF delay value.

Invalid file name

Cause. The specified file name contains extraneous characters.

Effect. The distributor fails to start up.

Recovery. Verify that you have accurately entered the specified file name.

Invalid filterfile name

Cause. The specified filter file is either invalid or does not exist.

Effect. The distributor fails to start up.

Recovery. Verify that you have provided a valid filter file name.

Invalid indentation

Cause. An indentation value is specified outside of the valid range (0 through 9999).

Effect. The distributor fails to start up.

Recovery. Specify a valid indentation value.

Invalid logfile name

Cause. The specified log file is either invalid or does not exist.

Effect. The distributor fails to start up.

Recovery. Verify that you have provided a valid log file name.

Invalid process name

Cause. The specified process is either invalid or does not exist.

Effect. The distributor fails to start up.

Recovery. Verify that you have provided a valid process name.

Invalid source collector name

Cause. The specified source collector is either invalid or does not exist.

Effect. The distributor fails to start up.
Recovery. Verify that you have provided a valid source collector name.

Invalid system name for source collector

Cause. The specified system name for the source collector is invalid.

Effect. The distributor fails to start up.

Recovery. Verify that you have specified a valid system name for the source collector.

Invalid system name for target collector

Cause. The specified system name for the target collector is invalid.

Effect. The distributor fails to start up.

Recovery. Verify that you have specified a valid system name for the target collector.

Invalid target collector name

Cause. The specified target collector is either invalid or does not exist.

Effect. The distributor fails to start up.

Recovery. Verify that you have specified a valid target collector name.

Invalid textout filetype for file-name

Cause. The textout destination has an incorrect filetype and cannot be written to.

Effect. The distributor fails to start up.

Recovery. Specify the correct textout file.

Invalid textout name

Cause. The specified textout name is either invalid or does not exist.

Effect. The distributor fails to start up.

Recovery. Verify that you have provided a valid textout name.

Invalid textout wait time

Cause. A textout wait time is specified outside of the valid range (0 through 32767).

Effect. The distributor fails to start up.
Recovery. Specify a valid textout wait time.

Keyword not found
Select from: TYPE, BACKUP, COLLECTOR, FILTER, LOGFILE, TARGET, TEXTOUT, TIME, STOP, DELAY, SBUF, AUTOSTOP, DUMP, WAIT, INDENT, GMT, RETRY

Cause. A necessary keyword is not supported.

Effect. The distributor fails to start up.

Recovery. Specify one of the provided keywords.

LOGFILE and COLLECTOR cannot coexist

Cause. The specified LOGFILE and COLLECTOR cannot coexist, because LOGFILE and COLLECTOR are mutually exclusive keywords.

Effect. The distributor fails to start up.

Recovery. Do not specify LOGFILE and COLLECTOR at the same time.

Log problem proc-name error x on file-name
[ COLLECTOR collector-name ]

Cause. A log problem has occurred with the specified OPEN or READ procedure.

Effect. The distributor fails to start up.

Recovery. Check the log-file and determine the cause of the OPEN or READ error.

Must be named to run in NonStop mode

Cause. A backup CPU is specified, but the process is not named.

Effect. The distributor fails to start up.

Recovery. Either specify the NAME parameter in the TACL RUN statement, or do not specify a backup CPU.

Must specify distributor type: C[ONSUMER], F[ORWARDING], or P[RINTING]

Cause. A distributor type (either C[ONSUMER], F[ORWARDING], or P[RINTING]) must be specified, but was not.

Effect. The distributor fails to start up.

Recovery. Specify the distributor type.

Must specify ON or OFF
Cause. An incorrect value is specified for the SBUF, DUMP, or GMT option.

Effect. The distributor fails to start up.

Recovery. Specify ON or OFF for the necessary options.

Name or value missing

Cause. A required name or value is missing in the command.

Effect. The distributor fails to start up.

Recovery. Check that all required names and values are present in your command.

No matching parenthesis

Cause. A parenthesis needs to be cancelled by another, as in “(xyz,” or “xyz).”

Effect. The distributor fails to start up.

Recovery. Check that all parentheses are used in pairs.

None of the specified TEXTOUT can be opened

Cause. None of the TEXTOUT destinations specified in the TEXTOUT list can be opened; that is, events cannot be printed anywhere.

Effect. The distributor fails to start up.

Recovery. Check the specified TEXTOUT destinations to determine why they could not be opened. Check that they are referred to correctly in the command.

No source collector open

Cause. The distributor attempted to fetch an event, but no source collector was specified.

Effect. The distributor does not start up.

Recovery. Check that a source collector is specified.

Position time too early  log file-name
[ collector collector-name ]
Continuing...

Cause. The position time for the specified log file is too early.

Effect. The distributor displays events from the earliest log available.
Recovery. None needed.

```
Position time too late log file-name
    [ collector collector-name ]
Continuing...
```

**Cause.** The position time for the specified log file is too late.

**Effect.** The distributor waits for the next event.

**Recovery.** None needed.

```
Pre-C00 collector not supported
```

**Cause.** A collector was specified as an event source that is a pre-C00 version collector.

**Effect.** The distributor fails to start up.

**Recovery.** Use a post-C00 collector as an event source.

```
Process must be named
```

**Cause.** A D-series distributor started without a name, and the RUNNAMED flag was reset with the finder.

**Effect.** The distributor fails to start up.

**Recovery.** Set the RUNNAMED flag with the finder to make NonStop Kernel assign a processname, or specify the name parameter in the TACL RUN command.

```
RETRY only allowed in collector mode
```

**Cause.** The RETRY option is specified in log file mode, which is not allowed.

**Effect.** The distributor fails to start up.

**Recovery.** Specify a collector in the startup line.

```
STOP time not allowed for consumer distributor
```

**Cause.** The STOP parameter is specified for a consumer distributor, which is not allowed.

**Effect.** The distributor fails to start up.

**Recovery.** Specify a printing or forwarding distributor.

```
STOP time smaller/equal start time
```

**Cause.** The STOP time cannot be smaller than or equal to the TIME parameter.
**Effect.** The distributor fails to start up.

**Recovery.** Check that the STOP time is greater than the TIME parameter.

TARGET and TEXTOUT can not coexist

**Cause.** TARGET and TEXTOUT are mutually exclusive distributor options and cannot be designated together.

**Effect.** The distributor fails to start up.

**Recovery.** Designate either TARGET or TEXTOUT, but not both in the same context.

TARGET can not be same as COLLECTOR

**Cause.** An attempt was made to forward event messages to a collector that is already a source, which creates a loop.

**Effect.** The distributor fails to start up.

**Recovery.** Verify that you do not target forwarded event messages to a collector that is already a source.

TIME may not follow STOP

**Cause.** The TIME parameter follows STOP, which is not allowed.

**Effect.** The distributor fails to start up.

**Recovery.** Check that the TIME parameter precedes STOP.

Too many blank spaces in time field

**Cause.** The specified time is larger than 80 characters.

**Effect.** The distributor fails to start up.

**Recovery.** Remove blanks from the time field to reduce its length to below 80 characters.

Too many filter files

**Cause.** More than 10 filter files are specified.

**Effect.** The distributor fails to start up.

**Recovery.** Reduce the number of filter files to below 10.

Too many source collectors

**Cause.** More than the maximum allowable (10) source collectors have been specified.
Effect. The distributor fails to start up.

Recovery. Specify no more than 10 source collectors.

Effect. The distributor fails to start up.

Recovery. Specify no more than 10 TEXTOUT destinations.

Effect. The distributor fails to start up.

Recovery. Specify a valid distributor type.

Translating an EMS Event Into an SNMP Trap

When a NonStop SNMP (Simple Network Management Protocol) agent is provided as a routing destination, EMSDIST can translate each event into an SNMP trap and send the trap to the NonStop SNMP agent for routing to manager stations configured to receive them. An example of this general procedure is:

1. Install and configure the NonStop agent and the EMS trap subagent.
2. Create a compiled filter that sets up the routing ID and identifies the events to translate into SNMP traps.
3. From TACL, load any DDL definition used by the filter, and compile the filter with EMF.
4. With the NonStop agent running, start the EMSDIST program.

EMSDIST automatically translates the EMS event into an SNMP trap when the destination is an SNMP agent process.

If no trap appears on the management console and the error “SNMP BAD-IPC-PDU-RCVD events” (SMP event number 9) appears in the NonStop Kernel event log, the problem is probably due to incorrect installation of the ZSMPMTPL template file that comes with the agent.

For detailed instructions about translating an EMS event into an SNMP trap, see the SNMP Configuration and Management Manual.
This section describes EMS tokens and values associated with event-message processing, and discusses conventions for their use, focusing on the reference needs of programmers who must retrieve information from event messages:

For information about writing a program to retrieve event information through a consumer distributor, see Section 4, Retrieving Event Messages Programmatically.

For a detailed discussion of the EMSGET and EMSGETTKN procedures and the special-operation tokens that they support, see EMSGET and EMSGETTKN Procedures on page 15-12.

### Event-Message Overview

This overview of terminology and token types builds a base for the definitions of specific event-message tokens in this section.

#### Terminology

A process is said to report an event if the process generates:

- An event message through the EMS procedures and sends the message to $0 through the WRITEREAD procedure. This is the standard way to report an event.
- A text message and sends the message to the collector ($0) through the WRITE procedure. Before the release of EMS, this was the only way to report events. $0 uses the text message to generate an event message for the reporting process.

A subject of an event message is a token that represents an object that is central to the event. An event message can have more than one subject but must have at least one. For example, if a device goes down, the event message that reports the event includes as an event subject a token that represents the device. The same token might occur in another event message but not be a subject token.

For information on retrieving subject tokens from event messages, see the description of the ZEMS-TKN-SUBJECT token in Definitions of EMS Special Token Codes on page 14-8.

Like command messages, event messages are based on tokens, and many general statements apply to both message types. For a discussion of general messages based on tokens, see the SPI Programming Manual.

Each event message consists of a header and a data portion. In many cases, you can ignore whether a token is a header token or a data-portion token. You get the value of
either header or data-portion tokens by calls to the EMSGET or EMSGETTKN procedures. For information, see EMSGET and EMSGETTKN Procedures on page 15-12.

Header Tokens Overview

Event-message header tokens represent values that occur in every event message. These tokens are prefixed by either ZEMS or ZSPI.

Header tokens are shared by all subsystems. That is, header token codes have the same values in every subsystem, and these values do not overlap the values of tokens that are private to a subsystem.

Therefore when you call EMSGET or EMSGETTKN to get the value of a header token, you can omit the ssid parameter. The header token is recognized regardless of the default subsystem ID.

Similarly, when you reference a header token in a filter specification, you do not need to qualify the token name, no matter what the default subsystem ID is at the place of reference.

Header Tokens Usage Restrictions

You cannot perform most EMSGET special operations, such as the ZSPI-TKN-ADDR and ZSPI-TKN-NEXTTOKEN operations, on header tokens. The ZSPI-TKN-LEN and ZSPI-TKN-COUNT operations are the only special operations defined for header tokens.

These restrictions are because the values of header tokens are stored in the buffer without their token codes in a compact form that is private to HP.

For information about special operations, see EMSGET and EMSGETTKN Procedures on page 15-12.

Event Message Requirements and Conventions

Knowing these conventions can help you retrieve event-message information in an application or help you examine event messages with a filter:

- To retrieve information from event messages, use only EMS procedures (and certain SPI procedures). For more information, see Section 15, EMS Procedures.

- An event message can contain an error list or a series of error lists, any of which can be nested. For information on how to retrieve tokens within a list, see the SPI Programming Manual.

- Event messages cannot be nested.

- The ZEMS-TKN-ACTION-NEEDED token used with the ZEMS-TKN-ACTION-ID token can help you coordinate related events. For information on the use of these tokens, see Section 8, Reporting Events.
In an event message, the reporting subsystem can include error lists with errors it encountered itself and error lists with errors encountered by a lower-level subsystem that has a different subsystem ID.

- Error lists do not have subjects. Therefore, all event-message subjects are visible to EMSGET or EMSGETTKN without entering a list.
- Each event message has one or more subjects.
- Each event message is self-contained and does not depend on another event message. That is, each event message describes exactly one event.

**Event-Message Restrictions**

The Event Message Subsystem restricts event messages to ZEMS-VAL-EVT-BUFLEN bytes (currently 4024 bytes: 2012 words). Other subsystems that process event messages, such as ViewPoint, can impose more restrictive length limits.

**Definitions of Event-Message Tokens**

All event-message header tokens and common event-message data-portion tokens are described in this subsection.

In these definitions, the types of the token values are shown as `int`, `enum`, `timestamp`, and so forth; these types correspond to ZSPI-TYP-INT, ZSPI-TYP-ENUM, and ZSPI-TYP-TIMESTAMP, defined in the standard-definition file for SPI.

**Definitions of SPI Tokens in the Event-Message Header**

Besides EMS tokens, the event-message header contains SPI tokens (found in all command messages):

- **ZSPI-TKN-MAX-FIELD-VERSION** (type `int`) is the highest field version among the non-null fields of structures added to the buffer with EMSADDTOKENMAPS.

- **ZSPI-TKN-SSID** (type `ssid`) is the subsystem ID used when the buffer was initialized with EMSINIT; that is, the subsystem ID of the subsystem that reported the event.

- **ZSPI-TKN-USEDLEN** (type `int`) is the number of bytes actually used in the buffer. A subsystem that reports events can use this to determine how many bytes to send to $0$ through the WRITEREAD procedure.
Definitions of EMS Tokens in the Event Message Header

The remainder of the event-message header tokens are EMS tokens:

- **ZEMS-TKN-EVENTNUMBER** (type `enum`; shared) is a number that a particular subsystem assigns to an event to identify it. Event numbers represent unique events only for a particular subsystem because each subsystem can use event numbers in the same range.

  Therefore, to identify an event, you must know both the event number and the subsystem ID of the subsystem that created the event message.

- **ZEMS-TKN-GENTIME** (type `timestamp`; shared) is the time (Greenwich mean time) that the reporting subsystem created the event message. In most cases, ZEMS-TKN-GENTIME is close to the time the event occurred.

- **ZEMS-TKN-LOGTIME** (type `timestamp`; shared) is the time (Greenwich mean time) that the collector wrote the event message to its log files. Event messages are ordered in the log files based on log time, not generation time.

  ZEMS-TKN-LOGTIME changes if the event message is forwarded to another collector. No other token values change in this way.

- **ZEMS-TKN-NODENUM** (type `int2`; shared) is the EXPAND node (system) number of the node in which the event was reported.

- **ZEMS-TKN-CPU** (type `uint`; shared) is the CPU of the reporting subsystem process.

- **ZEMS-TKN-PIN** (type `uint`; shared) is the PIN of the reporting subsystem process.

- **ZEMS-TKN-PROC-DESC** (type `string`; shared) is the process descriptor of the event creator.

- **ZEMS-TKN-USERID** (type `byte-pair`; shared) is the user ID of the reporting subsystem process.

- **ZEMS-TKN-CONSOLE-PRINT** (type `Boolean`; shared) is no longer used by the compatibility distributor.
ZEMS-TKN-EMPHASIS (type Boolean; shared) is designed to convey—from the perspective of the reporting subsystem—whether an event message should be considered critical. Set this token to TRUE to emphasize the event, though its perceived degree of seriousness will depend on your operations environment and the current situation.

For a pre-EMS message that can be critical, include this token in the tokenized event message. If the original message contains the ASCII character BEL, marking it as critical, set the token to TRUE to add the BEL character to the formatted output text of the message. The console receiving the BEL will beep.
ZEMS-TKN-SUPPRESS-DISPLAY  
(type Boolean; shared) if TRUE, tells the ViewPoint application not to display the event message; ViewPoint displays it if the token is either FALSE or missing.

Use this token in messages that report status or history that is useful for analysis but normally meaningless to operators. The token is also useful for suppressing a message that reports only an action-completion event. When ViewPoint receives such a message, with this token TRUE, it also clears the associated action-attention message from the display.

An event message with the SUPPRESS-DISPLAY token set to TRUE is still formatted by the printing distributor, so the message should have a format template associated with it or still contain a text token.

ZEMS-TKN-CONTENT-STANDARD  
(type enum; shared) indicates the type of standard event. Standard enumerated values are:

- ZEMS-VAL-NULL (default)
- ZEMS-VAL-TRANSIENT-FAULT
- ZEMS-VAL-OBJECT-UNAVAILABLE
- ZEMS-VAL-OBJECT-AVAILABLE
- ZEMS-VAL-OTHER-STATE-CHANGE
- ZEMS-VAL-ATTN-NEEDED
- ZEMS-VAL-ATTN-COMPLETED
- ZEMS-VAL-USAGE-THRESHOLD

If an event is one of the events defined in this standard, this token must be present, and its value must be its event type. If an event is not one of the standard events, this token must not be present, or its value must be EMS-VAL-NULL.

This token can co-exist with another type token, ZEMS-TKN-CONTENT-USER.
ZEMS-TKN-CONTENT-USER (type enum; shared) identifies the type of a subsystem-defined event. Standard enumerated values are:

- ZEMS-VAL-NULL (default)
- ZEMS-VAL-DATA-TRACE
- ZEMS-VAL-DATA-DEBUG
- ZEMS-VAL-DATA-DIAGNOSTIC

Subsystems can add their own values but they must be greater than or equal to ZEMS-VAL-MIN-USER-VALUE.

If an event is defined by a subsystem and contains trace, program debug, or diagnostic information for problems in a production environment, this token must be present to indicate the type of information conveyed by the event. Otherwise this token should not be present, or its value must be ZEMS-VAL-NULL.

This token can co-exist with the “Standard Content Type” ZEMS-TKN-CONTENT-STANDARD token described earlier. When both type tokens are present, it indicates the event contains tokens for the standard event and for the subsystem-defined event. This usually means the event was a private event defined by the subsystem initially and then enhanced subsequently—by adding other tokens—to become a standard event.

**Note.** Be careful when you use the ZEMS-TKN-NODENUM, ZEMS-TKN-CPU, ZEMS-TKN-PIN, and ZEMS-TKN-USERID tokens. These tokens do not reflect the reporting process if event text is written to $0 through a call to the WRITE procedure. Instead, these tokens reflect $0 because $0 creates the event message from the event text.

To get information about the reporting process, pass the ZEMS-TKN-XSENDERID and XSENDERID-PD tokens to the EMSGET (or EMSGETTKN) procedure. For more information, see the descriptions of XSENDERID and XSENDERID-PD in **Definitions of EMS Special Token Codes** on page 14-8.

The header tokens ZEMS-TKN-PROC-DESC, ZEMS-TKN-CPU, ZEMS-TKN-PIN, and ZEMS-TKN-NODENUM are related. Updating any of these four tokens does not automatically update the other three tokens. You must ensure that changes to these four tokens are consistent.
Definitions of EMS Special Token Codes

These special token codes are defined for use with the EMSGET and EMSGETTKN procedures to request information.

ZEMS-TKN-SENDERID (shared), passed to EMSGET, gets the process ID, in fname32 format, of the process that reported the event. For more information, see the Guardian Programmer’s Guide.

The normal mechanism to report events is through the WRITEREAD procedure. However, you can use text messages, as well as EMS event messages, to report events. Call the WRITE procedure with the text message, as you did in releases prior to the release of EMS.

If you report an event through a WRITE call to $0, $0 creates the tokenized event message that goes in the collector log file. The creation time of the message is reported in the header token ZEMS-TKN-GENTIME. The time when the event occurred is unavailable, so any time fields associated with the process ID of the message have the value 0.

Do not call WRITE from a remote system to report an event. If you do, information in the CPU and PIN tokens is meaningless. However, a WRITEREAD with an EMS event message from a remote system is perfectly acceptable.

The EMSGET and EMSGETTKN procedures recognize two special tokens that are equivalent to the old ZEMS-TKN-SENDERID token:

ZEMS-TKN-XSENDERID contains the node number, CPU, and PIN of the sender of the event.

ZEMS-TKN-XSENDERID-PD is a variable-length string containing the process descriptor of the event sender.

The EMSGET and EMSGETTKN procedures still support the ZEMS-TKN-SENDERID token, if possible. However, it might not be possible to extract ZEMS-TKN-SENDERID data from a D-series or later event. For example, the ZEMS-TKN-SENDERID token cannot contain a D-series PIN number larger than 255.

ZEMS-TKN-SUBJECT (shared), passed to EMSGET, gets the token code and subsystem ID of any subject token in an event message. To specify which subject token, use the index parameter. The value of the subject token is not returned.

To get the value of the subject token, call EMSGET again, passing the token code, subsystem ID, and index returned by the ZEMS-TKN-SUBJECT operation.

For more information about special-operation tokens that are defined for use with event messages, see the description of the EMSGET procedure in EMSGET and EMSGETTKN Procedures on page 15-12.
Definitions of EMS Data-Portion Tokens

These data-portion tokens occur frequently in event messages:

ZEMS-TKN-SUBJECT-MARK (type mark; shared) marks the token that follows it in the event-message buffer as a subject of the event message. An event message can have several subjects but must have at least one. The token code of ZEMS-TKN-SUBJECT-MARK is shared by all subsystems. For a description of how to find event message subjects, see the description of the ZEMS-TKN-SUBJECT token in Definitions of EMS Special Token Codes on page 14-8.

ZEMS-TKN-LDEV (type uint; nonshared) is the logical device number of the subsystem process that reported the event.

ZEMS-TKN-LDEVNAME (type fname32; shared) is the logical device name of the subsystem process that reported the event. The file name is in the internal format that includes the name of the system (node).

ZEMS-TKN-LDEVNUMBER (type int2; shared) is the logical device number of the subsystem process that reported the event.

ZEMS-TKN-TEXT (type string; nonshared) is the displayable text describing the event.

ZEMS-TKN-ACTION-ID (type int; nonshared) is a number that identifies a particular action-event message, distinguishing it from all other outstanding action-event messages that have the same subject, subsystem ID, and system number. All four values—values of the subject token and of the ZEMS-TKN-ACTION-ID, ZEMS-TKN-SYSTEM, and ZSPI-TKN-SSID tokens—are necessary to specify an action event uniquely.

ZEMS-TKN-ACTION-NEEDED (type Boolean; nonshared) is a flag. The value TRUE means that this message is a request for action; the value FALSE means that this message is a notification that the action taken is now complete. Whether ACTION-NEEDED is TRUE or FALSE, the ZEMS-TKN-ACTION-ID token identifies the particular request, numbering the request uniquely for a specified subject, subsystem ID, and system number.

Note. Be careful when you retrieve ZEMS-TKN-ACTION-ID, ZEMS-TKN-ACTION-NEEDED, and other EMS tokens that are not shared with other subsystems. For example, to call EMSGETTKN with ZEMS-TKN-ACTION-ID when the default subsystem ID is not EMS, you must specify ZEMS-VAL-SSID, the subsystem ID for EMS. Similarly, to refer to ZEMS-TKN-ACTION-ID or ZEMS-TKN-ACTION-NEEDED in a filter specification from a place where the default subsystem ID is not EMS, you must qualify the token name by the subsystem ID for EMS.
SPI Data-Portion Tokens

This SPI token is designed for use with the Event Management Service:

ZEMS-MAP-EXIOADDR (D-series RVUs only) (type exioaddr; nonshared) is an extensible structured token. It can hold an extended physical address for the I/O device associated with the subject of the event message. Because current and future releases of some subsystems for the NonStop server require such addresses, this token replaces the smaller CU token. Use the EXIOADDR token, not the CU token, in all your new event messages that will carry physical I/O addresses. Further, add the EXIOADDR token to any previously defined message containing the CU token and, if the latter was designated optional, remove it from the message.

This token also appears in some HP event messages that contain the ZEMS-TKN-OPMSG token.

ZEMS-TKN-CU (D-series RVUs only) (type uint; nonshared) is the subchannel address (controller and unit number) of the device associated with the subject of the event message.

ZEMS-MAP-PHYSICAL-ADDRESS (G-series RVUs only) is a nonshared token, private to HP.

ZEMS-TKN-SAC-NAME (G-series RVUs only) is a nonshared token, private to HP.

ZEMS-TKN-SCSI-TARGET-ID (G-series RVUs only) is a nonshared token, private to HP.

ZEMS-TKN-SCSI-LUN (G-series RVUs only) is a nonshared token, private to HP.

ZEMS-MAP-PLUG-IN-CARD (G-series RVUs only) is a nonshared token, private to HP.

ZEMS-TKN-SERVERNET-FABRIC (G-series RVUs only) is a nonshared token, private to HP.

ZEMS-TKN-OPMSG is a nonshared token, private to HP.
ZSPI-TKN-MANAGER (type $fname32$) is the process ID of a name manager. Certain subsystems, such as Pathway, create event messages that contain names that require qualification to be unique. For example, term6 might mean different things to different Pathway systems. A name manager (PATHMON for Pathway) specifies such names uniquely. For more information about name managers, see the *Distributed Name Service (DNS) Management Programming Manual* and the *Distributed Name Service (DNS) Management Operations Manual*. 
EMS procedures help you process event messages. You can:

- Retrieve information from event messages
- Produce display text derived from event messages
- Create event messages

These procedures work hand-in-hand with SPI procedures and are similar to them in many ways.

Event messages, like command messages, are based on SPI tokens. For a thorough discussion of issues that apply to both command messages and event messages, such as standard definition files, see the *SPI Programming Manual*.

Applications in TAL can call EMS procedures directly; COBOL85 applications can call them through the COBOL85 ENTER TAL feature. You can use TACL to retrieve or display event messages by invoking TACL built-ins #EMSGET, #EMSGETV, #EMSTEXT, and #EMSTEXTV, but you cannot use TACL to report events.

This section describes the procedures used by management applications to retrieve information from event messages and the procedures used by subsystems to produce event messages:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which Procedures to Use</td>
<td>15-2</td>
</tr>
<tr>
<td>Required Declarations and Standard Definitions</td>
<td>15-7</td>
</tr>
<tr>
<td>Procedure Descriptions</td>
<td>15-8</td>
</tr>
</tbody>
</table>

If your program will retrieve event messages, read Section 4, Retrieving Event Messages Programmatically; Section 17, Distributor Commands and Responses; and Section 5, Compiled Filters.

If your program will report event messages, read Section 8, Reporting Events; Section 9, Standard Events; Section 10, Generating Standard Events; and Section 11, Procedure Calls for Standard Events.
Which Procedures to Use

Because of the way TAL handles procedure parameters, it is inconvenient to use one procedure to handle all types of tokens. To understand this issue, you must know about passing tokens as parameters to EMS and SPI procedures.

Passing Token Parameters by Value or by Reference

Some tokens are identified by token codes and other tokens by token maps. The way your TAL applications pass tokens as parameters to the EMS or SPI procedures depends on the way the token is identified and, to a certain extent, on personal preference. To help decide whether to pass tokens by reference or by value, consider the following:

- Always pass a token map, which represents an extensible structured token, by reference (by address).
- It is usually more efficient to pass a token code by value.
- To pass a token code by reference, you must first store the code in a temporary variable if it is not already stored that way.

Procedure Summary Tables

Table 15-1 through Table 15-3 summarize procedures that help you process event messages.

EMS procedures begin with the letters EMS and are described in Procedure Descriptions on page 15-8. SPI procedures begin with the letters SS and are described in the SPI Programming Manual. Other procedures are Guardian system procedures and are described in the Guardian Procedure Calls Reference Manual.

<table>
<thead>
<tr>
<th>Name</th>
<th>Token-Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMSGET</td>
<td>Reference</td>
<td>Gets tokens and their values or related information from an event message</td>
</tr>
<tr>
<td>EMSGETTKN</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>EMSTEXT</td>
<td>(Does not apply)</td>
<td>Produces displayable text derived from an event message</td>
</tr>
</tbody>
</table>
### Table 15-2. EMS Procedures for Reporting Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Token-Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMSINIT</td>
<td>Value</td>
<td>Initializes an event-message buffer: first step in event message creation</td>
</tr>
<tr>
<td>EMSINITMAP</td>
<td>Reference</td>
<td></td>
</tr>
<tr>
<td>EMSADDTOKENS</td>
<td>Value</td>
<td>Adds one to four tokens to a buffer that has already been initialized by EMSINIT or EMSINITMAP</td>
</tr>
<tr>
<td>EMSADDTOKENMAPS</td>
<td>Reference</td>
<td></td>
</tr>
<tr>
<td>EMSADDSUBJECT</td>
<td>Value</td>
<td>Adds an additional subject to an event-message buffer</td>
</tr>
<tr>
<td>EMSADDSUBJECTMAP</td>
<td>Reference</td>
<td></td>
</tr>
</tbody>
</table>

### Table 15-3. Other EMS Procedures for Event-Message Processing

<table>
<thead>
<tr>
<th>Name</th>
<th>Token-Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN</td>
<td>(Does not apply)</td>
<td>Opens $0.#ZSPI, altcol.#ZSPI, or a distributor for command messages. Opens $0 or an alternate collector for event messages.*</td>
</tr>
<tr>
<td>CLOSE</td>
<td>(Does not apply)</td>
<td>Ends communication begun by the OPEN procedure.</td>
</tr>
<tr>
<td>WRITEREAD</td>
<td>(Does not apply)</td>
<td>Transports command messages to and from the distributor and collectors for the STATUS, GETEVENT, GETVERSION, and CONTROL commands. Reports standard event messages to $0 and any alternate collectors.**</td>
</tr>
<tr>
<td>WRITE</td>
<td>(Does not apply)</td>
<td>Transport TEXT messages to collectors. The text after 104 bytes is truncated.</td>
</tr>
<tr>
<td>SSPUT</td>
<td>Reference</td>
<td>Are used only for special operations such as positioning.</td>
</tr>
<tr>
<td>SSPUTTKN</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>SSMOVE</td>
<td>Reference</td>
<td>Move tokens from one message buffer to another.</td>
</tr>
<tr>
<td>SSMOVETKN</td>
<td>(Does not apply)</td>
<td></td>
</tr>
<tr>
<td>DEVICEINFO2</td>
<td>(Does not apply)</td>
<td>Determines whether a named process is an EMS collector.</td>
</tr>
</tbody>
</table>

* In COBOL85 programs, when sending EMS event messages and SPI commands to the primary collector (see WRITEREAD), first use COBOL85^SPECIAL^OPEN instead of EMS OPEN to open the primary collector’s $0.#ZSPI device. When sending EMS event messages and SPI commands to an alternate collector, first use COBOL OPEN instead of EMS OPEN to open the alternate collector’s #ZSPI device.

** In COBOL85 programs, use COBOL WRITE to send event messages to the primary or alternate collector. Use both COBOL WRITE with COBOL85^SPECIAL^OPEN to send SPI command messages to a primary or alternate collector. In COBOL85 programs, do not use READ WITH PROMPT to send event messages to a primary or alternate collector.
Examples

These examples show the use of the EMS procedures described in Table 15-3.

Example 1: Opening a Collector to Send Event Messages or Text

To open an alternate collector named $ACOL, use OPEN:

```
INT ALTCOLNAME[0:11] := ["$ACOL ", 9 * [" "]];
INT COL^FNUM;

CALL OPEN(ALTCOLNAME , COL^FNUM , 0  !Waited, READ/WRITE access, SHARED
         , 1);              !exclusion mode
         !Auto-retry path errors

IF COL^FNUM > 0 THEN
    BEGIN                !Open successful
```

Example 2: Using WRITEREAD to Send an Event

To create an event message and use the WRITEREAD procedure to send it to a collector:

```
INT COL^FNUM;                  !Initialized by OPEN
INT .EVENT^BUF[0:MAX^EVENT];
INT EVENT^SIZE;
INT ERROR;

IF NOT (ERROR := EMSINIT(BUF, .......)) THEN
    BEGIN              !Buffer successfully initialized
        IF NOT (ERROR := EMSADDTOKENS(BUF, .....)) THEN
            BEGIN  !Tokens successfully added
                CALL SSGETTKN(BUF, ZSPI^TKN^USEDLEN, EVENT^SIZE);
            CALL WRITEREAD ( FILENUM
                             , EVENT^BUF
                             , EVENT^SIZE
                             , 0);
            IF = THEN
                BEGIN          !Event successfully sent
```
Example 3: Issuing a Text Message Using the WRITE Procedure

To define a text message buffer and use the WRITE procedure to send it to a collector:

\[
\begin{align*}
\text{LITERAL TEXT}^\text{SIZE} &= 12; \\
\text{INT COL}^\text{FNUM}; &\quad \text{!Initialized by OPEN} \\
\text{INT .TEXT}^\text{BUF}[0:\text{TEXT}^\text{SIZE}-1] &= \text{"CODER HAS BECOME BUCOLIC"}; \\
\text{CALL WRITE (COL}^\text{FNUM} \\
&\quad , \text{TEXT}^\text{BUF} \\
&\quad , \text{TEXT}^\text{SIZE} '<' 1); \\
\text{IF } = \text{ THEN} \\
&\quad \text{BEGIN} \quad \text{!Text message sent successfully}
\end{align*}
\]

Example 4: Opening a Collector for the SPI Interface

To open an alternate collector named $ALTC for the SPI interface, use the OPEN procedure:

\[
\begin{align*}
\text{INT ALT}^\text{SPI}^\text{FNUM} &:= -1; \\
\text{INT ALT}^\text{SPI}^\text{FNAME}[0:11] &:= \text{"$ALTC \#ZSPI"}; \\
\text{CALL OPEN (ALT}^\text{SPI}^\text{FNAME, ALT}^\text{SPI}^\text{FNUM, 0, 1);} \\
\text{IF ALT}^\text{SPI}^\text{FNUM} > 0 \text{ THEN} \\
&\quad \text{BEGIN} \quad \text{!Open succeeded}
\end{align*}
\]
Example 5: Sending an SPI Command

To create an SPI command buffer and use the WRITEREAD procedure to send it to a collector:

INT ALT^SPI^FNUM; !From OPEN
INT .CMD^BUF[0:ZEMS^VAL^BUFSIZE - 1];
INT CMD^LEN;

IF NOT (ERROR := SSINIT(CMD^BUF, ......)) THEN
BEGIN !Buffer Initialized OK
IF NOT (ERROR := SSPUT(CMD^BUF, ......)) THEN
BEGIN !Token added OK
.
.
.
CALL SSGETTKN(CMD^BUF,
    ZSPI^TKN^USEDLEN,
    CMD^LEN);

CALL WRITEREAD(ALT^SPI^FNUM,
    CMD^BUF,
    CMD^LEN,
    ZEMS^VAL^BUFSIZE);

IF = THEN
BEGIN !SPI command sent OK

Example 6: Determining Whether a Process Is an EMS Collector

To determine whether a named process is an EMS collector, use DEVICEINFO2:

INT RECLEN;

CALL DEVICEINFO2 (PROCESSNAME,DEVTYPE,RECLN,,ERROR,2,1000D);
IF NOT ERROR THEN
IF DEVTYPE.<4:9> = 1 THEN
BEGIN ! Process is an EMS collector
    IF RECLN = 102 THEN
    BEGIN ! Process is a primary collector
    END
    ELSE
    BEGIN ! Process is an alternate collector
    END;
END
ELSE
BEGIN ! Process is not an EMS collector
END;
Required Declarations and Standard Definitions

Whether you program in TAL, COBOL85, C, Pascal, FORTRAN, or HP Tandem Advanced Command Language (TACL), consult the appropriate sections of the SPI Programming Manual. These sections generally apply to token-based messages (that is, to event messages as well as command and response messages).

This subsection is an overview. For more information, see the SPI Programming Manual.

Standard Definitions in TAL Programs

Each TAL module of your application must contain ?SOURCE directives to include the TAL versions of the SPI standard definition files for relevant subsystems. For example, this directive defines the symbols used by SPI and by EMS:

?SOURCE $SYSTEM.ZSPIDEF.ZSPITAL
?SOURCE $SYSTEM.ZSPIDEF.ZEMSTAL

You need a ?SOURCE directive for each subsystem whose symbols you use in the TAL module.

To make the standard definition files available to your COBOL85 program, use COPY statements referencing the relevant COBOL standard definition files.

Declaring Buffers and Subsystem IDs in TAL

To send command messages to a distributor, you need a buffer for distributor commands, such as GETEVENT. Use the template in the EMS standard definition file, for example:

STRUCT .my^spi^buffer (ZEMS^DDL^MSG^BUFFER^DEF);

If you are writing a subsystem and need a buffer in which to create event messages:

1. Use the same template to declare an event-message buffer:
   
   STRUCT .my^evt^msg^buffer (ZEMS^DDL^MSG^BUFFER^DEF);
   
   The resulting buffer is slightly longer than required.

2. Declare each subsystem ID that you pass to an EMS procedure in a declaration such as:
   
   STRUCT .ZEMS^VAL^SSID (ZEMS^VAL^SSID^DEF);

3. Initialize the subsystem ID with a statement such as:
   
   ZEMS^VAL^SSID ':=' [ZSPI^VAL^TANDEM, ! Z^OWNER
   ZSPI^SSN^ZEMS, ! Z^NUMBER
   ZEMS^VAL^VERSION ]; ! Z^VERSION

   This initializes the subsystem ID for EMS itself.
For more information about using SPI in TAL programs, see the *SPI Programming Manual*.

# Procedure Descriptions

This subsection describes the EMS procedures in alphabetical order.

## EMSADDSUBJECT and EMSADDSUBJECTMAP Procedures

The EMSADDSUBJECT and EMSADDSUBJECTMAP procedures let you add a subject token to the event-message buffer. Either procedure places the token that you provide—preceded by a *subject-mark* token—in the buffer.

In TAL programs, use EMSADDSUBJECT when supplying a token code for the *subject-token-id* parameter. This eliminates the need to store the token code in a temporary variable before passing it to EMSADDSUBJECTMAP. You must use EMSADDSUBJECTMAP when supplying a token map.

Note: These procedures can be called from 32-bit and 64-bit programs.

In COBOL85 programs, use EMSADDSUBJECTMAP through ENTER TAL.

```cobol
{ status : =  } { EMSADDSUBJECT   ( buffer       ! i/o
{ CALL      } { EMSADDSUBJECTMAP , subject-token-id ! i
                         , [ subject-value ]! i
                         , [ subject-length ]! i
                         , [ ssid     ]  );       ! i

status                returned value
        INT : value
is zero or an SPI error code. For a full list of SPI error numbers, see EMSGET and
EMSGETTKN Procedures on page 15-12.

buffer
        INT .EXT: ref:*
is the event-message buffer, which the caller must have allocated and initialized.

subject-token-id
        INT (32): value    (EMSADDSUBJECT)
        INT .EXT: ref:*   (EMSADDSUBJECTMAP)
is the token code or token map of the subject token to be added to the event
message.

subject-value
        STRING .EXT: ref:*
is the value of the new subject token. Include this parameter if a value is associated with the subject token.

*subject-length*

**INT:value**

is the length, in bytes, of the *subject-value* field. This parameter is ignored unless the *subject-token-code* is defined as a variable-length token. For alternative ways to specify the length of a variable-length token, see Considerations.

*ssid*

**INT .EXT:ref:6**

is a subsystem ID that qualifies the token code. If not supplied or equal to zero (6*[0]), *ssid* defaults to one of the following:

- If the current position is in a list, the subsystem ID of the current list
- If the current position is not in a list, the subsystem ID in the event-message header (ZSPI^TKN^SSID)

### Considerations

- Every event message has at least one subject, which you specify to the EMSINIT procedure. Use EMSADDSUBJECT to specify additional subjects.

- Choose one of these ways to specify the length, in bytes, if the subject token has a variable-length value:
  - Use the *subject-length* parameter to specify the length.
  - Place the token length in the two bytes that immediately precede the token value; make the *subject-value* parameter point to the extra bytes. (Do not count the extra bytes as part of the length.) When using this method, omit the *subject-length* parameter.

- If the length of the value of a variable-length subject token is zero, you cannot just omit both value and length parameters. Specify the length (zero) explicitly, by using one of the two ways described earlier.

- Each call to EMSADDSUBJECT inserts two tokens into the buffer:
  - The ZEMS^TKN^SUBJECT^MARK token, which always precedes a subject
  - The subject token that you specified

- If the subsystem that generates the event message needs to include tokens from another subsystem (often a lower-level subsystem), the call to EMSADDTOKENS must include the *ssid* parameter, which specifies the subsystem ID of the other subsystem. When you specify the *ssid* parameter, every token placed in the
buffer on that procedure call is in an extended form that includes the ssid that you specified.

EMSADDTOKENS and EMSADDTOKENMAPS Procedures

The EMSADDTOKENS and EMSADDTOKENMAPS procedures let you add one to four tokens to an event-message buffer.

In TAL programs, use EMSADDTOKENS when supplying a token code for the subject-token-id parameter. This eliminates the need to store the token code in a temporary variable before passing it to EMSADDTOKENMAPS. You must use EMSADDTOKENMAPS when supplying a token map.

Note: These procedures can be called from 32-bit and 64-bit programs.

In COBOL85 programs, use EMSADDTOKENMAPS through ENTER TAL. In TACL, use #EMSGET.

status returned value

INT:value

is zero or an SPI error code. For a full list of SPI error numbers, see EMSGET and EMSGETTKN Procedures on page 15-12.

buffer

INT .EXT64:ref:*

is the event-message buffer, which the caller must have allocated and initialized.

ssid

INT .EXT:ref:6

is a subsystem ID that qualifies the token code. If not supplied or equal to zero (6*[0]), ssid defaults to one of the following:

- If the current position is in a list, the subsystem ID of the current list
- If the current position is not in a list, the subsystem ID in the event-message header (ZSPI^TKN^SSID)
tkn-triplet

is the token code or token map of the token to be added to the buffer followed by the token value and the length of that value. tkn-triplet has the following syntax:

token-id, [ data-buf ], [ data-len ]

token-id

INT(32):value (EMSADDTOKENS)
INT .EXT:ref:* (EMSADDTOKENMAPS)

is the token code or token map of a token to be added to this event message.

data-buf

STRING .EXT:ref:*  

if present, is the value of the token to be added to the event message. This parameter must be present if the token takes a value.

data-len

INT:value  

is the length, in bytes, of the value in data-buf. This parameter is ignored unless token-id is defined as a variable-length token. For other ways to specify the length of a variable-length token, see Considerations.

Considerations

- For each token that you add—each tkn-triplet—use three commas, regardless of whether you include a token value or token length. You can omit trailing commas on the last tkn-triplet.

- Choose one of these ways to specify the length, in bytes, if data-buf contains a variable-length value:
  - Use the data-len parameter to specify the length.
  - Place the token length in the two bytes that immediately precede the token value; make the data-buf parameter point to the extra bytes. (Do not count the extra bytes as part of the length.) When using this method, omit the data-len parameter.

- If the length of the value of a variable-length token is zero, you cannot just omit both value and length parameters. Use either previously described method to specify the length (zero) explicitly.

- If the subsystem that generates the event message needs to include tokens from another subsystem (often a lower-level subsystem), the call to EMSADDTOKENS must include the ssid parameter, which specifies the subsystem ID of the other subsystem. When you specify the ssid parameter, every token placed in the
buffer on that procedure call is in an extended form that includes the ssid that you specified.

**EMSGET and EMSGETTKN Procedures**

The EMSGET and EMSGETTKN procedures extract tokens and related information from an event-message buffer. The two procedures produce the same results and are identical except for:

- The type of the token-id parameter—EMSGET passes token-id by reference and EMSGETTKN passes it by value.
- The consequent fact that EMSGETTKN cannot be used with a token map.

EMSGET and EMSGETTKN are similar in operation to SSGET and SSGETTKN.

In TAL programs, use EMSGETTKN when supplying a token code for the token-id parameter. This eliminates the need to store the token code in a temporary variable before passing it to EMSGET. You must use EMSGET when supplying a token map.

In COBOL85 programs, use EMSGET through ENTER TAL.

```plaintext
{ status := } { EMSGET } ( buffer ! i/o
{ CALL } { EMSGETTKN } , token-id ! i
, [ token-value ] ! i/o
, [ index ] ! i
, [ count ] ! i/o
, [ ssid ] ) ! i/o

status returned value
INT
is one of these SPI error numbers:

0 No error
-1 Invalid buffer format
-2 Illegal parameter value
-3 Missing parameter
-4 Illegal parameter address
-5 Token too large for buffer
-6 Invalid checksum
-7 Internal error
-8 Token not found
-9 Illegal token code or map
-10 Invalid subsystem ID
-11 Operation not supported
-12 Insufficient stack space
-30 Buffer length is larger than ZEMS-VAL-EVT-BUFLEN

buffer input, output
INT .EXT:*
is the SPI buffer from which information is to be extracted.

token-id               input
INT .EXT:ref:* (EMSGET)
INT(32):value (EMSGETTKN)

is a token code (EMSGETTKN) or else a pointer (EMSGET) to a token code or
token map. This parameter normally identifies the token to be retrieved. If the

token-id
is one of the SPI standard token codes indicating a special operation,

the interpretation of the token-value, count, and index parameters can vary

from the descriptions here. For more information, see Special Operations for

EMSGET and EMSGETTKN on page 15-15.

If token-id is a token that marks the beginning of a list (ZSPI^TKN^DATALIST,
ZSPI^TKN^ERRLIST, or ZSPI^TKN^LIST), EMSGET or EMSGETTKN selects the

list so that subsequent calls can retrieve tokens within the list.

token-value               input, output
STRING .EXT:ref:*

is normally the variable in which the requested token value is to be returned. For

control and positioning operations, token-value can be used as an input

parameter. Its data representation depends on the token-type field of the token-

id.

index               input
INT:value

if greater than zero, specifies an absolute index for token-id, starting from the

beginning of the buffer or list. That is, an index of 1 gets the first occurrence of

that token code; an index of 2 gets the second occurrence; and so on.

if zero or not supplied, directs EMSGET or EMSGETTKN to return the next

occurrence of the token code after the current position in the buffer. For example, if

the token occurred 5 times, calling EMSGET or EMSGETTKN once with an index

of 1 and 4 times with index 0 would return all 5 occurrences.

if less than zero, returns an error.

EMSGET then resets the current position to that of the token value returned.
**count** input, output

INT .EXT:ref:1

is normally used as an input and output count parameter:

- On the call, it specifies the maximum number of token values to return. The *token-value* parameter is an array of *count* elements, each of which is described by the *token-id*. If not supplied, it defaults to 1. If less than zero, it causes an error.

- On return, it specifies the actual number of token values returned.

If a count greater than 1 is specified, EMSGET or EMSGETTKN continues searching until it either satisfies the requested count or reaches the end of the buffer or list.

For certain tokens for special operations, EMSGET and EMSGETTKN use the *count* parameter to return attribute information such as length, byte offset, or number of occurrences.

**ssid** input, output

INT .EXT:ref:6

is a subsystem ID that qualifies the token code (see Section 4, Retrieving Event Messages Programatically). If not supplied or equal to zero ( 6*[0] ), *ssid* defaults to one of:

- If the current position is in a list, the subsystem ID of the current list

- If the current position is not in a list, the subsystem ID in the message header (ZSPI^TKN^SSID)

The version field of this parameter is not used when searching the buffer.

**Considerations**

- Tokens extracted by EMSGET and EMSGETTKN are not deleted or removed from the buffer.

- For checkpoint purposes, calls to EMSGET and EMSGETTKN can modify the message header. For instance, the header tokens ZSPI^TKN^LASTERR and ZSPI^TKN^LASTERRCODE change if an SPI error occurs on the call. Positioning information in the header also changes frequently, and future versions of SPI might introduce other kinds of change. Programs should always account for the possibility that any EMSGET or EMSGETTKN operation can change the buffer.

- When the current position is within a list, all EMSGET and EMSGETTKN calls pertain only to tokens within that list, except that the header tokens are always accessible. If the tokens are EMS tokens, for more information see Section 14, EMS Definitions. If the tokens are SPI tokens, see the SPI Programming Manual.
Your program can exit the list by calling EMSGET to get the ZSPI^TKN^ENDLIST token.

- If you want the search for a token to start at the beginning of the buffer or current list, your program must do one of:
  - Use EMSGET or EMSGETTKN, supplying a nonzero value for index.
  - Use SSPUT or SSPUTTKN, first resetting the initial position with ZSPI^TKN^RESET^BUFFER or ZSPI^TKN^INITIAL^POSITION. For more information, see the SPI Programming Manual.

- The index and count parameters have no effect when token-id is ZSPI^TKN^ENDLIST. But if supplied, index must be equal to 0 or 1, and count is always returned as 1.

- When you use a token map for the token-id parameter, the map can specify a structure version that is longer or shorter than the structure contained in the buffer. If the requested version is longer than the version in the buffer, EMSGET calls SSNULL to set to null values any new fields that are not obtained from the buffer. If the requested version is shorter than the one in the buffer, EMSGET returns only the requested length.

Special Operations from SSGET and SSGETTKN

EMSGET and EMSGETTKN recognize a number of token codes that perform special operations such as positioning within the event-message buffer. Think of these special token codes, which are not actually present in the event-message buffer, as commands to EMSGET and EMSGETTKN.

Most of these token codes and operations are identical to those defined for SSGET and SSGETTKN (see the SPI Programming Manual for a description of these operations). Some are defined only for EMSGET and EMSGETTKN.

Special Operations for EMSGET and EMSGETTKN

EMSGET and EMSGETTKN recognize these special token codes that are not recognized by SSGET and SSGETTKN:

**ZEMS^TKN^SENDERID**

Use this token code to get the internal file name of the process that originated the event message:

```c
EMSGETTKN(buffer, ZEMS^TKN^SENDERID, senderid );       ! output
```

senderid is of token type ZSPI^TYP^FNAME32.
ZEMS^TKN^XSENDERID

Use this token code to get the node number, CPU, and PIN of the sender of the event:

\[
\text{EMSGETTKN}(\text{buffer, ZEMS^TKN^XSENDERID}, \ xsenderid )\; ; \quad \text{! output}
\]

\(xsenderid\) is of token type ZSPI^TYP^STRUCT.

ZEMS^TKN^XSENDERID-PD

Use this token code to get the process descriptor of the process that generated the event:

\[
\text{EMSGETTKN}(\text{buffer, ZEMS^TKN^XSENDERID-PD}, \ xsenderidpd )\; ; \quad \text{! output}
\]

\(xsenderidpd\) is of token type ZSPI^TYP^STRING.

Use senderid-type tokens with EMSGET and EMSGETTKN. Using these tokens is the only way to get information about the process that reported an event when the event-message text was sent to $0 through a call to the WRITE procedure. By contrast, the PROC-DESC, CPU, PIN, USERID, and NODENUM tokens do not provide information about the process that produced the text if the event was reported by a WRITE to $0. Instead, these tokens provide information about $0 because $0 made a tokenized message from the text. If you use senderid-type tokens, you need not consider how events are reported.

For more information about ZEMS^TKN^SENDERID, ZEMS^TKN^XSENDERID, and ZEMS^TKN^XSENDERID-PD, see Section 14, EMS Definitions.

ZEMS^TKN^SUBJECT.

Use this token code to get the subject token specified by index:

\[
\text{EMSGETTKN}(\text{buf, ZEMS^TKN^SUBJECT}, \ subject-token, \ index, \ subject-token-index, \ subject-token-ssid )\; ; \quad \text{! output}
\]

If you specify an \(index\ i\), EMSGET finds the \(i\)th subject. If \(index\) is omitted or zero, EMSGET finds the first subject beyond the current position.

EMSGET returns the token code, index, and subsystem ID of the specified subject token. The value of the subject token is not returned. (The \text{subject-token} operation returns all the information needed for a subsequent call to EMSGET to fetch the subject token itself.)

ZEMS^TKN^D00.

EMSGET returns no value for this token code. Instead, EMSGET returns either a function value of ZSPI^ERR^OK if the event buffer contains an event for D-series
RVUs or later, or it returns ZSPI^ERR^MISTKN if the event buffer contains an event preceding the D-series RVU:

```
EMSGETTKN(buffer, ZEMS^TKN^D00, 
  d00 );       ! output
```

*d00* is of token type ZSPI^TYP^BOOLEAN.

**Usage Restrictions for Special-Operation Tokens**

These restrictions apply to special-operation tokens:

- In general, special operations are not defined for header tokens. However, the ZSPI^TKN^COUNT and ZSPI^TKN^LEN special operations are defined for all tokens (including header tokens).

- Perform special operations on special-operations tokens only in these (exceptional) cases:
  - To find the number of subjects in an event message, call EMGETSTKNN:
    ```
    EMSGETTKN(buf, ZSPI^TKN^COUNT, 
    ZEMS^TKN^SUBJECT, 1, count );                  ! output
    ```
  - To find the length of a particular subject in an event message, call EMSGETTKN:
    ```
    EMSGETTKN(buf, ZSPI^TKN^LEN, 
    ZEMS^TKN^SUBJECT, index, byte-length );            ! output
    ```

**EMSINIT and EMSINITMAP Procedures**

The first step in event-message creation is a call to the EMSINIT (or EMSINITMAP) procedure. Either procedure initializes a buffer as an event-message buffer.

In TAL programs, use EMSINIT when supplying a token code for the *subject-token-id* parameter. This eliminates the need to store the token code in a temporary variable before passing it to EMSINITMAP. You must use EMSINITMAP when supplying a token map.

Note: These procedures can be called from 32-bit and 64-bit programs.
In COBOL85 programs, use EMSINITMAP through ENTER TAL.

| { status := } | { EMSINIT } | ( buffer      ! i/o       |
| { CALL      } | { EMSINITMAP } | , buffer-length  ! i       |
|              |              | , generating-ssid ! i      |
|              |              | , event-number  ! i       |
|              |              | , subject-token-id ! i     |
|              |              | , [ subject-value ] ! i    |
|              |              | , [ subject-length ] ! i   |
|              |              | , [ subject-ssid ] ! i     |
|              |              | , [ time-stamp ] ! i       |

status returned value

INT: value

is zero or an SPI error code. For a full list of SPI error numbers, see EMSGET and EMSGETTKN Procedures on page 15-12.

buffer

INT .EXT: ref:*

is the event-message buffer, which the caller must have allocated and EMSINIT (or EMSINITMAP) initializes.

buffer-length

INT: value

is the length, in bytes, of the buffer—the length to be initialized. buffer-length should not be larger than the value of ZEMS-VAL-EVT-BUFLEN (currently 4024 bytes).

generating-ssid

INT .EXT: ref: 6

is the subsystem ID of the subsystem originating the event message. The version field of this parameter is stored in the header of the event message.

event-number

INT: value

is a number, specific to this subsystem, that identifies this event message.

subject-token-id

INT(32): value (EMSINIT)
INT .EXT: ref:* (EMSINITMAP)

is the token code or token map of the subject of this event message.
subject-value
STRING .EXT:ref:*  
if present, is the value of the subject to be added to the event message.

subject-length
INT:value  
is the length, in bytes, of the subject-value field. This parameter is ignored unless the subject-token-code is defined as a variable-length token. For more information, see Considerations on page 15-19.

subject-ssid
INT .EXT:ref:6  
is the subsystem ID of the subsystem to which the subject token belongs. If you omit this parameter or supply all zeros (6*[0]), the generating-ssid is used by default. The version field of this parameter is not stored in the event message.

timestamp
FIXED:value  
is the time stamp in Julian GMT (64-bit) format for the event message. If you omit this parameter, the current time is used.

Considerations

● Choose one of these ways to specify the length, in bytes, if the subject token has a variable-length value.

● Use the subject-length parameter to specify the length.

● Place the token length in the two bytes that immediately precede the token value; make the subject-value parameter point to the extra bytes. (Do not count the extra bytes as part of the length.) When using this method, omit the subject-length parameter.

● If the length of the value of a variable-length subject token is zero, you cannot just omit both value and length parameters. Use one of the previously described methods to specify the length (zero) explicitly.

EMSTEXT Procedure

The EMSTEXT procedure produces displayable text from an event message. In TACL, the equivalent command is #EMSTEXT or #EMSTEXTV. You can have text laid out:

● In the newer, more readable DSM display format
In a modified version of the display format in which the format of the message header and the indentation of the message body are specified by the EMSTEXT parameters header template key and indent.

For the steps to use EMSTEXT, and for error results, see Text Formatting Tools on page 2-12. For an explanation of the template compiler, see the DSM Template Services Manual.

```
{ status := } EMSTEXT( event-message-buffer ! i,
{ CALL } call (,
  displayable-text-buffer ! o,
  display-line-length ! i,
  number-display-lines ! i,
  actual-lengths ! i,
  [ header-template-key ] ! i,
  [ indent ] ! i,
  < RESERVED >,
  [ extended-status ] ) ); ! o
```

status

INT(32):value

is a pair of 16-bit integer status codes, as described in Table 15-4 on page 15-23. Compare to extended-status on page 15-22.

event-message-buffer

INT .EXT:ref:*

is the buffer containing the event message from which displayable text is derived.

displayable-text-buffer

STRING .EXT:ref:*

is the buffer in which EMSTEXT stores the displayable text. The buffer is a sequence of fixed-length logical lines. Each logical line will contain a line of text to be displayed, including indent blanks (if any), text characters, and trailing blanks (if any).

Here is how EMSTEXT places the text of an event message into this buffer. In the first logical line, EMSTEXT left-justifies the text, puts in as many words as fit, and fills any remaining spaces with blanks. In each logical line thereafter, EMSTEXT first puts the number of blanks specified by indent on page 15-22, then as many more words of the text as fit, then trailing blanks.

Consequently, this buffer contains \( d \times n \) character spaces, where:

- \( d \) is the number of character spaces in each logical line (the value of display-line-length)
EMS Procedures

- $n$ is the number of logical lines required to display the message text (the value of number-display-lines on page 15-21)

**Note.** You must make the buffer size, $d \times n$, large enough to hold the entire text, including trailing and indent blanks. You will probably set $d$ to the standard line length for your output device—80 characters, for instance. Making the buffer large enough then depends directly on the value you choose for $n$. If you make $n$ too small, the displayed text is truncated.

When EMSTEXT buffers the text of an event message, it also saves—in the array actual-lengths—the length of the text line to be displayed within each logical line. This length includes the number of blanks (if any) by which the text is indented, and the number of text characters, but not the number of trailing blanks. EMSTEXT saves this information, so your output device need not print or display trailing blanks when putting out the message.

**display-line-length**

- INT: value
  - is the number of character spaces (bytes) in each logical line in displayable-text-buffer. This number is referred to as $d$ previously.

  For $d$, choose the maximum number of characters displayable or printable on your output device, such as 80 or 132.

**number-display-lines**

- INT: value
  - is the number of logical lines needed in displayable-text-buffer to display the message text. This number is referred to as $n$.

  When estimating $n$, check that the number of text characters on the first line does not exceed $d$, and on succeeding lines it does not exceed $d - i$, where $i$ is the value of indent.

**actual-lengths**

- INT .EXT: ref:*  
  - is an integer array. The array contains $n$ integers, where $n$ is the value of number-display-lines. Each integer denotes the length of the display line—the number of indent blanks and text characters—within a logical line in displayable-text-buffer. If a logical line is not used (has no text), the corresponding integer in the array is -1.

  In other words, for each logical line in displayable-text-buffer, the corresponding integer in this array tells how many bytes to display, starting from the beginning of the line.

**header-template-key**

- INT .EXT: ref:10
if present, is the key of a format template. EMSTEXT uses the template to format the *headers* of all event messages for display as text.

With the header template key, you can customize the header information for all the event messages of an application. The header is the first part of the event message. It includes such information as the ID of the process that gave rise to the event and the date and time when it occurred.

If *header-template-key* is missing or all zeros, EMSTEXT has its own header template to provide the process ID, date, and time in text form.

The text describing the event, usually produced from a template indicated by the message number, follows the header text.

*header-template-key* is structured like:

```plaintext
STRUCT TEMPLATE^KEY
  BEGIN
    STRUCT    SSID(ZSPI^DDL^SSID^DEF);
    INT(32)   TOKENCODE;
    INT(32)   TOKENVALUE;
  END;

  The value in the SSID field comes from the SSID: (subsystem ID) statement in the template file that defines the format template—except that the version part of the subsystem ID is not used. The values in TOKENCODE and TOKENVALUE come from the first two parameters of the MSG: statement in the template file. (For information about template files and format templates, see the *DSM Template Services Manual*.)
```

*indent*

INT:value

if present, is a hanging indent value; it defines how many blanks precede the text in every logical line of *displayable-text-buffer* except the first. This number is referred to as *i*, described previously.

If you do not want the displayed message text indented, set *indent* to 0. If *indent* is omitted or -1, EMSTEXT uses a default value.

* < RESERVED >  This parameter is reserved for future use.

*extended-status*

```plaintext
INT(32) .EXT:ref:1
```

is a pair of 16-bit-integer status codes, as described in *Table 15-5*. Compare to *status*, described previously. The value of *extended-status* is undefined if the returned value of EMSTEXT (the simple status) is (0, *x*) and *x* > 0.
Considerations

- Because EMSTEXT requires extra data stack space, include this command or its equivalent in your TAL program:

  ?EXTENDSTACK 4 ! To ensure enough stack space for
  ! EMSTEXT

- Use the EMSGET procedure to extract information from event messages when it is feasible to do so, rather than scanning text produced by EMSTEXT. EMSTEXT is intended specifically for producing text for display purposes.

Table 15-4 lists the status code pairs returned by a function call to EMSTEXT.

### Table 15-4. EMSTEXT Status Codes

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Normal return</td>
</tr>
<tr>
<td>0 22</td>
<td>Bad parameter (address or value); no displayable text</td>
</tr>
<tr>
<td>0 29</td>
<td>Required parameter missing; no displayable text</td>
</tr>
<tr>
<td>0 632</td>
<td>Insufficient stack space; no displayable text</td>
</tr>
<tr>
<td>10 0</td>
<td>Template file problem; or no template and no TEXT token for event; or EMSTEXT error. For an explanation, see Text Formatting Tools on page 2-12.</td>
</tr>
<tr>
<td>11 0</td>
<td>Bad event buffer. For an explanation, see Text Formatting Tools on page 2-12.</td>
</tr>
</tbody>
</table>

If you specify the `extended-status` parameter, EMSTEXT returns extended status codes. Use them to analyze and improve the performance of your application rather than to produce end-user diagnostics.

Table 15-5 lists the EMSTEXT extended status codes. Several refer to information of which part (or all) is private to HP. If you need help, your service provider can help solve the problem if you report the exact status code received.

The template file mentioned in Table 15-5 is the file named TEMPLATE in the system load subvolume. For details, see the DSM Template Services Manual.
### Table 15-5. EMSTEXT Extended Status Codes (page 1 of 2)

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 Normal return</td>
</tr>
<tr>
<td>1</td>
<td>ALLOCATESEGMENT returned error (x) because it did not get the private segment for EMSTEXT</td>
</tr>
</tbody>
</table>
| 2      | Problem with the template file:  
|        | \(x > 0\) A file-management error sent from the OPEN procedure.  
|        | \(x = -1\) The file code is not 839 or 844.  
|        | \(x = -2\) The file is not a disk file.  
|        | \(x = -3\) The file is not key-sequenced.  
|        | \(x = -4\) The file has the wrong record size.  
|        | \(x = -5\) The file has the wrong primary-key definition. |
| 3      | File-management error \(x\) when reading the template file. |
| 4      | **event-message-buffer** is bad:  
|        | \(x = 0\) The first word in the buffer is not -28, or the buffer length is wrong.  
|        | \(x = 1\) SSPUT failed to position to front of buffer.  
|        | \(x = 2\) EMSGET for ZEMS^TKN^CRTPID failed.  
|        | \(x = 3\) EMSGET for ZEMS^TKN^SYSTEM failed.  
|        | \(x = 4\) EMSGET for ZEMS^TKN^GENTIME failed.  
|        | \(x = 5\) EMSGET for ZEMS^TKN^EVENTNUMBER failed.  
|        | \(x = 6\) EMSGET for ZSPI^TKN^SSID failed.  
|        | \(x = 7\) EMSGET for ZEMS^TKN^XSYSID failed with status code other than ZSPI^ERR^MISTKN (possible only for an EMS event that should contain the XSYSID token).  
|        | \(x = 8\) EMSGET for ZEMS^TKN^OPSMG failed with status code other than ZSPI^ERR^MISTKN.  
|        | \(x = 9\) EMSGET for ZEMS^TKN^TEXT failed with status code other than ZSPI^ERR^MISTKN.  
|        | \(x = 10\) EMSGET for ZSPI^TKN^DEFAULT^SSID failed.  
|        | \(x = 11\) EMSGET failed for a token referenced in the template, with status code ZSPI^ERR^INVBUF, ZSPI^ERR^NOSPACE, or ZSPI^ERR^XSUMERR.  
|        | \(x = 12\) EMSGET or SSPUT of ZSPI^TKN^POSITION failed (trying to save or restore the position within the buffer).  
|        | \(x = 13\) EMSGET for ZEMS^MAP^EXIOADDR failed with status code other than ZSPI^ERR^MISTKN.  
|        | \(x = 14\) EMSGET for ZEMS^TKN^EMPHASIS failed. |
| 5      | File-management error \(x\) while reading the nonresident template file for the initial template |
Table 15-5. EMSTEXT Extended Status Codes  (page 2 of 2) (continued)

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Problem with the format template:</td>
</tr>
<tr>
<td></td>
<td>(x = 0): Unrecognized version number for the template structure</td>
</tr>
<tr>
<td></td>
<td>(x = 1): Unrecognized item-type code in the template</td>
</tr>
<tr>
<td></td>
<td>(x = 2): No edit-descriptor item to match an edit-marker item in the template</td>
</tr>
<tr>
<td></td>
<td>(x = 3): Unrecognized edit code in the template</td>
</tr>
<tr>
<td></td>
<td>(x = 4): Unrecognized token type or structure-field type in an edit-descriptor item</td>
</tr>
<tr>
<td></td>
<td>(x = 5): Bad DATE edit-string</td>
</tr>
<tr>
<td></td>
<td>(x = 6): Bad TIME edit-string</td>
</tr>
<tr>
<td></td>
<td>(x = 7): Bad item length (extends past the end of the template)</td>
</tr>
<tr>
<td></td>
<td>(x = 8): EMSGET failed for a token referenced in the template, with status code other than ZSPI^ERR^MISTKN and other than one of those listed for error (4,11).</td>
</tr>
<tr>
<td></td>
<td>(x = 9): Cannot find the end of the true branch of an *IF</td>
</tr>
<tr>
<td></td>
<td>(x = 10): Cannot find the conditional-expression definition for an *IF</td>
</tr>
<tr>
<td></td>
<td>(x = 11): Unrecognized operation code in a conditional-expression item</td>
</tr>
<tr>
<td>7</td>
<td>Private-segment access-error (x)</td>
</tr>
<tr>
<td>8</td>
<td>Other error:</td>
</tr>
<tr>
<td></td>
<td>(x = 0): There are more than 10 levels of nesting of MSG edit codes.</td>
</tr>
<tr>
<td></td>
<td>(x = 1): The format template referenced a token that was not found data beyond the end of the token value.</td>
</tr>
<tr>
<td></td>
<td>(x = 3): The buffer is too small for the format template.</td>
</tr>
<tr>
<td>9</td>
<td>File-management error (x) while reading a nonresident template file for a template referenced by a MSG format code</td>
</tr>
<tr>
<td>1</td>
<td>PROCESS_HANDLE_TO_FILENAME error (x)</td>
</tr>
<tr>
<td>2</td>
<td>PROCESS_HANDLE_DECOMPOSE error (x)</td>
</tr>
<tr>
<td>3</td>
<td>FILENAME_DECOMPOSE error (x)</td>
</tr>
</tbody>
</table>
16 Event Routing

This section describes dynamic routing of events by the printing distributor to selected destinations:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing Capability</td>
<td>16-1</td>
</tr>
<tr>
<td>Launching of Destination Processes</td>
<td>16-1</td>
</tr>
<tr>
<td>Formatting Selection</td>
<td>16-2</td>
</tr>
<tr>
<td>Multiple Filters</td>
<td>16-2</td>
</tr>
<tr>
<td>User Interfaces</td>
<td>16-3</td>
</tr>
<tr>
<td>Selection of Collectors for Internal Events</td>
<td>16-3</td>
</tr>
<tr>
<td>Distributor Generated Messages</td>
<td>16-4</td>
</tr>
</tbody>
</table>

Routing Capability

The printing distributor allows distribution of events to selected destinations. The destinations are determined in the filter on an event basis, and a list of routing IDs is passed by the filter interpreter to the distributor event processing logic. All potential destinations are defined as profiles in the filter source and are stored in the filter object.

Note. A routing distributor is a printing distributor with a filter that contains destination profiles.

The maximum number of event sources allowed is ten. Likewise, the maximum number of destinations for printing distributors is ten. There is no memory allocation limitation. Buffer space is allocated from an extended segment. Future expansions are easily accommodated.

Launching of Destination Processes

A routing distributor can start or restart a destination process. This feature, originally required by TMDS for fault analyzers, is available for any process that has startup parameters defined in its configuration profile. Destination processes must be able to receive events, and overall response time is in direct relation to the slowest destination. A time limit is set so the distributor does not hang up on a specific destination.

A destination process is started or restarted when an event is received for it as a result of a filter PASS statement. Security has been provided to prevent unauthorized launching of processes. To safeguard against launching of processes that might inherit super.super access from the distributor, a SPI command requesting the loading of a filter that contains DESTINATION statements with startup information is accepted only if the sender's access ID matches the access ID of the distributor process. A filter ADD command is the only mechanism to introduce a new process for launching.
The startup failed event contains a special token, ZEMS-TKN-STARTUP-LOGTIME, which is set to the log time retrieved from the event that the distributor could not send. When the next event trigger occurs for this destination, the distributor attempts to start the process again. If it fails, the event is skipped, but no messages are generated. If eventually the startup succeeds, a message is sent to the OUT device indicating that the process launched. An event (ZEMS-EVT-STARTUP-OK) is also generated. It contains a token (ZEMS-TKN-STARTUP-LOGTIME) with the log time of the event to be forwarded to the destination. To determine which events were not forwarded, examine the ZEMS-EVT-STARTUP-FAILED and ZEMS-EVT-STARTUP-OK messages.

If the distributor cannot start or restart a destination, it retries after 10 seconds. If the retry fails, a message is sent to the OUT device indicating that the startup has failed. An event (ZEMS-EVT-STARTUP-FAILED) is also generated and sent to a user selected set of destination collectors; for instance, $0 or $ZLOG. These collectors are selected using ADD DEFINE. For details, see User Interfaces on page 16-3.

If the distributor cannot write an event to an existing destination, it retries every two seconds until a configurable time limit is reached (the default is two minutes). This time-out can also occur if the intended recipient of the event does not acknowledge the request by reading its $RECEIVE file. In either case, to prevent flooding of the event log, a message is generated when the first timeout occurs and after 24 hours if retries are unsuccessful. The event (ZEMS-EVT-WRITE-FAILED) contains a counter (ZEMS-TKN-WRITE-FAIL-COUNT) with the number of timed out write operations that occurred during the 24 hours. When the first timeout is detected, the event reports a fail count of one (1). After a timeout, the current event is skipped.

If a write error occurred, it is reported the first time it is encountered.

**Formatting Selection**

You can configure a routing distributor to send events formatted or unformatted; that is, it functions either as a printing or forwarding distributor, with the difference that events can be sent to a set of targets that are selected dynamically. For formatted events, you can also configure some other formatting attributes such as record length (number of columns) and indentation.

**Multiple Filters**

To allow filter tables and burst filters in conjunction with compiled filters, and to minimize maintenance cost and effort, the distributor supports loading of multiple filters. Decide how many destinations a specific filter will support; for instance, TMDS plans to collect separate filters from each group that is responsible for a particular fault analyzer. This way, there is no problem with merging filter sources for different fault analyzers; however, there might be a slight overhead in executing multiple filters. They are executed one after another, and each filter's exit conditions such as PASS/FAIL and routing IDs are preserved. You can specify filter file names in the startup line or as multiple objects in a SPI command. If parameters are required for more than one filter, you must submit them all as one set. Parameters can be shared among filters.
User Interfaces

This subsection describes the use of multiple filters and the filter PASS statement.

Startup of Multiple Filters

A routing distributor is started as a printing distributor:

```
RUN EMSDIST TYPE P[rinting], FILTER f1 | (f1, f2, f3, ..fn),
....
```

The filters can contain destination profiles to direct events to selected destinations. If the startup line contains a TEXTOUT parameter, specifying a filter with destination profiles results in an error. If multiple filters are used with no destination profiles, the distributor does not process any subsequent filters after encountering a filter with a PASS condition.

You can use consumer and forwarding distributors with multiple filters, but not with destination profiles.

Filter PASS and DESTINATION Statements

The filter PASS statement allows an optional list of routing IDs, in addition to the value that can currently be returned. A routing distributor also returns the PASS value itself to the destination if the event is unformatted. This was previously done for consumer distributors only. In the case of multiple filters, each filter can send a different PASS value to its destinations.

The PASS statement has been modified to accommodate routing IDs:

```
PASS n, m | PASS n, (m1, m2, ... mN)
```

where $n$ is the PASS value as currently defined, and

$m, m1...mN$ are routing IDs as defined in the DESTINATION statements.

The destination statement is part of the filter source and used to describe a distributor’s routing configuration. Filters with destination statements can be used only in printing distributors; multiple statements per filter are allowed. For details about the DESTINATION statement, see The Filter Language on page 5-7.

Selection of Collectors for Internal Events

Distributor generated events (such as ZEMS-EVT-STARTUP-FAILED) can be forwarded to up to two user selectable collectors. You can choose $0$, alternate collectors, or $ZLOG$ (TMDS). To make the selection, use the ADD DEFINE statement with identifier _ems_system_log or identifier _ems_alternate_log. For example, this statement selects the TMDS collector $ZLOG$:

```
<ADD DEFINE =_ems_system_log, file $ZLOG>
```
If a DEFINE is not specified, all distributor generated events except ZEMS^EVT^BURST^START and ZEMS^EVT^BURST^END are written to $0.

The burst start and burst end messages are written to the OUT file of the distributor, as specified in the EMSDIST run option, OUT. If the EMSDIST run option OUT is specified but does not match the hometerm, burst start and burst end messages are written to the OUT file as a text error list. If the EMSDIST run option OUT is specified without a value, burst start and burst end messages are written to the hometerm of the distributor as a text error list. If the value of the EMSDIST run options OUT and TERM are the same, and neither the =_ems_system_log or =_ems_alternate_log defines exist, burst start and burst end messages are not written.

Distributor Generated Messages

Previously, if an OUT file was not explicitly specified, all printing or forwarding distributor generated error or warning messages were sent to the home terminal. This included messages that were generated after successful distributor startup. These messages were also converted to equivalent events and sent to $0. Because messages sent to the OUT file are not identified as to what distributor they are associated with, they might have little meaning for an operator. This is especially true for the new destination process error messages.

The current strategy is to send only messages that report distributor startup problems to the home terminal. If an OUT file other than the home terminal is given, all messages are recorded as before. Also, if OUT is specified without a file name, all messages are sent to the home terminal. Event generation is not affected.

The distributor generates its own event and sends it to up to two user selectable destinations when:

- An event cannot be written to the destination because of a file system error (first occurrence).
- The destination process can not be started (first occurrence).
- The destination process was successfully started after previous failure and subsequent retries.
- All retries have failed and resulted in repeated timeouts over a period of 24 hours.
Part V: Using EMS Distributors and Collectors

This part of the manual describes the commands, responses, event messages, and errors for EMS collectors and distributors:

- Section 17, Distributor Commands and Responses
- Section 18, Distributor Event Messages
- Section 19, Collector Commands and Responses
- Section 20, Collector Event Messages
- Section 21, Distributor Errors
- Section 22, Collector Errors
This section describes the command and response messages for the consumer, printing, and forwarding distributors. For descriptions of commands for the compatibility distributor, which is controlled through the EMS collector, see Section 19, Collector Commands and Responses.

The commands described here are mainly used to:

- Manage consumer, printing, and forwarding distributors from a management application
- Add, alter, or delete specified object types (for example, distributor filters, event sources, and event destinations)
- Retrieve event messages

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended Programmatic Interface</td>
<td>17-2</td>
</tr>
<tr>
<td>Object Support Summary</td>
<td>17-3</td>
</tr>
<tr>
<td>Common Definitions for ZCOM- Commands</td>
<td>17-5</td>
</tr>
<tr>
<td>Common Definitions for ZEMS- Commands</td>
<td>17-9</td>
</tr>
<tr>
<td>Distributor Command Descriptions</td>
<td>17-12</td>
</tr>
</tbody>
</table>

Table 17-1. Distributor Commands Summary (page 1 of 2)

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-CMD-ADD*</td>
<td>Adds filters, event sources, and event destinations (textouts).</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER*</td>
<td>Alters distributor attributes, replaces filter parameters, and replaces the forwarding target collector.</td>
</tr>
<tr>
<td>ZEMS-CMD-CONTROL</td>
<td>Changes the operational environment of a distributor. CONTROL can change the source of event messages (and the position within the source at which the distributor begins its examination of event messages), the filter and filter parameters, and the destinations of event messages that pass the filter. The CONTROL command can be replaced by the ZCOM-ADD, -ALTER, and -DELETE commands, which support the extended interface.</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE*</td>
<td>Deletes filters, event sources, and event destinations (textouts).</td>
</tr>
</tbody>
</table>

* Command supports the extended programmatic interface. (All have a ZCOM- prefix.)
Extended Programmatic Interface

At any time, at most one application can be in charge of a distributor (can send it ADD, ALTER, DELETE, GETEVENT and CONTROL command messages). In addition, up to 15 secondary applications can receive information from the distributor (can send it STATUS and GETVERSION command messages). If the user has the proper access authority (that is, super group access or the same access ID as distributors), these secondary applications can also be used to send filter control commands (ADD, ALTER, DELETE, REPLACE, or CONTROL) and to change the EOF delay control parameter.

Extended Programmatic Interface

The EMS distributor supports an extended, object-oriented command set developed before the SPI common extensions standard was released. This command set is similar, but does not fully comply with the SPI common extensions (extended SPI) standard.

The object-oriented distributor commands provide additional functionality relative to the distributor commands that still use the basic SPI interface. The extended programmatic interface for the distributor supports multiple or field-level wild-card objects in these commands and, where appropriate, returns multiple-object information. The objects supported are the EMS distributor (ZCOM-OBJ-DIST), EMS filter (ZCOM-OBJ-FILTER), event source (ZCOM-OBJ-SOURCE), forwarding target (ZCOM-OBJ-TARGET), and event destination (ZCOM-OBJ-TEXTOUT).

For all of these object types, an object hierarchy is assumed. In these cases, ZCOM-OBJ-FILTER, -SOURCE, -TARGET, and -TEXTOUT are all considered to be subordinate to ZCOM-OBJ-DIST. However, no hierarchical naming is allowed or supported.

The extended programmatic interface supports the distributor commands ZCOM-CMD-ADD, ZCOM-CMD-ALTER, ZCOM-CMD-DELETE, and ZCOM-CMD-STATUS. The extended programmatic interface requires specification of a ZCOM object type and

---

Table 17-1. Distributor Commands Summary (page 2 of 2) (continued)

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-CMD-GETEVENT</td>
<td>Gets the next event message to pass the current filter. GETEVENT applies only to consumer distributors.</td>
</tr>
<tr>
<td>ZEMS-CMD-GETVERSION</td>
<td>Gets the version number of the programmatic interface.</td>
</tr>
<tr>
<td>ZEMS-CMD-REPLACE</td>
<td>Replaces one configured filter in a distributor with another filter.</td>
</tr>
<tr>
<td>ZCOM-CMD-STATUS*</td>
<td>Gets information about the distributor, its associated filters, event sources, and destinations.</td>
</tr>
<tr>
<td>ZEMS-CMD-STATUS</td>
<td>Gets information about the distributor, its associated filters, event sources, and destinations.</td>
</tr>
</tbody>
</table>

* Command supports the extended programmatic interface. (All have a ZCOM- prefix.)
verb when the command is initialized by SSINIT, as well as a ZCOM-TKN-OBJNAME token.

Distributor commands still supported by the original basic SPI interface are ZEMS-CMD-CONTROL, ZEMS-CMD-GETEVENT, ZEMS-CMD-GETVERSION, and ZEMS-CMD-REPLACE. While the ZEMS-CMD-REPLACE command requires an object name, the other ZEMS- commands do not.

The EMS distributor distinguishes between the two interfaces by checking for the presence of the ZSPI-TKN-OBJECT-TYPE token, which is only present in the ZCOM-commands.

You can use these SPI tokens and values to add or delete filters, event sources, event destinations (for example, textout or target), and to set distributor attributes.

<table>
<thead>
<tr>
<th>ZSPI-TKN-COMMAND</th>
<th>ZSPI-TKN-OBJ-TYPE</th>
<th>CONTROL Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>zcom-obj-source</td>
<td>ZEMS-TKN-CONNECT-SRC-COLL</td>
</tr>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>zcom-obj-source</td>
<td>ZEMS-TKN-CONNECT-LOG</td>
</tr>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>zcom-obj-filter</td>
<td>ZEMS-TKN-FILTERFILE</td>
</tr>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>zcom-obj-textout</td>
<td>ZEMS-TKN-ADD-TEXTOUT</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>zcom-obj-filter</td>
<td>ZEMS-TKN-REPLACE-PARAM</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>zcom-obj-dist</td>
<td>ZEMS-TKN-GMTTIME</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>zcom-obj-dist</td>
<td>ZEMS-TKN-LOGTIME</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>zcom-obj-target</td>
<td>ZEMS-TKN-REPLACE-TGT-COLL</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>zcom-obj-dist</td>
<td>ZEMS-TKN-EOFDELAY</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>zcom-obj-dist</td>
<td>ZEMS-TKN-SEQ-BLOCKING</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>zcom-obj-filter</td>
<td>ZEMS-TKN-RESET-FILTER</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>zcom-obj-source</td>
<td>ZEMS-TKN-DISCONNECT-SRC-COLL</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>zcom-obj-source</td>
<td>ZEMS-TKN-DISCONNECT-LOG</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>zcom-obj-textout</td>
<td>ZEMS-TKN-DELETE-TEXTOUT</td>
</tr>
</tbody>
</table>

Table 17-2. ZCOM- Commands to Replace CONTROL Commands

Object Support Summary

Table 17-3, Table 17-4, and Table 17-5 list which distributor commands can be issued to which object, whether multiple objects can be specified, and whether object name wild cards (*) are supported.

Table 17-3 lists distributor command support for the distributor object (ZCOM-OBJ-DIST) and the filter object (ZCOM-OBJ-FILTER).
Table 17-3. Distributor and Filter Object Support

<table>
<thead>
<tr>
<th>Distributor Command</th>
<th>ZCOM-OBJ-DIST</th>
<th>ZCOM-OBJ-FILTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>No</td>
<td>Yes, &gt;=1</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>Yes, Only 1</td>
<td>Yes, &gt;=1, *</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>No</td>
<td>Yes, &gt;=1, *</td>
</tr>
<tr>
<td>ZEMS-CMD-REPLACE</td>
<td>No</td>
<td>Yes, Only 1</td>
</tr>
<tr>
<td>ZCOM-CMD-STATUS</td>
<td>Yes, Only 1, *</td>
<td>Yes, Only 1, *</td>
</tr>
<tr>
<td>ZEMS-CMD-GETVERSION</td>
<td>Yes, Only 1</td>
<td>No</td>
</tr>
<tr>
<td>ZEMS-CMD-GETEVENT</td>
<td>Yes, Only 1</td>
<td>No</td>
</tr>
</tbody>
</table>

"Yes" signifies that the distributor command supports the object type.

"No" signifies that the distributor command does not support the object type.

"Only 1" signifies that only one instance of that object type is supported.

">=1" signifies that more than one instance of that object type is supported.

"*" signifies that wild-card object types are supported.

Table 17-4 lists distributor command support for the event source object (ZCOM-OBJ-SOURCE) and the event destination object (ZCOM-OBJ-TEXTOUT).

Table 17-4. Event Source and Event Destination Object Support

<table>
<thead>
<tr>
<th>Distributor Command</th>
<th>ZCOM-OBJ-SOURCE</th>
<th>ZCOM-OBJ-TEXTOUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>Yes, &gt;=1</td>
<td>Yes, &gt;=1</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>Yes, &gt;=1, *</td>
<td>Yes, &gt;=1, *</td>
</tr>
<tr>
<td>ZEMS-CMD-REPLACE</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ZCOM-CMD-STATUS</td>
<td>Yes, Only 1, *</td>
<td>Yes, Only 1, *</td>
</tr>
<tr>
<td>ZEMS-CMD-GETVERSION</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ZEMS-CMD-GETEVENT</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

"Yes" signifies that the distributor command supports the object type.

"No" signifies that the distributor command does not support the object type.

"Only 1" signifies that only one instance of that object type is supported.

">=1" signifies that more than one instance of that object type is supported.

"*" signifies that wild-card object types are supported.

Table 17-5 lists distributor command support for the forwarding target object (ZCOM-OBJ-TARGET).
Common Definitions for ZCOM- Commands

This subsection defines the command tokens and response tokens common to the ZCOM-CMD-ADD, ZCOM-CMD-ALTER, ZCOM-CMD-DELETE, and ZCOM-CMD-STATUS commands.

The value for the ZCOM-OBJNAME token must be a string file name and cannot be a pre-D00 internal file name.

Note. The ZCOM- commands go directly from the requestor to the EMS distributor. An SCP process is not used.

Common Command Tokens for ZCOM- Commands

Table 17-6. Common Command Tokens for ZCOM- Distributor Commands

<table>
<thead>
<tr>
<th>Command Token</th>
<th>Token Type</th>
<th>Special Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>ZSPI-TYP-ENUM</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-OBJECT-TYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
</tbody>
</table>

Command Token Usage Considerations

- SSINIT signifies that the value for this token must be supplied in the call to SSINIT to create the SPI command.
• Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses. The distributor is not extended SPI-compliant in this regard: if multiple object instances are returned, and MAXRESP is not equal to zero, all responses are framed by one data list. The EMS collector, by comparison, puts each response (object instance) in a separate data list.

• Required signifies that the token is required and must be supplied by the user.

Common Command Token Descriptions

ZSPI-TKN-COMMAND
contains the command number. Its value is of the form ZCOM-CMD-command and is specified in the call to SSINIT.

The header token can have these values for the distributor's extended programmatic interface commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>5005</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>5003</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>5007</td>
</tr>
<tr>
<td>ZCOM-CMD-STATUS</td>
<td>8</td>
</tr>
<tr>
<td>ZEMS-CMD-REPLACE</td>
<td>30000</td>
</tr>
</tbody>
</table>

ZSPI-TKN-OBJECT-TYPE
contains the type of object to which the command is applied. Its value is of the form ZCOM-OJB-objtype and is specified in the call to SSINIT. These object types are supported:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-OBJ-DIST</td>
<td>70</td>
</tr>
<tr>
<td>ZCOM-OJB-FILTER</td>
<td>75</td>
</tr>
<tr>
<td>ZCOM-OBJ-SOURCE</td>
<td>72</td>
</tr>
<tr>
<td>ZCOM-OBJ-TARGET</td>
<td>74</td>
</tr>
<tr>
<td>ZCOM-OBJ-TEXTOUT</td>
<td>73</td>
</tr>
</tbody>
</table>

ZSPI-TKN-SSID
identifies the subsystem that processes the command. In the ZCOM-distributor commands, this subsystem is always defined as ZSPI-VAL-TANDEM.ZSPI-SSN-ZEMS.ZEMS-VAL-VERSION. This value is specified in the call to SSINIT.

ZSPI-TKN-MAXRESP
if the value of this command header token is nonzero, the distributor encapsulates the response in a data list (that is, between the -DATALIST and the -ENDLIST tokens). If multiple object instances occur, they are all enclosed in the same data list. By comparison, the collector encapsulates each object instance (response) in its own data list to be compliant with the SPI common extensions interface standards.
**ZCOM-TKN-OBJNAME**

specifies the fully qualified name of the object to be selected for processing. At least one of these tokens must be in the command. Multiple occurrences of this token (with different object names) are allowed only for the ZCOM-CMD-ADD and ZCOM-CMD-DELETE commands. A ZCOM-CMD-STATUS command with a wild-card object name can have multiple responses. If it does, all occurrences are returned and encapsulated in a data list if MAXRESP is nonzero. This process deviates from the SPI common extension standards that are observed by EMS collectors. For EMS distributors, multiple response buffers with context tokens are not supported.

When more than one object name is specified, the same command is applied to all of the named objects. If different actions are desired on each named object, you must use separate commands. These examples illustrate these guidelines:

**Note.** When an object type is ZCOM-OBJ-DIST, only one object name is allowed. For example, the ZEMS-CMD-ALTER command can only affect a single distributor.

- To change the GENTIME setting for distributor $dist1 from 8:00 to 9:00, use a ZCOM-CMD-ALTER command (where the object type is ZCOM-OBJ-DIST and the -OBJNAME value is $dist1) and one -DIST-CONTROL token map with the GENTIME stamp set to 9:00.

- To add event sources $0 and \tsii.$0 to distributor $dist2, use a ZCOM-CMD-ADD command (where the object type is ZCOM-OBJ-SOURCE) and two -OBJNAME tokens with the values $0 and \tsii.$0, respectively. Because the command is sent to $dist2, there is no ambiguity as to which main object, with its subordinate object, the command applies.

**Common Response Tokens for ZCOM- Commands**

<table>
<thead>
<tr>
<th>Response Token</th>
<th>Token Type</th>
<th>Special Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>ZSPI-TYP-ENUM</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-OBJECT-TYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-SERVER-VERSION</td>
<td>ZSPI-TYP-VERSION</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INCTL</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>
Response Token Usage Considerations

- SSINIT signifies that the value for this token must be supplied in the call to SSINIT to create the SPI command.
- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses. The EMS distributor’s extended programmatic interface deviates from the SPI common extensions standard.
- Error only signifies that this token is present only when an error or warning is reported in the response record.
- The data list (all tokens from -DATALIST to -ENDLIST) forms a single response record. The distributor does not support multiple data lists. If multiple object instances do occur, they are all returned in the same data list. The data list enclosure is present only if MAXRESP is not equal to zero.

Common Response Token Descriptions

**ZSPI-TKN-COMMAND**
contains the command number for the request to which this is the reply. This value is specified in the call to SSINIT.

**ZSPI-TKN-OBJECT-TYPE**
contains the type of object from the request to which this is the reply. This value is specified in the call to SSINIT.

**ZSPI-TKN-SSID**
identifies the subsystem that processes the command. In the following response tokens, it is always defined as ZPI-VAL-TANDEM.ZSPI-SSN-ZEMS.ZEMS-VAL-VERSION. This value is specified in the call to SSINIT.

**ZSPI-TKN-SERVER-VERSION**
contains the release version of the server (the EMS distributor) that performed the command. For this release version, it is D30.

**ZEMS-TKN-DATALIST**
indicates the beginning of a data list. Multiple object instances can be contained in the list. The ZSPI-TKN-ENDLIST token indicates the end of the data list. These tokens are used only when a non-zero value was specified in the command MAXRESP token.

**ZSPI-TKN-RETCODE**
contains the return code, which indicates the outcome of the command processing. If it is zero (ZCOM-ERR-OK), the command completed successfully without error. If
it is non-zero, an error occurred, and the response record contains an error list that describes the error.

**ZSPI-TKN-ERRLIST**

indicates the beginning of an error list. ZSPI-TKN-ENDLIST indicates the end of the error list, and the error list can be nested. An error list is put in the response for each error or warning encountered during processing of the command. ZSPI-TKN-ERROR is the only required token for the error list. Other standard or subsystem-specific tokens can also be in the error list, depending on the type of error.

**ZSPI-TKN-ERROR**

specifies the error and identifies the subsystem that prepared the error list.

### Common Definitions for ZEMS- Commands

This subsection describes some tokens and error numbers that commonly occur in several ZEMS- command messages.

#### Common Command Tokens

Common command tokens are defined here in alphabetical order. Most are defined with the command message in which they are used.

Token types that are abbreviated in italics represent SPI types. For example, `int` represents ZSPI-TYP-INT.

**ZSPI-TKN-COMMAND (type enum)**

is the standard SPI token, which can have these values for the distributor’s basic SPI interface:

- ZEMS-CMD-GETVERSION: 0
- ZEMS-CMD-STATUS: 1
- ZEMS-CMD-CONTROL: 2
- ZEMS-CMD-GETEVENT: 3

**ZSPI-TKN-ENDLIST (type ssctl)**

is the SPI token used to end lists, including the command error lists in this section of the manual.

**ZSPI-TKN-ERRLIST (type list)**

is the SPI token that begins an error list.

**ZSPI-TKN-RETCODE (type enum)**

is the standard SPI return token. For ZSPI-TKN-RETCODE values and related information, see [Section 21, Distributor Errors](#).

---

EMS Manual—426909-005

17-9
ZSPI-TKN-SERVER-BANNER (type char50)
contains the standard banner information.

ZSPI-TKN-SERVER-VERSION (type uint)
is the subsystem version number of the EMS distributor.

ZSPI-TKN-SSID (type ssid)
consists of ZSPI-VAL-TANDEM.ZSPI-SSN-ZEMS.ZSPI-VAL-VERSION, the subsystem ID for the Event Management Service.

## Distributor Errors

Response messages to distributor commands, like responses to all command messages, contain the token ZSPI-TKN-RETCODE. A command was successful if the ZSPI-TKN-RETCODE value is zero, or was unsuccessful (received an unrecoverable error) if the token value is nonzero. The response message contains an error list if an error or a situation requiring a warning has occurred.

## Error Lists

A response message contains an error list if the distributor needs to specify errors or warnings. The seriousness of the error list is indicated by the ZSPI-TKN-RETCODE value: zero indicates warnings only; nonzero values indicate unrecoverable errors.

A response message with a ZSPI-TKN-RETCODE value of zero is accompanied by an error list in either of these cases:

- The command has succeeded after several unsuccessful tries.
- The distributor has encountered a minor problem.

Several error lists—one following the other—can occur in one response message. For example, an error list that represents a serious error can be followed by another error list explaining that the distributor recovered from the first error. Several warnings—each with its own error list—can also occur in one response message without the occurrence of an error.
Distributor Error Numbers

Table 17-8 and Table 17-9 list distributor errors and separate the errors in interpreting a command message from the errors in carrying out a command.

Table 17-8. Command-Interpretation Errors

<table>
<thead>
<tr>
<th>Name (ZEMS-ERR-)</th>
<th>Error Number</th>
<th>Parm Error Token?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION</td>
<td>1001</td>
<td>No</td>
<td>Version not supported</td>
</tr>
<tr>
<td>INV-CMD</td>
<td>1002</td>
<td>No</td>
<td>Invalid command</td>
</tr>
<tr>
<td>INV-SSID</td>
<td>1003</td>
<td>No</td>
<td>Invalid subsystem ID (SPI buffer not owned by EMS)</td>
</tr>
<tr>
<td>INV-TKN</td>
<td>1004</td>
<td>Yes</td>
<td>Extra or unrecognized token</td>
</tr>
<tr>
<td>INV-VALUE</td>
<td>1005</td>
<td>Yes</td>
<td>Invalid token value</td>
</tr>
<tr>
<td>DUP-TKN</td>
<td>1006</td>
<td>Yes</td>
<td>Duplicate token code</td>
</tr>
<tr>
<td>MODE-CONFLICT</td>
<td>1007</td>
<td>Yes</td>
<td>Command issued not available for this type of distributor</td>
</tr>
<tr>
<td>INV-OBJECT</td>
<td>1008</td>
<td>Yes</td>
<td>Invalid object type</td>
</tr>
<tr>
<td>INV-OP</td>
<td>1014</td>
<td>Yes</td>
<td>Token not allowed in this context</td>
</tr>
<tr>
<td>REQ-TKN</td>
<td>1015</td>
<td>Yes</td>
<td>Missing required token</td>
</tr>
<tr>
<td>INV-HEADERTYPE</td>
<td>1016</td>
<td>No</td>
<td>Invalid SPI header type</td>
</tr>
</tbody>
</table>

Table 17-9. Examples of Distributor Operation Errors

<table>
<thead>
<tr>
<th>Name (ZEMS-ERR-)</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLL-ACCESS</td>
<td>1018</td>
<td>Distributor cannot access a collector.</td>
</tr>
<tr>
<td>LOG-ACCESS</td>
<td>1031</td>
<td>Distributor cannot access a log file.</td>
</tr>
<tr>
<td>DEST-ACCESS</td>
<td>1036</td>
<td>Printing distributor cannot access a TEXTOUT destination, or forwarding distributor cannot access a collector destination.</td>
</tr>
<tr>
<td>DEVTYPE</td>
<td>1045</td>
<td>Distributor cannot access a collector’s log file.</td>
</tr>
<tr>
<td>COLL-PROTOCOL</td>
<td>1046</td>
<td>Distributor cannot interpret a collector status reply.</td>
</tr>
<tr>
<td>COLL-DISCONNECT</td>
<td>1050</td>
<td>Distributor has disconnected the specified collector.</td>
</tr>
</tbody>
</table>

For a complete list of distributor errors and warning numbers with their error lists, see Section 21, Distributor Errors.
Distributor Command Descriptions

This subsection describes the distributor commands, including both ZCOM- and ZEMS- types. Each command description includes a summary of the command and response tokens that are used. Many of these are common command and response tokens that are described in Common Definitions for ZCOM- Commands on page 17-5 and Common Definitions for ZEMS- Commands on page 17-9.

Note. In the command responses described in this subsection, the ZSPI-TKN-COMMAND, ZSPI-TKN-OBJECT-TYPE, ZSPI-TKN-SSID, and ZSPI-TKN-SERVER-VERSION tokens are not listed. Instead, their descriptions, which apply to all command responses, are provided in Common Response Tokens for ZCOM- Commands on page 17-7.

ADD Command (ZCOM-CMD-ADD)

The ZCOM-CMD-ADD command adds objects to the EMS distributor. Supported objects for the distributor are ZCOM-OBJ-FILTER, ZCOM-OBJ-SOURCE, and ZCOM-OBJ-TEXTOUT. If a filter is added, it can be a compiled filter, a filter table, or a burst filter. A maximum of 10 filters can be added to a single distributor.

If required parameter tokens are specified, these parameter tokens must be included in the ADD command. If these parameter tokens are optional, including them in the ADD command is optional.

If an event source is added, it can either be a primary or alternate collector or an event log file.

If an event destination is added, it can be a terminal, printer, disk file, or process. A maximum of 10 event sources or event destinations can be added.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>&lt;filter parameter token(s)&gt;</td>
<td>&lt;token-dependent&gt;</td>
<td>Filter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
</tbody>
</table>

Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Required signifies that the token is required and must be supplied by the user.
- Filter signifies that the required filter parameter tokens are required only if they are defined as required parameter tokens in the filter.
- Unconditional signifies that the token always appears in the response.
- The data list (all tokens from -DATALIST to -ENDLIST) forms a single response record.
- Error signifies that this token is present only when an error or warning is reported in the response record.

For a detailed description of these command and response tokens, see Common Definitions for ZCOM- Commands on page 17-5.

**ALTER Command (ZCOM-CMD-ALTER)**

The ZCOM-CMD-ALTER command changes objects in the EMS distributor. The supported objects are distributors (ZCOM-OBJ-DIST), EMS filters (ZCOM-OBJ-FILTER), and target collectors (ZCOM-OBJ-TARGET).

The ZCOM-CMD-ALTER can also be used to allow positioning by generation time or log time, and to alter the settings of the EOF delay and sequential blocking ON and OFF parameters. All of these attributes are submitted in a control structure; that is, more than one attribute at a time can be changed. The timestamp fields must be set to zero (0) if positioning is not required. For Boolean fields, the ZDIST-<XXX>-PRESENT field must be set to FALSE if a corresponding attribute value is not present, or to TRUE if it is present. For the ALTER command with object ZCOM-OBJ-DIST, the ZCOM-TKN-OBJNAME token is optional.

**DEFINITION ZEMS-DDL-CONTROL-DIST.**

```plaintext
02 ZDIST-GMTTIME TYPE ZSPI-DDL-TIMESTAMP.
02 ZDIST-LOGTIME TYPE ZSPI-DDL-TIMESTAMP.
02 ZDIST-DELAY-PRESENT TYPE ZSPI-DDL-BOOLEAN.
02 ZDIST-EOFDELAY TYPE ZSPI-DDL-INT.
02 ZDIST-BLOCKING-PRESENT TYPE ZSPI-DDL-BOOLEAN.
02 ZDIST-SEQ-BLOCKING TYPE ZSPI-DDL-BOOLEAN.
END
```

For definitions of the DDL fields, see ZEMS-MAP-CONTROL-DIST Definitions on page 17-15.

When the object is an EMS distributor, this command functions like its basic SPI counterpart (ZEMS-CMD-CONTROL). The only additional token required is the distributor control map token (ZEMS-DDL-CONTROL-DIST).

**Command Tokens (OBJ-DIST) | Token Type | Usage**
---|---|---
ZSPI-TKN-MAXRESP | ZSPI-TYP-INT | Basic SPI
ZCOM-TKN-OBJNAME | ZSPI-TYP-STRING | Optional
ZEMS-MAP-CONTROL-DIST | ZEMS-DDL-CONTROL-DIST | Required
When the object is an EMS filter, the additional tokens are filter parameter tokens when filter parameters are specified in the filter. Each ZCOM-CMD-ALTER command resets the filter parameters for the specified filters, so all required filter parameters must have tokens present in the command. The presence of tokens for the optional filter parameters is optional.

<table>
<thead>
<tr>
<th>Command Tokens (OBJ-FILTER)</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>&lt;filter parameter token(s)&gt;</td>
<td></td>
<td>Filter</td>
</tr>
</tbody>
</table>

When the object is a target collector for a forwarding distributor, the object name must specify the name of the new target collector that is to replace the current target collector.

<table>
<thead>
<tr>
<th>Command Tokens (OBJ-TARGET)</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

**Token Usage Considerations**

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Required signifies that the token is required and must be supplied by the user.
- Optional signifies that the token is optional and need not be supplied by the user.
- Filter signifies that the required filter parameter tokens are required only if they are defined as required parameter tokens in the filter.
- The data list (all tokens from -DATALIST to -ENDLIST) forms a single response record.
- Error only signifies that this token is present only when an error or warning is reported in the response record.
For a detailed description of these command and response tokens, see Common Definitions for ZCOM- Commands on page 17-5.

**ZEMS-MAP-CONTROL-DIST Definitions**

Use this Data Definition Language (DDL) structure when configuring an EMS distributor attribute with the ZCOM-CMD-ALTER command.

```
ZEMS-MAP-CONTROL-DIST
  DEF ZEMS-DDL-CONTROL-DIST.
    02 ZDIST-GMTTIME          token-type ZSPI-DDL-TIMESTAMP.
    02 ZDIST-LOGTIME          token-type ZSPI-DDL-TIMESTAMP.
    02 ZDIST-DELAY-PRESENT    token-type ZSPI-DDL-BOOLEAN.
    02 ZDIST-EOFDELAY         token-type ZSPI-DDL-INT.
    02 ZDIST-BLOCKING-PRESENT token-type ZSPI-DDL-BOOLEAN.
    02 ZDIST-SEQ-BLOCKING     token-type ZSPI-DDL-BOOLEAN.
  END
```

ZEMS-MAP-CONTROL-DIST

is an extensible structured token that controls these distributor attributes:

**ZDIST-GMTTIME (timestamp field)**

positions the distributor at a new place in the event-message source. The distributor examines event messages that have a generation time that is the same as (or later than) ZEMS-TKN-GMTTIME. A value of -1 requests that event messages be returned as the collector logs them. For more information, see Positioning Within Event-Message Sources on page 17-22. GMTIME is the time of the event’s origin.

**ZDIST-LOGTIME (timestamp field)**

positions the distributor at a new place in the event-message source. The distributor examines only messages that were written to the log at or after the logging time given by ZEMS-TKN-LOGTIME. For more information, see Positioning Within Event-Message Sources on page 17-22. Logtime is the time when the event is written to the log; events in the log file are sorted by logtime.

**ZDIST-DELAY-PRESENT (Boolean field)**

must be set to true if the EOF delay is allowed in ZDIST-EOFDELAY.

**ZDIST-EOFDELAY (int field)**

is the time the distributor delays after end of file (EOF) before rereading the log.
ZDIST-BLOCKING-PRESENT (Boolean field)
must be set to true if the sequential blocking flag is allowed in ZDIST-SEQ-BLOCKING.

ZDIST-SEQ-BLOCKING (Boolean field)
indicates whether sequential blocking is selected for all log files except the current file associated with the collector.
CONTROL Command (ZEMS-CMD-CONTROL)

The ZEMS-CMD-CONTROL command directs a distributor to change its operational environment: to change its source of event messages (and the position within the source at which examination of event messages begins), to install or change a filter or filter parameters, and to change the destinations of event messages that pass the filter.

Note. Because the ZCOM-CMD-ADD, -ALTER, and -DELETE commands use the distributor’s object-oriented extended interface, use these commands, whenever possible, instead of the ZEMS-CMD-CONTROL command. For example, the CONTROL command can only load a single filter, and in doing so deletes all existing filters in the distributor.

<table>
<thead>
<tr>
<th>Command</th>
<th>Tokens in Command Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-CMD-CONTROL</td>
<td>ZEMS-TKN-FILTERFILE token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-REPLACE-PARAM token-type ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-RESET-FILTER token-type ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-GMTTIME token-type ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-LOGTIME token-type ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-REPLACE-TGT-COLL token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-CONNECT-SRC-COLL token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-DISCONNECT-SRC-COLL token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-ADD-TEXTOUT token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-DELETE-TEXTOUT token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-CONNECT-LOG token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-DISCONNECT-LOG token-type ZSPI-TYP-INT</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-SEQ-BLOCKING token-type ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td></td>
<td>ZSPI-TKN-RETCODE token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td></td>
<td>ZSPI-TKN-ERRLIST token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>ZSPI-TKN-ENDLIST token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Tokens in Command Buffer

ZEMS-TKN-FILTERFILE (type \textit{fname})

is the name of a file that contains the filter object program. To specify a filter, include this token. In the case of a filter table or burst filter, you can give the EDIT file name instead.
ZEMS-TKN-REPLACE-PARAM (type Boolean)

if equal to ZSPI-VAL-TRUE, directs the distributor to replace the current parameter tokens with the parameter tokens supplied by this command message.

parameter-tokens are one or more user-defined tokens: the parameter tokens for the event-message filter. See Passing Filter Parameters on page 17-21.

A CONTROL command message that contains parameter tokens must also contain a ZEMS-TKN-FILTERFILE or ZEMS-TKN-REPLACE-PARAM token.

ZEMS-TKN-RESET-FILTER (type Boolean)

if equal to ZSPI-VAL-TRUE, directs a consumer or printing distributor to replace the current filter with the default filter. The default filter for consumer or printing distributors passes all event messages. This command is not defined for forwarding distributors; if attempted, the error ZEMS-ERR-NULL-FILTER is returned.

ZEMS-TKN-GMTTIME (type timestamp)

positions the distributor at a new place in the event-message source. The distributor examines event messages that have a generation time the same as (or later than) ZEMS-TKN-GMTTIME. A value of -1 requests that event messages be returned as the collector logs them. For more information, see Positioning Within Event-Message Sources on page 17-22. GMTTIME is the time of the event’s origin.

ZEMS-TKN-LOGTIME (type timestamp)

positions the distributor at a new place in the event-message source. The distributor examines only those messages written to the log at or after the logging time given by ZEMS-TKN-LOGTIME. For more information, see Positioning Within Event-Message Sources on page 17-22. Logtime is the time when the event is written to the log; events within the log file are sorted by their logtime.

ZEMS-TKN-REPLACE-TGT-COLL (type fname)

directs a forwarding distributor to disconnect the current target collector and to connect the new target collector. Subsequently, the distributor forwards event messages to the new collector.

ZEMS-TKN-CONNECT-SRC-COLL (type fname)

names a collector whose log files will serve as a source of event messages for this distributor. This token includes a log file event message source. For more information, see Specifying Event-Message Sources on page 17-22.
**Distributor Commands and Responses**

**CONTROL Command (ZEMS-CMD-CONTROL)**

**ZEMS-TKN-DISCONNECT-SRC-COLL** *(type* \( fname \))

is the name of a collector to be disconnected (from this distributor) as a source of event messages. The distributor continues to access event messages from any remaining source collectors. See also ZEMS-TKN-CONNECT-SRC-COLL.

**ZEMS-TKN-ADD-TEXTOUT** *(type* \( fname \))

directs a printing distributor to add this name to its destination list, which contains names of devices, processes, and disk files that receive event messages to be printed. Do not use this token with consumer or forwarding distributors. See also ZEMS-TKN-DELETE-TEXTOUT.

**ZEMS-TKN-DELETE-TEXTOUT** *(type* \( fname \))

directs a printing distributor to delete this name from its destination list, which contains names of devices, processes, and disk files that receive event messages to be printed. Do not use this token with consumer or forwarding distributors. See also ZEMS-TKN-ADD-TEXTOUT.

**ZEMS-TKN-CONNECT-LOG** *(type* \( fname \))

is the name of a log file to serve as the source of event messages for this distributor. This token directs the distributor to use the specified log file as its unique source of event messages. For more information about event-message sources, see Specifying Event-Message Sources on page 17-22.

You must issue a CONTROL command that disconnects any source collectors (use ZEMS-TKN-DISCONNECT-SRC-COL) before you can use ZEMS-TKN-CONNECT-LOG.

**ZEMS-TKN-DISCONNECT-LOG** *(type* \( fname \))

is the name of the log file to be disconnected as a source of event messages. See also ZEMS-TKN-CONNECT-LOG.

**ZEMS-TKN-EOFDELAY** *(type* \( int \))

is the amount of time the distributor delays after end of file (EOF) before rereading the log.

**ZEMS-TKN-SEQ-BLOCKING** *(type* \( Boolean \))

indicates if sequential blocking is selected for all log files except the current file associated with the collector.

**Tokens in Response Buffer**

**ZSPI-TKN-RETCODE, -ERRLIST, and -ENDLIST**

are standard SPI return tokens. For descriptions, see Common Definitions for ZEMS- Commands on page 17-9.
Error-Handling Notes

Table 17-10 lists the name, number, and description of each ZEMS-CMD-CONTROL error message (or warning). The messages are grouped by the command features to which they are each related.

Each name marked with an asterisk (*) is a warning—the value of the ZSPI-TKN-RETCODE token is set to zero. The error list contains the ZSPI-TKN-ERROR token, which contains the number of the warning.

For a more complete description of each error and for a description of the error-list tokens associated with each error, see Section 21, Distributor Errors.

<table>
<thead>
<tr>
<th>Name (ZEMS-ERR-)</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter Load and Related Options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLT-FORM</td>
<td>1019</td>
<td>Bad filter format</td>
</tr>
<tr>
<td>FLT-LOAD</td>
<td>1020</td>
<td>Failed to load filter</td>
</tr>
<tr>
<td>REQ-PARM</td>
<td>1022</td>
<td>Required parameter missing</td>
</tr>
<tr>
<td>Collector Connect/Disconnect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIST-MODE</td>
<td>1024</td>
<td>Bad operation on log-file source</td>
</tr>
<tr>
<td>MAX-COLLECTOR</td>
<td>1025</td>
<td>Ten collectors already</td>
</tr>
<tr>
<td>COLLECTOR-EXISTS</td>
<td>1026</td>
<td>Already connected</td>
</tr>
<tr>
<td>COLL-NOT-FOUND</td>
<td>1027</td>
<td>Not connected; cannot disconnect</td>
</tr>
<tr>
<td>NO-POOL</td>
<td>1043</td>
<td>No space for another collector</td>
</tr>
<tr>
<td>Positioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOO-EARLY*</td>
<td>502</td>
<td>Position too far in the past</td>
</tr>
<tr>
<td>TOO-LATE*</td>
<td>503</td>
<td>Position too far in the future</td>
</tr>
<tr>
<td>EOF</td>
<td>1032</td>
<td>Last record read from log file</td>
</tr>
<tr>
<td>FORWARD-SEARCH</td>
<td>1033</td>
<td>Missing next file to search</td>
</tr>
<tr>
<td>NO-EVENT SOURCE</td>
<td>1044</td>
<td>Specify source collector</td>
</tr>
<tr>
<td>Add/Delete TEXTOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX-DEST</td>
<td>1035</td>
<td>Exceeds 10 TEXTOUT destinations</td>
</tr>
<tr>
<td>DEST-EXISTS</td>
<td>1037</td>
<td>Destination already added</td>
</tr>
<tr>
<td>DEST-NOT-FOUND</td>
<td>1038</td>
<td>Destination not found</td>
</tr>
<tr>
<td>Distributor Access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAT-ONLY</td>
<td>1052</td>
<td>Access to the specified distributor is not permitted</td>
</tr>
</tbody>
</table>

Entries marked with an asterisk (*) are warnings (use the prefix ZEMS-WRN-) rather than errors (use the prefix ZEMS-ERR-).

Installing Filters

All types of distributors (consumer, forwarding, and printing) use EMS filters. These include compiled filters, filter tables, and burst filters. If you start a consumer or printing distributor without a filter, the system provides a default filter that passes all event
messages. If you start a forwarding distributor without a filter, the distributor does not begin event-message processing until a filter is specified.

To specify a single filter, use ZEMS-TKN-FILTERFILE, which contains the name of the filter object-file. Use ZEMS-TKN-FILTERFILE to provide a first filter or to replace a previously loaded filter.

To specify multiple filters, you must use the ZCOM-CMD-ADD, -ALTER, and -DELETE commands, which support multiple objects. For more details, see Extended Programmatic Interface on page 17-2 and the ZCOM-CMD -ADD, -ALTER, and -DELETE command descriptions. The basic SPI interface supports the addition or deletion of only one filter at a time.

If you have provided a printing or consumer distributor with a filter and want to reinstate the default filter, use the ZEMS-TKN-RESET-FILTER token. The ZEMS-TKN-RESET-FILTER token is not defined for forwarding distributors. If attempted, the error ZEMS-ERROR-NULL-FILTER is returned.

When you specify a filter, you must specify any required filter parameters.

---

**Note.** For new applications, you are strongly advised to use the distributor’s extended programmatic interface commands (ZCOM-ADD, -ALTER, -DELETE) instead of the basic SPI ZEMS- commands in all situations.

---

**Passing Filter Parameters**

An application can pass filter parameters that were previously declared in a filter specification. For detailed information about these EMS filters, see Section 5, Compiled Filters.

The application uses these tokens and the ZEMS-CMD-CONTROL command to pass parameter tokens:

- ZEMS-TKN-FILTERFILE to install a new filter
- ZEMS-TKN-REPLACE-PARAM to replace the parameters of the current filter

Parameter tokens supplied in the CONTROL command message must match—in token code and subsystem ID—the parameter tokens declared in the filter specification. Optional parameter tokens (parameters declared “OPTIONAL” in the filter specification) can be supplied or omitted. Required parameters must be supplied.

Treat filter parameters as a set; you cannot supply or replace them one by one. The REPLACE-PARAM command removes the values for all existing parameters and substitutes the submitted values.

---

**Note.** For new applications, HP strongly advises using the distributor’s extended programmatic interface commands (ZCOM-ADD, -ALTER, -DELETE) instead of the basic SPI ZEMS- commands to submit filter parameters.
Specifying Event-Message Sources

At any particular time, the distributor gets event messages from either—but not both—of these event-message sources:

- Specified collectors. The distributor gets its messages from the current log files of one or more collectors that you specify (see ZEMS-TKN-CONNECT-SRC-COLL and ZEMS-TKN-DISCONNECT-SRC-COLL). The distributor reads event messages from collector log files, proceeding from one log file to the next as appropriate.

The distributor merges the event messages from the collectors by log time if more than one collector is specified.

You must use ZEMS-TKN-DISCONNECT-LOG to disconnect any saved log file before you specify any collector as an event-message source.

- A specified log file. The distributor gets its messages from the log file that you specify with ZEMS-TKN-CONNECT-LOG. You can specify a log file that has been saved or one of several log files belonging to a collector.

You must use ZEMS-TKN-DISCONNECT-SRC-COLL to disconnect all collectors as event-message sources before you specify a log file as an event-message source.

For token definitions, see CONTROL Command (ZEMS-CMD-CONTROL) on page 17-17

**Note.** For new applications, HP strongly advises using the distributor’s extended programmatic interface commands (ZCOM-ADD, -ALTER, -DELETE) instead of the basic SPI ZEMS- commands in all situations.

Positioning Within Event-Message Sources

To examine past event messages, you can use the ZEMS-TKN-GMTTIME and ZEMS-TKN-LOGTIME tokens to position the distributor—within a collector or log file source—at the earliest event message of interest. Subsequently, the distributor examines consecutive event messages from the specified position onward.

You can use ZEMS-TKN-GMTTIME to specify the time the event messages were generated or use ZEMS-TKN-LOGTIME to specify the time the event messages were logged. (Event messages are saved on a log file as they are received and are only approximately in order by generation time). With either ZEMS-TKN-GMTTIME or ZEMS-TKN-LOGTIME, the distributor positions itself at the first event message that is at (or after) the time you specify.

If specified collectors are the event-message sources, you can direct the distributor to examine messages as they are logged by the specified collectors rather than positioning the distributor by time. If never positioned by time, the distributor examines event messages as logged. Once positioned by time, you can reset the distributor to
examine event messages as they are logged by specifying a ZEMS-TKN-GMTTIME of -1F.

**Note.** If a collector has received messages from a forwarding distributor, generation time and logging time can grow farther apart. The generation time depends on the system where the message was created. The logging time depends on the system where the message is logged.

### Using Separate CONTROL Commands for Unrelated Tasks

HP recommends that you use separate ZEMS-CMD-CONTROL command messages to perform unrelated tasks. For example, use one command message to reposition the distributor and another message to replace the filter parameters.

However, you can make several configuration changes in one ZEMS-CMD-CONTROL command. For instance, you can replace a filter, provide new filter parameters, set the log position, and add one or more source collectors in the same command message.

If you submit multiple configuration changes in a single ZEMS-CMD-CONTROL command message, regardless of the order of the tokens in the buffer, the distributor performs them in this order:

1. Filter operations
2. Connects and disconnects of source collectors; adds and deletes of TEXTOUT destinations
3. Positioning

You can greatly simplify error recovery by restricting each ZEMS-CMD-CONTROL command message to the tokens of a single configuration change. For example, use one CONTROL command message to replace a filter and another to add a source collector. When you use the distributor’s extended programmatic interface, configuration changes are restricted to the associated ZCOM- commands.

### Error-Handling Notes

You must resubmit some or all of a corrected ZEMS-CMD-CONTROL command message if the value of ZSPI-TKN-RETCODE is nonzero.

Resubmit the entire (corrected) command message if it involved a single operation. If you submitted a command message that involved unrelated operations (loading a filter and adding a new TEXTOUT destination, for example), resubmit only those tokens associated with failed operations.

In some cases, the distributor can recover from a log-file access error by switching to another log file and resuming event-message processing. The original error is reported in an error list, followed by a warning—warnings indicate recovery—in another error list. ZSPI-TKN-RETCODE will be zero, indicating a successful operation.
DELETE Command (ZCOM-CMD-DELETE)

The ZCOM-CMD-DELETE command deletes objects from the EMS distributor. The supported objects are the EMS filter (ZCOM-OBJ-FILTER), event source (ZCOM-OBJ-SOURCE), and event destination (ZCOM-OBJ-TEXTOUT).

At least one object name token is required. Use of the wild-card object name (*) is also supported.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Required signifies that the token is required and must be supplied by the user.
- Unconditional signifies that the token always appears in the response.
- Error only signifies that this token is present only when an error or warning is reported in the response record.
- The data list (all tokens from -DATALIST to -ENDLIST) forms a single response record.

For a detailed description of these command and response tokens, see Common Definitions for ZCOM-Commands on page 17-5.
GETEVENT Command (ZEMS-CMD-GETEVENT)

The ZEMS-CMD-GETEVENT command gets the next event message to pass the current filter. GETEVENT applies only to consumer distributors.

**Command**

ZEMS-CMD-GETEVENT

**Tokens in Command Buffer**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Token Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ZSPI-TKN-CONTEXT</td>
<td>type ZSPI-TYP-BYTESTRING</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-EOFSTOP</td>
<td>type ZSPI-TYP-BOOLEAN</td>
</tr>
</tbody>
</table>

**Tokens in Response Buffer**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Token Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ZEMS-TKN-EVENT</td>
<td>type ZSPI-TYP-BYTESTRING</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-PASSVAL</td>
<td>type ZSPI-TYP-INT</td>
</tr>
<tr>
<td></td>
<td>ZSPI-TKN-CONTEXT</td>
<td>type ZSPI-TYP-BYTESTRING</td>
</tr>
<tr>
<td></td>
<td>ZSPI-TKN-RETCODE</td>
<td>type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td></td>
<td>ZSPI-TKN-ERRLIST</td>
<td>type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td></td>
<td>ZSPI-TKN-ENDLIST</td>
<td>type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Tokens in Command Buffer**

**ZSPI-TKN-CONTEXT (type bytestring)**

is the standard SPI context token, as returned in the previous GETEVENT response message. You can include ZSPI-TKN-CONTEXT if the GETEVENT command message asks the distributor to examine the next sequential event message, the usual case.

Do not include ZSPI-TKN-CONTEXT if this is the first GETEVENT command message or if a CONTROL command message has just changed the distributor’s position within the source of event messages.

The distributor uses the context token for integrity checking. You can omit this token.

**ZEMS-TKN-EOFSTOP (type Boolean; nonshared)**

if present with a value of ZSPI-VAL-TRUE, requests that the distributor immediately return an end-of-file warning if an event message is not currently available. For more information, see Operational Notes on page 17-26.
Tokens in Response Buffer

ZEMS-TKN-EVENT (type bytestring)  
is the actual event message.

ZEMS-TKN-PASSVAL (type int)  
if present, is the number following PASS in the PASS statement executed by the current filter. If you omit the number on the PASS statement, the distributor omits the ZEMS-TKN-PASSVAL token.

ZSPI-TKN-CONTEXT (type bytestring)  
is the new context block for the event message just returned.

ZSPI-TKN-RETCODE, -ERRLIST, and -ENDLIST  
are standard SPI return tokens. See the common tokens description in Object Support Summary on page 17-3.

Operational Notes

- The ZEMS-TKN-EOFSTOP token is a no-wait feature available when collector log files are the source of event messages. A value of ZSPI-VAL-TRUE causes the distributor to return:
  - An event message if one is immediately available.
  - A ZSPI-TKN-RETCODE value of zero and an error list with the warning error number ZEMS-WRN-EOF—indicating the application should try again later—if no event message is immediately available.

If ZEMS-TKN-EOFSTOP is missing or has the value ZSPI-VAL-FALSE, GETEVENT waits for an event message to become available before responding to your application.

- If you want to limit the time your application waits for an event message, you can use NOWAIT I/O and later cancel the GETEVENT command. That is, use the CANCEL or CANCELREQ procedure. For more information, see the Guardian Procedure Calls Reference Manual.

- If the distributor has reached the end of one collector log file and cannot find the next file, it returns a ZSPI-TKN-RETCODE value of ZEMS-ERR-LOG-ACCESS. The ZEMS-TKN-ERROR token in the accompanying error list contains the ZEMS-ERR-FILES-LOST error number. The distributor continues to operate if it finds the next available log file in the series.

If the distributor fails to find any log file in the series, it shuts down the collector and returns a ZSPI-TKN-RETCODE value of ZEMS-ERR-COLL-DISCONNECT and the error lists with the ZEMS-ERR-FILES-LOST error numbers.
## Error-Handling Notes

### Table 17-11. GETEVENT Command Errors

<table>
<thead>
<tr>
<th>Name (ZEMS-ERR-)</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF*</td>
<td>501</td>
<td>End-of-file indication sent because requested by a ZEMS-TKN-EOFSTOP token.</td>
</tr>
<tr>
<td>LOG-ACCESS</td>
<td>1031</td>
<td>The distributor cannot access an event-message log file.</td>
</tr>
<tr>
<td>EOF</td>
<td>1032</td>
<td>GETEVENT called past the end-of-file position.</td>
</tr>
<tr>
<td>CONTEXT</td>
<td>1039</td>
<td>Mismatch between context token submitted and context token saved.</td>
</tr>
<tr>
<td>BAD-FILTER</td>
<td>1042</td>
<td>Filter failed on indicated event message; see your HP representative.</td>
</tr>
<tr>
<td>NO-EVENT-SOURCE</td>
<td>1044</td>
<td>Distributor has no source collector or log file.</td>
</tr>
<tr>
<td>BAD-EVENT</td>
<td>1047</td>
<td>Indicated event message is not recognized.</td>
</tr>
<tr>
<td>STAT-ONLY</td>
<td>1052</td>
<td>Access to the specified distributor is not permitted.</td>
</tr>
</tbody>
</table>

Entries marked by an asterisk (*) are prefixed by ZEMS-WRN- (warning) rather than ZEMS-ERR- (error) and are returned in an error list in association with a ZSPI-TKN-RETCODE value of zero.
GETVERSION Command (ZEMS-CMD-GETVERSION)

The ZEMS-CMD-GETVERSION command displays the version number and banner of the EMS programmatic interface.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-CMD-GETVERSION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tokens in Command Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tokens in Response Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SERVER-BANNER</td>
</tr>
<tr>
<td>token-type ZSPI-TYP-CHAR50</td>
</tr>
<tr>
<td>ZSPI-TKN-SERVER-VERSION</td>
</tr>
<tr>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
</tr>
<tr>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
</tr>
<tr>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
</tr>
<tr>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Tokens in Response Buffer

ZSPI-TKN-RETCODE, -ERRLIST, and -ENDLIST are standard SPI return tokens. See Common Response Tokens for ZCOM-Commands on page 17-7.

GETVERSION Banner

The banner displayed by this command is of the form:

EMSDIST - T9632D40 - 01MAR99 - EMS Distributor
The ZEMS-CMD-REPLACE command replaces a configured object in the EMS distributor with another object. The only supported object is an EMS filter (ZCOM-OBJ-FILTER). The filter can be a compiled filter, a filter table, or a burst filter. Only one filter at a time can be replaced using this command.

If required parameter tokens are specified in the new filter, these parameter tokens must be included in the command. If optional parameter tokens are specified in the filter, the parameter tokens are optional.

Although the ZEMS-CMD-REPLACE does not comply with the distributor’s extended command interface and does not support multiple filter objects, it is similar to the ZCOM-ADD-FILTER command.

### Command Tokens

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZEMS-TKN-NEW-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required Filter</td>
</tr>
<tr>
<td>&lt;filter parameter token(s)&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Response Tokens

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

### Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Required signifies that the token is required and must be supplied by the user.
- Filter signifies that the required filter parameter tokens are required only if they are defined as required parameter tokens in the filter.
- Unconditional signifies that the token always appears in the response.
- Error only signifies that this token is present only when an error or warning is reported in the response record.
- If the REPLACE command was not successful, the original filter is replaced.

For a detailed description of these command and response tokens, see Common Definitions for ZCOM-Commands on page 17-5.
STATUS Command (ZCOM-CMD-STATUS)

The ZCOM-CMD-STATUS command requests both configuration and status information from the EMS distributor. The supported object types are the EMS distributor (ZCOM-OBJ-DIST), EMS filters (ZCOM-OBJ-FILTER), event source (ZCOM-OBJ-SOURCE), event destination (ZCOM-OBJ-TEXTOUT), and forwarding target (ZCOM-OJB-TARGET). The basic SPI-compliant ZEMS-CMD-STATUS command does not support object types.

Exactly one object name token is required. The wild-card object name (*) is supported. If the object type is ZCOM-OBJ-DIST and the object name token is given, information about the distributor is returned. If the object name is omitted or set to the wild card, information about the distributor and all subordinate objects is returned.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZSPI-TKN-SUB</td>
<td>ZSPI-TYP-ENUM</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Command Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Required signifies that this token is required and must be supplied by the user.
- Optional signifies that this token is optional for the user.
For a detailed description of these command tokens, see Common Definitions for ZCOM- Commands on page 17-5.

<table>
<thead>
<tr>
<th>Response Tokens When Object Type Is ZCOM-OBJ-DIST</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>ZEMS-DDL-STATUS-DIST</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-SOURCE</td>
<td>ZEMS-DDL-STATUS-SOURCE</td>
<td>Conditional</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-FILTER</td>
<td>ZEMS-DDL-STATUS-FILTER</td>
<td>Conditional</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-TEXTOUT</td>
<td>ZEMS-DDL-STATUS-TEXTOUT</td>
<td>Error only</td>
</tr>
<tr>
<td>ZEMSP-MAP-STATUS-TARGET</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens When Object Type Is ZCOM-OBJ-FILTER</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-FILTER</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens When Object Type Is ZCOM-OBJ-SOURCE</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-SOURCE</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>
Response Tokens When Object Type Is ZCOM-OBJ-TEXTOUT

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-TEXTOUT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Response Tokens When Object Type Is ZCOM-OBJ-TARGET

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-TARGET</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Response Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Conditional signifies that this information is present only if the object name was omitted or set to wild card.
- Unconditional signifies that the token always appears in the response.
- Error only signifies that this token is present only when an error or warning is reported in the response record.

For a detailed description of these response tokens, see Common Definitions for ZCOM-Commands on page 17-5.

STATUS Command Features

To receive specific information about the new features, you can use the newer object-oriented interface of the distributor (command ZCOM-CMD-STATUS). If a STATUS command is sent without an object type, the distributor returns all available information (equivalent to the existing STATUS command, which is described later in this section). If the object type is specified as ZCOM-OBJ-DIST, and the object name is either omitted or set to the wild-card character (*), an asterisk, information about all objects...
and object occurrences is returned. If the name is set to the name of the distributor, only distributor attributes are returned.

If the object type is ZCOM-OBJ-SOURCE, ZCOM-OBJ-TEXTOUT, ZCOM-OBJ-FILTER, or ZCOM-OBJ-TARGET, and the object name is either omitted or set to (*), an asterisk, information about all object instances is returned. To receive only information for a particular object instance, set the object name accordingly.

These returned structures are described in STATUS Data Structure Details on page 17-34:

ZEMS-DDL-STATUS-SOURCE
ZEMS-DDL-STATUS-TEXTOUT
ZEMS-DDL-STATUS-TARGET
ZEMS-DDL-STATUS-FILTER
ZEMS-DDL-STATUS-DIST

For ZCOM-OBJ-FILTER, the distributor returns the filter object name and the filter name, size, and type. If available, a list of parameter tokens is attached. To retrieve the parameter list for all filters, a STATUS * (wild-card) command can be issued. Filter types are:

- COMPILED
- Basic TABLE/PASS
- Basic TABLE/FAIL
- Extended TABLE/PASS
- Extended TABLE/FAIL
- Burst filter

An extended table is a filter table that includes either pass values or header token key word directives.

The ZEMS-DDL-STATUS-FILTER structure is a part of the programmatic interface to support multiple filters. If more than one filter is installed, the ZEMS-DDL-STATUS-DIST specifies the first filter in the sequence.
STATUS Data Structure Details

The data definition lists for the five token maps returned by the ZCOM-CMD-STATUS command are shown in this table:

<table>
<thead>
<tr>
<th>Token Maps in ZCOM-CMD-STATUS Response Buffer (Page 1 of 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ZEMS-MAP-STATUS-DIST</strong></td>
</tr>
<tr>
<td>DEF ZEMS-DDL-STATUS-DIST.</td>
</tr>
<tr>
<td>02 ZDIST-NAME token-type ZSPI-DDL-PROCNAME.</td>
</tr>
<tr>
<td>02 ZDIST-TYPE token-type ZSPI-DDL-ENUM.</td>
</tr>
<tr>
<td>02 ZDIST-PRIMARY-CPU token-type ZSPI-DDL-INT.</td>
</tr>
<tr>
<td>02 ZDIST-BACKUP-CPU token-type ZSPI-DDL-INT.</td>
</tr>
<tr>
<td>02 ZDIST-PRIORITY token-type ZSPI-DDL-INT.</td>
</tr>
<tr>
<td>02 ZDIST-FILTERFILE token-type ZSPI-DDL-FNAME.</td>
</tr>
<tr>
<td>02 ZDIST-FILTERNAME token-type ZEMS-DDL-CHAR30.</td>
</tr>
<tr>
<td>02 ZDIST-LOGFILE token-type ZSPI-DDL-DISCNAME.</td>
</tr>
<tr>
<td>02 ZDIST-LAST-POSITION token-type ZSPI-DDL-TIMESTAMP.</td>
</tr>
<tr>
<td>02 ZDIST-CURRENT-POSITION token-type ZSPI-DDL-TIMESTAMP.</td>
</tr>
<tr>
<td>02 ZDIST-LAST-ERROR token-type ZSPI-DDL-INT.</td>
</tr>
<tr>
<td>02 ZDIST-EVENT-PASS token-type ZSPI-DDL-INT2.</td>
</tr>
<tr>
<td>02 ZDIST-EVENT-TOTAL token-type ZSPI-DDL-INT2.</td>
</tr>
<tr>
<td>02 ZDIST-LAST-POSITION token-type ZSPI-DDL-TIMESTAMP.</td>
</tr>
<tr>
<td>02 ZDIST-LAST-ERROR token-type ZSPI-DDL-INT.</td>
</tr>
<tr>
<td>02 ZDIST-EVENT-PASS token-type ZSPI-DDL-INT2.</td>
</tr>
<tr>
<td>02 ZDIST-EVENT-TOTAL token-type ZSPI-DDL-INT2.</td>
</tr>
<tr>
<td>02 ZDIST-FILTER-CHANGE token-type ZSPI-DDL-INT.</td>
</tr>
<tr>
<td>02 ZDIST-SUSPENDED token-type ZSPI-DDL-BOOLEAN.</td>
</tr>
<tr>
<td>02 ZDIST-SEQ-BLOCKING token-type ZSPI-DDL-BOOLEAN.</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

| **ZEMS-MAP-STATUS-FILTER**                                  |
| DEF ZEMS-DDL-STATUS-FILTER.                                 |
| 02 ZDIST-FILTERFILE-NAME token-type ZSPI-DDL-FNAME.         |
| 02 ZDIST-FILTER-NAME token-type ZSPI-DDL-CHAR30.             |
| 02 ZDIST-FILTER-SIZE token-type ZSPI-DDL-UINT.               |
| 02 ZDIST-FILTER-FLAGS token-type ZSPI-DDL-UINT.              |
| END                                                         |

| **ZEMS-MAP-STATUS-SOURCE**                                  |
| DEF ZEMS-DDL-STATUS-SOURCE.                                 |
| 02 ZDIST-COLL-NAME token-type ZSPI-DDL-PROCNAME.            |
| 02 ZDIST-COLL-LOGNAME token-type ZSPI-DDL-DISCNAME.         |
| 02 ZDIST-CUR-STATE token-type ZSPI-DDL-ENUM.                |
| 02 ZDIST-CUR-IO-PENDING token-type ZSPI-DDL-ENUM.           |
| 02 ZDIST-CUR-LOGNAME token-type ZSPI-DDL-DISCNAME.          |
| 02 ZDIST-CUR-RECORD-ADDRESS token-type ZSPI-DDL-INT2.       |
| 02 ZDIST-EVENT-PASS token-type ZSPI-DDL-INT2.                |
| 02 ZDIST-EVENT-TOTAL token-type ZSPI-DDL-INT2.               |
| 02 ZDIST-LOGSTOPPED token-type ZSPI-DDL-BOOLEAN.             |
| END                                                         |
ZEMS-MAP-STATUS-DIST

is an extensible structured token that provides information about the distributor’s log files, filters, and statistical information. The command box lists all fields and field types (DEF: ZEMS-DDL-STATUS-DIST). Definitions for the fields of ZEMS-MAP-STATUS-DIST are:

ZDIST-NAME (procname field)

is the name of the distributor process.

ZDIST-TYPE (enum field)

is the distributor type, which can have the values:

- ZEMS-VAL-CONSUMER-DIST: 0
- ZEMS-VAL-FORWARD-DIST: 1
- ZEMS-VAL-PRINTING-DIST: 2

ZDIST-PRIMARY-CPU (int field)

is the number of the CPU in which the primary process of the distributor is currently running.

Note. The following descriptions for certain fields, such as ZDIST-CUR-RECORD-ADDRESS, contain implementation information that is internal to HP. Do not access log files directly or make other explicit use of such information, which is provided to help you improve your system performance. HP reserves the right to change such information without notice.
**ZDIST-BACKUP-CPU** (int field)

is the number of the CPU in which the backup process of the distributor is currently running if a backup exists. This field is set to -1 if a backup does not exist.

**ZDIST-PRIORITY** (int field)

is the current execution priority of the distributor.

**ZDIST-FILTERFILE** (fname field)

is the name of the object filter file if the filter was loaded from a disk file. This is a 24-byte internal file name.

**ZDIST-FILTERNAME** (ZEMS-DDL-CHAR30 field)

is the name of the filter, as declared in the filter specification.

**ZDIST-LOGFILE** (discname field)

if present, is the name of the log file that is currently the sole source of event messages examined by this distributor.

**ZDIST-LAST-POSITION** (timestamp field)

is the saved time (in local civil time) of the last positioning command.

**ZDIST-CURRENT-POSITION** (timestamp field)

is the log time (in local civil time) of the event message last examined by the distributor.

**ZDIST-LAST-ERROR** (int field)

is the error number of the last error encountered.

**ZDIST-EVENT-PASS** (int2 field)

is the number of event messages that have passed the current filters since the last filter load.

**ZDIST-EVENT-TOTAL** (int2 field)

is the total number of event messages processed since the last filter load.

**ZDIST-FILTER-CHANGE** (int field)

is the number of filter changes since startup of the distributor.

**ZDIST-SUSPENDED** (Boolean field)

is FALSE. (This field is not used in the current release.)
**ZDIST-EOFDELAY** (uint field)

is the time that the distributor delays after end of file (EOF) before rereading the log. This value is given in sec/100.

**ZDIST-SEQ-BLOCKING** (Boolean field)

indicates if sequential blocking is selected for all log files except the current file associated with the collector.

**ZEMS-MAP-STATUS-FILTER**

is an extensible structured token that gives detailed information about the distributor’s EMS filters (DEF: ZEMS-DDL-STATUS-FILTER). The fields of ZEMS-MAP-STATUS-FILTER are defined as follows:

**ZDIST-FILTERFILE-NAME** (fname field)

is either the object name of a compiled filter, filter table, or burst filter.

**ZDIST-FILTER-NAME** (char30 field)

is a more descriptive name for a filter. For a filter table or burst filter, this name is equivalent to the file name part (maximum of eight characters) of the filter file name (ZDIST-FILTERFILE-NAME).

**ZDIST-FILTER-SIZE** (uint field)

is the size, in bytes, of the filter object.

**ZDIST-FILTER-FLAGS** (uint field)

FLAGS <11> is set to 1 if the filter is a filter table that includes either pass values or header token key word directives. This type of filter is referred to as extended. If the flag is zero, and the filter is a filter table, then it is referred to as basic.

For <12:15>

- 0 compiled
- 1 pass
- 2 failed

For <11>

- 0 basic
- 1 extended
**ZEMS-MAP-STATUS-SOURCE**

is an extensible structured token that gives information about each source collector (DEF: ZEMS-DDL-STATUS-SOURCE). The fields of ZEMS-MAP-STATUS-SOURCE are defined as:

**ZDIST-COLL-NAME** (procname field)

is the name of the collector process if the distributor is in collector mode. Otherwise, the field is blank.

**ZDIST-COLL-LOGNAME** (discname field)

is the name of a collector log file. The collector currently logs event messages to this file.

**ZDIST-CUR-STATE** (enum field)

is the current state of the collector. Collector state can have the following values:

- **ZEMS-VAL-NO-COLL** 0 No collector
- **ZEMS-VAL-COLL-IDLE** 1 Collector idle
- **ZEMS-VAL-COLL-GETEVENT** 2 Getting an event message
- **ZEMS-VAL-COLL-CONNECT** 3 Performing status request
- **ZEMS-VAL-COLL-POSITION** 4 Positioning
- **ZEMS-VAL-COLL-LOGNAMELIST** 5 Making a logname list
- **ZEMS-VAL-COLL-EOFDELAY** 6 Delaying after EOF

**ZDIST-CUR-IO-PENDING** (enum field)

is the collector I/O operation that is currently pending. This token can have these values:

- **ZEMS-VAL-NO-IO** 0 None
- **ZEMS-VAL-READLOG** 1 Reading the collector log
- **ZEMS-VAL-COLL-STATUS** 2 Normal status on collector
- **ZEMS-VAL-COLL-EVENTWAIT** 3 Waiting for collector to indicate it has a message

**ZDIST-CUR-LOGNAME** (discname field)

is the name of the log file containing the event message last examined by the distributor.
ZDIST-CUR-RECORD-ADDRESS (int2 field)

is the record address, an entry-sequenced file address within the current log file (in ZDIST-CUR-LOGNAME), of the event message last examined by the distributor.

Assume that blknum is the block number (numbering from zero), blklen is the block length (in bytes), and recnum is the record number (within the block). Then record-address is computed as:

\[( \text{blknum} \times \text{blklen} ) + \text{recnum} \]

With a block length of 4096 bytes—the block length of all log files—the number 4098 addresses the second record in the second block.

ZDIST-EVENT-PASS (int2 field)

is the number of event messages that have passed the filters for this collector since the last filter load.

ZDIST-EVENT-TOTAL (int2 field)

is the total number of event messages processed from this collector since the last filter load.

ZDIST-LOGSTOPPED (Boolean field)

is ZSPI-VAL-TRUE if this source collector has stopped logging, but it is normally ZSPI-VAL-FALSE.

ZEMS-MAP-STATUS-TEXTOUT

is an extensible structured token that gives information about the TEXTOUT destinations of a printing distributor (DEF: ZEMS-DDL-STATUS-TEXTOUT). The fields are defined as:

ZDIST-TEXTOUT-NAME (fname field)

is the name of this TEXTOUT destination.

ZDIST-TEXTOUT-TYPE (fname field)

is the device type of this TEXTOUT destination. Obtained from DEVICEINFO, it can have these values:

0    PROCESS
1    CONSOLE
2    not used
3    DISKFILE
4    TAPE
5    PRINTER
6    TERMINAL
or any other device type number
ZDIST-TEXTOUT-STATE (uint field)
is the state of this TEXTOUT destination, which can have these values:

- ZEMS-VAL-NO-TEXTOUT 0 Removed (after error)
- ZEMS-VAL-TXT-IDLE 1 Idle
- ZEMS-VAL-TXT-WAIT-ERROR 2 Waiting (after error)
- ZEMS-VAL-TXT-WAIT-TIMEDOUT 3 Waiting (timed out)
- ZEMS-VAL-TXT-PRINTING 4 Printing

ZDIST-TEXTOUT-RECLEN (uint field)
is the record length in bytes of the TEXTOUT destination.

ZEMS-MAP-STATUS-TARGET
is an extensible structured token that gives information about a target collector if
this is a forwarding distributor (DEF: ZEMS-DDL-STATUS-TARGET). The fields are
defined as:

ZDIST-TARGET-NAME (fname field)
is the name of the target collector.

ZDIST-TARGET-EVENTS-PER-BLOCK (uint field)
is the average number of event messages that the distributor has sent in a
block to this target collector.

ZDIST-TARGET-STATE (uint field)
is the state of the target collector, which can have these values:

- ZEMS-VAL-NO-TARGET 0 Target disconnected
- ZEMS-VAL-TGT-WRITE-PENDING 1 Write pending
- ZEMS-VAL-TGT-IDLE 2 Target collector idle
- ZEMS-VAL-TGT-RETRY 3 Retrying after an error
STATUS Command (ZEMS-CMD-STATUS)

The ZEMS-CMD-STATUS command gets information about several topics, each of which the distributor represents as one or more of the same extensible structured tokens that are described in STATUS Data Structure Details on page 17-34 for ZCOM-CMD-STATUS.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-CMD-STATUS</td>
</tr>
</tbody>
</table>

**Tokens in Command Buffer**

None

**Tokens in Response Buffer**

```
ZEMS-MAP-STATUS-DIST
DEF ZEMS-DDL-STATUS-DIST.
  02 ZDIST-NAME            token-type ZSPI-DDL-PROCNAME.
  02 ZDIST-TYPE            token-type ZSPI-DDL-ENUM.
  02 ZDIST-PRIMARY-CPU     token-type ZSPI-DDL-INT.
  02 ZDIST-BACKUP-CPU      token-type ZSPI-DDL-INT.
  02 ZDIST-PRIORITY        token-type ZSPI-DDL-INT.
  02 ZDIST-FILTERFILE      token-type ZSPI-DDL-FNAME.
  02 ZDIST-FILTERNAME      token-type ZEMS-DDL-CHAR30.
  02 ZDIST-LOGFILE         token-type ZSPI-DDL-DISCNAME.
  02 ZDIST-LAST-POSITION   token-type ZSPI-DDL-TIMESTAMP.
  02 ZDIST-CURRENT-POSITION token-type ZSPI-DDL-TIMESTAMP.
  02 ZDIST-LAST-ERROR      token-type ZSPI-DDL-INT.
  02 ZDIST-EVENT-PASS     token-type ZSPI-DDL-INT2.
  02 ZDIST-EVENT-TOTAL    token-type ZSPI-DDL-INT2.
  02 ZDIST-FILTER-CHANGE  token-type ZSPI-DDL-INT.
  02 ZDIST-SUSPENDED      token-type ZSPI-DDL-BOOLEAN.
  02 ZDIST-EOFDELAY        token-type ZSPI-DDL-BOOLEAN.
END
```

**Tokens in Response Buffer**

For the ZCOM-CMD-STATUS command, see the token descriptions in STATUS Data Structure Details on page 17-34.
This section describes event messages generated by printing or forwarding distributors and are related to their operation. (Consumer distributors do not generate event messages.)

Distributor event messages help you monitor and manage network resources from a ViewPoint console or through an application. Two examples of distributor events that require operational attention are a bad log file or a bad printer destination (for a print distributor).

The type of distributor determines, to some extent, which events cause the generation of an event message. For example, when a printing distributor cannot send event-message text to a print device, it generates an event message. Similarly, when a forwarding distributor cannot send an event message to a collector, it generates an event message. A consumer distributor reports its problems through an error list returned to the controlling application. Therefore, consumer distributors generate no event messages.

Certain distributor event messages are critical (their ZEMS-TKN-EMPHASIS tokens are set to ZSPI-VAL-TRUE). An event that threatens an event-message source—such as the destruction of a log file or the failure of a collector to deliver its event messages—triggers a critical event message.

Many distributor event messages are not critical but require some response from the operations environment. In addition, some event messages are only informative.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Message Descriptions</td>
<td>18-3</td>
</tr>
<tr>
<td>Token and Data Type Definitions</td>
<td>18-3</td>
</tr>
</tbody>
</table>
Table 18-1. Distributor Event Messages

<table>
<thead>
<tr>
<th>Event Number</th>
<th>Symbolic Name (ZEMS-EVT-)</th>
<th>Description</th>
<th>Critical</th>
</tr>
</thead>
<tbody>
<tr>
<td>538</td>
<td>BURST-START</td>
<td>Event burst detected</td>
<td>Yes</td>
</tr>
<tr>
<td>539</td>
<td>BURST-END</td>
<td>Event burst ended</td>
<td>No</td>
</tr>
<tr>
<td>1000</td>
<td>LOG-ACCESS</td>
<td>Cannot access log file</td>
<td>Yes</td>
</tr>
<tr>
<td>1001</td>
<td>COLL-ACCESS</td>
<td>Cannot access collector</td>
<td>Yes</td>
</tr>
<tr>
<td>1002</td>
<td>DEST-ACCESS</td>
<td>Cannot access destination</td>
<td>No</td>
</tr>
<tr>
<td>1003</td>
<td>LOGFILE-EOF</td>
<td>Reached end of log file</td>
<td>No</td>
</tr>
<tr>
<td>1005</td>
<td>BAD-FILTER</td>
<td>Encountered filter error</td>
<td>No</td>
</tr>
<tr>
<td>1006</td>
<td>COLL-PROTOCOL</td>
<td>Received bad collector response</td>
<td>Yes</td>
</tr>
<tr>
<td>1007</td>
<td>BAD-EVENT</td>
<td>Read bad event message</td>
<td>No</td>
</tr>
<tr>
<td>1008</td>
<td>DEVTYPE</td>
<td>Detected bad device</td>
<td>No</td>
</tr>
<tr>
<td>1009</td>
<td>INTERNAL-ERROR</td>
<td>Encountered internal error</td>
<td>No</td>
</tr>
<tr>
<td>1010</td>
<td>CHECKOPEN-FAILED</td>
<td>Notified of bad backup</td>
<td>No</td>
</tr>
<tr>
<td>1011</td>
<td>TAKEOVER</td>
<td>Switched control to backup</td>
<td>No</td>
</tr>
<tr>
<td>1012</td>
<td>CREATEBACKUP-FAILED</td>
<td>Could not create backup</td>
<td>No</td>
</tr>
<tr>
<td>1013</td>
<td>BACKUP-CREATED</td>
<td>Created backup process</td>
<td>No</td>
</tr>
<tr>
<td>1014</td>
<td>BACKUP-ABENDED</td>
<td>Terminated backup process</td>
<td>No</td>
</tr>
<tr>
<td>1015</td>
<td>BACKUP-DELETED</td>
<td>Missing backup process</td>
<td>No</td>
</tr>
<tr>
<td>1016</td>
<td>CHECKPOINT-FAILED</td>
<td>Encountered I/O error</td>
<td>No</td>
</tr>
<tr>
<td>1017</td>
<td>BAD-LOG</td>
<td>Detected bad log file</td>
<td>No</td>
</tr>
<tr>
<td>1018</td>
<td>FILES-LOST</td>
<td>Detected missing log file</td>
<td>Yes</td>
</tr>
<tr>
<td>1019</td>
<td>COLL-DISCONNECT</td>
<td>Gave up on this collector</td>
<td>Yes</td>
</tr>
<tr>
<td>1020</td>
<td>STARTUP-FAILED</td>
<td>Cannot create destination process</td>
<td>No</td>
</tr>
<tr>
<td>1021</td>
<td>STARTUP-OK</td>
<td>Destination successfully started</td>
<td>No</td>
</tr>
<tr>
<td>1022</td>
<td>WRITE-FAILED</td>
<td>Write to destination failed</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Event Message Descriptions**

Every event-message description in this section contains:

- The event-message display text—the text that the EMSTEXT procedure generates if you select normal text style (For more information, see EMSTEXT Procedure on page 15-19)
- A list of unconditional tokens—tokens that are always in the message
- A list of conditional tokens, if any, that are present under certain conditions
- The cause of the event
- The effect that the event has on the distributor and/or system
- The recovery action an operator should take to respond to the event message

**Note.** The descriptions of tokens such as ZEMS-TKN-RECORD-ADDRESS or ZEMS-TKN-BLOCKLENGTH contain implementation information that is internal to HP. Do not access log files directly or make other explicit use of such information, which is included to help you improve the performance of your system. HP reserves the right to change such information without notice.

**Token and Data Type Definitions**

Distributor event messages include some tokens and values defined by SPI (those prefixed by ZSPI), as well as some defined by EMS (those prefixed by ZEMS). The tokens described here are common to several distributor event messages.

**SPI Token Codes**

These SPI tokens occur in many event messages:

- **ZSPI-TKN-SSID** (type: ZSPI-TYP-SSID)
  
  contains ZSPI-SSID-ZEMS, the subsystem ID for the Event Management Service.

- **ZSPI-TKN-PROC-ERR** (type: ZSPI-TYP-ENUM)
  
  specifies a procedure associated with the event. (The exact association depends on the particular event and is described with the related event message.) EMS assigns ZSPI-TKN-PROC-ERR one of these values:

  - ZEMS-VAL-FILTER-EVAL 20
  - ZEMS-VAL-ZFILWAITIO 21
  - ZEMS-VAL-ZFILOPEN 22
  - ZEMS-VAL-ZFILPOSITION 23
  - ZEMS-VAL-ZFILREAD 24
  - ZEMS-VAL-ZFILWRITE 25
  - ZEMS-VAL-ZFILWRITEREAD 26
  - ZEMS-VAL-FILTER-READ 27
EMS Token Codes

These are definitions of the EMS tokens used in distributor event messages (EMS tokens common to event messages from all subsystems are excluded):

ZEMS-MAP-STATUS-DIST (DEF: ZEMS-DDL-STATUS-DIST)

is an extensible structured token used to return status information in the response message for the distributor STATUS command. This token contains many fields concerning the distributor process and its current state.

ZEMS-TKN-BLOCKLENGTH (type: ZSPI-TYP-INT; nonshared)

is the block length of a log file.

ZEMS-TKN-DIST-NAME (type: ZSPI-TYP-FNAME; nonshared)

is the name of the distributor reporting the event.

ZEMS-TKN-COLNAME (type: ZSPI-TYP-FNAME; nonshared)

if specified, is the name of a collector associated with the distributor reporting the event; otherwise it is blank. See ZEMS-TKN-COLNAME-ENUM.

ZEMS-TKN-COLNAME-ENUM (type: ZSPI-TYP-ENUM; nonshared)

indicates the presence of a collector name in ZEMS-TKN-COLNAME:

ZEMS-VAL-COLNAME-PRESENT 0
ZEMS-VAL-COLNAME-NOTPRESENT 1

ZEMS-TKN-DEVTYPE-ENUM (type: ZSPI-TYP-ENUM; nonshared)

uses one of these values to indicate a device-type problem with either a log file (0 to 3) or a collector process (4 or 5):

ZEMS-VAL-FILECODE-BAD 0
ZEMS-VAL-BLOCKLENGTH-BAD 1
ZEMS-VAL-DEVICE-TYPE-BAD 2
ZEMS-VAL-LOGNAME-BAD 3
ZEMS-VAL-VERSION-INCOMPATIBLE 4
ZEMS-VAL-DUP-SOURCE-TARGET 5
ZEMS-TKN-FAILFILENAME (type: ZSPI-TYP-FNAME; nonshared)

is the file name of a bad log file.

ZEMS-TKN-FILECODE (type: ZSPI-TYP-INT; nonshared)

is the file code of a log file.

ZEMS-TKN-FILTER-ERROR (type: ZSPI-TYP-ENUM; nonshared)

is a token, private to HP, that might be useful to your HP representative.

ZEMS-TKN-FILTERNAME (type: ZEMS-TYP-CHAR30; nonshared)

is the name of the filter given in the filter specification at compilation time.

ZEMS-TKN-DEVICE-TYPE (type: ZSPI-TYP-INT; nonshared)

is the device-type of a device associated with the distributor reporting the event.

ZEMS-TKN-FAIL-REASON (type: ZSPI-TYP-ENUM; nonshared)

indicates why the distributor failed to access a collector log file older than the current one. One of these values is used to indicate a device-type problem with a log file (0 to 3):

- ZEMS-VAL-EVENT-GENERATED 0
- ZEMS-VAL-NO-LINK 1
- ZEMS-VAL-BAD-LINK 2

ZEMS-TKN-LASTLOGFILE (type: ZSPI-TYP-FNAME; nonshared)

is the name of a collector log file, older than the current one, that the distributor tried unsuccessfully to access.

ZEMS-TKN-LOGNAME (type: ZSPI-TYP-FNAME; nonshared)

is the file name of the log file in use when the event occurred.

ZEMS-TKN-NEWLOGFILE (type: ZSPI-TYP-FNAME; nonshared)

is a conditional token containing the name of a collector log file—later in sequence than an inaccessible log file—that the distributor did access.

ZEMS-TKN-NEWPROCESS-CPU (type: ZSPI-TYP-INT; nonshared)

is the CPU number in a NEWPROCESS command.

ZEMS-TKN-NEWPROCESS-ERROR (type: ZSPI-TYP-ENUM; nonshared)

is one of these errors that occurred while executing a NEWPROCESS command:

- ZEMS-VAL-NO-ERROR 0
- ZEMS-VAL-UNDEFINED-EXTERNALS 1
- ZEMS-VAL-NO-PCB-AVAILABLE 2
Distributor Event Messages

EMS Token Codes

ZEMS-VAL-PROGRAMFILE-ERROR 3
ZEMS-VAL-NO-MAP 4
ZEMS-VAL-BAD-SWAPFILE 5
ZEMS-VAL-BAD-FILE-FORMAT 6
ZEMS-VAL-UNLICENSED 7
ZEMS-VAL-BAD-PROCESS-NAME 8
ZEMS-VAL-LIBRARY-CONFLICT 9
ZEMS-VAL-MONITOR-COMM 10
ZEMS-VAL-LIBRARY-FILE 11
ZEMS-VAL-PROGRAM-FILE 12
ZEMS-VAL-NO-EXT-SEGMENT 13
ZEMS-VAL-EXT-SEGMENT-SWAP 14
ZEMS-VAL-BAD-HOMETERM 15

ZEMS-TKN-NEWPROCESS-PRIORITY (type: ZSPI-TYP-INT; nonshared)
is the new process priority in a NEWPROCESS command.

ZEMS-TKN-PROGRAMFILE (type: ZSPI-TYP-FNAME; nonshared)
is the file name of a program file.

ZEMS-TKN-RECORD-ADDRESS (type: ZSPI-TYP-INT2; nonshared)
is an entry-sequenced file address. Suppose that $blknum$ is the block number (numbering from zero), $blklen$ is the block length (in bytes), and $recnum$ is the record number (within the block). Then $record\text{-}address$ is computed as:

$$ (blknum \times blklen) + recnum $$

ZEMS-TKN-RECOVERY-ENUM (type: ZSPI-TYP-ENUM; nonshared)
indicates whether the distributor recovered—after failing to access a collector log file older than the current one—by accessing a more recent log file:

- ZEMS-VAL-RECOVERY-OK 0
- ZEMS-VAL-NO-RECOVERY 1

ZEMS-TKN-TAKEOVER-REASON (type: ZSPI-TYP-ENUM; nonshared)
is the reason for a takeover by the distributor backup CPU:

- ZEMS-VAL-PRIMARY-STOPPED 0
- ZEMS-VAL-PRIMARY-ABEND 1
- ZEMS-VAL-PRIMARY-CPU-DOWN 2
- ZEMS-VAL-CHECKSWITCH 3

ZEMS-TKN-ZFILERR (type: ZSPI-TYP-UINT; nonshared)
is the error code associated with an event.
### Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-EVT-NUM</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-CODE</td>
<td>ZSPI-TYP-TOKENCODE</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-VALUE</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-TIME-START</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-MAP-BDS-INFO</td>
<td>ZEMS-DDL-BDS-INFO</td>
</tr>
</tbody>
</table>

### Event-Message Text

EMS: EVENT BURST DETECTED FOR EVENT NO. `eventno` OF SUBSYSTEM `ssid`, BY DISTRIBUTOR `proc-desc`

### Unconditional Tokens

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

**ZEMS-TKN-EVENTNUMBER** (shared)

is the event number. Its value is ZEMS-EVT-BURST-START (538).

**ZEMS-TKN-EMPHASIS** (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.

**ZEMS-TKN-CONSOLE-PRINT** (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR

is the EMS process type. Value 1 is for the primary collector, 2 for an alternate collector, and 3 for a distributor.

ZEMS-TKN-BURST-EVT-NUM

is the event number of the bursting event.

ZEMS-TKN-BURST-SSID

is the subsystem ID of the bursting event.

ZEMS-TKN-BURST-SUBJ-CODE

is the token code of the first subject in the bursting event.

ZEMS-TKN-BURST-SUBJ-VALUE

is the value of the first subject of the bursting event.

ZEMS-TKN-BURST-SUBJ-SSID

is the subsystem ID of the first subject of the bursting event.

ZEMS-TKN-BURST-TIME-START

is the Julian timestamp of the burst start time.

ZEMS-MAP-BDS-INFO

contains the burst detection and suppression (BDS) configuration parameters in effect when the burst was detected.

Text Values

eventno

is from ZEMS-TKN-BURST-EVT-NUM.

ssid

is from ZEMS-TKN-BURST-SSID.
proc-desc

is from ZEMS-TKN-BURST-PROC-DESC.

**Cause.** An event burst was detected. In terms of BDS configuration parameters, N similar events have occurred within the T1 time interval. Similar events are those that have the same SSID, event number, and subject. The L burst parameter defines what “same subject” means. For a detailed description of BDS configuration parameters and their default values, see Section 7, Burst Detection and Suppression.

**Effect.** Until the event burst ends, subsequent occurrences of these similar events are counted and discarded.

**Recovery.** Examine the event to confirm that it is a repetitive event. If it is, suppress the event by preventing it from being generated. Use the timestamp stored in the BURST-TIME-START token as a unique burst ID for programmed operators that need to verify that an event burst has ended.

If the event is useful, but not its repetition, change the event sender so it does not send out the same event repeatedly.

If all occurrences of the bursting event are useful, the operator should change the BDS configuration parameters so that bursts are not suppressed.
Unconditional Tokens

<table>
<thead>
<tr>
<th>Token</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-EVT-NUM</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-CODE</td>
<td>token-type ZSPI-TYP-TOKENCODE.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-VALUE</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-TIME-START</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-TIME-END</td>
<td>token-type ZEMS-DDL-BDS-INFO.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-END</td>
<td>token-type ZEMS-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-EVTS-DELETED</td>
<td>token-type ZEMS-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-END-REASON</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Event-Message Text

If `ZEMS-TKN-BURST-END-REASON = ZEMS-VAL-BDS-DISABLED`, then the event message text is:

EMS: BURST SUPPRESSION TERMINATED: count OCCURRENCES OF EVENT NO. eventno OF SUBSYSTEM ssid WERE NOT PROCESSED BY DISTRIBUTOR proc-desc.

If `ZEMS-TKN-BURST-END-REASON = ZEMS-VAL-NO-EVENTS`, then the event message text is:

EMS: BURST END DETECTED: count OCCURRENCES OF EVENT NO. eventno OF SUBSYSTEM ssid WERE NOT PROCESSED BY DISTRIBUTOR proc-desc.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.
ZEMS-TKN-EVENTNUMBER (shared) is the event number. Its value is ZEMS-EVT-BURST-END (539).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR

is the EMS process type. Value 1 is for the primary collector, 2 for an alternate collector, and 3 for a distributor.

ZEMS-TKN-BURST-EVT-NUM

is the event number of the bursting event.

ZEMS-TKN-BURST-SSID

is the subsystem ID of the bursting event.

ZEMS-TKN-BURST-SUBJ-CODE

is the token code of the first subject in the bursting event.

ZEMS-TKN-BURST-SUBJ-VALUE

is the value of the first subject of the bursting event.

ZEMS-TKN-BURST-SUBJ-SSID

is the subsystem ID of the first subject of the bursting event.

ZEMS-TKN-BURST-TIME-START

is the Julian timestamp of the burst start time.
ZEMS-TKN-BURST-TIME-END
is the Julian timestamp of the burst end time.

ZEMS-TKN-BURST-EVTS-DELETED
are occurrences of the bursting event that were not logged during the event burst.

ZEMS-TKN-BURST-END-REASON
is the reason for the event burst end. A value of ZEMS-VAL-BDS-DISABLED
means the burst ended because BDS was terminated by the operator. A value of
ZEMS-VAL-NO-EVENTS means the burst ended because no further events were
detected during the last T2 time units.

Text Values

eventno
is from ZEMS-TKN-BURST-EVT-NUM.

ssid
is from ZEMS-TKN-BURST-SSID.

proc-desc
is from ZEMS-TKN-BURST-PROC-DESC.

count
is from ZEMS-TKN-BURST-EVTS-DELETED.

Cause. An event burst ended for one of these reasons:
  ● The bursting event has not occurred for the time interval T2 specified in the
    BDS configuration.
  ● BDS was disabled while bursting events were being suppressed, in which case
    a ZEMS-EVT-BURST-END event is generated for every monitored event burst.

Effect. The distributor stops suppressing future occurrences of the formerly bursting
event.

Recovery. None.

Note. The timestamp stored in the BURST-TIME-START token matches the timestamp stored
in the corresponding burst start event.
1000: ZEMS-EVT-LOG-ACCESS

### Unconditional Tokens

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>ZEMS-DDL-STATUS-DIST</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME-ENUM</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
</tbody>
</table>

### Event-Message Text

EMS: EVENT LOG logname CANNOT BE ACCESSED - GUARDIAN ERROR zfilerr (, COLLECTOR colname ), (ACCESSSED AS LOGFILE), USING PROCEDURE proc-name

**Note:** proc-name is the name of the procedure specified by proc-err.

### Unconditional Tokens

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

**ZEMS-TKN-EVENTNUMBER (shared)**

is the event number. Its value is ZEMS-EVT-LOG-ACCESS (1000).

**ZEMS-TKN-EMPHASIS (shared)**

is a standard EMS token. For more information, see [Section 14, EMS Definitions](#). Its value here is TRUE.

**ZEMS-TKN-CONSOLE-PRINT (shared)**

is a standard EMS token. For more information, see [Section 14, EMS Definitions](#). Its value here is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-LOGNAME (nonshared)

is the file name of a log file. Here it is the log file in use when the event occurred.

ZSPI-TKN-PROC-ERR (nonshared)

specifies a procedure associated with the event. Here EMS assigns one of these values:

ZEMS-VAL-ZFILOPEN  22
ZEMS-VAL-ZFILPOSITION  23
ZEMS-VAL-ZFILREAD  24

The extensible structured token ZEMS-MAP-STATUS-DIST and tokens ZEMS-TKN-ZFILERR, -COLNAME-ENUM, and COLNAME are described in Token and Data Type Definitions on page 18-3.

Cause. The distributor is using the log files of a collector as its source of event messages. While proceeding from one file to the next, the distributor finds it cannot access a log file. A security violation might have occurred, or a log file might have been purged.

Effect. The distributor closes the old log file and tries to access the file that follows the missing log file in the file series. If no other log file exists, the distributor disconnects the collector. See also 1019: ZEMS-EVT-COL-DISCONNECT on page 18-49.

Recovery. If a security violation occurred or a log file was purged, correct the problem. Otherwise, contact your service provider. After you make the log file available, rerun any applications that depend on it.
1001: ZEMS-EVT-COLL-ACCESS

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME-ENUM</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>ZEMS-DDL-STATUS-DIST</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: COLLECTOR colname CANNOT BE ACCESSED - GUARDIAN ERROR zfilerr, USING PROCEDURE proc-name

Note: proc-name is the name of the procedure specified by proc-err.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-COLL-ACCESS (1001).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.
Distributor Event Messages

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLNAME (nonshared)

is the name of a collector. Here, it is a collector that the distributor cannot access.

ZEMS-TKN-COLNAME-ENUM (nonshared)

is described in Token and Data Type Definitions on page 18-3.

ZSPI-TKN-PROC-ERR (nonshared)

specifies a procedure associated with the event. Here, EMS assigns one of these values:

ZEMS-VAL-ZFILOPEN 22
ZEMS-VAL-ZFILWRITEREAD 26

The extensible structured token ZEMS-MAP-STATUS-DIST and the token ZEMS-TKN-ZFILERR are described in Token and Data Type Definitions on page 18-3.

Cause. The distributor tries to access a collector—often to get an event message from one of the collector’s log files—and finds it cannot do so.

Effect. The distributor disconnects the failing collector but continues to run. (An application can direct the distributor to reconnect the same collector after the problem has been corrected.)

Recovery. If this error was caused by a security violation, restart the collector, then reconnect it. Otherwise, contact your service provider.
Unconditional Tokens

ZSPI-TKN-SSID  
ZSPI-TYP-SSID.

ZEMS-TKN-EVENTNUMBER  
ZSPI-TYP-INT.

ZEMS-TKN-EMPHASIS  
ZSPI-TYP-BOOLEAN.

ZEMS-TKN-CONSOLE-PRINT  
ZSPI-TYP-BOOLEAN.

ZEMS-TKN-LOGTIME  
ZSPI-TYP-TIMESTAMP.

ZEMS-TKN-CPU  
ZSPI-TYP-INT.

ZEMS-TKN-PROC-DESC  
ZSPI-TYP-STRING.

ZEMS-TKN-PIN  
ZSPI-TYP-INT.

ZEMS-TKN-NODENUM  
ZSPI-TYP-INT2.

ZEMS-TKN-USERID  
ZSPI-TYP-UINT.

ZEMS-TKN-SUBJECT-MARK  
ZSPI-TYP-SSCTL.

ZEMS-TKN-FAILFILENAME  
ZSPI-TYP-FNAME.

ZSPI-TKN-PROC-ERR  
ZSPI-TYP-ENUM.

ZEMS-MAP-STATUS-DIST  
ZEMS-DDL-STATUS-DIST.

ZEMS-TKN-ZFILERR  
ZSPI-TYP-UINT.

Event-Message Text

EMS: failfilename CANNOT BE ACCESSED FOR
  { PRINTING }
  { FORWARDING }
  - GUARDIAN ERROR zfilerr

Unconditional Tokens

ZSPI-TKN-SSID
  is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)
  is the event number. Its value is ZEMS-EVT-DEST-ACCESS (1002).

ZEMS-TKN-EMPHASIS (shared)
  is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)
  is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
  are standard EMS tokens described in Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-FAILFILENAME (nonshared)
is the file name of a bad file. Here it is the name of a destination device or process that the distributor could not access.

ZSPI-TKN-PROC-ERR (nonshared)
specifies a procedure associated with the event. Here EMS assigns one of these values:

ZEMS-VAL-ZFILOPEN 22
ZEMS-VAL-ZFILWRITE 25

The extensible structured token ZEMS-MAP-STATUS-DIST and tokens ZEMS-TKN-FAILFILENAME and -ZFILERR are described in Token and Data Type Definitions on page 18-3.

Cause. There are two possible causes: (1) a printing distributor cannot access a text destination, or (2) a forwarding distributor cannot access the collector to which event messages are forwarded.

Effect. The distributor performs, depending on its type:

- A forwarding distributor that tried unsuccessfully to open a target collector tries again periodically. The distributor resumes forwarding messages if a later attempt succeeds.

- A printing distributor that tried to communicate with a TEXTOUT destination and timed-out does one of:
  - If another TEXTOUT destination is operating, the distributor disconnects the current TEXTOUT destination.
  - If no other TEXTOUT destination is operating, the distributor tries periodically to communicate with the current one.

A printing distributor that has had communications problems other than time-out problems with a TEXTOUT destination tries again periodically.

An application can connect a disconnected TEXTOUT destination when the problem is corrected.

Recovery. If the distributor cannot access a collector or a destination, check for a security violation. If the distributor cannot access a printer, check the printer.

If another destination is available, disconnect the problem destination. After you fix the problem destination, reconnect it to the system.

If you cannot correct the problem, contact your service provider.
1003: ZEMS-EVT-LOGFILE-EOF

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>token-type ZEMS-DDL-STATUS-DIST.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: EOF ENCOUNTERED FOR EVENT LOG logname

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-LOGFILE-EOF (1003).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:
ZEMS-TKN-LOGNAME *(nonshared)*

is the file name of a log file. Here, it is the log file in use when the end-of-file was encountered.

ZEMS-MAP-STATUS-DIST

is an extensible structured token described in *Token and Data Type Definitions* on page 18-3.

**Cause.** The distributor reached an end-of-file while using a specified log file as its event-message source. The distributor generates this event message in this situation only if an application connected the log file.

**Effect.** The distributor continues to run. If the log file was connected through a distributor startup option, the distributor issues a startup message and stops.

**Recovery.** This event message informs the application supplying the log-file source that the distributor is finished with the file. The distributor is ready for the application to supply another event-message source.
1005: ZEMS-EVT-BAD-FILTER

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-FILTERNAME</td>
<td>token-type ZEMS-TYP-CHAR30.</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>token-type ZEMS-DDL-STATUS-DIST.</td>
</tr>
<tr>
<td>ZEMS-TKN-RECORD-ADDRESS</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-FILTER-ERROR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME-ENUM</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: FILTER ERROR filter-error ON FILTER filtername,
EVENT FROM logname {, COLLECTOR colname } {(ACCESSSED AS LOGFILE)}

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-BAD-FILTER (1005).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-FILTERNAME (nonshared)

is the name of the filter given in the filter specification at compilation time. Here it is the filter used by the distributor when it detected the filter error.

ZSPI-TKN-PROC-ERR (nonshared)

specifies a procedure associated with the event. Here EMS assigns one of these values:

ZEMS-VAL-ZFILOPEN           22
ZEMS-VAL-ZFILPOSITION       23
ZEMS-VAL-ZFILREAD           24

The extensible structured token ZEMS-MAP-STATUS-DIST and the tokens -RECORD-ADDRESS, -FILTER-ERROR, ZEMS-TKN-LOGNAME, -COLNAME-ENUM, and -COLNAME are described in Token and Data Type Definitions on page 18-3.

Cause. The distributor detected a filter error while filtering event messages.

Effect. The distributor suspends event-message processing.

Recovery. Load a new filter or change the filter parameters. If you still have problems, contact your HP representative.
1006: ZEMS-EVT-COLL-PROTOCOL

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-COLL-PROTOCOL (1006).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

Event-Message Text

EMS: INVALID RESPONSE FROM COLLECTOR colname USING PROCEDURE proc-name, COLLECTOR WAS CLOSED

Note: proc-name is the name of the procedure specified by proc-err.
ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLNAME (nonshared)

is the name of a collector. Here it is the collector that responded to the distributor with an invalid SPI response.

ZSPI-TKN-PROC-ERR (nonshared)

specifies a procedure associated with the event. Here EMS assigns the value of ZSPI-VAL-SSGET to 2.

ZEMS-MAP-STATUS-DIST is an extensible structured token described in Token and Data Type Definitions on page 18-3.

Cause. The distributor detected an invalid response to a command message issued to a collector. The collector is malfunctioning or has aborted.

Effect. The distributor disconnects the failing collector but continues to run.

Recovery. Make sure you specified a valid EMS collector name. If you did specify a valid name, contact your service provider. An application can direct the distributor to reconnect the same collector after the collector is again operational.
## 1007: ZEMS-EVT-BAD-EVENT

### Unconditional Tokens

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>ZEMS-DDL-STATUS-DIST</td>
</tr>
<tr>
<td>ZEMS-TKN-RECORD-ADDRESS</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME-ENUM</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
</tbody>
</table>

### Event-Message Text

```plaintext
EMS: BAD EVENT AT RECORD ADDRESS record address
IN LOG FILE logname {, COLLECTOR colname },
{(ACCESSSED AS LOGFILE)},
USING PROCEDURE proc-name
```

**Note:** `proc-name` is the name of the procedure specified by `proc-err`.

### Unconditional Tokens

- **ZSPI-TKN-SSID**
  
  is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

- **ZEMS-TKN-EVENTNUMBER (shared)**
  
  is the event number. Its value is ZEMS-EVT-BAD-EVENT (1007).

- **ZEMS-TKN-EMPHASIS (shared)**
  
  is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is FALSE.

- **ZEMS-TKN-CONSOLE-PRINT (shared)**
  
  is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.
Distributor Event Messages

1007: ZEMS-EVT-BAD-EVENT

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-LOGNAME (nonshared)

is the file name of a log file. Here it is the log file in use when the distributor read the bad event message.

ZSPI-TKN-PROC-ERR (nonshared)

specifies a procedure associated with the event. Here EMS assigns one of these values:

ZEMS-VAL-FILTER-EVAL        20
ZSPI-VAL-SSGETTKN            3

For descriptions of the extensible structured token ZEMS-MAP-STATUS-DIST and the tokens RECORD-ADDRESS, ZEMS-TOKEN-COLNAME, and -COLNAME-ENUM, see Token and Data Type Definitions on page 18-3.

Cause. A distributor reads an event message from a log file and finds the message is not a valid event message.

Effect. The distributor continues to process event messages, skipping the invalid event message. After five invalid event messages are detected, the event message (ZEMS-EVT-BAD-LOG) is generated.

Recovery. This event message is informative only. No corrective action is necessary.
1008: ZEMS-EVT-DEVTYPE

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>token-type ZEMS-DDL-STATUS-DIST.</td>
</tr>
<tr>
<td>ZEMS-TKN-DEVTYPE-ENUM</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Conditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-COLNAME-ENUM</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-FILECODE</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-BLOCKLENGTH</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-DEVICE-TYPE</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: failfilename IS INVALID FOR ACCESS,
dev-type-text IS INVALID
USING PROCEDURE proc-name

Note: dev-type-text depends on the value of devtype-enum and is one of the following:

```plaintext
{ FILECODE filecode }
{ BLOCK LENGTH blocklength }
{ DEVICE TYPE devicetype }
{ NAME }
```

Unconditional Tokens

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual.*

**ZEMS-TKN-EVENTNUMBER (shared)**

is the event number. Its value is ZEMS-EVT-DEVTYPE (1008).
**Distributor Event Messages**

**ZEMS-TKN-EMPHASIS (shared)**

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is FALSE.

**ZEMS-TKN-CONSOLE-PRINT (shared)**

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.

**ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID**

are standard EMS tokens described in Section 14, EMS Definitions.

**ZEMS-TKN-SUBJECT-MARK (shared)**

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

**ZEMS-TKN-FAILFILENAME (nonshared)**

is the file name of a bad file. Here it is the name of a log file or collector that the distributor could not access.

The extensible structured token ZEMS-MAP-STATUS-DIST and the token ZEMS-TKN-DEVTYPE-ENUM are described in Token and Data Type Definitions on page 18-3.

**Conditional Tokens**

The tokens ZEMS-TKN-COLNAME-ENUM and -COLNAME are present when the event-message source is a collector and are described in Token and Data Type Definitions on page 18-3.

The tokens ZEMS-TKN-FILECODE, -BLOCKLENGTH, and -DEVICE-TYPE are present if a bad file code, bad block length, or bad device type is responsible for the access problems. For more information, see Token and Data Type Definitions on page 18-3.

**Cause.** The distributor cannot access an event message log file or collector because the file code, device type, or block length is incorrect. Something is wrong with the specified source of event messages; for example, you might have specified the wrong name for a log file or for a collector process.

**Effect.** The associated collector is disconnected, but the distributor continues to run.

**Recovery.** Make sure you specified a correct EMS event-log file name. If you specified a correct name, contact your HP representative. An application can direct the distributor to reconnect the same collector after the problem is corrected.
1009: ZEMS-EVT-INTERNAL-ERROR

Unconditional Tokens

ZSPI-TKN-SSID token-type ZSPI-TYP-SSID.
ZEMS-TKN-EVENTNUMBER token-type ZSPI-TYP-INT.
ZEMS-TKN-EMPHASIS token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-CONSOLE-PRINT token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-GENTIME token-type ZSPI-TYP-TIMESTAMP.
ZEMS-TKN-LOGTIME token-type ZSPI-TYP-TIMESTAMP.
ZEMS-TKN-CPU token-type ZSPI-TYP-INT.
ZEMS-TKN-PROC-DESC token-type ZSPI-TYP-STRING.
ZEMS-TKN-PIN token-type ZSPI-TYP-INT.
ZEMS-TKN-NODENUM token-type ZSPI-TYP-INT2.
ZEMS-TKN-USERID token-type ZSPI-TYP-UINT.
ZEMS-TKN-SUBJECT-MARK token-type ZSPI-TYP-SSCTL.
ZEMS-TKN-PROGRAMFILE token-type ZSPI-TYP-FNAME.
ZEMS-TKN-SUBJECT-MARK token-type ZSPI-TYP-SSCTL.
ZSPI-TKN-ERROR token-type ZSPI-TYP-ERROR.
ZEMS-MAP-STATUS-DIST token-type ZEMS-DDL-STATUS-DIST.

Event-Message Text

EMS: INTERNAL ERROR IN DISTRIBUTOR, SSID ssid, ERROR error PROGRAM FILENAME failfilename

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-INTERNAL-ERROR (1009).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has two subject tokens: ZEMS-TKN-PROGRAMFILE and ZSPI-TKN-ERROR.

ZEMS-TKN-PROGRAMFILE (nonshared)

is the file name of the distributor object program.

For descriptions of the extensible structured token ZEMS-MAP-STATUS-DIST and the token ZSPI-TKN-ERROR, see Token and Data Type Definitions on page 18-3.

Cause. An unrecognizable internal software error occurred.

Effect. The distributor stops running.

Recovery. Contact your HP representative.

1010: ZEMS-EVT-CHECKOPEN-FAILED

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-DIST-NAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: CHECKOPEN FAILED, ERROR zfilerr ON failfilename

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.
ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-CHECKOPEN-FAILED (1010).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions.
Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token; for more information, see Section 14, EMS Definitions.
Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This
event message has two subject tokens:

ZEMS-TKN-FAILFILENAME (nonshared)
is the file name of a bad file. Here it is the name of the distributor program.

ZEMS-TKN-DIST-NAME (nonshared)
is the file name of a distributor process. Here it is the name of the primary
process, which will start a new backup process.

ZSPI-TKN-PROC-ERR (nonshared)
specifies a procedure associated with the event. Here EMS assigns the value of
ZEMS-VAL-EMSCHECKOPEN to 35.

Cause. The primary distributor determines that its backup process is malfunctioning
because the backup is unable to “checkopen” a file.

Effect. The distributor stops its backup process and attempts to create a new backup
after a 30-second delay.

Recovery. For errors 30 through 37, increase system resources. If this does not
correct the problem or results in a different error, contact your service provider.
1011: ZEMS-EVT-TAKEOVER

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Name</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-DIST-NAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-TAKEOVER-REASON</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: TAKEOVER BY BACKUP (takeover-text)

Note: takeover-text depends on the value of takeover-reason and is one of the following:

{ PRIMARY STOPPED }
{ PRIMARY ABENDED }
{ PRIMARY CPU IS DOWN }
{ PRIMARY CALLED CHECKSWITCH }

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-TAKEOVER (1011).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZSPI-TKN-PROC-ERR (nonshared)

specifies a procedure associated with the event. Here EMS assigns the value of
ZEMS-VAL-CHECKPOINT to 37.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This
event message has one subject token:

ZEMS-TKN-DIST-NAME (nonshared)

is the name of the distributor process. Here it is the name of the new distributor
process.

ZEMS-TKN-TAKEOVER-REASON (nonshared)

is described in Token and Data Type Definitions on page 18-3.

**Cause.** The primary distributor process, or the CPU in which it runs, failed, and the
distributor backup process has taken over.

**Effect.** If the primary distributor process failed, an attempt is made to create a new
backup after a 30-second delay. In the case of CPU failure, a new backup is created
30 seconds after the new primary process receives a CPU RELOADED system
message.

**Recovery.** If the primary process abnormally ended, investigate the event log to
determine why.
1012: ZEMS-EVT-CREATEBACKUP-FAILED

Unconditional Tokens

ZSPI-TKN-SSID                      token-type ZSPI-TYP-SSID.
ZEMS-TKN-EVENTNUMBER              token-type ZSPI-TYP-INT.
ZEMS-TKN-EMPHASIS                token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-CONSOLE-PRINT           token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-GENTIME                 token-type ZSPI-TYP-TIMESTAMP.
ZEMS-TKN-LOGTIME                token-type ZSPI-TYP-TIMESTAMP.
ZEMS-TKN-CPU                     token-type ZSPI-TYP-INT.
ZEMS-TKN-PROC-DESC               token-type ZSPI-TYP-STRING.
ZEMS-TKN-PIN                      token-type ZSPI-TYP-INT.
ZEMS-TKN-NODENUM                  token-type ZSPI-TYP-INT2.
ZEMS-TKN-USERID                   token-type ZSPI-TYP-UINT.
ZEMS-TKN-SUBJECT-MARK            token-type ZSPI-TYP-SSCTL.
ZEMS-TKN-PROGRAMFILE             token-type ZSPI-TYP-FNAME.
ZEMS-TKN-SUBJECT-MARK            token-type ZSPI-TYP-SSCTL.
ZEMS-TKN-DIST-NAME               token-type ZSPI-TYP-FNAME.
ZSPI-TKN-PROC-ERR                token-type ZSPI-TYP-ENUM.
ZEMS-TKN-NEWPROCESS-CPU          token-type ZSPI-TYP-ENUM.
ZEMS-TKN-NEWPROCESS-PRIORITY     token-type ZSPI-TYP-INT.
ZEMS-TKN-PROCCREATE-ERROR       token-type ZSPI-TYP-ENUM.

Conditional Tokens

ZEMS-TKN-ZFILERR                    token-type ZSPI-TYP-UINT.

Event-Message Text

EMS: FAILED TO CREATE BACKUP PROCESS IN CPU cpu,
PRIORITY pri, PROGRAM FILE name -
ERROR newprocerr : ertxt

Note: ertxt depends on the value of ZEMS-TKN-NEWPROCESS and is one of:

{ NO ERROR }
{ UNDEFINED EXTERNALS }
{ NO PROCESS CONTROL BLOCK AVAILABLE }
{ ERROR ON PROGRAMFILE: zfilerr }
{ UNABLE TO ALLOCATE MAP }
{ ERROR ON SWAP FILE : zfilerr }
{ ILLEGAL FILE FORMAT }
{ UNLICENSED PRIVILEGED PROGRAM }
{ PROCESS NAME ERROR: zfilerr }
{ LIBRARY CONFLICT }
{ UNABLE TO COMMUNICATE WITH SYSTEM MONITOR }
{ ERROR ON LIBRARY FILE: zfilerr }
{ LIBRARY AND PROGRAM FILE ARE THE SAME }
{ EXTENDED SEGMENT ERROR: zfilerr }
{ SWAP FILE ERROR: zfilerr }
{ ILLEGAL HOME TERMINAL, ERROR zfilerr }
Unconditional Tokens

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

**ZEMS-TKN-EVENTNUMBER** *(shared)*

is the event number. Its value is ZEMS-EVT-CREATEBACKUP-FAILED (1012).

**ZEMS-TKN-EMPHASIS** *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is FALSE.

**ZEMS-TKN-CONSOLE-PRINT** *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

**ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, **and **-USERID**

are standard EMS tokens described in Section 14, EMS Definitions.

**ZEMS-TKN-SUBJECT-MARK** *(shared)*

is the standard EMS token that immediately precedes each subject token. This event message has two subject tokens:

**ZEMS-TKN-PROGRAMFILE** *(nonshared)*

is the name of a program file. Here it is the program file for the distributor.

**ZEMS-TKN-DIST-NAME** *(nonshared)*

is the name of the distributor process.

**ZSPI-TKN-PROC-ERR** *(nonshared)*

specifies a procedure associated with the event. Here EMS assigns the value of ZEMS-VAL-NEWPROCESS to 36.

**ZEMS-TKN-NEWPROCESS-CPU, -NEWPROCESS-PRIORITY, and -PROCCREATE-ERROR**

are described in *Token and Data Type Definitions* on page 18-3.

Conditional Tokens

**ZEMS-TKN-ZFILERR** *(nonshared)*

is the error code associated with an event if such an error occurred.
**Cause.** The distributor cannot create a backup due to a `PROCESS_CREATE_` error. The event message is issued only if the initial attempt to create a backup fails; unsuccessful retries are not reported.

**Effect.** The primary distributor process retries this error unless the `PROCESS_CREATE_` error indicates that the backup CPU is down. In the latter case, an attempt is made to create a new backup 30 seconds after receipt of a CPU `RELOADED` system message for the backup CPU. For errors other than CPU down, the primary distributor delays for 30 seconds and retries; if this second attempt fails, the next attempt occurs after a 60-second delay. This process continues (the delay increases by 30 seconds at each failure) until the delay reaches 5 minutes. Thereafter, the primary distributor attempts to create a new backup every 5 minutes.

**Recovery.** Recovery action depends on the `PROCESS_CREATE_` error and, if present, the `ZFILEERR` error, as shown in the *Guardian Procedure Calls Reference Manual*.

For other errors, contact your service provider.
1013: ZEMS-EVT-BACKUP-CREATED

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-DIST-NAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROGRAMFILE</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-CPU</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-PRIORITY</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROCCREATE-ERROR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: CREATED BACKUP IN CPU newprocess-cpu,
    PRIORITY newprocess-pri, PROGRAM FILE programfile

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-BACKUP-CREATED (1013).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions.
Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions.
Its value here is TRUE.
1013: ZEMS-EVT-BACKUP-CREATED

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has two subject tokens:

ZEMS-TKN-DIST-NAME (nonshared)

is the name of the distributor process. Here it is the name of the new distributor process.

ZEMS-TKN-PROGRAMFILE (nonshared)

is the name of a program file. Here it is the program file for the distributor.

ZEMS-TKN-NEWPROCESS-CPU and -NEWPROCESS-PRIORITY

are described in Token and Data Type Definitions on page 18-3.

ZEMS-TKN-PROCCREATE-ERROR (nonshared)

is the process create error.

ZSPI-TKN-PROC-ERR (nonshared)

specifies a procedure associated with the event. Here EMS assigns the value of ZEMS-VAL-NEWPROCESS to 36.

Cause. The distributor’s primary process succeeded in starting a new backup process.

Effect. None. The new backup process begins.

Recovery. This event message is informative only. No corrective action is necessary.
1014: ZEMS-EVT-BACKUP-ABENDED

Unconditional Tokens

ZSPI-TKN-SSID token-type ZSPI-TYP-SSID.
ZEMS-TKN-EVENTNUMBER token-type ZSPI-TYP-INT.
ZEMS-TKN-EMPHASIS token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-CONSOLE-PRINT token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-LOGTIME token-type ZSPI-TYP-TIMESTAMP.
ZEMS-TKN-CPU token-type ZSPI-TYP-INT.
ZEMS-TKN-PROC-DESC token-type ZSPI-TYP-STRING.
ZEMS-TKN-PIN token-type ZSPI-TYP-INT.
ZEMS-TKN-NODENUM token-type ZSPI-TYP-INT2.
ZEMS-TKN-USERID token-type ZSPI-TYP-UINT.
ZEMS-TKN-SUBJECT-MARK token-type ZSPI-TYP-SSCTL.
ZEMS-TKN-DIST-NAME token-type ZSPI-TYP-FNAME.

Event-Message Text
EMS: BACKUP PROCESS ABENDED

Unconditional Tokens

ZSPI-TKN-SSID
is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-BACKUP-ABENDED (1014).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions.
Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions.
Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:
ZEMS-TKN-DIST-NAME (nonshared)
is the name of the distributor process.

**Cause.** The distributor’s backup process terminated abnormally.

**Effect.** The primary distributor process attempts to create a new backup after a 30-second delay.

**Recovery.** Contact your service provider. Provide your provider with the SAVEABEND file from the backup.
1015: ZEMS-EVT-BACKUP-DELETED

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-DIST-NAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: BACKUP PROCESS DELETED (CPU DOWN)

Unconditional Tokens

ZSPI-TKN-SSID
is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-BACKUP-DELETED (1015).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:
**Distributor Event Messages**

**1016: ZEMS-EVT-CHECKPOINT-FAILED**

**ZEMS-TKN-DIST-NAME (nonshared)**

is the name of the distributor process.

**Cause.** The CPU in which the distributor’s backup process was running failed.

**Effect.** The primary distributor process does not attempt to create a backup until 30 seconds after receiving a CPU RELOADED system message for its backup CPU.

**Recovery.** Follow the recommended procedure for CPU failure.

---

### Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Name</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-DIST-NAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
</tbody>
</table>

**Event-Message Text**

```
EMS: CHECKPOINT FAILED, ERROR zfilerr
```
ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions.
Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-DIST-NAME (nonshared)
is the name of the distributor process.

ZEMS-TKN-ZFILERR (nonshared)
is described in Token and Data Type Definitions on page 18-3.

ZSPI-TKN-PROC-ERR (nonshared)
specifies a procedure associated with the event. Here EMS assigns the value of ZEMS-VAL-CHECKPOINT to 37.

Cause. The distributor's backup process received an I/O error during a checkpoint operation.

Effect. The distributor backup process is stopped, and an attempt is made to create a new backup after a 30-second delay. Resource allocation failures (errors 30 through 37) might indicate that the primary and/or backup CPU is overloaded.

Recovery. For errors 30 through 37, free system resources. If this does not correct the problem, or results in a different error, contact your service provider.
Unconditional Tokens

ZSPI-TKN-SSID          token-type ZSPI-TYP-SSID.
ZEMS-TKN-EVENTNUMBER   token-type ZSPI-TYP-INT.
ZEMS-TKN-EMPHASIS      token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-CONSOLE-PRINT token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-GENTIME       token-type ZSPI-TYP-TIMESTAMP.
ZEMS-TKN-LOGTIME       token-type ZSPI-TYP-TIMESTAMP.
ZEMS-TKN-CPU           token-type ZSPI-TYP-INT.
ZEMS-TKN-PROC-DESC     token-type ZSPI-TYP-STRING.
ZEMS-TKN-PIN           token-type ZSPI-TYP-INT.
ZEMS-TKN-NODENUM       token-type ZSPI-TYP-INT2.
ZEMS-TKN-USERID        token-type ZSPI-TYP-UINT.
ZEMS-TKN-SUBJECT-MARK  token-type ZSPI-TYP-SSCTL.
ZEMS-TKN-LOGNAME       token-type ZSPI-TYP-FNAME.
ZEMS-MAP-STATUS-DIST   token-type ZEMS-DDL-STATUS-DIST.
ZEMS-TKN-COLNAME-ENUM  token-type ZSPI-TYP-ENUM.
ZEMS-TKN-COLNAME       token-type ZSPI-TYP-FNAME.

Event-Message Text

EMS: LOG CLOSED - TOO MANY BAD EVENTS ENCOUNTERED -
LOG FILE file-name {, COLLECTOR colname },
    (ACCESS AS LOGFILE),
LAST RECORD ADDRESS record-address

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-BAD-LOG (1017).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions.
Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions.
Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-LOGNAME (nonshared)

is the file name of a log file. Here, it is the log file that the distributor has declared invalid.

For descriptions of the extensible structured token ZEMS-MAP-STATUS-DIST and the tokens ZEMS-TKN-COLNAME-ENUM and -COLNAME, see Token and Data Type Definitions on page 18-3.

Cause. The distributor designated its source of event messages, a specified log file, as a bad file after reading five invalid event messages.

Effect. The distributor closes the log file and attempts to access the next log file. If no next file exists, the distributor generates another event message (ZEMS-EVT-COLLECTOR-DISCONNECT). The distributor disconnects the collector but continues to run. (An application can direct the distributor to reconnect the same collector after the problem is corrected.)

Recovery. Try to determine the nature of the invalid events and where they come from.
1018: ZEMS-EVT-FILES-LOST

Unconditional Tokens

ZSPI-TKN-SSID token-type ZSPI-TYP-SSID.
ZEMS-TKN-EVENTNUMBER token-type ZSPI-TYP-INT.
ZEMS-TKN-EMPHASIS token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-CONSOLE-PRINT token-type ZSPI-TYP-BOOLEAN.
ZEMS-TKN-LOGTIME token-type ZSPI-TYP-TIMESTAMP.
ZEMS-TKN-CPU token-type ZSPI-TYP-INT.
ZEMS-TKN-PROC-DESC token-type ZSPI-TYP-STRING.
ZEMS-TKN-PIN token-type ZSPI-TYP-INT.
ZEMS-TKN-NODENUM token-type ZSPI-TYP-INT2.
ZEMS-TKN-SUBJECT-MARK token-type ZSPI-TYP-SSCTL.
ZEMS-TKN-COLNAME token-type ZSPI-TYP-FNAME.
ZEMS-TKN-LASTLOGFILE token-type ZSPI-TYP-FNAME.
ZEMS-TKN-FAIL-REASON token-type ZSPI-TYP-ENUM.
ZEMS-TKN-RECOVERY-ENUM token-type ZSPI-TYP-ENUM.

Conditional Tokens

ZEMS-TKN-NEWLOGFILE token-type ZSPI-TYP-FNAME.

Event-Message Text

EMS: DISTRIBUTOR COULD NOT COMPLETE BACK CHAIN TO FILE lastlogfile BECAUSE OF failreason recoveryenum COLLECTOR: colname

Note: failreason can be:

{ LOG ERROR (SEE PREVIOUS EVENT) }
{ MISSING LINK }
{ INCOMPATIBLE LINK }

recoveryenum can be:

{ , NOW USING FILE newlogfile. }
{ . }

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-FILES-LOST (1018).
ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token; for more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLNAME (nonshared)

is the name of a collector. Here it is the collector whose log files the distributor cannot access.

ZEMS-TKN-LASTLOGFILE (nonshared)

is the name of the collector log file that the distributor tried unsuccessfully to access after it read the most recent event message in the previous log file.

ZEMS-TKN-FAIL-REASON (nonshared)

indicates why the log file being sought was inaccessible. The token can have one of three values:

ZEMS-VAL-EVENT-GENERATED 0
ZEMS-VAL-NO-LINK 1
ZEMS-VAL-BAD-LINK 2

The first value means an I/O error occurred. The second value means the link to the log file is missing. The third value means that the link is corrupted. For an I/O error or corrupted link, the problem is described in a previous event message. If the link is missing, no earlier message exists to indicate why.

ZEMS-TKN-RECOVERY-ENUM (nonshared)

indicates whether the distributor was able to recover by accessing a log file later in the sequence than the inaccessible file. If EMS does recover, the token NEWLOGFILE gives the name of the later file.

EMS assigns one of two values to ZEMS-TKN-RECOVERY-ENUM:

ZEMS-VAL-RECOVERY-OK 0
ZEMS-VAL-NO-RECOVERY 1
Conditional Tokens

ZEMS-TKN-NEWLOGFILE (nonshared)

is the name of the log file, later in sequence than the inaccessible file, that the distributor was able to access. This token is included only if the value of ZEMS-TKN-RECOVERY-ENUM is ZEMS-VAL-RECOVERY-OK; that is, only if the distributor could access a later file.

**Cause.** The distributor could not access one or more collector log files older than the current one.

The distributor generates this event message after:

1. The distributor receives a command to process event messages from a certain time forward.
2. It finds that all messages in the current collector log file are later than the specified time.
3. Following pointers in the files, the distributor searches back through the chain of log files containing older messages. As it searches, it keeps a list of the names of the files it checks.
4. It finds the oldest log and oldest event message it wants.
5. It reads and distributes event messages until it reaches the end of that log.
6. It looks for the next later log file in its list but cannot access that file.
7. It tries to access still later files until it either succeeds or reaches the end of its list, unable to access even the current log file.

**Effect.** The distributor sends ZEMS-EVT-FILES-LOST whether it succeeds or fails at Step 7. If it fails, it also generates event message ZEMS-EVT-COLLECTOR-DISCONNECT (see event 1019) and disconnects the collector but continues to run.

ZEMS-EVT-FILES-LOST specifies the name of the first log file the distributor failed to access (Step 6); the reason for the failure; whether the distributor continued by processing a later log file (Step 7); and, if so, which file it is.

**Recovery.** If an I/O error or corrupted link is the reason the log file is inaccessible, look for the earlier event message that describes the problem. If you still cannot locate the missing file, or if the link is missing, contact your service provider for help.
1019: ZEMS-EVT-COL-DISCONNECT

### Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>token-type ZEMS-DDL-STATUS-DIST.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME-ENUM</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

### Event-Message Text

EMS: COLLECTOR colname HAS BEEN DISCONNECTED DUE TO INACCESSIBLE EVENT LOG(S).

### Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-COL-DISCONNECT (1019).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLNAME (nonshared)

is the name of a collector. Here, it is the collector that the distributor is disconnecting.

The extensible structured token ZEMS-MAP-STATUS-DIST and the token ZEMS-TKN-COLNAME-ENUM are described in Token and Data Type Definitions on page 18-3.

**Cause.** The distributor is using collector log files as a source of event messages. Because of a hardware, software, or operations failure, the distributor cannot access any of the collector's log files.

**Effect.** The distributor disconnects the specified collector but continues running. An application can direct the distributor to reconnect the original collector after the problem is corrected.

**Recovery.** Make sure the line to the remote collector is up. For recovery information, see the description of the accompanying message. If you cannot solve the problem, contact your service provider.
Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>subsystem ID for EMS</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>event number</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>emphasis</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>console print</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>gentime</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>subject mark</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>filename</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>process error</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-CPU</td>
<td>CPU priority</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-PRIORITY</td>
<td>program file</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-ERROR</td>
<td>error</td>
</tr>
<tr>
<td>ZEMS-TKN-STARTUP-LOGTIME</td>
<td>startup log time</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>map status</td>
</tr>
</tbody>
</table>

Conditional Token

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>file error</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: FAILED TO CREATE DESTINATION PROCESS IN CPU cpu, PRIORITY pri, PROGRAM FILE name - ERROR error:
{NO ERROR | UNDEFINED EXTERNALS | NO PROCESS CONTROL BLOCK AVAILABLE | ERROR ON PROGRAMFILE: zfilerr | UNABLE TO ALLOCATE MAP | ERROR ON SWAP FILE: zfilerr | ILLEGAL FILE FORMAT | UNLICENSED PRIVILEGED PROGRAM | PROCESS NAME ERROR:
  zfilerr | LIBRARY CONFLICT | UNABLE TO COMMUNICATE WITH SYSTEM MONITOR | ERROR ON LIBRARY FILE: zfilerr | LIBRARY AND PROGRAM FILE ARE THE SAME | EXTENDED SEGMENT ERROR:
  zfilerr | SWAP FILE ERROR: zfilerr | ILLEGAL HOME TERMINAL, ERROR zfilerr }. TRIGGER LOGTIME time.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual. EMS assigns one of these values to ZSPI-TKN-SSID:

ZSPI-VAL-TANDEM
ZSPI-SSN-ZEMS
ZEMS-VAL-VERSION

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its assigned value is ZEMS-EVT-STARTUP-FAILED (1020).
ZEMS-TKN-EMPHASIS *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK *(shared)*

is the standard EMS token that immediately precedes each subject token. This event message has two subject tokens:

ZEMS-TKN-PROGRAMFILE *(type: ZSPI-TYP-FNAME; nonshared)*

is the file name of a program file.

ZEMS-TKN-FAILFILENAME *(type: ZSPI-TYP-FNAME; nonshared)*

is the destination process name.

ZSPI-TKN-PROC-ERR *(type: ZSPI-TYP-ENUM; nonshared)*

an ordinal identifying the failing procedure. Here EMS assigns the value ZEMS-VAL-NEWPROCESS.

ZEMS-TKN-NEWPROCESS-CPU *(type: ZSPI-TYP-INT; nonshared)*

is the CPU number in a NEWPROCESS command.

ZEMS-TKN-NEWPROCESS-PRIORITY *(type: ZSPI-TYP-INT; nonshared)*

is the new process priority in a NEWPROCESS command.

ZEMS-TKN-NEWPROCESS-ERROR *(type: ZSPI-TYP-ENUM; nonshared)*

is one of these system error numbers, most significant byte, one of:

- ZEMS-VAL-NO-ERROR
- ZEMS-VAL-UNDEFINED-EXTERNALS
- ZEMS-VAL-NO-PCB-AVAILABLE
- ZEMS-VAL-PROGRAMFILE-ERROR
- ZEMS-VAL-NO-MAP
- ZEMS-VAL-BAD-SWAPFILE
- ZEMS-VAL-BAD-FILE-FORMAT
- ZEMS-VAL-UNLICENSED
- ZEMS-VAL-BAD-PROCESS-NAME
- ZEMS-VAL-LIBRARY-CONFLICT
Distributor Event Messages

ZEMS-VAL-MONITOR-COMM
ZEMS-VAL-LIBRARY-FILE
ZEMS-VAL-PROGRAM-FILE
ZEMS-VAL-NO-EXT-SEGMENT
ZEMS-VAL-EXT-SEGMENT-SWAP
ZEMS-VAL-BAD-HOMETERM

ZEMS-TKN-STARTUP-LOGTIME (type: ZSPI-TYP-TIMESTAMP; nonshared)

is the log time of the event that triggered the startup.

For a description of the extensible structured token ZEMS-MAP-STATUS-DIST, see Token and Data Type Definitions on page 18-3.

Conditional Token

The token ZEMS-TKN-ZFILERR is described in Token and Data Type Definitions on page 18-3.

Cause. The distributor cannot create a destination process due to a PROCESS_CREATE_ error. One retry was attempted and failed. This event is similar to the ZEMS-EVT-CREATEBACKUP-FAILED event. This event message is issued only if the initial attempt to create the process fails; unsuccessful retries resulting from subsequent event triggers are not reported.

Effect. If the startup eventually succeeds, another message (ZEMS-EVT-STARTUP-OK) is generated.

Recovery. The action to be taken depends on the most significant byte of the PROCESS_CREATE_ error:

MSB = 2. The specified CPU cannot run any additional processes. Change the routing profile for this destination.

MSB = 3. Check the distributor's code file.

MSB = 4. Change the CPU in the routing profile as in (2).

MSB = 5. Most likely the disc volume selected for the destination's SWAPVOL is out of space. Purge unneeded files from the volume or change the startup message in the destination's profile to specify a different volume.

MSB = 10. Check that the CPU specified for the destination is one that exists on your system; if not, change the CPU as in (2). Otherwise, check the CPU.

For other errors, contact your service provider.
1021: ZEMS-EVT-STARTUP-OK

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-SENTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROGRAMEFILE</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-PRIORITY</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-ERROR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-STARTUP-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>token-type ZEMS-DDL-STATUS-DIST.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: DESTINATION PROCESS SUCCESSFULLY CREATED IN CPU cpu, PRIORITY pri, PROGRAM FILE name, TRIGGER LOGTIME time

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the *SPI Programming Manual*. EMS assigns one of these values to ZSPI-TKN-SSID:

- ZSPI-VAL-TANDEM
- ZSPI-SSN-ZEMS
- ZEMS-VAL-VERSION

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its assigned value is ZEMS-EVT-STARTUP-OK (1021).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see [Section 14, EMS Definitions](#). Its value here is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see [Section 14, EMS Definitions](#). Its value here is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has two subject tokens:

ZEMS-TKN-PROGRAMFILE (type: ZSPI-TYP-FNAME; nonshared)

is the file name of a program file.

ZEMS-TKN-FAILFILENAME (type: ZSPI-TYP-FNAME; nonshared)

is the destination process name.

ZSPI-TKN-PROC-ERR (type: ZSPI-TYP-ENUM; nonshared)

is an ordinal identifying the failing procedure. Here EMS assigns the value ZEMS-VAL-NEWPROCESS.

ZEMS-TKN-NEWPROCESS-CPU (type: ZSPI-TYP-INT; nonshared)

is the CPU number in a NEWPROCESS command.

ZEMS-TKN-NEWPROCESS-PRIORITY (type: ZSPI-TYP-INT; nonshared)

is the new process priority in a NEWPROCESS command.

ZEMS-TKN-NEWPROCESS-ERROR (type: ZSPI-TYP-ENUM; nonshared)

a system error number, most significant byte, should be 0 (zero).

ZEMS-TKN-STARTUP-LOGTIME (type: ZSPI-TYP-TIMESTAMP; nonshared)

is the log time of the event that triggered the startup.

The extensible structured token ZEMS-MAP-STATUS-DIST, and token ZEMS-TKN-ZFILERR are described in Token and Data Type Definitions on page 18-3.

Cause. A destination was successfully started after previous failure and retries.

Effect. None. The destination successfully started.

Recovery. Determine which events have not been reported. The log time of the trigger event is included to let the analyst determine this. Examine the ZEMS-EVT-STARTUP-FAILED event to obtain the time of the first event that was missed.
1022: ZEMS-EVT-WRITE-FAILED

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Token Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZSPI-TKN-WRITE-FAIL-COUNT</td>
<td>token-type ZSPI-TYP-_UINT.</td>
</tr>
<tr>
<td>ZEMS-MAP-STATUS-DIST</td>
<td>token-type ZEMS-DDL-STATUS-DIST.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: WRITE OPERATION TO DESTINATION name HAS TIMED OUT n TIMES DURING A 24-HOUR PERIOD.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual. EMS assigns one of these values to ZSPI-TKN-SSID:

ZSPI-VAL-TANDEM
ZSPI-SSN-ZEMS
ZEMS-VAL-VERSION

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its assigned value is ZEMS-EVT-WRITE-FAILED (1022).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens described in Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK *(shared)*

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-FAILFILENAME *(type: ZSPI-TYP-FNAME; nonshared)*

is the destination name.

ZSPI-TKN-PROC-ERR *(type: ZSPI-TYP-ENUM; nonshared)*

is an ordinal identifying the failing procedure. Here EMS assigns the value ZEMS-VAL-ZFILWRITEREAD.

ZSPI-TKN-WRITE-FAIL-COUNT *(type: ZSPI-TYP-UINT; nonshared)*

indicates the number of timeouts that have occurred within a 24-hour period, due to failed write operations.

For a description of the extensible structured token ZEMS-MAP-STATUS-DIST, see *Token and Data Type Definitions* on page 18-3.

**Cause.** A write operation to a destination repeatedly timed out over an interval of 24 hours. The timeout can occur either after a certain number of retries have been attempted or if the recipient did not acknowledge the request by reading its $receive file. The event contains the number of timeouts that occurred during the 24-hour period.

**Effect.** If the error is recovered, the message is not generated.

**Recovery.** Determine the nature of the problem and correct it. Information from the tokens might help in determining the nature of the problem.
Collector Commands and Responses

Both the basic and extended SPI interfaces support commands that let you manage and monitor the operational environment of the primary and alternate collectors and the compatibility distributor.

These interfaces let your application program send a command to or request information from a primary or alternate collector. The interfaces also let you interact indirectly (through $0) with the compatibility distributor.

This section describes the command and response messages for the EMS primary and alternate collectors:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI Command and Response Buffers</td>
<td>19-1</td>
</tr>
<tr>
<td>Sending a SPI Command</td>
<td>19-2</td>
</tr>
<tr>
<td>Summary of EMS Collector Commands</td>
<td>19-2</td>
</tr>
<tr>
<td>Common Definitions of ZCOM- Commands</td>
<td>19-4</td>
</tr>
<tr>
<td>Common Definitions of ZEMS- Commands</td>
<td>19-27</td>
</tr>
<tr>
<td>Collector Command Descriptions</td>
<td>19-29</td>
</tr>
</tbody>
</table>

SPI Command and Response Buffers

Commands are sent to a collector in an SPI command buffer. Responses are returned in an SPI response buffer. A maximum of 127 application programs can have the $ZSPI interface to a collector open at the same time.

*Figure 19-1* illustrates the interaction between a management application and a collector.

*Figure 19-1. Use of SPI Command and Response Buffers*

SPI command and response buffers consist of tokens. A variety of tokens is associated with the SPI commands. Some of these tokens are required; some are not.
A command buffer always contains a command token named ZSPI-TKN-COMMAND, which specifies the command to be carried out. A command buffer can also contain additional tokens that define parameters unique to the command being sent.

A response buffer is a group of tokens containing all the information that results when a command is performed. These tokens are:

- A return token. A response buffer always contains a token named ZSPI-TKN-RETCODE. If a command fails, this token contains an error number indicating the reason for the failure. This error number is defined by the subsystem that issues it—in this case, a collector.
- One or more tokens that contain requested information or result from the completed command.

### Sending a SPI Command

Your program should perform these steps to send a command and receive the response:

1. Define the buffer space.
2. Use SSINIT to initialize the buffer space.
3. Use SSPUT to add tokens to the buffer.
4. Use WRITEREAD to send the command buffer to $0 or to an alternate collector.
5. Use SSGET to retrieve tokens from the response buffer.

### Summary of EMS Collector Commands

Table 19-1 summarizes the primary and alternate collector commands.

Commands with a ZCOM- prefix are extended SPI compliant. They use the extended SPI interface, support specific object types (ZCOM-OBJ-COLL, ZCOM-OBJ-FILTER), and have been designed to handle BDS and PLF information.

Commands with a ZEMS- prefix use and comply with the basic SPI interface. They do not support specific object types (the object type is ZSPI-VAL-NULL-OBJECT-TYPE), and generally do not use BDS or PLF information; only the updated fields added to the ZEMS-MAP-COL-CONTROL and ZEMS-MAP-COL-STATUS token maps used by the CONTROL and STATUS commands contain BDS and PLF information. The new ZEMS-CMD-REPLACE command supports filter objects.
### Summary of EMS Collector Commands

#### Table 19-1. Collector Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>Lets you add objects to the collector. The only supported object is the EMS filter (ZCOM-OBJ-FILTER), which can be a compiled filter, a filter table, or a burst filter. You can add a maximum of 10 filters to a single collector.</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>Alters objects in the collector. Supported objects are EMS collectors (ZCOM-OBJ-COLL) and EMS filters (ZCOM-OBJ-FILTER).</td>
</tr>
<tr>
<td>ZEMS-CMD-CONTROL</td>
<td>Lets you control the operational environment of the primary and alternate collectors and the compatibility distributor. You can select attributes of new event-message log files, switch the CPU used by a collector, switch the event-message log subvolume, turn event-message buffering on or off, and make other operational decisions.</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>Lets you delete objects from the collector. The only supported object is the EMS filter (ZCOM-OBJ-FILTER).</td>
</tr>
<tr>
<td>ZCOM-CMD-GETVERSION</td>
<td>Returns the version number of the programmatic interface. This command returns the conditional response token ZCOM-TKN-GETVSN-LVL to indicate that the collector can process enlightened tokens and data structures.</td>
</tr>
<tr>
<td>ZEMS-CMD-GETVERSION</td>
<td>Returns the version number of the programmatic interface.</td>
</tr>
<tr>
<td>ZCOM-CMD-INFO</td>
<td>Requests the configuration values for the specified collector object. These values are initially set at collector startup and might have been modified by the ALTER or CONTROL commands. Supported objects are the collector (ZCOM-OBJ-COLL) and filter (ZCOM-OBJ-FILTER).</td>
</tr>
<tr>
<td>ZCOM-CMD-LISTOBJECTS</td>
<td>Returns the name and type of all or a subset of the objects known by the collector.</td>
</tr>
<tr>
<td>ZEMS-CMD-REPLACE</td>
<td>Replaces a configured object in the collector with another object. The only supported object is the EMS filter (ZCOM-OBJ-FILTER). You can replace only one filter at a time.</td>
</tr>
<tr>
<td>ZCOM-CMD-STATUS</td>
<td>Requests up-to-date configuration and status information from the collector. Supported objects are EMS collectors (ZCOM-OBJ-COLL) and filters (ZCOM-OBJ-FILTER). You must specify at least one object name with the STATUS command.</td>
</tr>
<tr>
<td>ZEMS-CMD-STATUS</td>
<td>Returns up-to-date primary and alternate collector operational information and event-message statistics.</td>
</tr>
<tr>
<td>ZCOM-CMD-STOP</td>
<td>Performs an orderly shutdown of the alternate collector and supports a collector object (ZCOM-OBJ-COLL).</td>
</tr>
<tr>
<td>ZEMS-CMD-STOP</td>
<td>Performs an orderly shutdown of the alternate collector.</td>
</tr>
</tbody>
</table>
Common Definitions of ZCOM- Commands

This subsection defines the tokens and error numbers common to the ZCOM-collector commands and the ZEMS-CMD-REPLACE command. While the REPLACE command is not extended SPI compliant, it supports filter objects (ZCOM-OBJ-FILTER) and you can use it to replace a configured filter in an EMS collector with another filter.

The ZCOM-collector commands provide functional support for burst detection and suppression (BDS) and pre-log filtration (PLF) that the basic ZEMS- commands do not provide. These commands let collectors support multiple objects or field-level wild-card objects and return multiple response records (one per object instance) where appropriate. Supported object types are collector (ZCOM-OBJ-COLL) and filter (ZCOM-OBJ-FILTER).

For ZCOM-collector commands that support the ZCOM-TKN-SUB token, an object hierarchy exists. In these cases, ZCOM-OBJ-FILTER is considered subordinate to ZCOM-OBJ-COLL. Hierarchical names are not allowed; however, object name ambiguity is not likely because collector object names are process names, and filter object names are file names.

Table 19-2 shows which ZCOM- commands (and the ZEMS-CMD-REPLACE command) you can issue for filter and collector objects, whether wild-card (*) objects are supported, and whether the ZCOM-TKN-SUB modifier token is supported. Values in the ZCOM-OBJ-COLL and ZCOM-OBJ-FILTER columns mean:

- Yes signifies that the command supports the associated object type; No signifies that the command does not support the associated object type.
- Only 1 signifies that only one of that object type is supported; >=1 signifies that more than one of that object type is supported; NO OBJNAME signifies that the object name need not be specified; * signifies that wild-card object types are supported.
- SUB signifies that the command supports the ZCOM-TKN-SUB modifier token.

Table 19-2. ZCOM- Command Capabilities

<table>
<thead>
<tr>
<th>Collector Command</th>
<th>ZCOM-OBJ-COLL</th>
<th>ZCOM-OBJ-FILTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-CMD-ADD</td>
<td>No</td>
<td>Yes, &gt;=1</td>
</tr>
<tr>
<td>ZCOM-CMD-ALTER</td>
<td>Yes, Only 1</td>
<td>Yes, Only 1</td>
</tr>
<tr>
<td>ZCOM-CMD-DELETE</td>
<td>No</td>
<td>Yes, &gt;=1</td>
</tr>
<tr>
<td>ZCOM-CMD-GETVERSION</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ZCOM-CMD-INFO</td>
<td>Yes, &gt;=1, *, SUB</td>
<td>Yes, &gt;=1, *, SUB</td>
</tr>
<tr>
<td>ZCOM-CMD-LISTOBJECTS</td>
<td>Yes, NO OBJNAME</td>
<td>YES, NO OBJNAME</td>
</tr>
<tr>
<td>ZEMS-CMD-REPLACE</td>
<td>No</td>
<td>Yes, Only 1</td>
</tr>
<tr>
<td>ZCOM-CMD-STATUS</td>
<td>Yes, &gt;=1, *, SUB</td>
<td>Yes, &gt;=1, * SUB</td>
</tr>
<tr>
<td>ZCOM-CMD-STOP</td>
<td>Yes, Only 1</td>
<td>No</td>
</tr>
</tbody>
</table>

EMS Manual—426909-005
19-4
These collector commands do not accommodate pre-D00 tokens that do not support a high PIN or a string file name. Such tokens generate a ZCOM-ERR-TKN-CODE-INV error.

**Note.** These commands are sent directly from the requestor to the EMS collector. An SCP process is not used.

### Common Command Tokens for ZCOM- Commands

Table 19-3 summarizes the common command tokens for the extended SPI-compliant collector commands, their ZSPI token type, and special usage considerations.

<table>
<thead>
<tr>
<th>Command Token</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>ZSPI-TYP-ENUM</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-OBJECT-TYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-PAID</td>
<td>ZSPI-TYP-INT</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-PHANDLE</td>
<td>ZSPI-TYP-PHANDLE</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-PPROGXFILE</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-PSTRING</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZCOM-TKN-SUB</td>
<td>ZSPI-TYP-ENUM</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-SEL-SUMSTATE</td>
<td>ZSPI-TYP-INT-PAIR</td>
<td>Ignored</td>
</tr>
<tr>
<td>ZCOM-TKN-CMD-POWER</td>
<td>ZSPI-TYP-ENUM</td>
<td>Ignored</td>
</tr>
<tr>
<td>ZCOM-TKN-CMD-TIMEOUT</td>
<td>ZSPI-TYP-TIMESTAMP</td>
<td>Ignored</td>
</tr>
<tr>
<td>ZSPI-TKN-RESPONSE-TYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Ignored</td>
</tr>
<tr>
<td>ZSPI-TKN-ALLOW-TYPE</td>
<td>ZSPI-TYP-ALLOW-TYPE</td>
<td>Ignored</td>
</tr>
<tr>
<td>ZSPI-TKN-ALLOW</td>
<td>ZSPI-TYP-ENUM</td>
<td>Ignored</td>
</tr>
<tr>
<td>ZSPI-TKN-COMMENT</td>
<td>ZSPI-TYP-STRING</td>
<td>Ignored</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

### Usage Considerations

- SSINIT signifies that the value for this token must be supplied in the call to SSINIT to create the SPI command.
Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.

Optional signifies that the token is optional and need not be supplied by the user.

Ignored signifies that this token is tolerated in the command and ignored.

**Common Command Token Descriptions**

**ZSPI-TKN-COMMAND**

is the header token that contains the command number. Its value is of the form ZCOM-CMD-command and is specified in the call to SSINIT.

**ZSPI-TKN-OBJECT-TYPE**

is the header token that contains the type of object to which the command is applied. Its value is of the form ZCOM-OBJ-objtype and is specified in the call to SSINIT.

**ZSPI-TKN-SSID**

is the header token that identifies the subsystem that processes the command. This subsystem is always defined as ZSPI-VAL-TANDEM.ZSPI-SSN-ZEMS.ZEMS-VAL-VERSION. This value is specified in the call to SSINIT.

**ZSPI-TKN-MAXRESP**

if the value of this command header token is not zero, the collector encapsulates the response record in a data list (between the -DATALIST and the -ENDLIST tokens). Multiple response records are supported. The default value for this token is zero.

**ZCOM-TKN-XMGR**

is not required, but is allowed. More than one of these tokens is not allowed and results in a ZCOM-ERR-TKN-DUP error.

**ZCOM-TKN-REQID-PAID**
**ZCOM-TKN-REQID-PHANDLE**
**ZCOM-TKN-REQID-PPROGXFILE**
**ZCOM-TKN-REQID-PSTRING**

are intended for security purposes and are not required. The EMS collectors use a different method for validating the requestor. However, if these tokens are present and the requestor has the proper security for the command, the contents of these tokens are checked for proper security. If this second check fails, the command is rejected with a ZCOM-ERR-SECURE-VIOL.
**ZCOM-TKN-OBJNAME**

specifies the fully qualified name of the object to be selected for processing. At least one of these tokens must be in the command. If there are multiple occurrences of this token, the collector returns a response record for each -OBJNAME token. The basic SPI command language rules about returning multiple response records are followed: the -CONTEXT token indicates when you must use more than one SPI buffer to return the full command response. When more than one object name is specified, the same command is applied to all named objects. If different actions are desired on each named object, then you must use separate commands.

**ZCOM-TKN-SUB**

is an optional token used to modify the STATUS and INFO commands. When it is used, the object hierarchy is assumed to be one where ZCOM-OBJ-COLL is at the top and ZCOM-OBJ-FILTER is a subordinate to ZCOM-OBJ-COLL. The permissible values for the ZCOM-TKN-SUB token are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Command Applies to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM-VAL-SUB-ONLY</td>
<td>Only the named object’s subordinates</td>
</tr>
<tr>
<td>ZCOM-VAL-SUB-NONE</td>
<td>Only the named object</td>
</tr>
<tr>
<td>ZCOM-VAL-SUB-ALL</td>
<td>Both the named object and its subordinates</td>
</tr>
<tr>
<td>ZCOM-OBJ-FILTER</td>
<td>Only the named object’s filter objects</td>
</tr>
</tbody>
</table>

**ZCOM-TKN-SEL-SUMSTATE**

**ZCOM-TKN-CMD-POWER**

**ZCOM-TKN-CMD-TIMEOUT**

**ZSPI-TKN-RESPONSE-TYPE**

**ZSPI-TKN-ALLOW-TYPE**

**ZSPI-TKN-ALLOW**

**ZSPI-TKN-COMMENT**

if present, these seven tokens are allowed and ignored.

**ZSPI-TKN-CONTEXT**

is a response continuation indicator. If a response cannot fit into the SPI response buffer, this token is present in the incomplete response. This token is also required in the next command to retrieve the remaining response information.

### Common Response Tokens for ZCOM- Commands

**Table 19-4** summarizes the common response tokens for the extended SPI-compliant collector commands, their ZSPI token type, and special usage considerations. Detailed descriptions of these common response tokens and their usage considerations are provided after the table.
Common Response Tokens for ZCOM- Collector Commands

### Table 19-4. Common Response Tokens for ZCOM- Collector Commands

<table>
<thead>
<tr>
<th>Response Token</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>ZSPI-TYP-ENUM</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-OBJECT-TYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-SERVER-VERSION</td>
<td>ZSPI-TYP-VERSION</td>
<td>SSINIT</td>
</tr>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

### Usage Considerations

- **SSINIT** signifies that the value for this token must be supplied in the call to SSINIT to create the SPI command.
- **Basic SPI** signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- **Error only** signifies that this token is present only when an error or warning is reported in the response record (that is, ZSPI-TKN-RETCODE contains a value other than zero).
- The data list (all tokens from -DATALIST to -ENDLIST) forms a single response record. Multiple data lists (response records) can occur in a response.

### Common Response Token Descriptions

**ZSPI-TKN-COMMAND**

contains the command number for the request to which this is the reply. This value is specified in the call to SSINIT.
ZSPI-TKN-OBJECT-TYPE
contains the type of object from the request to which this is the reply. This value is specified in the call to SSINIT. Its value is of the form ZCOM-OBJ-objtype.

ZSPI-TKN-SSID
identifies the subsystem that processes the command. In the following response tokens, the subsystem is always defined as ZSPI-VAL-TANDEM.ZSPI-SSN-ZEMS.ZEMS-VAL-VERSION. This value is specified in the call to SSINIT.

ZSPI-TKN-SERVER-VERSION
contains the release identifier of the server (the EMS collector) that performed the command. For this version, it is D31 or F40.

ZSPI-TKN-DATALIST
indicates the beginning of a data list. One response record is contained in each data list. The ZSPI-TKN-ENDLIST token indicates the end of the data list. These tokens are used only when a value other than zero was specified in the command -MAXRESP token. If the -MAXRESP value is other than zero and there are multiple responses, there might be multiple data lists (response records) in the response.

ZCOM-TKN-XMGR
is returned for all commands and contains the name of the collector that processed the SPI command.

ZSPI-TKN-RETCODE
contains the return code, which indicates the outcome of the command processing. If the value is zero (ZCOM-ERR-OK), the command completed without error. If the value is not zero, an error occurred, and the response record contains an error list that describes the error.

ZCOM-TKN-OBJTYPE
specifies the object type for ZCOM-TKN-OBJNAME. This value is usually the same as the value specified in the ZSPI-TKN-OBJECT-TYPE token in the command.

ZCOM-TKN-OBJNAME
specifies the name of the object acted upon by the command. If the command acted upon multiple objects, multiple occurrences of this token are returned, each within its own response record.

ZSPI-TKN-ERRLIST
indicates the beginning of an error list. ZSPI-TKN-ENDLIST indicates the end of the error list, and the error list can be nested. An error list is put in the response for
each error or warning encountered during processing of the command. The following are the required tokens for the error list. Other standard or subsystem-specific tokens can also be in the error list, depending on the type of error.

**ZSPI-TKN-ERROR**
- specifies the error and identifies the subsystem that prepared the error list.

**ZCOM-TKN-OBJTYPE**
- specifies the object type of the object in error.

**ZCOM-TKN-OBJNAME**
- specifies the object name of the object in error.

**ZSPI-TKN-ENDLIST**
- signifies the end of a data list.

**ZSPI-TKN-CONTEXT**
- if present, indicates that not all of the response records generated by the command could be returned in the response buffer. The requester must resend the original command with this token inserted in it to receive the remainder of the response.

**EMS-Specific Tokens**

These structured and simple tokens are used by collectors to support BDS and PLF:

**ZEMS-MAP-BDS-CONFIG**
- is a command token used in the ZCOM-CMD-ALTER command to change the BDS configuration in a collector. The fields in this token describe the parameter values currently used in the BDS configuration.

```
DEFINITION zems-ddl-bds-config
  02 zbds-flags TYPE zspi-ddl-uint.
  03 enabled  TYPE BIT 1.
  03 tmds     TYPE BIT 1.
  03 use-subj  TYPE BIT 1.
  03 process-type TYPE BIT 1.
  02 zbds-num-events TYPE zspi-ddl-uint.
  02 zbds-interval-start TYPE zspi-ddl-uint.
  02 zbds-interval-end   TYPE zspi-ddl-uint.
  02 zbds-interval-chk-end TYPE zspi-ddl-uint. SPI-NULL 255.
  02 zbds-num-sim-bursts TYPE zspi-ddl-uint SPI NULL 255.
  02 zbds-num-subj-bytes TYPE zspi-ddl-uint
END
```
zbds-flags
   is a field that contains several bit fields that further describe the particular BDS configuration. These bit fields are:

zbds-f.enabled
   is set to 1 if BDS is enabled and set to 0 if BDS is disabled.

zbds-f.tmds
   in the EMS alternate collector, this bit is set to 1 if it is running in TMDS mode; otherwise, it is set to 0. The EMS primary collector always sets this bit to 0.

zbds-f.use-subj
   is set to 1 if the subject token code is used to determine similar events; this bit is set to 0 if the subject token code is ignored in determining similar events.

zbds-f.process-type
   indicates the process type of the process performing BDS. The alternate collector stores ZEMS-SUBJ-ACOLL in this field. The primary collector stores ZEMS-SUBJ-PCOLL.

zbds-num-events
   identifies the number of events that constitute an event burst. This field provides the same information as the N parameter in the EMSACOLL and EMSCCTRL RUN commands.

zbds-interval-start
   the occurrence of N events during this time interval (in seconds) constitutes the start of an event burst. This field provides the same information as the T1 parameter in the EMSACOLL and EMSCCTRL RUN commands.

zbds-interval-end
   the occurrence of no events during this time interval (in seconds) constitutes the end of an event burst. This field provides the same information as the T2 parameter in EMSACOLL and EMSCCTRL.

zbds-interval-chk-end
   indicates the maximum interval (in seconds) in which the EMS collector checks for a burst end condition. In other words, the collector waits no longer than this interval between burst end checks. This is the same as the T3 parameter in EMSACOLL and EMSCCTRL.
zbds-num-sim-burst

indicates the maximum number of simultaneous event bursts that can be detected. This is the same as the S parameter in EMSACOLL and EMSCCTRL.

zbds-num-subj-bytes

indicates the maximum number of bytes of the first subject value that are used in determining same events for burst detection and suppression. A length of 0 signifies that the subject value is not used in determining same events. For BDS purposes, events are the same if they have the same SSID and event number. The maximum value for this field is 254. This field provides the same information as the L parameter (when L is not negative) used in EMSACOLL and EMSCCTRL.

ZEMS-MAP-BDS-INFO

is the token map returned in the ZCOM-CMD-STATUS and ZCOM-CMD-INFO commands when the object type is ZCOM-OBJ-COLL (an EMS collector). The fields in this token describe the BDS configuration in the collector.

```
DEFINITION zems-ddl-bds-info
  02 zbds-flags TYPE zspi-ddl-uint.
  02 zbds-f redefines zbds-flags
    03 enabled TYPE BIT 1.
    03 tmds TYPE BIT 1.
    03 use-subj TYPE BIT 1.
    03 process-type TYPE BIT 4.
  02 zbds-num-events TYPE zspi-ddl-uint.
  02 zbds-interval-start TYPE zspi-ddl-uint.
  02 zbds-interval-end TYPE zspi-ddl-uint.
  03 zbds-interval-chk-end TYPE zspi-ddl-uint.
  02 zbds-num-sim-bursts TYPE zspi-ddl-uint SPI-NULL 255.
  02 zbds-num-subj-bytes TYPE zspi-ddl-uint SPI-NULL 255.
END
```

zbds-flags

contains several bit fields that further describe the particular BDS configuration. These bit fields are:

zbds-f.enabled

is set to 1 if BDS is enabled and set to 0 if BDS is disabled.

zbds-f.tmds

in the EMS alternate collector, is set to 1 if it is running in TMDS mode. Otherwise, it is set to 0. The EMS primary collector always sets this bit to 0.
zbds-f.use-subj

is set to 1 if the subject token code is used to determine similar events. This bit is set to 0 if the subject token code is ignored in determining similar events.

zbds-f.process-type

indicates the process type of the process performing BDS. The alternate collector stores ZEMS-SUBJ-ACOLL in this field; the primary collector stores ZEMS-SUBJ-PCOLL.

zbds-num-events

indicates the number of events that constitute a burst. It provides the same information as the N parameter in the EMSACOLL and EMSCCTRL RUN commands.

zbds-interval-start

the occurrence of N events during this time interval (in seconds) constitutes the start of an event burst. This field provides the same information as the T1 parameter in the EMSACOLL and EMSCCTRL RUN commands.

zbds-interval-end

no events occurring in this time interval (in seconds) constitute the end of an event burst. This is the same as the T2 parameter in EMSACOLL and EMSCCTRL.

zbds-interval-chk-end

indicates the maximum interval (in seconds) in which the EMS collector checks for a burst end condition. In other words, the collector waits no longer than this interval between burst end checks. This is the same as the T3 parameter in EMSACOLL and EMSCCTRL.

zbds-num-sim-burst

indicates the maximum number of simultaneous event bursts that can be detected. This is the same as the S parameter in EMSACOLL and EMSCCTRL.

zbds-num-subj-bytes

indicates the maximum number of bytes of the first subject value that are used in determining same events for burst detection and suppression. A length of 0 signifies that the subject value is not used in determining same events. For BDS purposes, events are the same if they have the same SSID and event number. The maximum number of bytes is 254. This provides the same information as the L parameter (when L is not negative) used in EMSACOLL and EMSCCTRL.
ZEMS-MAP-BDS-STATS

is a token map returned in the ZEMS-CMD-STATUS command when the object type is ZCOM-OBJ-COLL (an EMS collector). The fields in this token contain various BDS statistics that are accumulated by the collector.

```
DEFINITION zems-ddl-bds-stats
  02 zbds-eventsdiscarded         TYPE zspi-ddl-int2.
  02 zbds-eventspassed            TYPE zspi-ddl-int2.
  02 zbds-peaksimultbursts        TYPE zspi-ddl-uint.
  02 zbds-peakwatchcount          TYPE zspi-ddl-int2.
  02 zbds-peakdiscardcount        TYPE zspi-ddl-int2.
  02 zbds-numberburststartevts    TYPE zspi-ddl-int2.
  02 zbds-numberburstendevts      TYPE zspi-ddl-int2.
END
```

zbds-eventsdiscarded

is the number of events that have been discarded by the BDS feature since BDS was enabled. This counter is reset whenever BDS is enabled.

zbds-eventspassed

is the number of events that have been passed by the BDS feature since BDS was enabled. This counter is reset whenever BDS is enabled.

zbds-peaksimultbursts

is the highest number of simultaneous event bursts that have occurred since BDS was enabled. This counter is reset whenever BDS is enabled.

zbds-peakwatchcount

is the highest number of events being watched that have occurred since BDS was enabled. Watched events are events that have occurred more than once within T1 time units and so are potential bursts. This count is reset whenever BDS is enabled.

zbds-peakdiscardcount

is the highest number of events that have been discarded in a single event burst. This count is reset whenever BDS is enabled.

zbds-numberburststartevts

is the number of BURST-START events that have been generated by the EMS collector since BDS was enabled. This count is reset whenever BDS is enabled.
zbds-numberburstendevts

is the number of BURST-END events that have been generated by the collector since BDS was enabled. This count is reset whenever BDS is enabled.

ZEMS-MAP-PLF-STATS

is the token map returned in the ZCOM-CMD-STATUS command when the object type is ZCOM-OBJ-COLL (an EMS collector). The fields in this token contain various PLF statistics accumulated by the collector.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zplf-eventspassed</td>
<td>is the number of events that have been passed by the pre-log filtration (PLF) feature since PLF was enabled. This counter is reset whenever PLF is enabled.</td>
</tr>
<tr>
<td>zplf-eventsfailed</td>
<td>is the number of events failed (discarded) by the PLF feature since PLF was enabled. This counter is reset whenever PLF is enabled.</td>
</tr>
</tbody>
</table>

ZEMS-MAP-BURST-STATUS

is the token map returned in the ZCOM-CMD-STATUS command when the object type is ZCOM-OBJ-COLL (an EMS collector). The fields in this token describe a particular or potential event burst. Many of these tokens can be returned in the ZCOM-CMD-STATUS response. The number returned is limited by the configured value for simultaneous event bursts, and it might be even less because only entries in the BURST state are returned. Each of these tokens corresponds to an entry in the burst table that resides in the EMS collector data space.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zburst-state</td>
<td>TYPE zspi-didl-uint.</td>
</tr>
<tr>
<td>zburst-count</td>
<td>TYPE zspi-didl-int2.</td>
</tr>
<tr>
<td>zburst-last-time</td>
<td>TYPE zspi-didl-timestamp.</td>
</tr>
<tr>
<td>zburst-first-time</td>
<td>TYPE zspi-didl-timestamp.</td>
</tr>
<tr>
<td>zburst-event-ssid</td>
<td>TYPE zspi-didl-ssid.</td>
</tr>
<tr>
<td>zburst-event-number</td>
<td>TYPE zspi-didl-uint.</td>
</tr>
<tr>
<td>zburst-subj-ssid</td>
<td>TYPE zspi-didl-ssid.</td>
</tr>
<tr>
<td>zburst-subj-tkncode</td>
<td>TYPE zspi-didl-tokencode.</td>
</tr>
<tr>
<td>zburst-subj-length</td>
<td>TYPE zspi-didl-uint.</td>
</tr>
</tbody>
</table>
zburst-state

is the state of the event burst. The legal values are:

- ZEMS-VAL-BSTATE-AVAILABLE
- ZEMS-VAL-BSTATE-PASS
- ZEMS-VAL-BSTATE-WATCH
- ZEMS-VAL-BSTATE-BURST

ZEMS-VAL-BSTATE-AVAILABLE signifies that the burst table entry is not in use.

ZEMS-VAL-BSTATE-PASS signifies that the burst table entry contains data on a particular event but that no more than one has been encountered in the last zbds-start-interval (T1) seconds.

ZEMS-VAL-BSTATE-WATCH signifies that between 2 and the <value of ZBDS- NUM EVENTS minus 1> occurrences of the event described in this entry have been encountered in the last zbds-start-interval (T1) seconds.

ZEMS-VAL-BSTATE-BURST signifies that zbds-num-events or more occurrences of the event described in this entry have been encountered in the last zbds-start-interval (T1) seconds. This is an active event burst, and the event is no longer being logged to disk.

**Note.** Normally, the ZCOM-CMD-STATUS command only returns tokens for burst table entries that are in the BURST state.

zburst-count

is the number of occurrences of this event since the last state change.

zburst-last-time

when this entry is in the BURST state, this field contains the generation timestamp of the last encountered event described in this entry.

zburst-first-time

the meaning of this field depends on this entry’s state:

<table>
<thead>
<tr>
<th>State</th>
<th>Generation timestamp of the...</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS</td>
<td>Last encountered event described in this entry</td>
</tr>
<tr>
<td>WATCH</td>
<td>First event of the potential event burst</td>
</tr>
<tr>
<td>BURST</td>
<td>First event discarded because of this event burst</td>
</tr>
</tbody>
</table>

zburst-event-ssid

is the SSID of the event that this entry is monitoring.

zburst-event-number

is the event number of the event that this entry is monitoring.
zburst-subj-ssid
is the SSID of the first subject of the event that this entry is monitoring.

zburst-subj-tkncode
is the token code of the first subject of the event that this entry is monitoring.

zburst-subj-length
is the byte length of the value of the first subject of the event that this entry is monitoring.

ZEMS-MAP-COL-CONTROL
is the command token used by the ZEMS-CMD-ALTER command when the object type is ZCOM-OBJ-COLL (an EMS collector).

```
DEFINITION zems-ddl-col-control
  02 zcol-primarycpu         TYPE zspi-ddl-uint.
  02 zcol-logsubvol          TYPE zspi-ddl-subvol.
  02 zcol-nextlogfile        TYPE zspi-ddl-boolean.
  02 zcol-rotatefiles        TYPE zspi-ddl-boolean
  02 zcol-maxfilennn         TYPE zspi-ddl-uint SPI-NULL 255.
  02 zcol-primaryextent      TYPE zspi-ddl-uint SPI-NULL 255.
  02 zcol-secondaryextent    TYPE zspi-ddl-boolean.
  02 zcol-writethrucache     TYPE zspi-ddl-boolean.
  02 zcol-discaccessid       TYPE zspi-ddl-boolean.
  02 zcol-protection         TYPE zspi-ddl-uint.
  02 zcol-eventblocking      TYPE zspi-ddl-boolean.
  02 zcol-burstsupdetect     TYPE zspi-ddl-boolean.
  02 zcol-prelogfilter       TYPE zspi-ddl-boolean.
END
```

zcol-primarycpu
is the current primary CPU of the collector process.

zcol-logsubvol
is the current log subvolume.

zcol-nextlogfile
if TRUE, directs a collector to close the current log file, create the next sequential file, and open it. The collector generates a ZEMS-EVT-FILESWITCH event message and resumes event-message logging with the new file.

zcol-rotatefiles
a value of TRUE (ZSPI-VAL-TRUE) indicates that ROTATEFILES is on. A value of FALSE (ZSPI-VAL-FALSE) indicates that ROTATEFILES is off.
**zcol-maxfilennn**

is the maximum number of log files that are allowed in the current subvolume.

**zcol-primaryextent**

is the primary extent of the next log file that is created.

**zcol-secondaryextent**

is the secondary extent of the next log file that is created.

**zcol-writethrucache**

a value of TRUE (ZSPI-VAL-TRUE) indicates that BUFFERED is off. A value of FALSE (ZSPI-VAL-FALSE) indicates that BUFFERED is on.

**zcol-eofrefresh**

selects a new value for the EOFREFRESH attribute. A value of true in this field sets EOFREFRESH to ON, which directs a collector to update the end-of file pointer on disk for each block written to the log file. This method is safer than updating the pointer less often, but is less efficient because it involves more writes to disk. The default value of this attribute is OFF.

**zcol-discaccessid**

is the user ID used in the next log file created or opened.

**zcol-protection**

is the security used in the next log file created.

**zcol-eventblocking**

a value of TRUE (ZSPI-VAL-TRUE) indicates that BLOCKING is on. A value of FALSE (ZSPI-VAL-FALSE) indicates that BLOCKING is off.

**zcol-burstsupdetect**

a value of TRUE (ZSPI-VAL-TRUE) indicates that BDS is enabled. A value of FALSE (ZSPI-VAL-FALSE) indicates that BDS is disabled.

**zcol-prelogfilter**

a value of TRUE (ZSPI-VAL-TRUE) indicates that PLF is enabled. A value of FALSE (ZSPI-VAL-FALSE) indicates that PLF is disabled.
Common Response Tokens for ZCOM- Commands

`ZEMS-MAP-COL-STATUS` is a token map returned in the ZCOM-CMD-STATUS command when the object type is ZCOM-OBJ-COLL (a collector).

```
DEFINITION zems-ddl-col-status
  02 zcol-primarycpu               TYPE zspi-ddl-uint.
  02 zcol-backupcpu                TYPE zspi-ddl-uint.
  02 zcol-priority                 TYPE zspi-ddl-uint.
  02 zcol-logdiscerror             TYPE zspi-ddl-uint.
  02 zcol-currentfilename          TYPE zspi-ddl-fname.
  02 zcol-currentrecord            TYPE zspi-ddl-int2.
  02 zcol-defaultfilename          TYPE zspi-ddl-fname.
  02 zcol-primaryextent            TYPE zspi-ddl-uint.
  02 zcol-secondaryextent          TYPE zspi-ddl-uint.
  02 zcol-rotatefiles              TYPE zspi-ddl-boolean.
  02 zcol-maxfilennnn              TYPE zspi-ddl-uint.
  02 zcol-eofrefresh               TYPE zspi-ddl-boolean.
  02 zcol-writethrcache            TYPE zspi-ddl-boolean.
  02 zcol-protection               TYPE zspi-ddl-boolean.
  02 zcol-eventsreceived           TYPE zspi-ddl-int2.
  02 zcol-eventslogged             TYPE zspi-ddl-int2.
  02 zcol-eventsdiscarded          TYPE zspi-ddl-int2.
  02 zcol-opensreceived            TYPE zspi-ddl-uint.
  02 zcol-closesreceived           TYPE zspi-ddl-uint.
  02 zcol-fileswitches             TYPE zspi-ddl-uint.
  02 zcol-discerrors               TYPE zspi-ddl-uint.
  02 zcol-invaliddevents           TYPE zspi-ddl-uint.
  02 zcol-bufferfailures           TYPE zspi-ddl-uint.
  02 zcol-eventblocking            TYPE zspi-ddl-boolean.
  02 zcol-burstsupdetect           TYPE zspi-ddl-boolean.
  02 zcol-prelogfilter             TYPE zspi-ddl-boolean.
  02 zcol-currentbursts            TYPE zspi-ddl-uint.
END
```

- `zcol-primarycpu` is the current primary CPU of the collector process.
- `zcol-backupcpu` is the backup CPU number of the collector responding to the command.
- `zcol-priority` is the current execution priority of the collector responding to the command.
- `zcol-logdiscerror` is the status of the last disk operation on the log file of the collector responding to the command. A value of zero indicates a successful operation. A nonzero value is the file management number that causes logging to stop.
zcol-currentfilename
This is the file name of the current log file for the collector responding to the command.

zcol-currentrecord
is the record number (as received from the FILEINFO procedure) of the last record written to the log file of the collector responding to the command. If the collector is currently switching log files, ZCOL-CURRENTRECORD is -1.

zcol-defaultfilename
is the file name of the last file in use in the default subvolume; this field can be the same as ZCOL-CURRENTFILENAME.

zcol-primaryextent
is the current setting of the PRIMARYEXTENT attribute.

zcol-secondaryextent
is the current setting of the SECONDARYEXTENT attribute.

zcol-rotatefiles
is the current setting of the ROTATEFILES attribute.

zcol-maxfilen
is the (current) maximum number of log files that the responding collector can create in this log subvolume.

zcol-eofrefresh
is a flag that indicates whether frequent updating of the end-of-file position—each time the disk process writes a block to the log file—has been selected.

zcol-writethru
is a flag that indicates whether log-file buffering has been selected. TRUE signifies that buffering is off.

zcol-protection
is the set of file-security values for new log files.

zcol-eventsreceived
is the number of event notifications received by an alternate collector or by a primary collector since the last system load. These notifications are not all event messages.
**zcol-eventslogged**

is the total number of event messages an alternate collector has received or a primary collector logged to disk log files since the last system load.

**zcol-eventsdiscarded**

is the number of event messages that a collector had to discard because of event message flooding or because BDS or PLF was enabled.

**zcol-opensreceived**

is the total number of #ZSPI subdevice OPENs accepted by a collector (since the last system load for a primary collector).

**zcol-closesreceived**

is the total number of #ZSPI CLOSEs received by a collector (since the last system load for a primary collector).

**zcol-fileswitches**

is the total number of times a collector has switched log files. This number includes both requested switches and switches triggered by log files becoming full.

**zcol-discerrors**

is the total number of disk errors returned to a collector by the disk process.

**zcol-invalidevents**

is the total number of event messages a collector discarded because of event-message format errors.

**zcol-bufferfailures**

is the total number of event messages a collector discarded because of insufficient memory-pool capacity.

**zcol-eventblocking**

the EMS collector returns TRUE if blocking is ON and FALSE if blocking is OFF.

**zcol-burstsupdetect**

the EMS collector returns a value of TRUE (ZSPI-VAL-TRUE) if BDS is enabled and a value of FALSE (ZSPI-VAL-FALSE) if BDS is disabled.

**zcol-prelogfilter**

the EMS collector returns a value of TRUE (ZSPI-VAL-TRUE) if BDS is enabled and a value of FALSE (ZSPI-VAL-FALSE) if BDS is disabled.
**zcol-currentbursts**

is the number of the currently detected event bursts.

**ZEMS-MAP-COL-INFO**

is the token map returned in the ZCOM-CMD-INFO command when the object type is ZCOM-OBJ-COLL (a collector). This contains all the configuration information common to both EMS collectors. The information returned is similar to what is returned in the -COL-STATUS structure of the ZCOM-CMD-STATUS command.

```
DEFINITION zems-ddl-col-info
  02 zcol-primarycpu           TYPE zspi-ddl-uint.
  02 zcol-backupcpu            TYPE zspi-ddl-uint.
  02 zcol-logsubvol            TYPE zspi-ddl-subvol.
  02 zcol-defaultlogsubvol     TYPE zspi-ddl-subvol.
  02 zcol-primaryextent        TYPE zspi-ddl-uint.
  02 zcol-secondaryextent      TYPE zspi-ddl-uint.
  02 zcol-rotatefiles          TYPE zspi-ddl-boolean.
  02 zcol-maxfilenfnn          TYPE zspi-ddl-uint.
  02 zcol-eofrefresh           TYPE zspi-ddl-boolean.
  02 zcol-writethrucache       TYPE zspi-ddl-boolean.
  02 zcol-protection           TYPE zspi-ddl-uint.
  02 zcol-discaccessid         TYPE zspi-ddl-uint.
  02 zcol-eventblocking        TYPE zspi-ddl-boolean.
  02 zcol-burstsupdetect       TYPE zspi-ddl-boolean.
  02 zcol-prelogfilter         TYPE zspi-ddl-boolean.
END
```

**zcol-primarycpu**

is the current primary CPU of the collector process.

**zcol-backupcpu**

is the current backup CPU of the collector process.

**zcol-logsubvol**

is the current log subvolume.

**zcol-defaultlogsubvol**

is the current default log subvolume. Currently, this configuration attribute cannot be altered after the collector has started.

**zcol-primaryextent**

is the primary extent of the next log file created.

**zcol-secondaryextent**

is the secondary extent of the next log file created.
zcol-rotatefiles
this value of TRUE (ZSPI-VAL-TRUE) indicates that ROTATEFILES is on. A value of FALSE (ZSPI-VAL-FALSE) indicates that ROTATEFILES is off.

zcol-maxfilennnn
is the maximum number of log files that are allowed in the current subvolume.

zcol-eofrefresh
this value of TRUE (ZSPI-VAL-TRUE) indicates that EOFREFRESH is on. A value of FALSE (ZSPI-VAL-FALSE) indicates that EOFPREFRESH is off.

zcol-writethrucache
this value of TRUE (ZSPI-VAL-TRUE) indicates that BUFFERED is off. A value of FALSE (ZSPI-VAL-FALSE) indicates that BUFFERED is on.

zcol-protection
is the security used in the next log file created.

zcol-discaccessid
is the user ID used in the next log file created or opened.

zcol-eventblocking
this value of TRUE (ZSPI-VAL-TRUE) indicates that BLOCKING is on. A value of FALSE (ZSPI-VAL-FALSE) indicates that BLOCKING is off.

zcol-burstsupdetect
this value of TRUE (ZSPI-VAL-TRUE) indicates that BDS is enabled. A value of FALSE (ZSPI-VAL-FALSE) indicates that BDS is disabled.

zcol-prelogfilter
this value of TRUE (ZSPI-VAL-TRUE) indicates that PLF is enabled. A value of FALSE (ZSPI-VAL-FALSE) indicates that PLF is disabled.

ZEMS-MAP-ACOL-INFO
is a token map returned in the ZCOM-CMD-INFO command when the object type is ZCOM-OBJ-COLL (an EMS collector) and the collector is an alternate collector. This token map contains all of the configuration information unique to the alternate
collector. The information returned is similar to that returned in the -ACOL-STATUS structure of the ZCOM-CMD-STATUS command.

```
DEFINITION zems-ddl-acol-info12345
  02 zcol-poolpages     TYPE zspi-ddl-uint.
  02 zcol-replyafterwrite TYPE zspi-ddl-boolean.
END
```

**zcol-poolpages**

indicates the number of 2-KB pages that the alternate collector has available for buffering events before logging them to disk. This configuration attribute cannot be altered after the alternate collector has started.

**zcol-replyafterwrite**

this value of TRUE (ZSPI-VAL-TRUE) indicates that REPLYAFTERWRITE is enabled. A value of FALSE (ZSPI-VAL-FALSE) indicates that REPLYAFTERWRITE is disabled. This configuration attribute cannot be altered after the collector has started.

**ZEMS-MAP-COL-CDIST-INFO**

is the token map returned in the ZCOM-CMD-INFO command when the object type is ZCOM-OBJ-COLL and the collector is the primary collector. This token map contains all of the configuration information for the compatibility distributor (process $Z0), which is controlled by the primary collector. The information returned is similar to that returned in the -COL-CDIST-STATUS structure of the ZCOM-CMD-STATUS command.

```
DEFINITION zems-ddl-col-cdist-info
  02 zcol-cdist-pricpu       TYPE zspi-ddl-uint SPI-NULL 255.
  02 zcol-cdist-bkupcpu      TYPE zspi-ddl-uint SPI-NULL 255.
  02 zcol-cdist-mode         TYPE zspi-ddl-enum.
  02 zcol-cdist-textout      TYPE zspi-ddl-fname.
  02 zcol-cdist-user         TYPE zspi-ddl-userid.
  02 zcol-cdist-def-textout  TYPE zspi-ddl-fname.
END
```

**zcol-cdist-pricpu**

is the current primary CPU of the compatibility distributor.

**zcol-cdist-bkupcpu**

is the current backup CPU of the compatibility distributor.

**zcol-cdist-mode**

is the current operating mode of the compatibility distributor.
**zcol-cdist-textout**

is the current textout device of the compatibility distributor.

**zcol-cdist-user**

is the current userid employed by the compatibility distributor when writing to the textout device.

**zcol-cdist-def-textout**

is the default textout device of the compatibility distributor. Currently, this configuration attribute cannot be altered after the primary collector has started.

**ZEMS-MAP-STATUS-FILTER**

is the token map returned in the ZCOM-CMD-STATUS command when the object type is ZCOM-OBJ-FILTER. This same token map is returned by the EMS distributor when it processes the same command.

```plaintext
DEFINITION zems-ddl-status-filter
  02 zdist-filterfile-name TYPE zspi-ddl-fname.
  02 zdist-filter-name   TYPE zems-ddl-char30.
  02 zdist-filter-size   TYPE zems-ddl-uint.
  02 zdist-filter-flags  TYPE zems-ddl-uinit.
END
```

**zdist-filterfile-name**

contains the name of the relevant filter object file for a compiled filter. For filter tables and burst filters, this is the name for the converted EDIT file.

**zdist-filter-name**

contains the name of the filter, if any. This is the name specified in the filter source code for a compiled filter.

**zdist-filter-size**

contains the file size, in bytes, of the filter.

**zdist-filter-flags**

identifies the filter type. The valid values are ZEMS-VAL-FILTYPE-COMPILED, ZEMS-VAL-FILTYPE-PASS, ZEMS-VAL-FILTYPE-FAIL, and ZEMS-VAL-FILTYPE-BURST.

**ZEMS-TKN-BURST-EVT-NUM**

is a token of the type ZSPI-TYP-ENUM that contains the event number of a bursting event.
ZEMS-TKN-BURST-SSID
is a token of the type ZSPI-TYP-SSID that contains the subsystem ID (SSID) of a bursting event.

ZEMS-TKN-BURST-SUBJ-CODE
is a token of the type ZSPI-TYP-TOKENCODE that contains the first subject token code of a bursting event.

ZEMS-TKN-BURST-SUBJ-VALUE
is a token of the type ZSPI-TYP-STRING that contains the value of the first subject token of a bursting event.

ZEMS-TKN-BURST-SUBJ-SSID
is a token of the type ZSPI-TYP-SSID that contains the subsystem ID (SSID) of the first subject token of a bursting event.

ZEMS-TKN-BURST-TIME-START
is a token of the type ZSPI-TYP-TIMESTAMP that contains the Julian timestamp of the event burst start time.

ZEMS-TKN-BURST-TIME-END
is a token of the type ZSPI-TYP-TIMESTAMP that contains the Julian timestamp of the event burst end time.

ZEMS-TKN-BURST-EVTS-DELETED
is a token of the type ZSPI-TYP-INT2 that contains the number of burst events deleted during the duration of the specified event burst.

ZEMS-TKN-BURST-END-REASON
is a token of the type ZSPI-TYP-ENUM that contains the reason for the event burst end. This token can have values of ZEMS-VAL-BDS-DISABLED and ZEMS-VAL-NO-EVENTS.

ZEMS-TKN-XFILTERFILE
is a token of the type ZSPI-TYP-BYTESTRING that contains the fully qualified name of a filter file. One of these tokens is stored in the ZEMS-EVT-FILESWITCH event for each filter that is operating when PLF in enabled.

ZEMS-TKN-EVT-EVTNUM
is a token of the type ZSPI-TYP-ENUM that contains the event number of the event being examined when a PLF error occurred. This token is stored in the ZEMS-TKN-PLF-ERROR event.
ZEMS-TKN-EVT-GENTIME

is a token of the type ZSPI-TYP-TIMESTAMP that contains the generation timestamp of the event being examined when a PLF error occurred. This token is stored in the ZEMS-TKN-FILTER-ERROR event.

ZEMS-TKN-NEW-OBJNAME

is a token of the type ZSPI-TYP-STRING that contains the fully qualified name of a new object. This token is used in the ZEMS-CMD-REPLACE command where it contains the name of the filter that is to replace the filter specified in the ZCOM-TKN-OBJNAME token.

ZEMS-TKN-ZOPR-CMD

is a token of the type ZSPI-TYP-ENUM that is put into a SPI reply when an error occurs during communication with the $ZOPR process. The value stored in this token is the specific $ZOPR command.

Common Definitions of ZEMS- Commands

This subsection defines the tokens and error numbers common to the older, basic SPI-compliant collector commands: ZEMS-CMD-CONTROL, ZEMS-CMD-STATUS, ZEMS-CMD-GETVERSION, and ZEMS-CMD-STOP.

Common Tokens for ZEMS- Commands

Tokens common to all of these SPI commands are defined here in alphabetical order. All SPI commands use additional tokens defined with the command in which they are used.

Token types abbreviated in italics represent SPI types. For example, int represents ZSPI-TYP-INT.

ZSPI-TKN-COMMAND (type enum)

is the standard SPI token for defining the command to be executed. It is contained in both the request and reply buffers. It can have these values for the primary and alternate collector:

ZEMS-CMD-GETVERSION       0
ZEMS-CMD-STATUS            1
ZEMS-CMD-CONTROL           2
ZEMS-CMD-STOP              9 (alternate collector only)

ZSPI-TKN-RETCODE (type enum)

is the standard SPI return token. For ZEMS-TKN-RETCODE values and related information, see Common ZSPI-TKN-RETCODE Values on page 19-28.
ZSPI-TKN-SERVER-VERSION (type uint)

is the subsystem version number of the EMS collector. It is contained in a response buffer.

ZSPI-TKN-SSID (type ssid)

contains ZSPI-SSN-ZEMS, the subsystem ID for the Event Management Service. It is contained in a response buffer.

**Common ZSPI-TKN-RETCODE Values**

Table 19-5 shows the names and numbers of ZSPI-TKN-RETCODE values returned by a primary or alternate collector if a command message cannot be correctly interpreted. Errors that occur in a particular command are defined with that command.

For detailed descriptions of collector error codes and related error lists, see Section 22, Collector Errors.

<table>
<thead>
<tr>
<th>Name (ZEMS-ERR-)</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION</td>
<td>1001</td>
<td>Version not supported</td>
</tr>
<tr>
<td>INV-CMD</td>
<td>1002</td>
<td>Invalid command</td>
</tr>
<tr>
<td>INV-SSID</td>
<td>1003</td>
<td>Invalid subsystem ID (SPI buffer not owned by EMS)</td>
</tr>
<tr>
<td>INV-TKN</td>
<td>1004</td>
<td>Extra or unrecognized token</td>
</tr>
<tr>
<td>INV-VALUE</td>
<td>1005</td>
<td>Illegal token value</td>
</tr>
<tr>
<td>DUP-TKN</td>
<td>1006</td>
<td>Duplicate token code</td>
</tr>
<tr>
<td>INV-OBJECT</td>
<td>1008</td>
<td>Invalid object type</td>
</tr>
<tr>
<td>INV-OCCURS</td>
<td>1017</td>
<td>Application buffer too small for response</td>
</tr>
<tr>
<td>ZSPI</td>
<td>1041</td>
<td>SPI error while decoding command buffer or building response buffer</td>
</tr>
</tbody>
</table>
Collector Command Descriptions

Each command description in this subsection contains:

- A brief summary of what the command does
- Information about command tokens—tokens placed in the command buffer when building a command message
- Information about response tokens—tokens returned in the buffer as a response message
- When appropriate, this additional information:
  - Token usage considerations
  - Operational notes
  - Programming notes
  - Error-handling notes

Note. In the ZEMS- prefixed commands in this subsection:

- The command tokens ZSPI-TKN-COMMAND, ZSPI-TKN-OBJECT-TYPE, and ZSPI-TKN-SSID are not listed or described because they are specified in the SSINIT call that creates the command. For descriptions of these command tokens, see Common Tokens for ZEMS- Commands on page 19-27.

- The response tokens ZSPI-TKN-COMMAND, ZSPI-TKN-OBJECT-TYPE, ZSPI-TKN-SSID, and ZSPI-TKN-SERVER-VERSION are not listed or described because their descriptions in Common Definitions of ZEMS- Commands on page 19-27 apply to all command responses.
ADD Command (ZCOM-CMD-ADD)

The ZCOM-CMD-ADD command adds one or more filter objects (ZCOM-OBJ-FILTER) to a collector. The filter can be a compiled filter, a filter table, or a burst filter. A maximum of ten filters can be added to a collector, including any combination of compiled filters, filter tables, and one burst filter.

If required parameter tokens are specified in the filter being added, these parameter tokens must be included in the ADD command. If these parameter tokens are optional, including them in the ADD command is optional.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxx</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>&lt;filter parameter token(s)&gt;</td>
<td>ZSPI-TYP-ENUM</td>
<td>Filter</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Optional signifies that the token is optional and need not be supplied by the user.
- Required signifies that the token is required and must be supplied by the user.
- Filter signifies that the required filter parameter tokens are required only if they are defined as required parameter tokens in the filter.
- Unconditional signifies that the token always appears in the response.
The data list (all tokens from -DATALIST to -ENDLIST) forms a single response record. There can be multiple data lists (response records) in a response.

Error only signifies that this token is present only when an error or warning is reported in the response record (ZSPI-TKN-RETCODE contains a nonzero value).

For a detailed description of these command and response tokens, see Common Definitions of ZCOM- Commands on page 19-4.

ALTER Command (ZCOM-CMD-ALTER)

The ZCOM-CMD-ALTER command alters objects in the EMS collector. The supported objects are collectors (ZCOM-OBJ-COLL) and EMS filters (ZCOM-OBJ-FILTER). Exactly one object name token is required with this command.

When the object is the EMS collector, this command functions like its basic SPI counterpart (ZEMS-CMD-CONTROL). The only additional tokens required are the collector control maps.

When the object is an EMS filter, the additional tokens are filter parameter tokens when filter parameters are specified in the filter. Each ZCOM-CMD-ALTER command resets the filter parameters for the specified filters, so all required filter parameters must have tokens present in the command. The presence of tokens for the optional filter parameters is optional.

<table>
<thead>
<tr>
<th>Command Tokens (-OBJ-COLL)</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxx</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZEMS-MAP-COL-CONTROL</td>
<td>ZEMS-DDL-ACOL-CONTROL</td>
<td>Required</td>
</tr>
<tr>
<td>ZEMS-MAP-ACOL-CONTROL</td>
<td>ZEMS-DDL-ACOL-CONTROL</td>
<td>AC only</td>
</tr>
<tr>
<td>ZEMS-TKN-CDISTPRICPU</td>
<td>ZSPI-TYP-UINT</td>
<td>PC only</td>
</tr>
<tr>
<td>ZEMS-MAP-COL-CONTROL-CDIST</td>
<td>ZEMS-DDL-BDS-CONFIG</td>
<td>Required</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>
Token Usage Considerations

- **Basic SPI** signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.

- **Optional** signifies that the token is optional and need not be supplied by the user.

- **Required** signifies that the token is required and must be supplied by the user.

- **AC only** signifies that the item is required for the alternate collector and is invalid for the primary collector.

- **PC only** signifies that the item is required for the primary collector and is invalid for the alternate collector.

- The `-CDISTPRICPU` and `-COL-CONTROL-CDIST` tokens cannot both be present. One token or the other (or neither) must be present.

- **Filter** signifies that the required filter parameter tokens are required only if they are defined as required parameter tokens in the filter.

- The data list (all tokens from `-DATALIST` to `-ENDLIST`) forms a single response record. **Unconditional** signifies that the token always appears in the response.

- **Unconditional** signifies that the token always appears in the response.
• Error only signifies that this token is present only when an error or warning is reported in the response record (ZSPI-TKN-RETCODE contains a nonzero value).

For a detailed description of these command and response tokens, see Common Definitions of ZCOM- Commands on page 19-4.
CONTROL Command (ZEMS-CMD-CONTROL)

The ZEMS-CMD-CONTROL command lets you programmatically manage the operation of the primary and alternate collectors and the compatibility distributor. You do this by changing the settings of one or more operational attributes for the appropriate process. Table 19-6 shows the attributes you can set when you use the CONTROL command and the EMS process controlled by each attribute.

Table 19-6. Attributes Changed by the CONTROL Command

<table>
<thead>
<tr>
<th>Option</th>
<th>Primary Collector</th>
<th>Alternate Collector</th>
<th>Compatibility Distributor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary CPU</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Log subvolume</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Next logfile</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Primary extent</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Secondary extent</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Rotate files</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Maxfile</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>EOF refresh</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Write through cache</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Protection</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Blocking</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>BDS</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>PLF</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Discaccessid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocate</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Console out</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Textout set</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Mode set</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>User set</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

In addition to the required token ZSPI-TKN-COMMAND, which specifies the command type to the SPI interface, each CONTROL command buffer must contain tokens or token maps that define the values of attributes to be changed. You include only the values of the attributes to be changed. The values of the attributes not included remain the same. Two fields in the ZEMS-MAP-COL-CONTROL token map control the enabling and disabling of BDS and PLF.
This is detailed information about the tokens and token maps in a CONTROL command buffer:

<table>
<thead>
<tr>
<th>Command</th>
<th>Required Tokens or Token Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-CMD-CONTROL</td>
<td>ZEMS-MAP-COL-CONTROL</td>
</tr>
<tr>
<td></td>
<td>ZEMS-MAP-COL-CONTROL or ZEMS-MAP-ACOL-CONTROL</td>
</tr>
<tr>
<td></td>
<td>ZEMS-TKN-CDISTPRICPU or ZEMS-MAP-COL-CONTROL-CDIST</td>
</tr>
</tbody>
</table>

**Tokens in Command Buffer**

ZEMS-MAP-COL-CONTROL

```ini
DEF ZEMS-DDL-COL-CONTROL.
  02 ZCOL-PRIMARYCPU type ZSPI-DDL-UINT.
  02 ZCOL-LOGSUBVOL type ZSPI-DDL-SUBVOL.
  02 ZCOL-NEXTLOGFILE type ZSPI-DDL-BOOLEAN.
  02 ZCOL-ROTATEFILES type ZSPI-DDL-BOOLEAN.
  02 ZCOL-MAXFILENNNN type ZSPI-DDL-UINT.
  02 ZCOL-PRIMARYEXTENT type ZSPI-DDL-UINT.
  02 ZCOL-SECONDARYEXTENT type ZSPI-DDL-UINT.
  02 ZCOL-WRITETHRUCACHE type ZSPI-DDL-BOOLEAN.
  02 ZCOL-EOFREFRESH type ZSPI-DDL-BOOLEAN.
  02 ZCOL-DISCCACHE types ZSPI-DDL-BOOLEAN.
  02 ZCOL-PROTECTION type ZSPI-DDL-BOOLEAN.
  02 ZCOL-EVENTBLOCKING type ZSPI-DDL-BOOLEAN.
  02 ZCOL-BURSTSUPDETECT type ZSPI-DDL-BOOLEAN.
  02 ZCOL-PRELOGFILTER type ZSPI-DDL-BOOLEAN.
END
```

ZEMS-MAP-ACOL-CONTROL

```ini
DEF ZEMS-DDL-ACOL-CONTROL.
  02 ZCOL-ALLOCATE type ZSPI-DDL-BOOLEAN.
END
```

ZEMS-TKN-CDISTPRICPU type ZSPI-TYP-UINT.

ZEMS-MAP-COL-CONTROL-CDIST

```ini
DEF ZEMS-DDL-COL-CONTROL-CDIST.
  02 ZCOL-CDIST-PRICPU type ZSPI-DDL-UINT.
  02 ZCOL-CDIST-CONSOLE-OUT type ZSPI-DDL-BOOLEAN.
  02 ZCOL-CDIST-TEXTOUT-SET type ZSPI-DDL-FNAME.
  02 ZCOL-CDIST-MODE-SET type ZSPI-DDL-ENUM.
  02 ZCOL-CDIST-USER-SET type ZSPI-DDL-USERID.
END
```
CONTROL Command (ZEMS-CMD-CONTROL)

Tokens in Command Buffer

ZEMS-MAP-COL-CONTROL

is an extensible structured token that lets you change collector attributes or give
commands to a primary or alternate collector. If you supply one or more collector-
attribute fields within the structured token, the collector changes the corresponding
attributes. If, in a particular command message, you do not supply a particular field,
the collector leaves the corresponding attribute unchanged. The attribute is left at
the value last given it, or at a system-supplied default value.

Some fields specify a new value for a collector attribute; for example, ZCOL-
MAXFILennn tells the collector how many log files the log subvolume may
contain. Other fields contain commands to the collector. For example, the ZCOL-
NEXTLOGFILE field, when set to TRUE, tells the collector to switch log files, and
ZCOL-PRIMARYCPU, when set to the backup CPU number, tells a collector to
switch to that CPU.

ZCOL-PRIMARYCPU (uint field)

contains the number of the CPU in which a collector is to operate. This is a
request for the collector to switch CPUs—that is, to begin operating in its
backup CPU. The only valid value for this field is the number of the current
backup CPU.

ZCOL-LOGSUBVOL (subvol field)

contains the names of the volume and subvolume a collector is to use to create
log files. These names are the first two parts of an HP internal-format file
name. If access to the identified subvolume is not possible, logging continues
in the current subvolume.

ZCOL-NEXTLOGFILE (Boolean field)

if TRUE, tells a collector to close the current log file, create the next sequential
file, and open it. The collector then generates a ZEMS-EVT-FILESWITCH
event message and resumes event-message logging with the new file.

ZCOL-ROTATEFILES (Boolean field)

sets a new value for the ROTATEFILES attribute. A value of TRUE in this field
causes ROTATEFILES to be set to ON, while FALSE sets the attribute to OFF.
**ZCOL-MAXFILENNNN** *(uint field)*

specifies the maximum number of files that can exist at one time in the log subvolume. Allowed values for this field are 2 through 1000; the default is 4.

**ZCOL-PRIMARYEXTENT** *(uint field)*

selects the value for the primary extent size used when a collector creates a log file. The value must be expressed as an even number of pages. The default value for this attribute is 20. For more information about file creation, see the CREATE procedure in the *Guardian Procedure Calls Reference Manual.*

**ZCOL-SECONDARYEXTENT** *(uint field)*

selects the value for the secondary extent size used when a collector creates a log file. The value must be expressed as an even number of pages. The default value for this attribute is 100. For more information about file creation, see the CREATE procedure in the *Guardian Procedure Calls Reference Manual.*

**ZCOL-WRITETHRUCACHE** *(Boolean field)*

selects a new value for the WRITETHRUCACHE attribute. A value of TRUE in this field sets WRITETHRUCACHE to ON, which causes the disk process to write records immediately to disk rather than store them in the disk cache buffer. A value of FALSE sets the WRITETHRUCACHE attribute to OFF, causing log records to be written to a disk cache buffer. Setting the attribute ZCOL-WRITETHRUCACHE to TRUE is equivalent to running EMSCCTRL with the BUFFERED option set to OFF. The default value for this attribute is FALSE.

**ZCOL-EOFREFRESH** *(Boolean field)*

selects a new value for the EOFREFRESH attribute. A value of TRUE in this field sets EOFREFRESH to ON, which tells a collector to update the end-of-file pointer on disk for each block written to the log file. This is safer than updating the pointer less often, but less efficient because it involves more writes to disk. The default value of this attribute is OFF.

**ZCOL-DISCACCESSID** *(uint field)*

specifies the user ID to be used when gaining access to primary collector log files. Unless you specify otherwise, the default ID is -1 (super ID).

This field must be left null when issuing commands to the alternate collector.
**ZCOL-PROTECTION** *(uint field)*

defines the read, write, execute, and purge security used when creating disk log files. The format of this word is:

0:3 = 0
4:6 = ID allowed for reading
7:9 = ID allowed for writing
10:12 = ID allowed for executing
13:15 = ID allowed for purging

Each field can have one of these values:

0 = Any local ID
1 = Local member of owner’s group
2 = Local owner
4 = Any ID
5 = Member of owner’s group (local or remote)
6 = Owner (local or remote)
7 = Local super ID only

**ZCOL-EVENTBLOCKING** *(Boolean field)*

if ON, tells a collector to write a number of messages to a buffer and then write the entire group of messages to the log file. If OFF, it tells the collector to write messages one at a time to the log file.

**ZCOL-BURSTSUPDETECT**

a value of TRUE (ZSPI-VAL-TRUE) specifies to the EMS collector that BDS should be enabled. A value of FALSE (ZSPI-VAL-FALSE) indicates that BDS should be disabled.

**ZCOL-PRELOGFILTER**

a value of TRUE (ZSPI-VAL-TRUE) specifies to the EMS collector that PLF should be enabled. A value of FALSE (ZSPI-VAL-FALSE) indicates that PLF should be disabled.

**ZEMS-MAP-ACOL-CONTROL**

is an extensible structured token that provides the value for the allocate control options used with the alternate collector. This token has one field:

**ZCOL-ALLOCATE** *(Boolean field)*

if TRUE, causes the alternate collector to attempt to create and allocate MAXFILE log files in the log subvolume. The range of file numbers created or
allocated is the same as that described in EMSACOLL—Alternate Collector Program on page 13-2.

**Note.** You must include the ZEMS-MAP-ACOL-CONTROL or ZEMS-MAP-COL-CONTROL token in the command buffer when you use the CONTROL command to manage the alternate collector. You can include both tokens in the command buffer if you want to both change alternate collector attributes and allocate MAXFILE log files.

ZEMS-TKN-CDISTPRICPU *(type uint; nonshared)*

is the token that contains the number of the CPU in which the compatibility distributor is to operate. This is a request for the compatibility distributor to switch CPUs—that is, to begin operating in its backup CPU.

This token and the token map ZEMS-MAP-COL-CONTROL-CDIST are mutually exclusive: do not use both in the command buffer.

ZEMS-MAP-COL-CONTROL-CDIST

is an extensible structured token that helps you give commands to the compatibility distributor to change its operating characteristics; for example, ZCOL-CDIST-PRICPU tells it to switch CPUs. In a command message, if you do not supply a particular field, the distributor leaves that operating characteristic unchanged; that is, the characteristic is left at the value last given it or at its default value.

This token map and ZEMS-TKN-CDISTPRICPU are mutually exclusive: do not use both in the command buffer.

ZCOL-CDIST-PRICPU *(uint field)*

contains the number of the CPU in which the distributor is to operate. This is a request for the distributor to switch CPUs—that is, to begin operating in its backup CPU.

ZCOL-CDIST-CONSOLE-OUT *(Boolean field)*

if TRUE, tells the distributor to ignore the ZCOL-CDIST-TEXTOUT-SET field and to use the default console, selected at SYSGEN, for text output.

ZCOL-CDIST-TEXTOUT-SET *(fname field)*

gives the destination file name of the console to which $Z0$ sends text output if the ZCOL-CDIST-CONSOLE-OUT field is FALSE. You can also specify the destination file name with the EMSCCTRL utility (keyword TEXTOUT—see EMSCCTRL—Control Collector Utility on page 13-9) or with the PUP CONSOLE command in D-series RVUs. This field allows certain device types only.

This field is ignored if the ZCOL-CDIST-CONSOLE-OUT field is TRUE.
ZCOL-CDIST-MODE-SET (enum field)
indicates that $Z0 displays only critical events if ON, or all events if OFF.

ZCOL-CDIST-USER-SET (userid field)
specifies the user ID that the compatibility distributor must use when forwarding event messages to the console from a remote collector (one on another node). You can perform the same operation using the EMSCCTRL command (or the PUP CONSOLE command on systems running the D-series).

Tokens in Response Buffer

ZSPI-TKN-RETCODE (type enum)
is the standard SPI return token. A zero value for this token signals the successful execution of the command. A nonzero value signals that the command failed. For a list of the values that can be returned in the ZSPI-TKN-RETCODE token, see Error-Handling Notes on page 19-40.

ZSPI-TKN-SERVER-VERSION (type uint)
is the subsystem version number of the EMS collector.

ZSPI-TKN-SSID (type ssid)
contains ZSPI-SSN-ZEMS, the subsystem ID for the Event Management Service.

ZSPI-TKN-COMMAND (type enum)
is the original command.

Error-Handling Notes

Table 19-7 shows the names and numbers of ZSPI-TKN-RETCODE values specific to the CONTROL command.

<table>
<thead>
<tr>
<th>Name (ZEMS-ERR-)</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV-CPU</td>
<td>1009</td>
<td>CPU number in ZCOL-PRIMARYCPU was not the backup CPU for $0 or for the alternate collector.</td>
</tr>
<tr>
<td>CPU-RANGE</td>
<td>1010</td>
<td>CPU number in ZCOL-PRIMARYCPU is invalid; it must be 0 to 15.</td>
</tr>
<tr>
<td>CDIST-CPU</td>
<td>1013</td>
<td>CPU number in ZEMS-TKN-CDISTPRICPU was not the primary or backup CPU for $Z0.</td>
</tr>
</tbody>
</table>
Collector Commands and Responses

CONTROL Command (ZEMS-CMD-CONTROL)

Table 19-7. Values of ZSPI-TKN-RETCODE Specific to CONTROL Command (page 2 of 2) (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>(ZEMS-ERR-)</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-TKN</td>
<td></td>
<td>1015</td>
<td>Collector CONTROL command needs ZEMS-TKN-CDISTPRICPU, ZEMS-MAP-COL-CONTROL, ZEMS-MAP-COL-CONTROL-CDIST, or ZEMS-MAP-ACOL-CONTROL.</td>
</tr>
<tr>
<td>LOG-ACCESS</td>
<td></td>
<td>1031</td>
<td>Command could not be completed because an I/O error occurred while the collector was accessing a log file or a ZZEVCONF file. The response error list contains ZEMS-TKN-ERROR, ZEMS-TKN-ZFILERR, and ZEMS-TKN-FAILFILENAME.</td>
</tr>
<tr>
<td>OPEN-LOG</td>
<td></td>
<td>1034</td>
<td>Subvolume specified in ZCOL-LOGSUBVOL could not be accessed.</td>
</tr>
<tr>
<td>ACC-VIOL</td>
<td></td>
<td>1048</td>
<td>Security violation occurred.</td>
</tr>
<tr>
<td>INV-MODE</td>
<td></td>
<td>1053</td>
<td>Mode for $Z0 is invalid. It must be CRITICAL-ONLY, ALL, or STOP.</td>
</tr>
<tr>
<td>CDIST-DOWN</td>
<td></td>
<td>1054</td>
<td>$Z0 is down.</td>
</tr>
<tr>
<td>NO-BACKUP</td>
<td></td>
<td>1055</td>
<td>Contents of the ZCOL-PRIMARYCPU field is the alternate collector’s backup CPU, but the alternate collector does not currently have a backup process.</td>
</tr>
<tr>
<td>ALLOC-LOG</td>
<td></td>
<td>1056</td>
<td>Log file could not be allocated. The response error list contains ZEMS-TKN-ERROR, ZEMS-TKN-ZFILERR and ZEMS-TKN-FAILFILENAME.</td>
</tr>
</tbody>
</table>
DELETE Command (ZCOM-CMD-DELETE)

The ZCOM-CMD-DELETE command deletes objects from the EMS collector. The only supported object is the EMS filter (ZCOM-OBJ-FILTER).

At least one object name token is required. Use of the wild-card object name (*) is also supported.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxx</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Optional signifies that the token is optional and need not be supplied by the user.
- Required signifies that the token is required and must be supplied by the user.
- Unconditional signifies that the token always appears in the response.
- Error only signifies that this token is present only when an error or warning is reported in the response record (ZSPI-TKN-RETCODE contains a nonzero value).
- The data list (all tokens from -DATALIST to -ENDLIST) forms a single response record. Multiple data lists (response records) can occur in a response.

For a detailed description of these command and response tokens, see Common Definitions of ZCOM-Commands on page 19-4.
GETVERSION Command (ZCOM-CMD-GETVERSION)

The ZCOM-CMD-GETVERSION command returns the EMS collector version information and optional supplemental information. Because the values of ZCOM-CMD-GETVERSION and ZEMS-CMD-GETVERSION are equal (=0), the basic form of this command is identical to ZEMS-CMD-GETVERSION. However, the newer ZCOM-CMD-GETVERSION also returns the ZCOM-TKN-GETVSN-LVL token.

The -OBJTYPE token in the command is ignored.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxx</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZCOM-TKN-GETVSN-SUPP-PARAMS</td>
<td>ZSPI-TYP-BOOLEAN</td>
<td>Optional</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-SERVER-BANNER</td>
<td>ZSPI-TYP-CHAR50</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>
Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Optional signifies that the token is optional and need not be supplied by the user.
- Required signifies that the token is required and must be supplied by the user.
- Unconditional signifies that the token always appears in the response.
- The presence of the ZCOM-TKN-GETVSN-SUPP-PARAMS command token identifies this as the supplemental form of the ZEMS-CMD-GETVERSION command. If the value of this command token is true, the supplemental token ZCOM-TKN-GETVSN-LVL is returned in the response.
- The ZCOM-TKN-GETVSN-LVL response token is returned only if the ZCOM-TKN-GETVSN-SUPP-PARAMS token is in the command, and its value is true. The value returned is ZCOM-VAL-GETVSN-ENLIT, which signifies that the collector is capable of processing enlightened (post-D00) tokens and data structures.
- The data list (all tokens from -DATALIST to -ENDLIST) forms a single response record.
- Error only signifies that this token is present only when an error or warning is reported in the response record (ZSPI-TKN-RETCODE contains a nonzero value).
- AC only signifies that the item is required for the alternate collector and is invalid for the primary collector.

For a detailed description of these command and response tokens, see Common Definitions of ZCOM-Commands on page 19-4.
GETVERSION Command (ZEMS-CMD-GETVERSION)

The ZEMS-CMD-GETVERSION command returns the version number of the programmatic interface to the primary or alternate collector.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-CMD-GETVERSION</td>
</tr>
</tbody>
</table>

Tokens in Command Buffer

None

Tokens in Response Buffer

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SERVER-BANNER</td>
<td>token-type ZSPI-TYP-CHAR50.</td>
</tr>
<tr>
<td>ZSPI-TKN-SERVER-VERSION</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>token-type ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Tokens in Response Buffer

ZSPI-TKN-SERVER-BANNER (type char50)
contains the standard banner information.

ZSPI-TKN-SERVER-VERSION (type uint)
is the subsystem version number of the EMS collector.

ZSPI-TKN-RETCODE (type enum)
is the standard SPI return token. A zero value for this token signals the successful execution of the command. A nonzero value signals that the command failed.

ZSPI-TKN-SSID (type ssid)
contains ZSPI-SSN-ZEMS, the subsystem ID for the Event Management Service.

ZSPI-TKN-COMMAND (type enum)
is the original command.

ZEMS-TKN-COLLECTOR (type enum)
contains the value ZEMS-SUBJ-ACOLL (2) when the alternate collector returns a response. This token is not returned by the primary collector.

Error-Handling Notes

No error codes are defined that are unique to this command.
INFO Command (ZCOM-CMD-INFO)

The ZCOM-CMD-INFO command requests the configuration values of the specified object. These values are initially set at collector startup (performed by SYSGEN or by the TACL RUN command) and might have been modified by ZCOM-CMD-ALTER or ZEMS-CMD-CONTROL. All the configuration attributes are returned in an extensible INFO structure (see EMS-Specific Tokens on page 19-10).

The supported objects are EMS collectors (ZCOM-OBJ-COLL) and EMS filters (ZCOM-OBJ-FILTER). At least one object name is required. The wild-card object name (*) and the ZCOM-TKN-SUB command modifier token are also supported.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxxx</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZCOM-TKN-SUB</td>
<td>ZSPI-TYP-ENUM</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-INFO-ENQ</td>
<td>ZSPI-TYP-ENUM</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-INFO-DFLT</td>
<td>ZSPI-TYP-BOOLEAN</td>
<td>Optional</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Command Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Optional signifies that the token is optional and need not be supplied by the user.
- Required signifies that the token is required and must be supplied by the user.
- The optional ZCOM-TKN-INFO-DFLT token is a Boolean that specifies whether default or current configuration values should be returned in the response. A value of TRUE requests the default values; a value of FALSE requests the current values.
- The optional ZCOM-TKN-ENQ token specifies the token number of the token desired. There may be multiple occurrences of this token. Each occurrence causes a separate response record to be returned. If the specified token is not related to the specified object, a ZCOM-ERR-CMD-MISMATCH error is returned.
INFO Command (ZCOM-CMD/INFO)

For a detailed description of these command tokens, refer to Common Definitions of ZCOM-Commands on page 19-4.

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-xxx-xxxx</td>
<td></td>
<td>Conditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-ENQ</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Response Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Unconditional signifies that the token always appears in the response.
- Conditional signifies that the token does not always appear in the command response; the condition under which it appears is command dependent.
- Error only signifies that this token is present only when an error or warning is reported in the response record.
- The ZEMS- tokens and token maps that are returned depend on the specified object and whether any ZCOM-TKN-ENQ tokens were present in the command. Table 19-8 identifies the response tokens returned with ZCOM-CMD/INFO, depending on the object type specified (-COLL or -FILTER) and the presence or absence of the ZCOM-TKN-ENQ token. If this token is present in the command, the response token is determined by the value of the ZEMS-TKN-ENQ token. Under the table’s Tokens Returned column, AC only signifies the alternate collector only; PC only signifies the primary collector only.

For a detailed description of these response tokens, see Common Definitions of ZCOM-Commands on page 19-4.
LISTOBJECTS Command (ZCOM-CMD-LISTOBJECTS)

The ZCOM-CMD-LISTOBJECTS command returns the name and type of all or a subset of the objects known by the EMS collector. Two forms of the LISTOBJECTS command are supported. The first form requests all object types and names; the second form requests only the object names of a specified object type. Object types supported are ZCOM-OBJ-NULL, ZCOM-OBJ-COLL, and ZCOM-OBJ-FILTER.

<table>
<thead>
<tr>
<th>Object Type (ZCOM-OBJ)</th>
<th>ZCOM-TKN-ENQ</th>
<th>Tokens Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>-COLL</td>
<td>Missing</td>
<td>ZEMS-MAP-COL-INFO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZEMS-MAP-ACOL-INFO (AC only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZEMS-MAP-COL-CDIST-INFO (PC only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZEMS-MAP-BDS-INFO</td>
</tr>
<tr>
<td>-COLL</td>
<td>(&lt;z&gt;-COL-INFO)</td>
<td>ZEMS-MAP-COL-INFO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZEMS-MAP-ACOL-INFO (AC only)</td>
</tr>
<tr>
<td>-COLL</td>
<td>(&lt;z&gt;-ACOL-INFO)</td>
<td>ZEMS-MAP-COL-INFO (AC only)</td>
</tr>
<tr>
<td>-COLL</td>
<td>(&lt;z&gt;-COL-CDIST-INFO)</td>
<td>ZEMS-MAP-COL-CDIST-INFO (PC only)</td>
</tr>
<tr>
<td>-COLL</td>
<td>(&lt;z&gt;-BDS-INFO)</td>
<td>ZEMS-MAP-BDS-INFO</td>
</tr>
<tr>
<td>-FILTER</td>
<td>Missing</td>
<td>ZEMS-MAP-STATUS-FILTER</td>
</tr>
<tr>
<td>-FILTER</td>
<td>(&lt;z&gt;-STATUS-FILTER)</td>
<td>ZEMS-MAP-STATUS-FILTER</td>
</tr>
</tbody>
</table>

Command Tokens

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxx</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Command Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Optional signifies that the token is optional and need not be supplied by the user.
- If the object type is ZCOM-OBJ-NULL, all object types and names are requested.
- If the object type is ZCOM-OBJ-COLL, the name of the collector is requested.
- If the object type is ZCOM-OBJ-FILTER, the name of all filters in use by the collector are requested.
Collector Commands and Responses

LISTOBJECTS Command (ZCOM-CMD-LISTOBJECTS)

For a detailed description of these command tokens, see Common Definitions of ZCOM- Commands on page 19-4.

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Response Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Unconditional signifies that the token always appears in the response.
- Error only signifies that this token is present only when an error or warning is reported in the response record.
- If all object types and names are requested (the first form of ZCOM-CMD-LISTOBJECTS), a response record is built for each object type. The EMS collector returns only two types of response records: -COLL and -FILTER. Each response record contains an -OBJNAME token for each object of the type -OBJTYPE. There can be zero, one, or many occurrences of the -OBJNAME token.

For a detailed description of these response tokens, see Common Definitions of ZCOM- Commands on page 19-4.
REPLACE Command (ZEMS-CMD-REPLACE)

The ZEMS-CMD-REPLACE command replaces a configured object in the EMS collector with another object. The only supported object is an EMS filter (ZCOM-OBJ-FILTER). The filter can be a compiled filter, a filter table, or a burst filter. Only one filter at a time can be replaced in this command.

If required parameter tokens are specified in the new filter, these parameter tokens must be included in the command. If optional parameter tokens are specified in the filter, the parameter tokens are optional.

Although not extended SPI compliant, ZEMS-CMD-REPLACE is similar to the ZCOM-ADD-FILTER command.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxxx</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZEMS-TKN-NEW-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>&lt;filter parameter token(s)&gt;</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Filter</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-STRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Token Usage Considerations

- **Basic SPI** signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- **Optional** signifies that the token is optional and need not be supplied by the user.
- **Required** signifies that the token is required and must be supplied by the user.
- **Filter** signifies that the required filter parameter tokens are required only if they are defined as required parameter tokens in the filter.
Collector Commands and Responses

STATUS Command (ZCOM-CMD-STATUS)

The ZCOM-CMD-STATUS command requests both configuration and status information from the EMS collector. The supported object types are EMS collectors (ZCOM-OBJ-COLL) and EMS filters (ZCOM-OBJ-FILTER). The ZEMS-CMD-STATUS command does not support object types.

At least one object name token is required. The wild-card object name (*) and the ZCOM-TKN-SUB command modifier are supported.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxx</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZCOM-TKN-SUB</td>
<td>ZSPI-TYP-ENUM</td>
<td>Optional</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

Command Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Optional signifies that the token is optional and need not be supplied by the user.
- Required signifies that the token is required and must be supplied by the user.

Unconditional signifies that the token always appears in the response.

Error only signifies that this token is present only when an error or warning is reported in the response record.

If the REPLACE command was successful, the value in the ZCOM-TKN-OBJNAME token is the name of the new filter file.

If the REPLACE command was not successful, the value of the first ZCOM-TKN-OBJNAME contains the name of the old filter file (the filter that was supposed to be replaced). The ZCOM-TKN-OBJNAME token in the error list contains the name of the new filter table.

For a detailed description of these command and response tokens, see Common Definitions of ZCOM-Commands on page 19-4.
For a detailed description of these command tokens, see "Common Definitions of ZCOM- Commands" on page 19-4.

<table>
<thead>
<tr>
<th>Response Tokens When Object Type is ZCOM-OBJ-COLL</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJSTATE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-COL-STATUS</td>
<td>ZEMS-DDL-COL-STATUS</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-ACOL-STATUS</td>
<td>ZEMS-DDL-ACOL-STATUS</td>
<td>AC only</td>
</tr>
<tr>
<td>ZEMS-MAP-COL-CDIST-STATUS</td>
<td>ZEMS-DDL-COL-CDIST-STATUS</td>
<td>PC only</td>
</tr>
<tr>
<td>ZEMS-MAP-BDS-INFO</td>
<td>ZEMS-DDL-BDS-INFO</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-BDS-STATS</td>
<td>ZEMS-DDL-BDS-STATS</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-PLF-STATS</td>
<td>ZEMS-DDL-PLF-STATS</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-BURST-STATUS</td>
<td>ZEMS-DDL-BURST-STATUS</td>
<td>Conditional-1</td>
</tr>
<tr>
<td>ZEMS-MAP-BURST-SUBJ-VALUE</td>
<td>ZSPI-TYP-STRING</td>
<td>Conditional-1</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Tokens When Object Type is ZCOM-OBJ-FILTER</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJSTATE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-COL-STATUS</td>
<td>ZEMS-DDL-COL-STATUS</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-COL-CDIST-STATUS</td>
<td>ZEMS-DDL-COL-CDIST-STATUS</td>
<td>AC only</td>
</tr>
<tr>
<td>ZEMS-MAP-ACOL-STATUS</td>
<td>ZEMS-DDL-ACOL-STATUS</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-ACOL-CDIST-STATUS</td>
<td>ZEMS-DDL-ACOL-CDIST-STATUS</td>
<td>PC only</td>
</tr>
<tr>
<td>ZEMS-MAP-BDS-INFO</td>
<td>ZEMS-DDL-BDS-INFO</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-BDS-STATS</td>
<td>ZEMS-DDL-BDS-STATS</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-PLF-STATS</td>
<td>ZEMS-DDL-PLF-STATS</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZEMS-MAP-BURST-STATUS</td>
<td>ZEMS-DDL-BURST-STATUS</td>
<td>Conditional-1</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-STRING</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
</tbody>
</table>

| ZSPI-TKN-SSCTL                                    | ZSPI-TYP-STRING  | Basic SPI      |
| ZSPI-TKN-SSCTL                                    | ZSPI-TYP-STRING  | Basic SPI      |
Response Token Usage Considerations

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Unconditional signifies that the token always appears in the response.
- AC only signifies that the item is returned only by the alternate collector.
- PC only signifies that the item is returned only by the primary collector.
- Conditional 1 signifies that one of these tokens is present for every burst table entry that is in the BURST state.
- Error only signifies that this token is present only when an error or warning is reported in the response record.
- If all object types and names are requested (the first form of ZCOM-CMD-LISTOBJECTS), a response record is built for each object type. The EMS collector returns only two types of response records: -COLL and -FILTER. Each response record contains an -OBJNAME token for each object of the type -OBJTYPE. There can be zero, one, or many occurrences of the -OBJNAME token.

For a detailed description of these response tokens, see Common Definitions of ZCOM- Commands on page 19-4.

STATUS Command (ZEMS-CMD-STATUS)

The STATUS command returns event-message statistics and current information about collector attributes to a distributor or other application programs. Distributors use this command to synchronize access to the log file. Applications programs (including EMSCINFO) use it to retrieve display information for primary and alternate collector status displays. Three fields in the ZEMS-MAP-COL-STATUS token map indicate whether BDS and PLF are enabled or disabled.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-CMD-STATUS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tokens in Command Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
</tbody>
</table>
Collector Commands and Responses

STATUS Command (ZEMS-CMD-STATUS)

Tokens in Response Buffer

ZEMS-MAP-COL-STATUS
DEF ZEMS-DDL-COL-STATUS.
  02 ZCOL-PRIMARYCPU type ZSPI-DDL-UINT.
  02 ZCOL-BACKUPCPU type ZSPI-DDL-UINT.
  02 ZCOL-PRIORITY type ZSPI-DDL-UINT.
  02 ZCOL-LOGDISCERROR type ZSPI-DDL-UINT.
  02 ZCOL-CURRENTFILENAME type ZSPI-DDL-FNAME.
  02 ZCOL-CURRENTRECORD type ZSPI-DDL-INT2.
  02 ZCOL-DEFAULTFILENAME type ZSPI-DDL-FNAME.
  02 ZCOL-PRIMARYEXTENT type ZSPI-DDL-UINT.
  02 ZCOL-SECONDARYEXTENT type ZSPI-DDL-UINT.
  02 ZCOL-ROTATEFILES type ZSPI-DDL-BOOLEAN.
  02 ZCOL-MAXFILENNNN type ZSPI-DDL-UINT.
  02 ZCOL-EOFREFRESH type ZSPI-DDL-BOOLEAN.
  02 ZCOL-WRITETHRUCACHE type ZSPI-DDL-BOOLEAN.
  02 ZCOL-PROTECTION type ZSPI-DDL-UINT.
  02 ZCOL-EVENTSRECEIVED type ZSPI-DDL-INT2.
  02 ZCOL-EVENTSLOGGED type ZSPI-DDL-INT2.
  02 ZCOL-EVENTSDISCARDED type ZSPI-DDL-INT2.
  02 ZCOL-OPENSRECEIVED type ZSPI-DDL-UINT.
  02 ZCOL-CLOSESRECEIVED type ZSPI-DDL-UINT.
  02 ZCOL-FILESWITCHES type ZSPI-DDL-UINT.
  02 ZCOL-DISCERRORS type ZSPI-DDL-UINT.
  02 ZCOL-INVALIDEVENTS type ZSPI-DDL-UINT.
  02 ZCOL-BUFFERFAILURES type ZSPI-DDL-UINT.
  02 ZCOL-EVENTBLOCSIZE type ZSPI-DDL-UINT.
  02 ZCOL-CDIST-ERROR type ZSPI-DDL-UINT.
  02 ZCOL-OPRLOG-ERROR type ZSPI-DDL-UINT.
  02 ZCOL-CDIST-MODE type ZSPI-DDL-ENUM.
  02 ZCOL-CDIST-TEXTOUT type ZSPI-DDL-FNAME.
  02 ZCOL-CDIST-USER type ZSPI-DDL-USERID.
  02 ZCOL-CDIST-DEF-TEXTOUT type ZSPI-DDL-FNAME.
END

ZEMS-MAP-ACOL-STATUS
DEF ZEMS-DDL-ACOL-STATUS.
  02 ZCOL-POOLPAGES type ZSPI-DDL-UINT.
  02 ZCOL-REPLYAFTERWRITE type ZSPI-DDL-BOOLEAN.
END

ZEMS-MAP-COL-CDIST-STATUS
DEF ZEMS-DDL-COL-CDIST-STATUS.
  02 ZCOL-CDISTPRICPU type ZSPI-DDL-UINT.
  02 ZCOL-CDISTBKUPCPU type ZSPI-DDL-UINT.
  02 ZCOL-STOPCOMPATDIST type ZSPI-DDL-BOOLEAN.
  02 ZCOL-DISTR-ERROR type ZSPI-DDL-UINT.
  02 ZCOL-OGRPLOG-ERROR type ZSPI-DDL-UINT.
  02 ZCOL-CDIST-MODE type ZSPI-DDL-ENUM.
  02 ZCOL-CDIST-TEXTOUT type ZSPI-DDL-FNAME.
  02 ZCOL-CDIST-USER type ZSPI-DDL-USERID.
  02 ZCOL-CDIST-DEF-TEXTOUT type ZSPI-DDL-FNAME.
END

ZSPI-TKN-RETCODE token-type ZSPI-TYP-ENUM.
ZSPI-TKN-SERVER-VERSION token-type ZSPI-TYP-UINT.
ZSPI-TKN-SSID token-type ZSPI-TYP-SSID.
ZSPI-TKN-COMMAND token-type ZSPI-TYP-ENUM.
Tokens in Response Buffer

ZEMS-MAP-COL-STATUS

is an extensible structured token that gives you the current values of a collector’s operational attributes and some event-message statistics.

For a more detailed description of the collector attribute fields, see CONTROL Command (ZEMS-CMD-CONTROL) on page 19-34.

ZCOL-PRIMARYCPU (uint field)

is the primary CPU number of the collector responding to the STATUS command.

ZCOL-BACKUPCPU (uint field)

is the backup CPU number of the collector responding to the STATUS command.

ZCOL-PRIORITY (uint field)

is the current execution priority of the collector responding to the STATUS command.

ZCOL-LOGDISCERROR (uint field)

is the status of the last disk operation on the log file of the collector responding to the STATUS command. A value of zero indicates a successful operation. A nonzero value is the file management number that caused logging to stop.

ZCOL-CURRENTFILENAME (fname field)

is the file name of the current log file for the collector responding to the STATUS command.

ZCOL-CURRENTRECORD (int2 field)

is the record number (as received from the FILEINFO procedure) of the last record written to the log file of the collector responding to the STATUS command. If the collector is currently switching log files, ZCOL-CURRENTRECORD is -1.

ZCOL-DEFAULTFILENAME (fname field)

is the file name of the last file in use in the default subvolume; this field can be the same as ZCOL-CURRENTFILENAME.

ZCOL-PRIMARYEXTENT (uint field)

is the current setting of the PRIMARYEXTENT attribute.
ZCOL-SECONDARYEXTENT (uint field)
is the current setting of the SECONDARYEXTENT attribute.

ZCOL-ROTATEFILES (Boolean field)
is the current setting of the ROTATEFILES attribute.

ZCOL-MAXFILENNNN (uint field)
is the (current) maximum number of log files that the responding collector can create in this log subvolume.

ZCOL-EOFREFRESH (Boolean field)
is a flag that tells whether frequent updating of the end-of-file position—each time the disk process writes a block to the log file—has been selected.

ZCOL-WRITETHRUCACHE (Boolean field)
is a flag that tells whether log-file buffering has been selected. TRUE indicates that buffering is off.

ZCOL-PROTECTION (uint field)
is the set of file-security values for new log files.

ZCOL-EVENTSRECEIVED (int2 field)
is the number of event notifications received by an alternate collector or by a primary collector since the last system load. These notifications are not all event messages.

ZCOL-EVENTSLOGGED (int2 field)
is the total number of event messages an alternate collector received or that a primary collector logged to disk log files since the last system load.

ZCOL-EVENTSDISCARDED (int2 field)
is the number of event messages that a collector had to discard because of event message flooding, or because BDS or PLF was enabled.

ZCOL-OPENSRECEIVED (uint field)
is the total number of #ZSPI subdevice OPENs accepted by a collector (since the last system load for a primary collector).

ZCOL-CLOSESRECEIVED (uint field)
is the total number of #ZSPI CLOSEs received by a collector (since the last system load for a primary collector).
**ZCOL-FILESWITCHES** (uint field)
is the total number of times a collector has switched log files. This total includes both requested switches and switches triggered by log files becoming full.

**ZCOL-DISCERRORS** (uint field)
is the total number of disk errors returned to a collector by the disk process.

**ZCOL-INVALIDEVENTS** (uint field)
is the total number of event messages a collector discarded because of event-message format errors.

**ZCOL-BUFFERFAILURES** (uint field)
is the total number of event messages a collector discarded because of insufficient memory-pool capacity.

**ZCOL-EVENTBLOCKING** (Boolean field)
the EMS collector returns a value of TRUE if blocking is ON and a value of FALSE if blocking is OFF.

**ZCOL-BURSTSUPDETECT** (Boolean field)
the EMS collector returns a value of TRUE (ZSPI-VAL-TRUE) if BDS is enabled and a value of FALSE (ZSPI-VAL-FALSE) if BDS is disabled.

**ZCOL-PRELOGFILTER** (Boolean field)
the EMS collector returns a value of TRUE (ZSPI-VAL-TRUE) if BDS is enabled and a value of FALSE (ZSPI-VAL-FALSE) if BDS is disabled.

**ZCOL-CURRENTBURSTS** (uint field)
is the number of the currently detected event bursts stored in this field.

**ZEMS-MAP-ACOL-STATUS**
is a structured token used to return values from the alternate collector.

**ZCOL-POOLPAGES** (uint field)
returns the number of pages in the event-buffer pool. This field contains the setting of the POOLPAGES attribute.

**ZCOL-REPLYAFTERWRITE** (Boolean field)
contains the current setting of the REPLYAFTERWRITE attribute.
ZEMS-MAP-COL-CDIST-STATUS

is an extensible structured token that gives you the current values of the
operational environment of $Z0 and some event-message statistics.

ZCOL-CDISTPRICPU (uint field)

is the number of the CPU in which the primary process of the compatibility
distributor is running.

ZCOL-CDISTBKUPCPU (uint field)

is the number of the CPU in which the backup process of the compatibility
distributor is running.

ZCOL-STOPCOMPATDIST (Boolean field)

is the current run state of the compatibility distributor. This field is -1 if the
compatibility distributor is stopped.

ZCOL-DISTR-ERROR (uint field)

is the current error condition of the compatibility distributor console device. This
field is 0 if no error exists.

ZCOL-OPRLOG-ERROR (uint field)

is not used starting in D20 systems.

ZCOL-CDIST-MODE (enum field)

indicates that $Z0 displays only critical events if ON, or all events if OFF.

ZCOL-CDIST-TEXTOUT (fname field)

is the destination file name for CONSOLE—the console device that the
compatibility distributor uses for text output.

ZCOL-CDIST-USER (userid field)

is the user ID that the compatibility distributor uses when forwarding event
messages to the console from a remote collector (one on another node).

ZCOL-CDIST-DEF-TEXTOUT (fname field)

is the name of the default operator console selected during system load.

ZSPI-TKN-RETCODE (type enum)

is the standard SPI return token. A zero value for this token signals the successful
execution of the command. A nonzero value signals that the command failed. For a
list of the values that can be returned in the ZSPI-TKN-RETCODE token, see
Error-Handling Notes on page 19-61.
**ZSPI-TKN-SERVER-VERSION** (type uint)

This is the subsystem version number of the EMS collector.

**ZSPI-TKN-SSID** (type ssid)

Contains **ZSPI-SSN-ZEMS**, the subsystem ID for the Event Management Service.

**ZSPI-TKN-COMMAND** (type enum)

This is the original command.

---

**STOP Command (ZCOM-CMD-STOP)**

The command ZCOM-CMD-STOP is the extended SPI-compliant counterpart to the SPI-basic compliant ZEMS-CMD-STOP and is its functional equivalent. Like ZEMS-CMD-STOP, the ZCOM-CMD-STOP command can only be used to stop the alternate collector. Unlike ZEMS-CMD-STOP, the command ZCOM-CMD-STOP supports use of the collector (ZCOM-OBJ-COLL) as its object type.

<table>
<thead>
<tr>
<th>Command Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-MAXRESP</td>
<td>ZSPI-TYP-INT</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-REQID-xxxxx</td>
<td>ZSPI-TYP-STRING</td>
<td>Optional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Required</td>
</tr>
<tr>
<td>ZCOM-TKN-CMD-POWER</td>
<td>ZSPI-TYP-ENUM-CMD-POWER</td>
<td>Optional</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

---

**Command Token Usage Considerations**

- **Basic SPI** signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- **Optional** signifies that the token is optional and need not be supplied by the user.
- **Required** signifies that the token is required and must be supplied by the user.
- If **ZCOM-TKN-CMD-POWER** is present, only the value **ZCOM-VAL-CMD-POWER.ORDERLY** is supported.
For a detailed description of these command tokens, see **Common Definitions of ZCOM- Commands** on page 19-4.

<table>
<thead>
<tr>
<th>Response Tokens</th>
<th>Token Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-DATALIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZCOM-TKN-XMGR</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-RETCODE</td>
<td>ZSPI-TYP-INIT</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Unconditional</td>
</tr>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>ZSPI-TYP-ENUM</td>
<td>Error only</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>ZSPI-TYP-STRING</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Error only</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
<td>Basic SPI</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>ZSPI-TYP-BYTESTRING</td>
<td>Basic SPI</td>
</tr>
</tbody>
</table>

**Response Token Usage Considerations**

- Basic SPI signifies that the presence of this token in the command is dictated by the basic SPI command language standards that govern multiple objects and responses.
- Unconditional signifies that the token always appears in the response.
- Error only signifies that this token is present only when an error or warning is reported in the response record.

For a detailed description of these response tokens, see **Common Definitions of ZCOM- Commands** on page 19-4.

**STOP Command (ZEMS-CMD-STOP)**

The ZEMS-CMD-STOP command is used to perform an orderly shutdown of the alternate collector.

When the STOP command is accepted, the alternate collector denies further OPEN commands by returning FEOPENSTOP (61). The alternate collector process stops when all OPEN commands of the process are closed or when two minutes pass without the collector receiving an event request, whichever occurs first.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-CMD-STOP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tokens in Command Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
</tbody>
</table>
Tokens in Response Buffer

<table>
<thead>
<tr>
<th>Token Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SERVER-VERSION</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZSPI-TKN-RETCODE</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-SERVER-VERSION** (type uint)

is the subsystem version number of the EMS collector.

**ZSPI-TKN-COMMAND** (type enum)

is the original command.

**ZSPI-TKN-SSID** (type ssid)

contains ZSPI-SSN-ZEMS, the subsystem ID for the Event Management Service.

**ZSPI-TKN-RETCODE** (type enum)

is the standard SPI return token. A zero value for this token signals the successful execution of the command. A nonzero value signals that the command failed. For a list of the values that can be returned in ZSPI-TKN-RETCODE, see Error-Handling Notes on page 19-61.

**Error-Handling Notes**

The STOP command returns two error codes in ZSPI-TKN-RETCODE. One of these error codes, ZEMS-ERR-LOGGING-STOPPED, is unique to the STOP command. Table 19-9 shows the names and numbers of ZSPI-TKN-RETCODE values specific to the STOP command.

**Table 19-9. Values of ZSPI-TKN-RETCODE Specific to STOP Command**

<table>
<thead>
<tr>
<th>Name (ZEMS-ERR-)</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC-VIOL</td>
<td>1048</td>
<td>Requester's user ID is not allowed to use the STOP command. ZSPI-TKN-ERROR is returned in the response error list.</td>
</tr>
<tr>
<td>LOGGING-STOPPED</td>
<td>1057</td>
<td>Alternate collector is currently holding events in the event-buffer pool and cannot write them to disk because logging has stopped.</td>
</tr>
</tbody>
</table>
STOP Command (ZEMS-CMD-STOP)
Primary and alternate collectors receive and log event messages sent to them by subsystems. Collectors also generate event messages for their own log files. In general, these event messages report events occurring in the collector environment.

The subsystem ID (ZSPI-TKN-SSID) for collector-generated messages is EMS (Event Management Service). Each message has an event number that uniquely identifies it within the group of EMS event messages.

This section describes event messages generated by EMS primary and alternate collectors:

### Types of Collector-Generated Messages

There are four types of collector-generated messages: tokenized operator-console messages, tokenized text messages, primary collector event messages, and alternate collector event messages.

#### Tokenized Operator-Console Messages

Numbered operator-console messages are written to the primary collector by privileged routines. These privileged routines are internal to HP and are not in this manual. The primary collector converts these messages to a tokenized form and gives them an event number that is the same as the console-message number. The collector then writes the messages to the primary collector log file.

#### Tokenized Text Messages

Subsystems can write text messages to a primary or alternate collector using the WRITE procedure. The collector converts these messages to tokenized form, gives them the event number 512, and writes them to that collector’s log file.

#### Primary Collector Event Messages

The primary collector generates event messages to describe occurrences in its environment. These messages are then written to the primary collector log file. They inform you of these kinds of events:
The start of event message logging after a system load
A log file switch because the current log file was full

Alternate Collector Event Messages

The alternate collector generates event messages to describe occurrences in the alternate collector environment. The collector writes all these messages to the primary collector log file. It also writes most of the messages to the alternate collector log file. These messages inform you of these kinds of events:

- An alternate collector shutdown after a STOP command
- The successful creation of a new collector process in the backup CPU

Summary of Collector-Generated Messages

In these tables listing collector events, the column labeled Critical identifies each message as being critical (Y) or not critical (N). The EMPHASIS token in the message header is set to TRUE if the event is critical, and to FALSE if the message is not critical.

Messages 1 through 512 are shown with BEL? in the Critical column. The presence or absence of a BEL character in the text of the message determines whether or not the message is critical. A numbered console message or a text message sent with a WRITE command can contain a BEL character in the text. If the BEL character is present, the collector that generates the tokenized message sets the EMPHASIS token to TRUE. If the BEL character is not present, the EMPHASIS token retains its default value of FALSE.

For descriptions of the content of the operator console messages, see the Operator Messages Manual.

<table>
<thead>
<tr>
<th>Event Number</th>
<th>Symbolic Name (ZEMS-EVT-)</th>
<th>Description</th>
<th>Critical</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-511</td>
<td>(Operator Messages)</td>
<td>Numbered operator-console messages</td>
<td>BEL?</td>
</tr>
<tr>
<td>512</td>
<td>WRITE-TO-0</td>
<td>Events reported by WRITEs of text to $0</td>
<td>BEL?</td>
</tr>
<tr>
<td>513</td>
<td>COLD-LOAD</td>
<td>System loaded</td>
<td>Y</td>
</tr>
<tr>
<td>514</td>
<td>FILESUCHITCH</td>
<td>Next log file used</td>
<td>Y</td>
</tr>
<tr>
<td>515</td>
<td>COLL-DISC-FAILED</td>
<td>Log file inaccessible</td>
<td>Y</td>
</tr>
<tr>
<td>517</td>
<td>COMPAT-DISTR-STOPPED</td>
<td>$Z0 inaccessible</td>
<td>Y</td>
</tr>
<tr>
<td>518</td>
<td>COL-EVENT-DISCARDS</td>
<td>Flood recovery</td>
<td>Y</td>
</tr>
<tr>
<td>519</td>
<td>MSGR-EVENTS-DISCARDED</td>
<td>Special flood recovery</td>
<td>Y</td>
</tr>
</tbody>
</table>
### Table 20-1. Primary Collector Event Messages (page 2 of 2) (continued)

<table>
<thead>
<tr>
<th>Event Number</th>
<th>Symbolic Name (ZEMS-EVT-)</th>
<th>Description</th>
<th>Critical</th>
</tr>
</thead>
<tbody>
<tr>
<td>520</td>
<td>FILE-ROTATE-PURGE</td>
<td>Automatic file purge</td>
<td>Y</td>
</tr>
<tr>
<td>521</td>
<td>LOGGING-STOPPED</td>
<td>Logging has stopped</td>
<td>Y</td>
</tr>
<tr>
<td>524</td>
<td>LOGTIME-DECREASE</td>
<td>Log time decreased</td>
<td>Y</td>
</tr>
<tr>
<td>525</td>
<td>INVALIDEVENT</td>
<td>Invalid event message</td>
<td>N</td>
</tr>
<tr>
<td>538</td>
<td>BURST-START</td>
<td>An event burst, as defined by the BDS configuration parameters, was detected.</td>
<td>Y</td>
</tr>
<tr>
<td>539</td>
<td>BURST-END</td>
<td>An event burst, as defined by the BDS configuration parameters, ended.</td>
<td>N</td>
</tr>
<tr>
<td>540</td>
<td>PLF-ERROR</td>
<td>A filter evaluation error occurred when a filter was applied to an event.</td>
<td>Y</td>
</tr>
</tbody>
</table>

### Table 20-2. Alternate Collector Event Messages (page 1 of 2)

<table>
<thead>
<tr>
<th>Event Number</th>
<th>Symbolic Name (ZEMS-EVT-)</th>
<th>Description</th>
<th>Critical</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>WRITE-TO-0</td>
<td>Events reported by WRITEs of text to the alternate collector</td>
<td>BEL?</td>
</tr>
<tr>
<td>514</td>
<td>FILESWITCH</td>
<td>Next log file used</td>
<td>Y</td>
</tr>
<tr>
<td>520</td>
<td>FILE-ROTATE-PURGE</td>
<td>Automatic file purge</td>
<td>Y</td>
</tr>
<tr>
<td>521</td>
<td>LOGGING-STOPPED</td>
<td>Logging has stopped</td>
<td>Y</td>
</tr>
<tr>
<td>522</td>
<td>COLLECTOR-RUN</td>
<td>First event after RUN</td>
<td>N</td>
</tr>
<tr>
<td>523</td>
<td>ACOL-EVENT-DISCARDS</td>
<td>Number of events discarded after flooding</td>
<td>Y</td>
</tr>
<tr>
<td>524</td>
<td>LOGTIME-DECREASE</td>
<td>Log time decreased</td>
<td>Y</td>
</tr>
<tr>
<td>525</td>
<td>INVALIDEVENT</td>
<td>Invalid event message</td>
<td>N</td>
</tr>
<tr>
<td>526</td>
<td>ACOL-INTERNAL-ERR</td>
<td>Inconsistency in alternate collector operation</td>
<td>Y</td>
</tr>
<tr>
<td>527</td>
<td>ACOL-SHUTDOWN</td>
<td>Collector shutdown</td>
<td>N</td>
</tr>
<tr>
<td>528</td>
<td>ACOL-ALLOCATESEG-ERR</td>
<td>Cannot allocate extended data segment</td>
<td>Y</td>
</tr>
<tr>
<td>529</td>
<td>ACOL-CHECKOPEN-FAILED</td>
<td>Unable to open file</td>
<td>Y</td>
</tr>
<tr>
<td>530</td>
<td>ACOL-TAKEOVER</td>
<td>Takeover by collector backup process</td>
<td>Y</td>
</tr>
<tr>
<td>531</td>
<td>ACOL-CREATEBACKUP-ERR</td>
<td>Cannot create backup</td>
<td>Y</td>
</tr>
<tr>
<td>532</td>
<td>ACOL-BACKUP-CREATED</td>
<td>Backup process created</td>
<td>N</td>
</tr>
</tbody>
</table>
Tokenized Operator Console Messages (1–511)

These event messages represent the numbered operator console messages generated by the primary collector. For descriptions of individual messages, see the Operator Messages Manual.

Each event message corresponds to one numbered console message; the event message number is the same as the number of the console message.
Because these event messages all have the same format—differing only in the ZEMS-TKN-EVENTNUMBER and ZEMS-TKN-EMPHASIS flags—after this syntax box is a description of the general format rather than the individual event message.

### Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-OPMSG</td>
<td>ZEMS-DDL-OPMSG.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-XSYSPID</td>
<td>ZEMS-DDL-XSYSPID.</td>
</tr>
</tbody>
</table>

### Conditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-MAP-EXIOADDR</td>
<td>ZSPI-TYP-EXIOADDR.</td>
</tr>
</tbody>
</table>

### Event-Message Text

Depends on opmsg; see Event Notes on page 20-6.

### Unconditional Tokens

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

**ZEMS-TKN-EVENTNUMBER (shared)**

is the number of a numbered console message.

**ZEMS-TKN-EMPHASIS (shared)**

is a standard EMS token. For more information, see Section 14, EMS Definitions. If the value is TRUE, the original message and the display text contain the BEL character.

**ZEMS-TKN-CONSOLE-PRINT (shared)**

is a standard EMS token. For more information about this token, see Section 14, EMS Definitions. For numbered console messages, the value of this token is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, -USERID, and -OPMSG

are all standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. Event messages 1 through 511 each have one subject token: ZEMS-TKN-XSYSPID.

ZEMS-TKN-XSYSPID

is private to HP. It contains the system number and PID of some process associated with the event message.

Conditional Tokens

ZEMS-MAP-EXIOADDR

(D-series RVUs only) is an extensible structured token, private to HP. It is found in some messages that include the ZEMS-TKN-OPMSG token. When present, the ZEMS-MAP-EXIOADDR token contains the extended physical address of the I/O device associated with the subject of the event message.

Cause. For the cause of each of these event messages, see the Operator Messages Manual.

Effect. For the effect of each of these event messages, see the Operator Messages Manual.

Recovery. The action recommended depends on the particular event message.

Event Notes

- In its original form, a nontokenized operator message that is critical contains a BEL character to draw the operator’s attention. In its tokenized form, the message contains an EMPHASIS token set to TRUE (see Section 14, EMS Definitions), and the formatted display text contains a BEL character.

- The ZEMS-TKN-EVENTNUMBER token for these event messages is in the range 1 through 511.
Tokenized Text Messages (512)

These tokenized text messages are generated by the primary and alternate collectors.

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-XSYSPID</td>
<td>ZEMS-DDL-XSYSPID.</td>
</tr>
<tr>
<td>ZEMS-TKN-TEXT</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
</tbody>
</table>

Event-Message Text
Depends on ZEMS-TKN-TEXT.

Unconditional Tokens

ZSPI-TKN-SSID
is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-WRITE-TO-0 (512).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. If the value is TRUE, the original message and the display text contain the BEL character.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information about this token, see Section 14, EMS Definitions. For tokenized text messages, the value of this token is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-XSYSPI

is private to HP. It contains the system number and PID of some process associated with the event message.

ZEMS-TKN-TEXT (nonshared)
is the text that was sent to the collector through the call to the WRITE procedure. If the original text is more than 102 bytes, it is truncated to 102 bytes.

Cause. The cause of the event varies with the contents of the text written to the collector. The primary and alternate collectors generate an event message (ZEMS-EVT-WRITE-TO-0) when any process calls the WRITE procedure, rather than the WRITEREAD procedure, to write a text message to the collector. Each text message written to a collector becomes an ZEMS-EVT-WRITE-TO-0 event message. In this way, text messages that come from programs written before EMS are available in event-message form. Before EMS, the WRITE procedure was the usual mechanism for reporting events.

Effect. The effect depends on the reported text.

Recovery. The action recommended depends on the reported text.

Programming Notes

- HP recommends that you not write text messages to the primary or alternate collector in new programs unless you are doing it as part of a transition to the EMS system. This method of reporting events is currently supported only for compatibility with older programs and is increasingly obsolete.

- In its original form, a critical nontokenized operator message contains a BEL character to attract the operator’s attention. In its tokenized form, the message contains an EMPHASIS token set to TRUE (see Section 14, EMS Definitions), and the formatted display text contains a BEL character.

- When a process writes text to a collector using the WRITE procedure, the collector is the generator of the resulting event message. In this case, the EMS event message header tokens used to identify the reporting process describe the collector as the reporting process. The header tokens defining the reporting process are ZEMS-TKN-SYSTEM, ZEMS-TKN-CPU, ZEMS-TKN-PIN, ZEMS-TKN-PROC-DESC, and ZEMS-TKN-USERID. You cannot use these header tokens to identify the process sending the text to the collector.

- When obtaining the name of the reporting process, for the best results, pass the ZEMS-TKN-SENDERID token to the EMSGET (or EMSGETTKN) procedure.
The ZEMS-TKN-TEXT token holds 102 bytes or fewer of text. If the original text was longer, it is truncated.

**Collector-Specific Event Messages**

These event messages describe events associated with the primary and alternate collectors and their environments. The event-message text for each message describes events occurring in each collector’s environment. It contains a ZEMS-TKN-PROC-DESC token representing the system and PID of the primary collector, and the system, process name, and PID of the alternate collector.

All collector-specific messages are written to the primary collector log file. In addition, the alternate collector writes all the messages that it generates to its own log file, except for message numbers 526 and 528.

**513: ZEMS-EVT-COLD-LOAD**

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
</tbody>
</table>

**Event-Message Text**

EMS: PRIMARY COLLECTOR $0,
DISK EVENT LOGGING STARTED

**Unconditional Tokens**

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

**ZEMS-TKN-EVENTNUMBER (shared)**

is the event number. Its value is ZEMS-EVT-COLD-LOAD (513).
ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token: ZEMS-TKN-COLLECTOR.

ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token has the value ZEMS-SUBJ-PCOLL (or 1).

Cause. An operator has performed a system load that generated the event message.

Effect. The primary collector generates this event message when a system load is performed and disk logging begins.

Recovery. This event message is primarily informative and is useful as a place marker in the event log.
514: ZEMS-EVT-FILESWITCH

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SYSTEM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-LASTLOGFILE</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWLOGFILE</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGSWITCHREASON</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-MAP-COL-STATUS</td>
<td>ZEMS-DDL-COL-STATUS.</td>
</tr>
<tr>
<td>ZEMS-TKN-LCT-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conditional Tokens</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-ACTION-NEEDED</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-ACTION-ID</td>
<td>ZSPI-TYP-INT.</td>
</tr>
</tbody>
</table>

**Event-Message Text**

EMS: COLLECTOR proc-desc SWITCHED LOG FILES
FROM FILE lastlogfile TO newlogfile
  { BY OPERATOR } 
  { BECAUSE DISK INACCESSIBLE } 
  { BECAUSE OLD FILE FULL }

Depends on logswitchreason.

Note: proc-desc is the collector’s system name and PID.

**Unconditional Tokens**

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual. Its value is ZSPI-VAL-TANDEM, ZSPI-SSN-ZEMS, or ZEMS-VAL-VERSION.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-FILESWITCH (514).
ZEMS-TKN-EMPHASIS *(shared)*

is a standard EMS token. For more information, see [Section 14, EMS Definitions](#). Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT *(shared)*

is a standard EMS token. For more information, see [Section 14, EMS Definitions](#). Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see [Section 14, EMS Definitions](#).

ZEMS-TKN-SUBJECT-MARK *(shared)*

is the standard EMS token that immediately precedes each subject token. This event message has one subject token: ZEMS-TKN-COLLECTOR.

ZEMS-TKN-COLLECTOR *(nonshared)*

is the collector type. This token has the value ZEMS-SUBJ-PCOLL (or 1) for the primary collector, and ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-LASTLOGFILE *(nonshared)*

is the file name of the log file in use before the switch.

ZEMS-TKN-NEWLOGFILE *(nonshared)*

is the file name of the log file in use after the switch.

ZEMS-TKN-LOGSWITCHREASON *(nonshared)*

is the reason for the file switch, shown by one of the following values:

ZEMS-VAL-OPRSWITCH

is the file switch by operator.

ZEMS-VAL-DISCACCESS

is the disk file that is not accessible.

ZEMS-VAL-FILEFULL

is the previous file filled.

ZEMS-MAP-COL-STATUS

is an extensible structured token that contains the current status of the collector; see [STATUS Command (ZEMS-CMD-STATUS)](#) on page 19-53.
Conditional Tokens

ZEMS-TKN-ACTION-NEEDED (nonshared)

tells whether some action is required (in this case, because the collector stopped
logging). The token value is FALSE because logging already resumed with a log-
file switch.

ZEMS-TKN-ACTION-ID (nonshared)

has the value 0. This token has the same value in the corresponding action-
attention message, ZEMS-EVT-LOGGING-STOPPED (521).

Cause. A log-file switch, either preplanned (through the ROTATEFILES option) or as
needed, was made by an operator or a management application.

The usual cause of this event is that the previous file was filled. Another cause might
be that the system operator requested a switch to a new log file or subvolume. These
causes are considered normal. No exceptional action is required of the operator,
except normal procedures to ensure that adequate disk capacity is available for
continued logging.

A third possible reason for this event is that the disk log file in use has become
inaccessible due to a disk I/O error, causing logging to be switched to the default log
subvolume. After the disk problem has been corrected, you can use the LOGSUBVOL
command of the EMSCCTRL program to return to the normal log subvolume.

Effect. None. The log-file switch has occurred. This message can be an action-
completion message for the event message ZEMS-EVT-LOGGING-STOPPED (521).

Recovery. This event message is an informative message only; no corrective action is
necessary.

Event Notes

- The collector saves this event message as the only record in the ZZEVCONF files
  and as the first record of each ZZEV_nnnn file. This mechanism provides backward
  links to all currently active log files. That is, ZZEVCONF points to log file _x_ (the
current log file), the first record of log file _x_ points to log file _y_ (the previous log file),
  and so forth.

- When present, the ACTION-NEEDED and ACTION-ID tokens mean that this
FILESWITCH event message was the action-completion response to a LOGGING-
STOPPED message. These tokens were in the LOGGING-STOPPED message.
When the operator responded, logging resumed with a file switch, and the tokens
were passed to the FILESWITCH message.
515: ZEMS-EVT-COLL-DISC-FAILED

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: COLLECTOR proc-desc DISK LOG FILE INACCESSIBLE, GUARDIAN FILE ERROR zfilerr REPORTED ON FILE failfilename

Note: proc-desc is the collector's system name and PID.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-COLL-DISC-FAILED (515).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)

specifies the collector type. This token can have only one value: ZEMS-SUBJ-PCOLL (or 1).

ZEMS-TKN-ZFILERR (nonshared)

is the error code associated with the event. For an explanation of this error code, see the Operator Messages Manual.

ZEMS-TKN-FAILFILENAME (nonshared)

is the file name of the inaccessible log file.

Cause. The primary collector generates this event message whenever a log file becomes inaccessible. Either disk failure or operational errors can cause this event. In the case of disk failure, the collector switches logging to the default subvolume. In the case of operational errors, the collector does not switch logging subvolumes. It is assumed that the operator will act to correct the error.

These error-message elements—number and text—indicate operational errors. Each number is a ZEMS-TKN-ZFILERR token value, labeled GUARDIAN ERROR in the event-message text. The accompanying text interprets the number in the collector context:

010 MAXFILE files already exist
043 Unable to obtain disk space for file extent
044 Disk directory is full

The value of ZEMS-TKN-ZFILERR is 10 if ZCOL-ROTATEFILES is FALSE (you asked that logging stop), and if the (ZCOL-MAXFILENNNN) log files in the logging subvolume are full. (See CONTROL Command (ZEMS-CMD-CONTROL) on page 17-17.)

Effect. The collector can no longer log event messages to disk. It continues to accept event messages in its internal queue until it runs out of space. Then it rejects incoming event messages with error 33.

Recovery. If logging has stopped because of too many log files, purge the oldest file, and logging resumes.

If you need more disk space, archive some files (log files or other files) and purge them from the volume. You can also use EMSCCTRL with keyword MAXFILE to adjust the number of files permitted or EXT to adjust the size of file extents.

For problems with the current disk volume—if you need a larger logging volume or have hardware problems with the current volume—locate a disk volume appropriate for the creation of new log files. Then use EMSCCTRL with the LOGSUBVOL keyword to give the collector the names of the new volume and subvolume.
517: ZEMS-EVT-COMPAT-DISTR-STOPPED

<table>
<thead>
<tr>
<th>Unconditional Token</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-COMPATDISTCRTPID</td>
<td>ZSPI-TYP-CRTPID.</td>
</tr>
<tr>
<td>ZEMS-TKN-CDIST-CPU-PIN</td>
<td>ZSPI-TYP-UINT-PAIR.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: COMPATIBILITY DISTRIBUTOR FAILED - GUARDIAN ERROR - zfilerr

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

ZEMS-TKN-EVENTNUMBER *(shared)*

is the event number. Its value is ZEMS-EVT-COMPAT-DISTR-STOPPED (517).

ZEMS-TKN-EMPHASIS *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)

specifies the collector type. This token has the value ZEMS-SUBJ-PCOLL (or 1).

ZEMS-TKN-ZFILERR (nonshared)

is the error code associated with this event. For an explanation of this error code, see the Operator Messages Manual.

ZEMS-TKN-COMPATDISTCRTPID (nonshared)

is the last known CRTPID for the compatibility distributor.

ZEMS-TKN-CDIST-CPU-PIN (nonshared)

is the CPU and PIN of the compatibility distributor.

**Cause.** The primary collector cannot successfully access the compatibility distributor. The primary collector detected an error, ZFILERR = 201, which means the $Z0 process no longer exists. This event can have various causes, including:

- A compatibility distributor that was stopped by this command:
  
  \texttt{EMSCCTRL 0 CDISTSTOP}

- Failure of the processors in which the compatibility distributor and its backup process were running

**Effect.** The compatibility distributor was terminated—normally or abnormally.

**Recovery.** If the CPUs are down, reload them. Restart the compatibility distributor.
Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-COL-EVENT-DISCARDS</td>
<td>ZEMS-TYP-COL-EVENT-DISCARDS.</td>
</tr>
<tr>
<td>ZEMS-TKN-OPMSG</td>
<td>ZEMS-TYP-OPMSG.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: count-0 SENDOPMSGs, count-1 ESPSENDs, count-2 WRITES, AND count-3 WRITEREADS DISCARDED BY COLLECTOR proc-desc, PID cpu, pin

Note: count-0 through count-3 correspond to the fields of ZEMS-TKN-COL-EVENT-DISCARDS. proc-desc is the collector’s system name and PID.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-COL-EVENT-DISCARDS (518).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)

specifies the collector type. This token has the value ZEMS-SUBJ-PCOLL (or 1).

ZEMS-TKN-COL-EVENT-DISCARDS (nonshared)

is an array of four integers with this DDL definition:

```
DEFINITION ZEMS-DDL-COL-EVENT-DISCARDS.
  02 zsendopmsg       TYPE zspi-ddl-uint.
  02 zmessenger       TYPE zspi-ddl-uint.
  02 zusertext        TYPE zspi-ddl-uint.
  02 zuserevent       TYPE zspi-ddl-uint.
END
```

These four fields of ZEMS-TKN-COL-EVENT-DISCARDS represent these categories of discarded messages:

- Operator console messages
- Other operating-system messages
- Text messages written to $0 with the WRITE procedure
- Event messages written to $0 with the WRITEREAD procedure

ZEMS-TKN-OPMSG (nonshared)

is private to HP.

**Cause.** The primary collector has recovered from an event-flooding condition. So many events were reported at about the same time that the collector had to discard some of them and reply to the event issuer with an error. A process might be looping, which can cause event-message flooding if the loop generates an event message. A forwarding distributor can cause flooding if the distributor forwards an event message to the system that originally reported the event.

**Effect.** None. The primary collector has recovered.

**Recovery.** Try to determine the cause of event-message flooding. You might succeed in decreasing the rate of event-message generation, especially if many messages come from a single source. For example, if you determine that a process is sending many messages (the process might be looping), you can stop the process or place it under a debugger. Similarly, if a device malfunctions and produces a flood of event messages, you can stop the device and have the problem fixed.

If no particular device or process is responsible for the flood, in some cases you can change collector attributes to increase the rate at which event messages are logged. EMSCCTRL can help you change these attributes:
**Collector Event Messages**

**EMS Manual—426909-005**

519: ZEMS-EVT-MSGR-EVENTS-DISCARDED

- **WRITETHRUCACHE.** By using EMSCCTRL with the BUFFERED ON option, you are selecting a buffering mode used by the disk process. When buffering, event messages accumulate in the disk cache-buffer before they are written to disk. This mode of operation involves significantly fewer writes to disk and provides substantial gains in efficiency.

  **Note.** The buffering, however, involves a slight risk. If the disk’s CPU and backup both fail, the event messages in the disk cache buffer are lost.

Whether greater efficiency or greater security is more important depends on your particular programming environment.

- **EOFREFRESH.** By using EMSCCTRL with the REFRESH ON option, you are directing the collector to update the end-of-file pointer on disk for each block written. This is somewhat safer but involves more writes to the disk. The change in efficiency is less substantial than in the BUFFERED ON/OFF decision. If efficiency is your primary goal, use REFRESH OFF.

- **EVENTBLOCKING.** By using EMSCCTRL with the BLOCKING ON option, you are directing the collector to block events to disk, if possible. At high event rates, this substantially increases the collector’s maximum sustained logging rate.

519: ZEMS-EVT-MSGR-EVENTS-DISCARDED

**Unconditional Tokens**

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-_UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-MSGR-EVENTS-DISCARDED</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-MESSENGERCPU</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
</tbody>
</table>

**Event-Message Text**

EMS: count EVENTS DISCARDED BY SENDVIAMESSENGER IN CPU messenger-cpu

**Unconditional Tokens**

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the *SPI Programming Manual*. 

EMS Manual—426909-005

20-20
ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-MSGR-EVENTS-DISCARDED (519).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token has the value ZEMS-SUBJ-PCOLL (or 1).

ZEMS-TKN-MSGR-EVENTS-DISCARDED (nonshared)
is the number of messages that the SENDVIAMESSENGER procedure discarded.

ZEMS-TKN-MESSENGERCPU (nonshared)
is the CPU of the SENDVIAMESSENGER procedure that discarded the event messages.

**Cause.** The primary collector has been informed that the SENDVIAMESSENGER procedure (details internal to HP) has discarded event information because it received too many event messages (event-message flooding). The message states how many messages were discarded. For more information about event message flooding, see the cause for event message 518: ZEMS-EVT-COL-EVENT-DISCARDS on page 20-18.

**Effect.** The messages specified were discarded.

**Recovery.** Try to determine the process or device that is sending the ZEMS-TKN-MESSENGERCPU so many messages. If a process is sending too many messages—it might be creating messages in a loop—you can stop it or place it under a debugger. Likewise, if a device malfunctions and produces a flood of event messages, you can stop the device and have the problem fixed.
Changing collector attributes cannot help this problem because messages are
discarded before they reach the collector.

Event Notes
At any time, ZEMS-TKN-MSGR-EVENTS-DISCARDED does not exceed 32767
although more events might have already been discarded.

The SENDVIAMESSENGER procedure reports events from:

- System library procedures
- Privileged I/O processes
- Privileged user processes

520: ZEMS-EVT-FILE-ROTATE-PURGE

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-PURGEDLOGFILE</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
</tbody>
</table>

Event-Message Text
EMS: LOG FILE purgedlogfile PURGED BY COLLECTOR proc-desc, ROTATEFILES OPTION

Note: proc-desc is the collector’s system name and PID.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-FILE-ROTATE-PURGE (520).
ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token has one of two values: ZEMS-SUBJ-PCOLL (or 1) for the primary collector and ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-PURGEDLOGFILE (nonshared)
is the file name of the file that the collector purged or renamed.

**Cause.** These conditions, all of which must occur, cause the ZEMS-EVT-FILE-ROTATE-PURGE event message:

- The value of the ZCOL-ROTATEFILES collector attribute is TRUE.
- The collector, in trying to write an event message to its current log file, reached the end of file.
- The last of the available (ZCOL-MAXFILENNNN) log files in the logging subvolume has been used. (See CONTROL Command (ZEMS-CMD-CONTROL) on page 19-34.)
- The collector just purged or renamed the oldest log file to provide more log file space.

For more information about changing the ZCOL-ROTATEFILES collector attribute, see CONTROL Command (ZEMS-CMD-CONTROL) on page 19-34 or EMSCCTRL—Control Collector Utility on page 13-9.

**Effect.** The collector has continued to the next log file.

**Recovery.** This event message is informative only. No corrective action is necessary. If this event message occurs too frequently, increase the extent sizes of your log files.
521: ZEMS-EVT-LOGGING-STOPPED

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGSTPREASON</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-ACTION-NEEDED</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-ACTION-ID</td>
<td>ZSPI-TYP-INT.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: COLLECTOR proc-desc EVENT LOGGING STOPPED BECAUSE
  { DISK FAILED     }
  { NORotate OPTION }

  Depends on logstpreason.

Note: proc-desc is the collector’s system name and PID.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-LOGGING-STOPPED (521).

ZEMS-TKN-EMPHASIS (shared)

is the standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)

specifies the collector type. This token has one of two values: ZEMS-SUBJ-PCOLL (or 1) for the primary collector and ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-LOGSTOPREASON (nonshared)

specifies the reason the collector stopped logging. The two main causes of this event are expressed by the token values ZEMS-VAL-DISCFAILED and ZEMS-VAL-NORotate.

ZEMS-TKN-ACTION-NEEDED (nonshared)

specifies whether some action is required (in this case because the collector stopped logging). The token value here is TRUE.

ZEMS-TKN-ACTION-ID (nonshared)

has the value 0. The value is the same in the corresponding action-completion message, ZEMS-EVT-FILESWITCH (514).

Cause. A primary or alternate collector cannot log event messages to a log file. This event message is written to the log file when access to the file is restored, and it is forwarded to any distributor waiting on a STATUS command from the collector. This message is an action-attention message. The corresponding action-completion message is ZEMS-EVT-FILESWITCH (514). The two main causes of this event are:

- The last of the available log files in the logging subvolume is full, and the files cannot be rotated (purged and recycled). See the fields ZCOL-MAXFILENNNN and -ROTATEFILES in CONTROL Command (ZEMS-CMD-CONTROL) on page 19-34.

Effect. Logging stops. The primary collector eventually runs out of buffer space.

Recovery. If the maximum number of log files (given by ZCOL-MAXFILENNNN) is already in use, you can do one of four things to resume logging:

- Increase the value of ZCOL-MAXFILENNNN, using the CONTROL command or the MAXFILE parameter of EMSCCTRL.
- Change ZCOL-ROTATEFILES to TRUE using the CONTROL command or the ROTATEFILES parameter of EMSCCTRL.
- Purge the oldest file.
- Change the logging volume or subvolume using the LOGSUBVOL parameter of EMSCCTRL.

If a disk fails, locate space for new log files on another volume. You can change the logging volume using the LOGSUBVOL parameter of EMSCCTRL.

522: ZEMS-EVT-COLLECTOR-RUN

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: ALTERNATE COLLECTOR `processname`,
DISK EVENT LOGGING STARTED

Note: `processname` comes from the token ZEMS-TKN-PROC-DESC.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-COLLECTOR-RUN (522).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions.
Here, its value is FALSE.
ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

Cause. An operator started an alternate collector by issuing the TACL command EMSACOLL.

Effect. This event message is the first message placed in the alternate collector log file after a collector RUN command is issued.

Recovery. This event message is for information only. No corrective action is necessary.
523: ZEMS-EVT-ACOL-EVENT-DISCARDS

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-ACOL-EVENTS-DISCARDS</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
</tbody>
</table>

Event-Message Text
EMS: count EVENTS DISCARDED
       BY COLLECTOR name, PID cpu, pin

Note: name comes from the token ZEMS-TKN-PROC-DESC.

Unconditional Tokens

ZSPI-TKN-SSID
is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-ACOL-EVENT-DISCARDS (523).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared) is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared) specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

ZEMS-TKN-ACOL-EVENTS-DISCARDS (nonshared) contains a count of the number of events discarded by the alternate collector.

**Cause.** The alternate collector has recovered from event-flooding or has discarded one or more invalid events (see 525: ZEMS-EVT-INVALIDEVENT on page 20-33). The total number of events discarded is reported in the ZEMS-TKN-ACOL-EVENT-DISCARDS field. Events can be discarded by:

- A process might be looping, which can cause event-message flooding if the loop generates an event message.
- A forwarding distributor can cause flooding if the distributor forwards an event message to the system that originally reported the event.
- A message was sent to the alternate collector that is not a proper event message.
- Logging stops, causing the alternate collector to run out of space in its pool (pool size is determined by POOLPAGES). When the out-of-space condition occurs, event messages can be discarded after they have been forwarded to all distributors. When logging resumes, the alternate collector issues this event to tell the operator the number of events discarded while logging was stopped.

**Effect.** The specified events were discarded.

**Recovery.** Evaluate the cause of the event flooding and perform one of these actions:

- Cause the event generators to decrease the rate of event generation.
- Change collector attributes to allow higher logging rates.
- Reassign some of the events to other collectors.
- Stop or fix the generator of invalid events.
524: ZEMS-EVT-LOGTIME-DECREASE

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-OLD-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEW-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: LOGTIME ERROR ENCOUNTERED - TIME DECREASED
FROM time-1 date-1 TO time-2 date-2

Note: time-1 date-1 depends on the value of ZEMS-TKN-OLD-LOGTIME, and
time-2 date-2 depends on the value of ZEMS-TKN-NEW-LOGTIME.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in Spi Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-LOGTIME-DECREASE (524).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK **(shared)**

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR **(nonshared)**

specifies the collector type. This token has one of two values: ZEMS-SUBJ-PCOLL (or 1) for the primary collector, or ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-OLD-LOGTIME **(nonshared)**

is the log time (Julian timestamp) of the event message logged just before a system-time change.

ZEMS-TKN-NEW-LOGTIME **(nonshared)**

is the log time (Julian timestamp) of the event message logged just after a system-time change.

**Cause.** The log timestamp of a just-received event message is out of sequence. This sequence of events describes conditions that can cause an ZEMS-EVT-LOGTIME-DECREASE message to be issued:

1. The collector is writing event messages to the current log file. It puts a log timestamp in each message. This is an example timestamp:

   Event A (19:00)

2. The collector compares the log timestamp of each message to that of the preceding message. If each timestamp is greater (later) than that in the previous message, the messages appear in the order expected by the collector, and logging continues. In the following, event B is an example of a greater (later) timestamp:

   Event A (19:00)  ...  Event B (19:01)

3. The system operator resets the system time to an earlier time, using a SETTIME command.

4. The collector timestamps the next event message:

   Event C (18:58)

   This strategy creates a temporary reversal in the sequence of timestamps. (If the system time is reset to a later value, there is no problem.)

5. The collector compares this timestamp to the timestamp of the previous message and determines that the timestamp of the current message is less (earlier) than the timestamp of the previous message. In this example, event C has an earlier timestamp:

   Event B (19:01)  ...  Event C (18:58)
6. The collector issues a LOGTIME-DECREASE event message, with a log timestamp later than that of event C:

   Event C (18:58) ... Event L-D (18:59)

   (At least one event message, C, necessarily has a log timestamp that is out of sequence. In this example, there are two or more such messages: C, L-D, and so on.)

7. One of the distributors issues a positioning command to the collector, requesting all event messages with log timestamps of 19:00 and later.

8. The distributor locates, filters, and sends event messages that have suitable timestamps if the conditions of the filter are met. In this case, messages A and B are candidates to be sent:

   Event A (19:00) ... Event B (19:01)

9. When it checks event message C, the distributor sees a log timestamp earlier than those requested, so it does not attempt to pass message C.

10. When the distributor checks event message L-D, it sees that this is a LOGTIME-DECREASE message. Despite the timestamp, 18:59, it passes L-D to its destination unless its filter has been programmed to reject LOGTIME-DECREASE event messages.

**Effect.** The log timestamps are no longer in an ascending pattern. This might cause distributor positioning errors later.

**Recovery.** The purpose of this message is to alert system users and management applications to an out-of-sequence event message or messages. Eventually, each type of distributor sees the LOGTIME-DECREASE message and sends it to each destination (unless the filter rejects it). Its meaning depends on that destination.

An operator who sees this message will be concerned with whether the time was correctly reset, or whether a new resetting is needed. ($Z0 always passes LOGTIME-DECREASE messages to the console for display or printing.)

An analyst who sees the LOGTIME-DECREASE message in the print-out from an application will realize that at least one event message—the preceding one—was not distributed that probably should have been. The analyst can have the distributor issue a new positioning command, seeking all messages with times early enough so that (in all likelihood) messages that were skipped are now filtered and distributed. In the Cause example, seeking log timestamps of 18:57 and later might pick up some messages earlier than A. In any case, it would pick up message C, 18:58.
Unconditional Tokens Table

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK (shared)</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-XSYSPID</td>
<td>ZEMS-DDL-XSYSPID.</td>
</tr>
</tbody>
</table>

Event-Message Text
EMS: COLLECTOR RECEIVED BAD EVENT BUFFER FROM SYSTEM sysnum, PID cpu, pin

Unconditional Tokens

ZSPI-TKN-SSID
is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-INVALIDEVENT (525).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here its value is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:
ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token has one of two values: ZEMS-SUBJ-PCOLL (or 1) for the primary collector, or ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-XSYSPI D (nonshared)
is private to HP. It contains the system number and PID of the process that sent the invalid message.

**Cause.** A primary or alternate collector received an improper event message, usually caused by one of:

- A process called the WRITEREAD procedure with an old-style text message.
- A process called WRITEREAD with an incorrect count.
- A programming error in the sending process created an improperly formatted event message.

**Note.** The alternate collector issues this event message once for each event issuer. The primary collector issues this event message for each invalid event received.

**Effect.** The event message is not logged.

**Recovery.** Stop the process that generated the invalid message.
526: ZEMS-EVT-ACOL-INTERNAL-ERR

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-CODESEG</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PREG</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: INTERNAL ERROR AT %codeseg.preg

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-ACOL-INTERNAL-ERR (526).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:
**ZEMS-TKN-COLLECTOR** (nonshared) specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

**ZEMS-TKN-CODESEG** (nonshared) is the code segment number where the error is detected.

**ZEMS-TKN-PREG** (nonshared) is the P register contents where the error is detected.

**Cause.** The alternate collector has encountered an internal error. This event message should never occur. This event message is sent to the primary collector on the alternate collector’s system. The alternate collector does not attempt to write this event to its own logging file.

**Effect.** The collector stops.

**Recovery.** Contact your HP representative.

### 527: ZEMS-EVT-ACOL-SHUTDOWN

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

**Event-Message Text**

EMS: COLLECTOR processname SHUTDOWN

**Note:** processname comes from the token ZEMS-TKN-PROC-DESC.

**Unconditional Tokens**

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual*. 

---

527: ZEMS-EVT-ACOL-SHUTDOWN

EMS Manual—426909-005

20-36
ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-ACOL-SHUTDOWN (527).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

Cause. The alternate collector has completed processing a SPI STOP command. This event is not generated when the alternate collector is stopped by other means, such as with the TACL STOP command.

Effect. None.

Recovery. This event message is informational only. No action is necessary.
528: ZEMS-EVT-ACOL-ALLOCATESEG-ERR

Unconditional Tokens

<table>
<thead>
<tr>
<th>ZSPI-TKN-SSID</th>
<th>ZSPI-TYP-SSID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SEGALLOC-ERROR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: SEGMENT ALLOCATION FAILURE segalloc-error
ON VOLUME failfilename

Where segalloc-error can be one of the following:

{ CREATE OR OPEN ERROR }  
{ PARAMETER ERROR }  
{ BOUNDS ERROR }  
{ ILLEGAL SEGMENT ID }  
{ ILLEGAL SEGMENT SIZE }  
{ UNABLE TO ALLOCATE SEGMENT SPACE }  
{ UNABLE TO ALLOCATE SEGMENT PAGE TABLE SPACE }  

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-ACOL-ALLOCATESEG-ERR (528).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.
ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions.
Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has two subject tokens:

ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

ZEMS-TKN-FAILFILENAME (nonshared)
is the name of the swap volume.

ZEMS-TKN-ZFILERR (nonshared)
is the error code associated with an event. For an explanation of this error code, see the Operator Messages Manual.

ZEMS-TKN-SEGALLOC-ERROR (nonshared)
is the type of allocation error. The different types are listed in Event Message Text in the event message box for this event message.

Cause. An alternate collector process cannot allocate its extended data segment. The collector does not attempt to write this event to its own log. The most likely cause is lack of space on the alternate collector’s swap volume.

Effect. The alternate collector terminates abnormally after sending this event message to the primary collector.

Recovery. Either run the alternate collector, specifying a different SWAP volume, or make space available on the original volume.
## 529: ZEMS-EVT-ACOL-CHECKOPEN-FAILED

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
</tbody>
</table>

### Event-Message Text

EMS: CHECKOPEN FAILED, ERROR zfilerr ON failfilename

### Unconditional Tokens

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

**ZEMS-TKN-EVENTNUMBER (shared)**

is the event number. Its value is ZEMS-EVT-ACOL-CHECKOPEN-FAILED (529).

**ZEMS-TKN-EMPHASIS (shared)**

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

**ZEMS-TKN-CONSOLE-PRINT (shared)**

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

**ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID**

are standard EMS tokens. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared) 
is the standard EMS token that immediately precedes each subject token. This 
event message has two subject tokens:

ZEMS-TKN-COLLECTOR (nonshared) 
specifies the collector type. This token can have only one value: ZEMS-SUBJ- 
ACOLL (or 2).

ZEMS-TKN-FAILFILENAME (nonshared) 
is the name of the file that could not be opened by the backup alternate 
collector.

ZEMS-TKN-ZFILERR (nonshared) 
is the error code associated with an event. For an explanation of this error code, 
see the Operator Messages Manual.

Cause. The alternate collector has determined that its backup process is 
malfunctioning because it cannot open a file previously opened by the primary process.

Effect. The collector stops its backup process and attempts to create a new backup 
after a 30-second delay.

Recovery. For file system errors 30 through 37, wait and try again. If the error occurs 
again, increase system resources. If this does not correct the problem, or if it results in 
a different error, contact your service provider.
530: ZEMS-EVT-ACOL-TAKEOVER

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-TAKEOVER-REASON</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: TAKEOVER BY BACKUP (takeover text)

Where takeover text depends on the value of takeover-reason. The values of takeover-reason are as follows:

{ PRIMARY STOPPED }
{ PRIMARY ABENDED }
{ PRIMARY CPU IS DOWN }
{ OPERATOR REQUESTED SWITCH }

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-ACOL-TAKEOVER (530).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)

specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

ZEMS-TKN-TAKEOVER-REASON (nonshared)

is the reason for the takeover.

**Cause.** The primary alternate collector process, or the CPU in which it runs, failed. This event is generated after the backup alternate collector process takes over.

**Effect.** If the primary alternate collector process fails, the backup process becomes the primary alternate collector. The primary alternate collector attempts to create a new backup after a 30-second delay. In the case of a CPU failure, a new backup is created 30 seconds after the new primary alternate collector process receives a CPU RELOADED system message.

**Recovery.** If the primary process terminates abnormally, investigate the event log to determine why.
531: ZEMS-EVT-ACOL-CREATEBACKUP-ERR

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Name</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROGRAMFILE</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-CPU</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-PRIORITY</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROCCREATE-ERROR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Conditional Tokens

<table>
<thead>
<tr>
<th>Token Name</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: UNABLE TO CREATE BACKUP PROCESS IN CPU cpu, ERROR errnum: errText

Where err text depends on the value of process_create_-error. The values of process_create_-error are as follows:

- NO PROCESS CONTROL BLOCK AVAILABLE
- ERROR ON PROGRAMFILE: zfilerr
- UNABLE TO ALLOCATE MAP
- ERROR ON SWAP FILE : zfilerr
- ILLEGAL FILE FORMAT
- PROCESS NAME ERROR : zfilerr
- LIBRARY CONFLICT
- UNABLE TO COMMUNICATE WITH SYSTEM MONITOR
- ERROR ON LIBRARY FILE : zfilerr
- LIBRARY AND PROGRAM FILE ARE THE SAME
- EXTENDED SEGMENT ERROR : zfilerr
- SWAP FILE ERROR : zfilerr
- ILLEGAL HOME TERMINAL, ERROR zfilerr

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.
Collector Event Messages

531: ZEMS-EVT-ACOL-CREATEBACKUP-ERR

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-ACOL-CREATEBACKUP-ERR (531).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has two subject tokens:

ZEMS-TKN-COLLECTOR (nonshared)

specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

ZEMS-TKN-PROGRAMFILE (nonshared)

is the alternate collector object file name.

ZEMS-TKN-NEWPROCESS-CPU (nonshared)

is the CPU specified in the PROCESS_CREATE_ procedure call.

ZEMS-TKN-NEWPROCESS-PRIORITY (nonshared)

is the alternate collector priority.

ZEMS-TKN-PROCCREATE-ERROR (nonshared)

is the process create error.

Conditional Tokens

ZEMS-TKN-ZFILERR (nonshared)

is the error code associated with an event. For an explanation of this error code, see the Operator Messages Manual.

Cause. The alternate collector cannot create a backup due to a PROCESS_CREATE_ error.
Effect. The primary alternate collector process tries to create the backup again, unless the PROCESS_CREATE_ error indicates that the backup CPU is down. If the backup CPU is not down, the primary alternate collector tries to create a new backup 30 seconds after it receives a CPU RELOADED system message for the backup CPU. For errors other than the CPU being down, the primary alternate collector delays for 30 seconds and retries; if the second attempt fails, the next attempt occurs after a 60-second delay. This process continues (the delay increases by 30 seconds at each failure) until the delay reaches 5 minutes. Thereafter, the primary alternate collector attempts to create a new backup every 5 minutes.

The event message is issued only if the initial attempt to create a backup fails; unsuccessful retries are not reported.

Recovery. The recommended action depends on the PROCESS_CREATE_ error as described in the Guardian Procedure Calls Reference Manual.

532: ZEMS-EVT-ACOL-BACKUP-CREATED

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROGRAMFILE</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-CPU</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-PRIORITY</td>
<td>ZSPI-TYP-INT.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: BACKUP CREATED IN CPU cpu

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-ACOL-BACKUP-CREATED (532).
ZEMS-TKN-EMPHASIS *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is FALSE.

ZEMS-TKN-CONSOLE-PRINT *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK *(shared)*

is the standard EMS token that immediately precedes each subject token. This event message has two subject tokens:

ZEMS-TKN-COLLECTOR *(nonshared)*

specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

ZEMS-TKN-PROGRAMFILE *(nonshared)*

is the name of the program file. Here, it is the program file for the alternate collector.

ZEMS-TKN-NEWPROCESS-CPU *(nonshared)*

is the CPU number specified in the PROCESS_CREATE_ procedure call.

ZEMS-TKN-NEWPROCESS-PRIORITY *(nonshared)*

is the alternate collector priority.

**Cause.** The alternate collector’s primary process successfully started a new backup process.

**Effect.** A new backup process is started and begins to function normally.

**Recovery.** This event message is for information only. No corrective action is necessary.
533: ZEMS-EVT-ACOL-BACKUP-ABENDED

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT (shared)</td>
<td>ZSPI-TYP-BOOLEAN</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME (shared)</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU (shared)</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC (shared)</td>
<td>ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN (shared)</td>
<td>ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM (shared)</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID (shared)</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK (shared)</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR (shared)</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
</tbody>
</table>

**Event-Message Text**

EMS: BACKUP PROCESS ABENDED

**Unconditional Tokens**

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-ACOL-BACKUP-ABENDED (533).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:
ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

Cause. The alternate collector’s backup process has terminated abnormally.

Effect. The primary alternate collector process attempts to create a new backup after a 30-second delay.

Recovery. Contact your HP representative for assistance; provide the SAVEABEND file from the backup.

534: ZEMS-EVT-ACOL-BACKUP-DELETED

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Event-Message Text
EMS: BACKUP PROCESS DELETED (CPU DOWN)

Unconditional Tokens

ZSPI-TKN-SSID
is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-ACOL-BACKUP-DELETED (534).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.
ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared)
specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

Cause. The CPU in which the alternate collector's backup process was running failed.

Effect. The primary alternate collector process does not attempt to create a backup until 30 seconds after it receives a CPU RELOADED system message for its backup CPU.

Recovery. Follow the recommended procedure for CPU failure.
535: ZEMS-EVT-ACOL-CHECKPOINT-ERR

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR (nonshared)</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: CHECKPOINT FAILED, ERROR zfilerr

Unconditional Tokens

ZSPI-TKN-SSID
is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)
is the event number. Its value is ZEMS-EVT-ACOL-CHECKPOINT-ERR (535).

ZEMS-TKN-EMPHASIS (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)
is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID
are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-ZFILERR (nonshared)
is the error code associated with an event. For an explanation of this error code, see the Operator Messages Manual.
ZEMS-TKN-SUBJECT-MARK (shared) is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared) specifies the collector type. This token can have only one value: ZEMS-SUBJ-ACOLL (or 2).

Cause. The alternate collector’s backup process received an I/O error during a checkpoint operation.

Effect. The alternate collector stops its backup process and attempts to create a new backup after a 30-second delay. Resource allocation failures (file system errors 30 through 37) might indicate that the primary or backup CPU is overloaded.

Recovery. For file system errors 30 through 37, wait and try again. If the error occurs again, free the system resources. If this does not correct the problem, or results in a different error, contact your service provider.
536: ZEMS-EVT-COL-PURGETABLE-OVRFLO

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-BYTE-PAIR.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-PURGEDLOGFILE</td>
<td>ZSPI-TYP-FNAME.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: COLLECTOR proc-desc COULD NOT PURGE LOG FILE purged-logfile AUTOMATICALLY. FILE MUST BE PURGED MANUALLY.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-COL-PURGETABLE-OVRFLO (536).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see *Section 14, EMS Definitions*. Here, its value is FALSE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see *Section 14, EMS Definitions*. Here, its value is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see *Section 14, EMS Definitions*. 
ZEMS-TKN-SUBJECT-MARK (shared)  
is, for this message, a valueless token.

ZEMS-TKN-COLLECTOR (nonshared)  
specifies the collector type. This token has the value 1.

ZEMS-TKN-PURGEDLOGFILE (nonshared)  
is the file name of the file that needs to be purged manually.

**Cause.** The primary collector, $0$, is creating log files much faster than the distributors can read them. When $0$ cannot purge the oldest log file (because a distributor is still accessing it), it saves the log file name in an internal table so that it can attempt to purge it later. This table overflowed and the oldest entry was deleted. The token ZEMS-TKN-PURGEDLOGFILE contains the log file named that was stored in the deleted entry.

**Effect.** The oldest entry in the $0$ internal table containing log file records to be purged, must be purged manually.

**Recovery.** HP recommends two possible actions:

- Determine why so many events are being generated, and attempt to reduce the number of events being generated.
- Make the log files larger so they hold more event messages. Use EMSCCTRL with the keyword EXT (to adjust the size of file extents) to increase the size of the log files.
537: ZEMS-EVT-COL-CONFIG-WARNING (D-series Only)

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-BYTE-PAIR.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONFIG-ITEM</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-SPEC-CONFIG-VALUE</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-USED-CONFIG-VALUE</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: SYSTEM_PROCESS_MODIFIERS SECTION OF THE SYSGEN CONFIGURATION FILE CONTAINS AN INCORRECT VALUE.
ITEM: config-item, SPECIFIED VALUE: spec-config-value, USED VALUE: used-config-value.

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER

is the event number. Its value is ZEMS-EVT-COL-CONFIG-WARNING (537).

ZEMS-TKN-EMPHASIS

is a standard EMS token. For more information, see Section 14, EMS Definitions. Here, its value is FALSE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK

is, for this message, a valueless token.

ZEMS-TKN-COLLECTOR

specifies the collector type. This token has the value 1.
ZEMS-TKN-CONFIG-ITEM
    has the value: ZEMS-VAL-CONFIG-EMSFLAGS.

ZEMS-TKN-SPEC-CONFIG-VALUE
    has the value specified by SYSGEN.

ZEMS-TKN-USED-CONFIG-VALUE
    is the actual value used by $0.

**Cause.** Bit 15 of the EMSFLAGS word in $0 has been set to 1 by SYSGEN during process initialization, just after system load. The configuration file is specifying EMS display format, which is no longer necessary.

**Effect.** None.

**Recovery.** Examine the displayed message to determine which bit caused this event and eliminate it before the next system generation.
## 538: ZEMS-EVT-BURST-START

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT (shared)</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT (shared)</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK (shared)</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-EVT-NUM (shared)</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SSID (shared)</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-CODE (shared)</td>
<td>token-type ZSPI-TYP-TOKENCODE.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-VALUE (shared)</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-SSID (shared)</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-TIME-START (shared)</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-MAP-BDS-INFO</td>
<td>token-type ZEMS-DDL-BDS-INFO.</td>
</tr>
</tbody>
</table>

### Event-Message Text

EMS: EVENT BURST DETECTED FOR EVENT NO. eventno OF SUBSYSTEM ssid, BY COLLECTOR proc-desc

### Unconditional Tokens

**ZSPI-TKN-SSID**

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

**ZEMS-TKN-EVENTNUMBER (shared)**

is the event number. Its value is ZEMS-EVT-LOG-ACCESS (1000).

**ZEMS-TKN-EMPHASIS (shared)**

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

**ZEMS-TKN-CONSOLE-PRINT (shared)**

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.
ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR

is the EMS process type (value 1 for the primary collector; value 2 for an alternate collector).

ZEMS-TKN-BURST-EVT-NUM

is the event number of the bursting event.

ZEMS-TKN-BURST-SSID

is the subsystem ID of the bursting event.

ZEMS-TKN-BURST-SUBJ-CODE

is the token code of the first subject in the bursting event.

ZEMS-TKN-BURST-SUBJ-VALUE

is the value of the first subject of the bursting event.

ZEMS-TKN-BURST-SUBJ-SSID

is the subsystem ID of the first subject of the bursting event.

ZEMS-TKN-BURST-TIME-START

is the Julian timestamp for the burst start time.

ZEMS-MAP-BDS-INFO

contains the burst detection and suppression (BDS) configuration parameters in effect when the burst was detected.

Text Values

eventno

is from ZEMS-TKN-BURST-EVT-NUM.

ssid

is from ZEMS-TKN-BURST-SSID.
**proc-desc**

is from ZEMS-TKN-BURST-PROC-DESC.

**Cause.** An event burst was detected. In BDS configuration parameters terminology, a number (N) of similar events occurred within the T1 time interval. Similar events are those that have the same SSID, event number, and subject. The L burst parameter defines what same subject means.

**Effect.** Until the event burst ends, subsequent occurrences of these similar events are counted and discarded (that is, are not logged to disk).

**Recovery.** Examine the event to confirm that it is a repetitive event. If it is, totally suppress the event, by doing one of:

- Load a compiled filter or filter table into the collector (that is, using pre-log filtration) to prevent it from being logged and monitored by BDS.
- Prevent the event from being sent to the collector.

You can use the timestamp stored in the BURST-TIME-START token as a unique burst ID for programmed operators that need to verify that an event burst has ended.

If the event is useful, but its repetition is not, change the sending subsystem so it does not send out the same event repeatedly.

If all occurrences of the bursting event are useful, change the BDS configuration parameters so bursts are not suppressed. For detailed BDS configuration parameter information, see [Section 7, Burst Detection and Suppression](#).
539: ZEMS-EVT-BURST-END

### Event-Message Text

If ZEMS-TKN-BURST-END-REASON = ZEMS-VAL-BDS-ENABLED, then the event message text is:

**EMS: BURST SUPPRESSION TERMINATED:** count OCCURRENCES OF EVENT NO. eventno OF SUBSYSTEM ssid WERE NOT LOGGED BY COLLECTOR proc-desc.

If ZEMS-TKN-BURST-END-REASON = ZEMS-VAL-NO-EVENTS, then the event message text is:

**EMS: BURST END DETECTED:** count OCCURRENCES OF EVENT NO. eventno OF SUBSYSTEM ssid WERE NOT LOGGED BY COLLECTOR proc-desc.

### Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENBOX</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-EVT-NUM</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-CODE</td>
<td>ZSPI-TYP-TOKENCODE.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-VALUE</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-SUBJ-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-TIME-START</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-TIME-END</td>
<td>ZEMS-DDL-BDS-INFO.</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-END</td>
<td>ZEMS-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-ETS-DELETED</td>
<td>ZEMS-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-BURST-END-REASON</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
</tbody>
</table>

**Unconditional Tokens**

- **ZSPI-TKN-SSID**
  - is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

- **ZEMS-TKN-EVENTNUMBER** *(shared)*
  - is the event number. Its value is ZEMS-EVT-LOG-ACCESS (1000).
ZEMS-TKN-EMPHASIS *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions.
Its value here is TRUE.

ZEMS-TKN-CONSOLE-PRINT *(shared)*

is a standard EMS token. For more information, see Section 14, EMS Definitions.
Its value here is TRUE.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK *(shared)*

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR

is the EMS process type (value 1 for the primary collector; value 2 for an alternate collector).

ZEMS-TKN-BURST-EVT-NUM

is the event number of the bursting event.

ZEMS-TKN-BURST-SSID

is the subsystem ID of the bursting event.

ZEMS-TKN-BURST-SUBJ-CODE

is the token code of the first subject in the bursting event.

ZEMS-TKN-BURST-SUBJ-VALUE

is the value of the first subject of the bursting event.

ZEMS-TKN-BURST-SUBJ-SSID

is the subsystem ID of the first subject of the bursting event.

ZEMS-TKN-BURST-TIME-START

is the Julian timestamp for the burst start time.

ZEMS-TKN-BURST-TIME-END

is the Julian timestamp for the burst end time.
ZEMS-TKN-BURST-EVTS-DELETED

are occurrences of the bursting event that were not logged during the event burst.

ZEMS-TKN-BURST-END-REASON

is the reason for the event burst end. A value of ZEMS-VAL-BDS-DISABLED means that the burst ended because BDS was terminated by the operator. A value of ZEMS-VAL-NO-EVENTS means that the burst ended because no further events were detected during the last T2 time units.

Text Values

eventno

is from ZEMS-TKN-BURST-EVT-NUM.

ssid

is from ZEMS-TKN-BURST-SSID.

proc-desc

is from ZEMS-TKN-BURST-PROC-DESC.

count

is from ZEMS-TKN-BURST-EVTS-DELETED.

Cause. An event burst ended for one of these reasons:

- The bursting event did not occur for the time interval T2 specified in the BDS configuration (during which an event burst must occur at least once).
- BDS was disabled while bursting events were being suppressed, in which case an Event Burst Ended event is generated for every monitored event burst.

Effect. The next occurrence of the formerly bursting event is logged to disk.

Recovery. None. This is an informational message.

The timestamp stored in the BURST-TIME-START token matches the timestamp that is stored in the corresponding burst start event.
540: ZEMS-EVT-PLF-ERROR

Unconditional Tokens

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT (shared)</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-COLLECTOR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-FILTER-ERROR</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVT-EVTNUM</td>
<td>ZSPI-TYP-ENUM.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVT-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: FILTER ERROR filter-error ON FILTER filtername ON EVENT eventno,TIMESTAMP gentime, BY COLLECTOR proc-desc

Unconditional Tokens

ZSPI-TKN-SSID

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-LOG-ACCESS (1000).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions. Its value here is TRUE.

ZEMS-TKN-GENTIME, LOGTIME, CPU, PROC-DESC, PIN, NODENUM, and USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-SUBJECT-MARK (shared)  
is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR  
is the EMS process type (value 1 for the primary collector and value 2 for an alternate collector).

ZEMS-TKN-FILTER-ERROR  
is the filter evaluation error. The actual error code denotes a specific problem in interpreting the filter (most likely a compiled filter). This error might indicate an EMF filter compiler problem. For additional support, contact your service provider.

ZEMS-TKN-FILTERNAME  
is the name of the compiled filter or filter table in which the error occurred. This is not the filter’s file name.

ZEMS-TKN-EVT-EVTNUM  
is the event number from the event where the error occurred.

ZEMS-TKN-EVT-GENTIME  
is the generation timestamp from the event where the error occurred.

Text Values

filter-error  
is from ZEMS-TKN-FILTER-ERROR

filtername  
is from ZEMS-TKN-FILTERNAME

eventno  
is from ZEMS-TKN-EVT-EVTNUM

gentime  
is from ZEMS-TKN-EVT-GENTIME

proc-desc  
is from ZEMS-TKN-BURST-PROC-DESC

Cause. A filter evaluation error occurred when a filter was applied to a specific event.
Effect. The collector deletes the filter and generates this event. The event is examined by any subsequent filters in the collector and logged if the event passes all the filters.

Recovery. This condition can be caused by a corrupted filter (most likely) or event. If neither the filter nor event appears to be corrupt, contact your service provider, who might request that you provide a magnetic tape copy of:

- The number and description of this event message
- The log file that contains the event message
- The collector filter identified in the event message text
- If possible, a copy of the EMSCINFO collector-name, DETAIL display generated at the time the event message was received.

541: ZEMS-EVT-PLF-RELOAD-ERROR

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>token-type ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENITIVE</td>
<td>token-type ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>token-type ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>token-type ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>token-type ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>token-type ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>token-type ZSPI-TYP-SSCTL.</td>
</tr>
</tbody>
</table>

Event-Message Text
EMS: FILTER filtername COULD NOT BE RE-ADDED, BY COLLECTOR collector-name

Unconditional Tokens

ZSPI-TKN-SSID (shared)

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-PLF-RELOAD-ERROR (541).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions.
ZEMS-TKN-CONSOLE-PRINT (shared) is a standard EMS token. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared) is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

ZEMS-TKN-COLLECTOR (nonshared) specifies the collector type. This token has one of two values: ZEMS-SUBJ-PCOLL (or 1) for the primary collector or ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-FILTERNAME (nonshared) is the name of the compiled filter or filter table where the error occurred.

**Text Values**

*filtername* is the name of the filter that could not be added.

*collectorname* comes from ZEMS-TKN-PROC-DESC.

**Cause.** The event is generated when:

- At least one filter is loaded into the alternate collector.
- The filter file (filter object for burst filters or filter tables) is deleted.
- The primary CPU goes down.
- The backup process is stopped.
- The backup CPU goes down and is reloaded.

If the backup becomes the primary and attempts to reload the deleted filter file, it does not find it, and the message is displayed.

**Effect.** None.

**Recovery.** If you want to filter the events using the deleted filter, add the filter to the alternate collector.
Unconditional Tokens

ZSPI-TKN-SSID (shared)

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-PLF-RELOAD-ERROR (542).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see *Section 14, EMS Definitions*.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see *Section 14, EMS Definitions*.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see *Section 14, EMS Definitions*.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

EMS: LOG FILE *filename* IS NOT PURGED BY COLLECTOR *proc-desc*,

**ROTATEFILES** OPTION
ZEMS-TKN-COLLECTOR (nonshared) specifies the collector type. This token has one of two values: ZEMS-SUBJ-PCOLL (or 1) for the primary collector or ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-PURGEDLOGFILE (nonshared) is the purged log file.

Text Values

filename is the name of the unpurged log file.

proc-desc is the system name of the EMS collector process.

Cause. The collector did not purge the log files with the ROTATEFILE option set.

Effect. Extra files are generated and can be manually purged.

Recovery. Informational message; no corrective action is necessary.
Unconditional Tokens

**ZSPI-TKN-SSID** (shared)

is the subsystem ID for EMS, as described in the *SPI Programming Manual*.

**ZEMS-TKN-EVENTNUMBER** (shared)

is the event number. Its value is ZEMS-EVT-PLF-RELOAD-ERROR (543).

**ZEMS-TKN-EMPHASIS** (shared)

is a standard EMS token. For more information, see [Section 14, EMS Definitions](#).

**ZEMS-TKN-CONSOLE-PRINT** (shared)

is a standard EMS token. For more information, see [Section 14, EMS Definitions](#).

**ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID**

are standard EMS tokens. For more information, see [Section 14, EMS Definitions](#).

**ZEMS-TKN-SUBJECT-MARK** (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:

---

**Event-Message Text**

EMS: BURST END DETECTED: count1 OCCURANCES OF EVENT NO. eventnumber WERE NOT LOGGED AFTER Date1 time1

---

**Unconditional Tokens**

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-ZLOGFILTERDELETED</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-ZLOGEVENT</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-ZLOGFILTERSTART</td>
<td>ZSPI-TYP-TIMESTAMP</td>
</tr>
</tbody>
</table>
ZEMS-TKN-COLLECTOR (nonshared)  
specifies the collector type. This token has one value: ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-ZLOGFILTERDELETED (nonshared)  
is the number of events that are deleted.

ZEMS-TKN-ZLOGEVENT (nonshared)  
is the filter event number.

ZEMS-TKN-ZLOGFILTERSTART (nonshared)  
is the filter start date.

ZEMS-TKN-ZLOGFILTERDELETED (nonshared)  
is the number of events that are deleted.

Text Values

\( \text{count1} \)

determines the number of suppressed occurrences.

\( \text{eventnumber} \)

determines the suppressed event.

\( \text{Date1} \)

is the date the burst started.

\( \text{time1} \)

is the time the burst started.

**Cause.** EMSACOLL generates this event when it runs in TMDS mode ($Zlog) and a burst end event was detected.

**Effect.** None.

**Recovery.** Informational message; no corrective action is necessary.
544: ZEMS-EVT-ZLOGLOSTEVENT

<table>
<thead>
<tr>
<th>Unconditional Tokens</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-SSID (shared)</td>
<td>ZSPI-TYP-SSID.</td>
</tr>
<tr>
<td>ZEMS-TKN-EVENTNUMBER (shared)</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-EMPHASIS (shared)</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-CONSOLE-PRINT (shared)</td>
<td>ZSPI-TYP-BOOLEAN.</td>
</tr>
<tr>
<td>ZEMS-TKN-GENTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGTIME</td>
<td>ZSPI-TYP-TIMESTAMP.</td>
</tr>
<tr>
<td>ZEMS-TKN-CPU</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-PROC-DESC</td>
<td>ZSPI-TYP-STRING.</td>
</tr>
<tr>
<td>ZEMS-TKN-PIN</td>
<td>ZSPI-TYP-INT.</td>
</tr>
<tr>
<td>ZEMS-TKN-NODENUM</td>
<td>ZSPI-TYP-INT2.</td>
</tr>
<tr>
<td>ZEMS-TKN-USERID</td>
<td>ZSPI-TYP-UINT.</td>
</tr>
<tr>
<td>ZEMS-TKN-SUBJECT-MARK (shared)</td>
<td>ZSPI-TYP-SSCTL.</td>
</tr>
<tr>
<td>ZEMS-TKN-ZLOGREASON</td>
<td>ZSPI-TYP-CHAR24</td>
</tr>
<tr>
<td>ZEMS-TKN-ZLOGNUMBERLOSS</td>
<td>ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-ZLOGLOSSCPU</td>
<td>ZSPI-TYP-INT</td>
</tr>
</tbody>
</table>

Event-Message Text

EMS: reason count1 NO. OF EVENTS LOST IN CPU cpunumber

Unconditional Tokens

ZSPI-TKN-SSID (shared)

is the subsystem ID for EMS, as described in the SPI Programming Manual.

ZEMS-TKN-EVENTNUMBER (shared)

is the event number. Its value is ZEMS-EVT-PLF-RELOAD-ERROR (544).

ZEMS-TKN-EMPHASIS (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-CONSOLE-PRINT (shared)

is a standard EMS token. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-GENTIME, -LOGTIME, -CPU, -PROC-DESC, -PIN, -NODENUM, and -USERID

are standard EMS tokens. For more information, see Section 14, EMS Definitions.

ZEMS-TKN-SUBJECT-MARK (shared)

is the standard EMS token that immediately precedes each subject token. This event message has one subject token:
ZEMS-TKN-COLLECTOR (nonshared) specifies the collector type. This token has one value: ZEMS-SUBJ-ACOLL (or 2) for the alternate collector.

ZEMS-TKN-ZLOGREASON (nonshared) states the reason for the loss of the event messages.

ZEMS-TKN-ZLOGNUMBERLOSS (nonshared) is the number of events lost.

ZEMS-TKN-ZLOGLOSSCPU (nonshared) is the CPU that lost the events.

**Text Values**

*reason* states why the events were not logged.

*count1* is the number of lost events.

*cpunumber* is the CPU number.

**Cause.** This event is generated by EMSACOLL when it runs in TMDS mode ($Zlog) and some events are lost due to one of the following:

- File system did not have PFS space
- Logger was too slow.
- Logger was not reading for more than 30 minutes.

**Effect.** Any messages sent to the alternate collector in TMDS mode are not logged.

**Recovery.** Check the event log to determine the reason for lost events and try to resolve the problem.
This section lists error codes and warning codes returned by the distributor:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token and Data Type Definitions</td>
<td>21-1</td>
</tr>
<tr>
<td>Distributor Warning Codes</td>
<td>21-4</td>
</tr>
<tr>
<td>Distributor Error Codes</td>
<td>21-6</td>
</tr>
</tbody>
</table>

Following the code of each error or warning is a box with the name and type of tokens that constitute the related error list. Following the box a description of the error’s cause, effect, and recovery actions that can be taken.

All distributor response messages contain ZSPI-TKN-RETCODE and can include one or more error lists even when the command is successful (ZSPI-TKN-ERROR value is zero) if something unusual occurred. Each error list contains a ZSPI-TKN-ERROR token and often contains other tokens as well, depending on the situation.

For more information about distributor error lists, see Section 17, Distributor Commands and Responses.

**Token and Data Type Definitions**

Distributor error lists include some tokens and values defined by SPI (prefixed by ZSPI) as well as those defined by EMS (prefixed by ZEMS). These tokens and values are defined here, rather than with each error list, because most of them occur in several error lists.

**Note.** To express SPI token types such as ZSPI-TYP-UINT, the following listings omit ZSPI-TYP- and place the remainder in italicized, lowercase letters, as in (type \textit{uint}).

**SPI Token Codes**

These SPI tokens (tokens with prefix ZSPI) occur in distributor error lists:

- **ZSPI-TKN-COMMAND** \texttt{(type enum)}
  - specifies the distributor command.

- **ZSPI-TKN-CONTEXT** \texttt{(type bytestring)}
  - is the token saved by the distributor to record its position in the log file.

- **ZSPI-TKN-ENDLIST** \texttt{(type ssctl)}
  - is the SPI token that ends a list—an error list in this section. The DDL for ZSPI-TKN-ENDLIST is:
    
    \texttt{DEF IS ZSPI-DDL-INT4.}
ZSPI-TKN-ERRLIST (type list) is the SPI token that begins an error list.

ZSPI-TKN-ERROR (type error) is the standard SPI error token. The DDL for ZSPI-TKN-ERROR is:

```plaintext
DEFINITION ZSPI-DDL-ERROR.
  02 Z-SSID     TYPE ZSPI-DDL-SSID.
  02 Z-ERROR    TYPE ZSPI-DDL-ENUM.
END
```

ZSPI-TKN-PARM-ERR is the standard SPI error token. The DDL for ZSPI-TKN-PARM-ERR is:

```plaintext
DEF ZSPI-DDL-PARM-ERR.
  02 Z-TOKENCODE TYPE ZSPI-DDL-TOKENCODE.
  02 Z-INDEX     TYPE ZSPI-DDL-UINT.
  02 Z-OFFSET    TYPE ZSPI-DDL-UINT.
END
```

The ZSPI-TKN-PARM-ERR token gives the token-code and index (but not the value) of a parameter token that is incorrect.

If the value was incorrect, the error list also includes the token and its erroneous value.

ZSPI-TKN-PROC-ERR (type enum) specifies a procedure associated with the error. (The exact association depends on the particular error and is described with the related ZSPI-TKN-ERROR value.) ZSPI-TKN-PROC-ERR can have these values:

- ZEMS-VAL-FILTER-EVAL 20
- ZEMS-VAL-ZFILWAITIO 21
- ZEMS-VAL-ZFILOPEN 22
- ZEMS-VAL-ZFILOPPOSITION 23
- ZEMS-VAL-ZFILREAD 24
- ZEMS-VAL-ZFILWRITE 25
- ZEMS-VAL-ZFILWRITEREAD 26
- ZEMS-VAL-FILTER-READ 27
- ZEMS-VAL-FILTER-VERIFY 28
- ZEMS-VAL-EMSADDBUFFER 29
- ZEMS-VAL-EMSADDSUBJECT 30
- ZEMS-VAL-EMSADDTOKENS 31
- ZEMS-VAL-EMSGET 32
- ZEMS-VAL-EMSINIT 33
- ZEMS-VAL-EMSSEND 34
- ZEMS-VAL-CHECKOPEN 35
- ZEMS-VAL-NEWPROCESS 36
- ZEMS-VAL-CHECKPOINT 37
- ZEMS-VAL-CHECKMONITOR 38
- ZEMS-VAL-FINDDEV 39
EMS Token Codes

These EMS tokens occur in distributor error lists:

ZEMS-TKN-BLOCKLENGTH (type int; nonshared)

is the block length of a log file.

ZEMS-TKN-COLNAME (type fname; nonshared)

if specified, is the name of a collector associated with the distributor reporting the error; otherwise it is set to blanks. See ZEMS-TKN-COLNAME-ENUM.

ZEMS-TKN-COLNAME-ENUM (type enum; nonshared)

indicates the presence or absence of a collector name in ZEMS-TKN-COLNAME:

ZEMS-VAL-COLNAME-PRESENT 0
ZEMS-VAL-COLNAME-NOTPRESENT 1

ZEMS-TKN-CONNECT-SRC-COLL (type fname; nonshared)

names a collector whose log files are to serve as a source of event messages for this distributor.

ZEMS-TKN-DEVICE-TYPE (type int; nonshared)

is the device type of a device associated with the distributor reporting the error.

ZEMS-TKN-DEVTYPE-ENUM (type enum; nonshared)

uses one of these values to indicate a device-type problem:

ZEMS-VAL-FILECODE-BAD 0
ZEMS-VAL-BLOCKLENGTH-BAD 1
ZEMS-VAL-DEVICE-TYPE-BAD 2
ZEMS-VAL-LOGNAME-BAD 3
ZEMS-VAL-VERSION-INCOMPATIBLE 4
ZEMS-VAL-DUP-SOURCE-TARGET 5

ZEMS-TKN-DISCONNECT-SRC-COLL (type fname; nonshared)

is the name of a collector to be disconnected (from this distributor) as a source of event messages.

ZEMS-TKN-FAILFILENAME (type fname; nonshared)

is the file name of a bad log file.

ZEMS-TKN-FILECODE (type int; nonshared)

is the file code of a log file.
ZEMS-TKN-FILTER-ERROR (type nonshared)
   is private to HP.

ZEMS-TKN-LOGNAME (type fname; nonshared)
   is the file name of the log file in use when the error occurred.

ZEMS-TKN-RECORD-ADDRESS (type int2; nonshared)
   is an entry-sequenced file address. Suppose blknum is the block number
   (numbering from zero), blklen is the block length (in bytes), and recnum is the
   record number (within the block). Then record-address is computed as follows:
   \[( \text{blknum} \times \text{blklen} ) + \text{recnum} \]

ZEMS-TKN-ZFILERR (type uint; nonshared)
   is the error code associated with some value of ZSPI-TKN-ERROR for the
   distributor.

**Distributor Warning Codes**

These distributor codes represent warnings:

**501: ZEMS-WRN-EOF**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** You sent the ZEMS-TKN-EOFSTOP token, and the distributor reached an
end-of-file indication before another event message passed the filter.

**Effect.** No more event messages are distributed.

**Recovery.** None.

**502: ZEMS-WRN-TOO-EARLY**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A log file position command was issued, but the time specified is earlier than
the first event in the oldest log.
Effect. The distributor starts event retrieval with the first event in the oldest log in the log chain attached to a collector.

Recovery. Ignore this warning, or specify a time that falls within the range of the log chain.

503: ZEMS-WRN-TOO-LATE

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Cause. A log file position command was issued, but the time specified is later than the last event in the most recent log.

Effect. The distributor waits for the next event to arrive, switching to event monitoring mode. If the event source is a log file, an end-of-file error is reported.

Recovery. Ignore this warning, or specify a time that falls within the range of the log chain.

504: ZEMS-WRN-STARTUP-OK

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-PROGRAMFILE</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-STARTUP-LOGTIME</td>
<td>token-type ZSPI-TYP-TIMESTAMP</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-ERROR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-CPU</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-NEWPROCESS-PRIORITY</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

ZSPI-TKN-PROC-ERR

can have the value:

| ZEMS-VAL-NEWPROCESS        | 36 |

Cause. A routing distributor attempted to launch an application and failed. After some retries, the process creation succeeds.

Effect. Routing of event messages can proceed.

Recovery. None.
Distributor Error Codes

These distributor errors vary in seriousness, depending on the context in which they occur:

1001: ZEMS-ERR-VERSION

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor does not support the version you supplied in the command message.

**Effect.** The command is not executed.

**Recovery.** Supply a recent version that the distributor supports in your command.

1002: ZEMS-ERR-INV-CMD

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor received an invalid command message.

**Effect.** The command is not executed.

**Recovery.** Check the syntax and parameters specified in your command.

1003: ZEMS-ERR-INV-SSID

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID-ERR</td>
<td>token-type ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor received a command message with an invalid subsystem ID (SSID) from a command-message buffer not owned by EMS.

**Effect.** The command is not executed.

**Recovery.** Specify a valid subsystem ID (SSID) in your command.
1004: ZEMS-ERR-INV-TKN

Cause. The distributor received a command message with an unnecessary or unrecognized token.

Effect. The command is not executed.

Recovery. Check that you specify only needed and recognizable tokens in your command.

1005: ZEMS-ERR-INV-VALUE

Cause. The distributor received a command message with an invalid token.

Effect. The command is not executed.

Recovery. Make sure you provide a valid token value in your command.

1006: ZEMS-ERR-DUP-TKN

Cause. The distributor received a command message with duplicate tokens. This error occurs, for example, when a filter object is added that already exists in the distributor.

Effect. The object is not added.

Recovery. None. If an updated version of the object is added, delete the old object first.
1007: ZEMS-ERR-MODE-CONFLICT

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor received a command message that is inappropriate for this type of distributor.

**Effect.** The command is not executed.

**Recovery.** Make sure your command is appropriate for this type of distributor.

1008: ZEMS-ERR-INV-OBJECT

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor received a command message with an invalid object type.

**Effect.** The command is not executed.

**Recovery.** Make sure to specify a valid object type.

1014: ZEMS-ERR-INV-OP

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor received a command message that is not allowed in this context.

**Effect.** The command is not executed.

**Recovery.** Make sure you issue commands that are allowed in this context.

1015: ZEMS-ERR-REQ-TKN

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor received a command message that lacks a required token.

**Effect.** The command is not executed.
Recovery. Make sure you provide all tokens required in the specified command.

1016: ZEMS-ERR-INV-HEAEDERTYPE

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Cause. The distributor received a command message with an invalid header type.

Effect. The command is not executed.

Recovery. Use a valid header type in the command.

1018: ZEMS-ERR-COLL-ACCESS

ZSPI-TKN-PROC-ERR

can have the values:

<table>
<thead>
<tr>
<th>Value Description</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-VAL-ZFILOPEN</td>
<td>22</td>
</tr>
<tr>
<td>ZEMS-VAL-ZFILWRITEREAD</td>
<td>26</td>
</tr>
</tbody>
</table>

Cause. The distributor tries to access a collector—often to get an event message from one of the collector’s log files—and finds it cannot.

Effect. The distributor disconnects the failing collector but continues to run. An application can direct the distributor to reconnect the same collector after the problem is corrected. Automated source recovery is available.

Recovery. If this error was caused by a security violation, restart the collector, then reconnect it. Otherwise, contact your service provider.

1019: ZEMS-ERR-FLT-FORM

ZEMS-TKN-FILTER-ERROR

can have the values:

<table>
<thead>
<tr>
<th>Value Description</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-FILTER-ERROR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>
Distributor Errors

1020: ZEMS-ERR-FLT-LOAD

-1 Filter too big to fit into supplied buffer.
-2 Filter format invalid.
-3 Version incompatibility.

Cause. The distributor received a command message for processing a filter that is not in the expected format. Either the filter is too big to fit in the supplied buffer, the filter format is invalid, or a version incompatibility was detected. The filter possibly was compiled with a version newer than the version supported by the distributor.

Effect. The filter is not added. If the specified filter is a burst filter or filter table edit file, it cannot be converted to an object form.

Recovery. Correct the filter source, or use an appropriate compiler.

1020: ZEMS-ERR-FLT-LOAD

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

ZSPI-TKN-PROC-ERR

can have the values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-VAL-ZFILOPEN</td>
<td>22</td>
</tr>
<tr>
<td>ZEMS-VAL-ZFILREAD</td>
<td>24</td>
</tr>
</tbody>
</table>

Cause. The distributor could not load the filter because an I/O problem occurred, an incorrect file name was given, or the file was not the correct type (845 for filter objects, 101 for filter table source files).

Effect. The filter is not added.

Recovery. Specify a correct name, then generate a correct filter file.

1022: ZEMS-ERR-REQ-PARAM

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Cause. One or more required filter parameters are missing. Either a filter was added that requires parameters, but no (or insufficient) parameters were submitted; or a filter was altered, but not all required parameters were submitted.

Effect. The filter is not added, or the submitted set or parameters is not applied.

Recovery. Provide all required parameters.
1024: ZEMS-ERR-HIST-MODE

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor cannot perform this operation in log file mode. An attempt was made to add or delete a source collector while the distributor is currently connected to a log file source.

**Effect.** The operation is not performed.

**Recovery.** Delete the log file source first before adding a collector source.

1025: ZEMS-ERR-MAX-COLLECTOR

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor cannot connect another collector because the number of collectors is already at its maximum (10).

**Effect.** The collector is not connected.

**Recovery.** Disconnect a connected collector to accommodate the specified collector.

1026: ZEMS-ERR-COLLECTOR-EXISTS

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZEMS-TKN-CONNECT-SRC-COLL</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

** Cause.** The specified collector is already associated with the distributor and cannot be connected.

**Effect.** None.

**Recovery.** None.
1027: ZEMS-ERR-COLL-NOT-FOUND

<table>
<thead>
<tr>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
</tr>
<tr>
<td>ZEMS-DISCONNECT-SRC-COLL</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
</tr>
</tbody>
</table>

ZSPI-TKN-TERM-ERR

The token code and index (but not the value) of the ZEMS-TKN-DISCONNECT-SRC-COLL token, which caused the error. The distributor includes ZEMS-TKN-DISCONNECT-SRC-COLL (value included) to provide the collector name.

**Cause.** The collector is not associated with the distributor and cannot be disconnected.

**Effect.** No collector connections are altered.

**Recovery.** Make sure you specify a collector associated with the specified distributor.

1031: ZEMS-ERR-LOG-ACCESS

<table>
<thead>
<tr>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
</tr>
</tbody>
</table>

ZSPI-TKN-PROC-ERR

ZEMS-VAL-ZFILOPEN 22
ZEMS-VAL-ZFILPOSITION 23
ZEMS-VAL-ZFILREAD 24

**Cause.** The distributor is using a collector’s log files as its event message source. While proceeding from one file to the next, the distributor cannot access a log file. Either a security violation occurred, or a log file was purged.

**Effect.** The distributor closes the old log file and tries to access the log file that follows the missing log file in the file series. If no other log file exists, the distributor disconnects the collector and generates event ZEMS-EVT-COLLECTOR-DISCONNECT.

**Recovery.** If a security violation occurred or a log file was purged, correct the problem. Otherwise, contact your service provider. When the log file is available again, rerun any applications that depend on it.
### 1032: ZEMS-ERR-EOF

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor is in log file mode, and you requested a log file position that is past the end-of-file position on the log file. The distributor reached an end-of-file while using the specified log file as its event message source.

**Effect.** If an application connected the log file, the distributor continues to run. If the log file was connected through a distributor startup option, the distributor stops.

**Recovery.** When the distributor is finished with the file, the application is notified that the distributor is ready for another event message source.

### 1033: ZEMS-ERR-FORWARD-SEARCH

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

ZEMS-TKN-COLNAME and ZEMS-TKN-LOGNAME give the names of the associated collector and the log file last accessed.

**Cause.** The distributor cannot find the next file to be searched after a positioning by time command. Either a legitimate file is missing, or the distributor’s first event message after the cold-load event message contains an erroneous file name as its previous log name. The log file and the collector last accessed are given.

**Effect.** The positioning command terminates. The current log position is not changed.

**Recovery.** Determine why the file is not accessible. If the file no longer exists, position past it into the next log.

### 1035: ZEMS-ERR-MAX-DEST

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A printing distributor received more than the maximum (10) number of add (connect) TEXTOUT destinations.

**Effect.** The specified TEXTOUT destination is not added.
**Distributor Errors**

**1036: ZEMS-ERR-DEST-ACCESS**

**Recovery.** Remove another TEXTOUT destination first, or use another printing distributor.

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** Either a printing distributor cannot access a TEXTOUT destination, or a forwarding distributor cannot access a TARGET collector.

**Effect.** If there is one assigned destination, it continues to retry accessing. If there are multiple assigned destinations, it retries accessing once then, if it fails, it disconnects the destination.

**Recovery.** Check the destination to see why it might be inaccessible. If the destination was disconnected, reconnect it.

**1037: ZEMS-ERR-DEST-EXISTS**

**Recovery.** None.

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZEMS-TKN-ADD-TEXTOUT</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The user tried to add a TEXTOUT destination to the list of print destinations for this printing distributor that already exists there.

**Effect.** The specified TEXTOUT destination is not added because it is already present.

**Recovery.** None.
1038: ZEMS-ERR-DEST-NOT-FOUND

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZEMS-TKN-DELETE-TEXTOUT</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-PARM-ERR**

gives the token code and index (but not the value) of the token that caused the error (ZEMS-TKN-DELETE-TEXTOUT). The distributor includes the ZEMS-TKN-DELETE-TEXTOUT token in the error list to provide the erroneous destination (the token's value).

**Cause.** The TEXT destination is not in the list of print destinations for this printing distributor and cannot be deleted.

**Effect.** None.

**Recovery.** None.

1039: ZEMS-ERR-CONTEXT

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-CONTEXT</td>
<td>token-type ZSPI-TYP-BYTESTRING</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-CONTEXT**

includes the context token saved by the distributor.

**Cause.** The context submitted did not match the context saved in the distributor.

**Effect.** The application does not receive the requested event.

**Recovery.** Make sure the contexts are not being tampered with once they are created.

1041: ZEMS-ERR-ZSPI

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID-ERR</td>
<td>token-type ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-PROC-ERR**

can have these values for the ZEMS-ERR-ZSPI error:
Distributor Errors

1042: ZEMS-ERR-BAD-FILTER

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-FILTER-ERROR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-RECORD-ADDRESS</td>
<td>token-type ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZEMS-TKN-FILTERNAME</td>
<td>token-type ZEMS-TYP-CHAR30</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The distributor detected a SPI error while processing a command or control buffer.

**Effect.** The command is not executed.

**Recovery.** This is an internal error. Report it to your service provider.

1043: ZEMS-ERR-NO-POOL

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The pool space for additional resources is not available.

**Effect.** The event source or destination is not added.

**Recovery.** Remove event sources or destinations to free resources.

1044: ZEMS-ERR-NO-EVENT-SOURCE

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A GETEVENT or POSITION by logtime command has been issued, but the distributor has no event-message source collector or log file source.
Effect. The event cannot be retrieved, or positioning cannot be done.

Recovery. Add an event source.

### 1045: ZEMS-ERR-DEVTYPE

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-DEVTYPE-ENUM</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME-ENUM</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-FILECODE</td>
<td>token-type ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZEMS-TKN-BLOCKLENGTH</td>
<td>token-type ZSPI-TYP-INT</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

ZEMS-TKN-DEVTYPE-ENUM (nonshared)

indicates what failed. It can have the values:

- ZEMS-VAL-FILECODE-BAD 0
- ZEMS-VAL-BLOCKLENGTH-BAD 1
- ZEMS-VAL-DEVICE-TYPE-BAD 2
- ZEMS-VAL-LOGNAME-BAD 3
- ZEMS-VAL-VERSION-INCOMPATIBLE 4
- ZEMS-VAL-DUP-SOURCE-TARGET 5

Cause. The collector log file serving as the source of event messages has an incorrect file-name format, file type, or block length. You might have specified the wrong name for a log file or for a collector process. The log file connected to a collector might have been tampered with. ZEMS-VAL-DUP-SOURCE-TARGET means an inappropriate target collector is selected for a forwarding distributor: a collector that was already an event-message source for that distributor.

Effect. The associated collector or log file is disconnected.

Recovery. Check that you specified a correct EMS event-log file name. After the problem with the log or collector process is corrected, the source might be reconnected.

### 1046: ZEMS-ERR-COLL-PROTOCOL

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

ZSPI-TKN-PROC-ERR

can have the value:
1047: ZEMS-ERR-BAD-EVENT

**Cause.** The distributor cannot interpret the reply from a collector following a status request because the collector malfunctioned or aborted. This can occur when adding an event source collector or a process that is not a collector, or during event processing after end-of-file detection.

**Effect.** The distributor disconnects the failing collector but continues to run.

**Recovery.** Automatic source recovery is available through a special startup parameter.

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-TKN-COLNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-LOGNAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-RECORD-ADDRESS</td>
<td>token-type ZSPI-TYP-INT2</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-PROC-ERR** can have the following values:

<table>
<thead>
<tr>
<th>ZSPI-VAL-SSGETTKN</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-VAL-FILTER-EVAL</td>
<td>20</td>
</tr>
</tbody>
</table>

**Cause.** The distributor finds the event message that it read from the log is invalid.

**Effect.** The distributor continues to process event messages, skipping the invalid event. After five invalid messages are detected, event ZEMS-EVT-BAD-LOG is generated.

**Recovery.** Determine how the event message got corrupted and was not detected by the collector. An application might have generated the log, which HP does not recommend.

1049: ZEMS-ERR-BAD-LOG

**Cause.** The distributor declares the current log inaccessible after five consecutive bad event messages are encountered.

**Effect.** The distributor closes the file and attempts to access the next log file. If none exist, event ZEMS-EVT-COLLECTOR-DISCONNECT is generated, and the associated...
source collector is disconnected. The distributor continues to run. The application may
reconnect the source after the problem is identified and corrected.

**Recovery.** Try to determine the nature of the invalid events and where they came
from. An application might have generated the log file, which HP does not recommend.

### 1050: ZEMS-ERR-COLL-DISCONNECT

| ZSPI-TKN-ERRLIST | token-type ZSPI-TYP-LIST |
| ZEMS-TKN-COLNAME-ENUM | token-type ZSPI-TYP-ENUM |
| ZEMS-TKN-COLNAME | token-type ZSPI-TYP-FNAME |
| ZSPI-TKN-ERROR | token-type ZSPI-TYP-ERROR |
| ZSPI-TKN-ENDLIST | token-type ZSPI-TYP-SSCTL |

**Cause.** Because of a hardware, software, or operations failure, the distributor can
access none of the log files of the collector it is using as an event messages source.

**Effect.** The distributor disconnects the specified collector but continues running. An
application can direct the distributor to reconnect the original collector after the problem
is corrected. Automatic source recovery is available through a special startup
parameter.

**Recovery.** Refer to the event message that was generated relating to this error. It
indicates the underlying reason for the failure. If there is no such event message,
contact your service provider.

### 1051: ZEMS-ERR-FILES-LOST

| ZSPI-TKN-ERRLIST | token-type ZSPI-TYP-LIST |
| ZEMS-TKN-COLNAME | token-type ZSPI-TYP-FNAME |
| ZEMS-TKN-LASTLOGFILE | token-type ZSPI-TYP-FNAME |
| ZEMS-TKN-NEWLOGFILE | token-type ZSPI-TYP-FNAME |
| ZEMS-TKN-RECOVERY-ENUM | token-type ZSPI-TYP-ENUM |
| ZEMS-TKN-FAIL-REASON | token-type ZSPI-TYP-ENUM |
| ZSPI-TKN-ERROR | token-type ZSPI-TYP-ERROR |
| ZSPI-TKN-ENDLIST | token-type ZSPI-TYP-SSCTL |

**ZEMS-TKN-NEWLOGFILE**

is omitted if no new log file can be found. In that case, ZEMS-TKN-RECOVERY-ENUM is set to FALSE (no recovery). For details, see distributor event 1018: ZEMS-EVT-FILES-LOST on page 18-46.

The ZEMS-TKN-COLNAME token gives the name of the collector associated with
the log file.

**Cause.** The distributor cannot find or access the log file that should be next.

**Effect.** The distributor proceeds to the next file in the queue of log file names.

**Recovery.** Check that the log file is available and accessible and, then try again.
1052: ZEMS-ERR-STATUS-ONLY

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** An application issued a CONTROL command, or an ADD, ALTER, GETEVENT, or DELETE command to a consumer, printing, or forwarding distributor for which it did not have permission or is in use by another user. Only info or status commands are allowed for a secondary opener. Exceptions are the filter add/delete commands, which are allowed for a super-group user or a user with the same access ID as the distributor’s ID.

**Effect.** Access for this command is denied.

**Recovery.** Log on as the proper user to gain filter control access.

1059: ZEMS-ERR-MAXFLT

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** An ADD FILTER command was issued exceeding the maximum number of filters (10).

**Effect.** The filter is not added.

**Recovery.** Remove filters to accommodate new filters.

1060: ZEMS-ERR-FLT-ALLOC

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A resource for a filter could not be allocated, such as filter buffer or filter parameter buffer. The failing filter name is displayed. An ADD FILTER command was issued, but the combined resources for event source, destination, and filter parameter buffers exceed the buffer pool size.

**Effect.** The filter is not added.

**Recovery.** Remove sources, destinations, or filters to allow room in the buffer pool.
1061: ZEMS-ERR-DIST-ALLOC

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** An ADD FILTER command was issued to allocate a routing distributor destination, such as a print buffer, but the combined resources for event source, destination, and filter parameter buffers exceed the available buffer pool size.

**Effect.** The filter is not added.

**Recovery.** Remove sources, destinations, or filters to allow room in the buffer pool.

1062: ZEMS-ERR-INV-PROFILE

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** FORMAT ON was specified, and the destination is a collector or a process that is set up for launching. The ADD FILTER command was issued.

**Effect.** The filter is not added.

**Recovery.** Set FORMAT to OFF.

1063: ZEMS-ERR-DEST-CONFLICT

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A conflict between a destination profile and a runtime TEXTOUT specification was detected when an ADD FILTER command was issued. A destination profile was specified in the filter, and TEXTOUT was also given as a runtime parameter.

**Effect.** The filter is not added.

**Recovery.** Start the distributor without a TEXTOUT specification or delete the destination profile in the filter.
1064: ZEMS-ERR-STARTUP-FAILED

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROGRAMFILE</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-PROC-ERR**

can have the values:

- ZEMS-VAL-NEWPROCESS
- ZEMS-VAL-ZFILOPEN
- ZEMS-VAL-ZFILWRITE

**Cause.** An attempt to launch a destination process was unsuccessful. Either a process create failure, an open error, or an error occurred during the write of the startup message.

**Effect.** Only the initial launching error is reported, not subsequent retry errors. If the startup eventually succeeds, ZEMS-WRN-STARTUP-OK is issued.

**Recovery.** Determine from the event ZEMS-EVT-STARTUP-FAILED what value the ZEMS-TKN-NEWPROCESS-ERROR has.

1065: ZEMS-ERR-WRITE-FAILED

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROGRAMFILE</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A timeout occurred after an attempt was made to write an event to a destination process. The error could be the result of unsuccessful retries, or the recipient did not acknowledge the request by reading the $RECEIVE file. The cause might be an abended destination process.

**Effect.** The error message is repeated every 24 hours, and a count of the failed write operations is attached.

**Recovery.** Determine the nature of the problem and correct it. Monitor the event messages.
This section lists the error codes returned by the primary and alternate EMS collectors in their three categories:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCOM- Errors (Over Extended SPI Interface)</td>
<td>22-2</td>
</tr>
<tr>
<td>ZEMS- Errors (Over Extended SPI Interface)</td>
<td>22-10</td>
</tr>
<tr>
<td>ZEMS- Errors (Over Basic SPI Interface)</td>
<td>22-19</td>
</tr>
</tbody>
</table>

These categories are based on the type of SPI-compliant collector commands to which the collectors send the errors in response.

For descriptions of all these collector commands, see Section 19, Collector Commands and Responses.

There is some commonality among the error names listed for ZEMS- errors generated by the extended SPI-compliant interface and ZEMS- errors generated over the basic SPI interface. However, the error list tokens generated are different. Accordingly, the ZEMS- errors are described twice, once for each SPI interface.

Following the code of each error or warning is a box with the name and type of tokens that constitute the related error list. Following the box is a description of the error’s cause, effect, and recovery actions that can be taken.

All collector command responses contain the command header and the ZSPI-TKN-RETCODE token. They contain an error list if the ZSPI-TKN-RETCODE value is nonzero.

Use the SSGET and SSGETTKN procedures to get the ZSPI-TKN-RETCODE or error-list tokens. These procedures also provide special operations that enable you to extract information from the command header. For information about SSGET and SSGETTKN, see the SPI Programming Manual and the SPI Common Extensions Manual.
ZCOM- Errors (Over Extended SPI Interface)

These errors can be returned by a collector through the SPI common extension interface in response to extended SPI-compliant (ZCOM-) collector commands documented in Section 19, Collector Commands and Responses.

Token and Data Type Definitions for ZCOM- Errors

Collector error lists returned in response to extended SPI-compliant ZCOM- commands include tokens and values defined by SPI (prefixed by ZSPI). These tokens and values are defined here, rather than with each error list, because most of them occur in several error lists.

Note. To express SPI token types such as ZSPI-TYP-UINT, the following listings omit ZSPI-TYP- and place the remainder in italicized, lowercase letters, as in (type uint).

SPI Token Codes

These SPI tokens (with prefix ZSPI) occur in collector error lists:

ZSPI-TKN-ERRLIST (type list)

is the SPI token that begins an error list.

ZSPI-TKN-ERROR (type error)

is the standard SPI error token. The DDL for ZSPI-TKN-ERROR is:

```plaintext
DEFINITION ZSPI-DDL-ERROR.
   02 Z-SSID      TYPE ZSPI-DDL-SSID.
   02 Z-ERROR     TYPE ZSPI-DDL-ENUM.
END
```

ZCOM-TKN-OBJNAME (type string)

is the SPI token that returns the name of the object specified in the extended SPI-compliant ZCOM- command and is included if it is relevant to the analysis of the error condition.

ZCOM-TKN-OBJTYPE (type enum)

is the SPI token that returns the object type (ZCOM-OBJ-objtype) specified in the extended SPI-compliant ZCOM- command and is included if relevant to the analysis of the error condition.

ZCOM-TKN-OBJSTATE (type enum)

contains the summary reported by the object to which the command could not be applied.
**ZSPI-TKN-PARM-ERR**

is the standard SPI error token and identifies the token used to establish context. The ZSPI-TKN-PARM-ERR token gives the token code and index (but not the value) of a command parameter token used in error.

**ZSPI-TKN-PROC-Err (type enum)**

specifies a procedure associated with the error. (The exact association depends on the particular error and is described with the related ZSPI-TKN-ERROR value.) ZSPI-TKN-PROC-ERR can have these values for EMS collectors using the extended or basic SPI interface:

- **ZEMS-VAL-FILTER-EVAL**  20
- **ZEMS-VAL-ZFILAWAITIO**  21
- **ZEMS-VAL-ZFILOPEN**  22
- **ZEMS-VAL-ZFILPOSITION**  23
- **ZEMS-VAL-ZFILREAD**  24
- **ZEMS-VAL-ZFILWRITE**  25
- **ZEMS-VAL-ZFILWRITEREAD**  26
- **ZEMS-VAL-FILTER-READ**  27
- **ZEMS-VAL-FILTER-VERIFY**  28
- **ZEMS-VAL-EMSADDBUFFER**  29
- **ZEMS-VAL-EMSADDSUBJECT**  30
- **ZEMS-VAL-EMSADDTOKENS**  31
- **ZEMS-VAL-EMSGET**  32
- **ZEMS-VAL-EMSINIT**  33
- **ZEMS-VAL-EMSSEND**  34
- **ZEMS-VAL-CHECKOPEN**  35
- **ZEMS-VAL-NEWPROCESS**  36
- **ZEMS-VAL-CHECKPOINT**  37
- **ZEMS-VAL-CHECKMONITOR**  38
- **ZEMS-VAL-FINDDEV**  39
- **ZEMS-VAL-ALLOCATESEGMENT**  40
- **ZEMS-VAL-FILTER-LOAD**  41
- **ZEMS-VAL-FILTER-PARAM**  42
- **ZEMS-VAL-MSG-READDATA**  43
- **ZEMS-VAL-FILNM-TO-OFILNM**  44
- **ZEMS-VAL-OFILNM-TO-FILNM**  45
- **ZEMS-VAL-FILNM-RESOLVE**  46
- **ZEMS-VAL-ADD-FILTER**  47

**ZSPI-TKN-SSID-ERR (type ssid)**

is the subsystem ID used in error.

**ZSPI-TKN-ERRLIST (type list)**

marks an embedded error list that contains this SPI error information:

**ZSPI-TKN-ERROR.Z-SSID (type ssid)**

contains the SPI SSID.
ZSPI-TKN-ERROR.Z-ERROR (type enum)
contains the SPI error number (ZSPI-ERROR-error).

ZSPI-TKN-ERROR.PROC-ERROR (type int)
identifies the SPI procedure that encountered the error (ZSPI-VAL-PROC).

ZSPI-TKN-ENDLIST (type ssctl)
marks the end of the nested file-system error list.

ZSPI-TKN-ENDLIST (type ssctl)
is the SPI token that ends a list—an error list in this section.

-3: ZCOM-ERR-CMD-INV-IN-SUMSTATE

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJSTATE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The command cannot be applied to the object in its current summary state. You either attempted to activate SUPPRESS ON or change a SUPPRESS parameter when a burst filter was installed, or you attempted to change the S or L suppress parameters or add a burst filter while BDS was enabled.

**Effect.** The desired action does not take effect.

**Recovery.** In the first case, remove the burst filter before using the SUPPRESS parameter, or alter the burst filter then replace it. In the latter case, add a burst filter, or disable BDS and change the S or L parameter.

-4: ZCOM-ERR-CMD-MISMATCH

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A mismatch occurred between the command and object type. The command does not support the specified object type.

**Effect.** The command is not executed.

**Recovery.** Check that the object type you specify is supported by the command you use.
-5: ZCOM-ERR-CMD-NOT-SUPP

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The request specifies a command that is the EMS collector does not support.

**Effect.** The request is not executed.

**Recovery.** Make sure to specify commands supported by the EMS collector.

-15: ZCOM-ERR-OBJ-ALRDY-DEF

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The specified object is already configured. Because there is only one object (filter), the user issued an ADD FILTER command, and the filter was already installed.

**Effect.** The filter is not added.

**Recovery.** To replace a filter with an updated version of the same name:

- Use the SPI replace operation to replace it directly.
- Delete the filter with the SPI delete operation, then add it again using the SPI add operation.

**Note.** The filter order is significant, and SPI add always adds the filter to the end of the filter list. SPI replace puts the new filter in the same place in the list as the old filter.

-17: ZCOM-ERR-OBJ-NOT-FOUND

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The specified object was not found.

**Effect.** The command does not execute because it could not find the specified object.

**Recovery.** Check that the object exists and that you referred to it correctly.
-22: ZCOM-ERR-SECUR-VIOL

| ZSPI-TKN-ERRLIST          | token-type ZSPI-TYP-LIST |
| ZSPI-TKN-ERROR            | token-type ZSPI-TYP-ERROR |
| ZCOM-TKN-OBJNAME          | token-type ZSPI-TYP-STRING |
| ZCOM-TKN-OBJTYPE          | token-type ZSPI-TYP-ENUM   |
| ZSPI-TKN-ENDLIST          | token-type ZSPI-TYP-SSCTL  |

**Cause.** An unauthorized user issued a sensitive command. The command requires that the requester have a super ID access privilege in the case of a primary collector, or have the same access ID as the alternate collector process. The error is returned after an attempt is made to change any of the collector’s configuration parameters.

**Effect.** The desired action does not take effect.

**Recovery.** Have the system administrator perform the specified control command or obtain the correct access ID.

-23: ZCOM-ERR-SPI-ERROR

| ZSPI-TKN-ERRLIST          | token-type ZSPI-TYP-LIST |
| ZSPI-TKN-ERROR            | token-type ZSPI-TYP-ERROR |
| ZCOM-TKN-OBJNAME          | token-type ZSPI-TYP-STRING |
| ZCOM-TKN-OBJTYPE          | token-type ZSPI-TYP-ENUM   |
| ZSPI-TKN-ERRLIST          | token-type ZSPI-TYP-ERROR  |
| ZSPI-TKN-ERROR,Z-SSID     | token-type ZSPI-DDL-ENUM   |
| ZSPI-TKN-ERROR.Z-ERROR    | token-type ZSPI-DDL-ENUM   |
| ZSPI-TKN-PROC-ERR         | token-type ZSPI-TYP-INT    |
| ZSPI-TKN-ENDLIST          | token-type ZSPI-TYP-SSCTL  |

**Cause.** An SPI error occurred while processing this command.

**Effect.** The command is not executed.

**Recovery.** Contact your service provider.

-24: ZCOM-ERR-SSID-INV

| ZSPI-TKN-ERRLIST          | token-type ZSPI-TYP-LIST |
| ZSPI-TKN-ERROR            | token-type ZSPI-TYP-ERROR |
| ZCOM-TKN-OBJNAME          | token-type ZSPI-TYP-STRING |
| ZCOM-TKN-OBJTYPE          | token-type ZSPI-TYP-ENUM   |
| ZSPI-TKN-SSID-ERR         | token-type ZSPI-TYP-SSID   |
| ZSPI-TKN-ENDLIST          | token-type ZSPI-TYP-SSCTL  |

**Cause.** The subsystem ID (SSID) in the command message header is invalid.

**Effect.** The command is not executed.

**Recovery.** Make sure to use a valid subsystem ID (SSID).
-25: ZCOM-ERR-SUB-NOT-FOUND

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The command contained the ZCOM-TKN-SUB token, but no subordinate objects of the specified type were found.

**Effect.** The command is not executed.

**Recovery.** If you use ZCOM-TKN-SUB, be sure the specified object type has subordinate objects configured. In the collectors, only the ZCOM-OBJ-COLL object type can have subordinate objects.

-26: ZCOM-ERR-TKN-CODE-INV

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-TYP-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID-ERR</td>
<td>token-type ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The specified token code is invalid (that is, it should not be in the command).

**Effect.** The command is not executed.

**Recovery.** Make sure to use token codes that are valid with the specified command.

-27: ZCOM-ERR-TKN-DUP

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-TYP-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The command contains duplicate occurrences of a token that can appear only once.

**Effect.** The command is not executed.

**Recovery.** Remove the duplicate occurrence of the token.
**-28: ZCOM-ERR-TKN-LEN-INV**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-TYP-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The specified variable length token is too large. The collector fully resolves object name token values. If the resolved name is larger than 50 bytes, this error is reported.

**Effect.** The desired action is not taken.

**Recovery.** This error is most likely internal. Report it to your service provider.

**-29: ZCOM-ERR-TKN-REQ**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The command is missing a required token.

**Effect.** The command is not executed.

**Recovery.** Check that the command has all the required tokens.

**-30: ZCOM-ERR-TKN-VAL-INV**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-TYP-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The command contains a token with an illegal value.

**Effect.** The command is not executed.

**Recovery.** Check that all the tokens in the command have legal values.
-32: ZCOM-ERR-VSN-INCOMP

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-SERVER-VERSION</td>
<td>token-type ZSPI-TYP-VERSION</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The command contains token maps with a higher version number than this collector can support.

**Effect.** The command is not executed.

**Recovery.** Check that your token maps are of a version compatible with your collector.

-33: ZCOM-ERR-EMPT-RSP

**Cause.** The EMS collector has no response records to return.

The only token in this response is ZSPI-TKN-RETCODE, which contains the error number. Unlike other ZCOM-ERR tokens, ZCOM-ERR-EMPT-RSP does not contain an error list. It does not indicate an error condition, and no error message should be displayed when this response is received.

The EMS collector returns this response when it receives a command with a context token indicating that more objects await processing, but no qualifying objects are found (for example, if there was another object to process but that object was since deleted). The EMS collector also returns this response when a command includes the value ZSPI-VAL-ERR-WARN, but no errors or warnings apply to any object referred to by the command.

**Effect.** None.

**Recovery.** None needed.

-39: ZCOM-ERR-CMD-NOT-SUPP-BY-OBJ

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-TYP-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The specified object does not support the specified command. The collector received an alter filter command, but the filter was not a compiled filter with parameters. Only such compiled filters can be altered.

**Effect.** The command is not executed.

**Recovery.** Check that the filter you want to alter is a compiled filter with parameters.
ZEMS- Errors (Over Extended SPI Interface)

These errors can be returned by EMS collectors through the extended SPI-compliant interface in response to ZCOM- command.

Token and Data Type Definitions for ZEMS- Errors

Collector error lists returned in response to extended SPI-compliant ZCOM- collector commands include tokens and values defined by SPI (prefixed by ZSPI). These tokens and values are defined here, rather than with each error list, because most of them occur in several error lists.

Note. To express SPI token types such as ZSPI-TYP-UINT, the following listings omit ZSPI-TYP- and place the remainder in italicized, lowercase letters, as in (type uint).

SPI Token Codes

These SPI tokens (with prefix ZSPI) occur in collector error lists:

ZSPI-TKN-ERRLIST (type list)

is the SPI token that begins an error list.

ZSPI-TKN-ERROR (type error)

is the standard SPI error token. The DDL for ZSPI-TKN-ERROR is:

```
DEFINITION  ZSPI-DDL-ERROR.
  02 Z-SSID      TYPE ZSPI-DDL-SSID.
  02 Z-ERROR     TYPE ZSPI-DDL-ENUM.
END
```

ZCOM-TKN-OBJNAME (type string)

is the SPI token that returns the name of the object specified in the extended SPI-compliant ZCOM- command and is included if it is relevant to the analysis of the error condition.

ZCOM-TKN-OBJTYPE (type enum)

is the SPI token that returns the object type (ZCOM-OBJ-\textit{objtype}) specified in the extended SPI-compliant ZCOM- command and is included if relevant to the analysis of the error condition.

ZSPI-TKN-PARM-ERR

is the standard SPI error token and identifies the token used to establish context. The ZSPI-TKN-PARM-ERR token gives the token code and index (but not the value) of a command parameter token used in error.
ZSPI-TKN-PROC-ERR (type enum)
specifies a procedure associated with the error. (The exact association depends on the particular error and is described with the related ZSPI-TKN-ERROR value.)
ZSPI-TKN-PROC-ERR can have these values for EMS collectors using the extended SPI interface:

- ZEMS-VAL-FILTER-EVAL  20
- ZEMS-VAL-ZFILWAITIO   21
- ZEMS-VAL-ZFILOPEN     22
- ZEMS-VAL-ZFILPOSITION 23
- ZEMS-VAL-ZFILREAD    24
- ZEMS-VAL-ZFILWRITE   25
- ZEMS-VAL-ZFILWRITEREAD 26
- ZEMS-VAL-FILTER-READ 27
- ZEMS-VAL-FILTER-VERIFY 28
- ZEMS-VAL-EMSADDBUFFER 29
- ZEMS-VAL-EMSADDSUBJECT 30
- ZEMS-VAL-EMSADDTOKES 31
- ZEMS-VAL-EMSGET       32
- ZEMS-VAL-EMSINIT      33
- ZEMS-VAL-EMSSEND      34
- ZEMS-VAL-CHECKOPEN    35
- ZEMS-VAL-NEWPROCESS   36
- ZEMS-VAL-CHECKPOINT   37
- ZEMS-VAL-CHECKMONITOR 38
- ZEMS-VAL-FINDDEV      39
- ZEMS-VAL-ALLOCATESEGMENT 40
- ZEMS-VAL-FILTER-LOAD 41
- ZEMS-VAL-FILTER-PARAM 42
- ZEMS-VAL-MSG-READDATA 43
- ZEMS-VAL-FILNM-TO-OFILNM 44
- ZEMS-VAL-OFILNM-TO-FILNM 45
- ZEMS-VAL-FILNM-RESOLVE 46
- ZEMS-VAL-ADD-FILTER   47

ZSPI-TKN-SSID-ERR (type ssid)
is the subsystem ID used in error.

ZSPI-TKN-ENDLIST (type ssctl)
is the SPI token that ends a list—an error list in this section.

EMS Token Codes

These EMS tokens (with prefix ZEMS) occur in collector error lists:

ZEMS-TKN-PCOLL-NUM-FILTERS (type uint)
identifies the number of filters installed in the primary collector when $0$ and $ZOPR$ are not synchronized with respect to filter configuration data.
1008: ZEMS-ERR-INV-OBJECT

**Cause.** This error can be returned by a pre-D31 release collector for an object that it does not support, which would be FILTER.

**Effect.** The desired action does not take place.

**Recovery.** Use a collector from the D31 RVUs or later.

1009: ZEMS-ERR-INV-CPU

**Cause.** A switch CPU command was issued, but the CPU number was not equal to the backup CPU.

**Effect.** The switch does not take place.

**Recovery.** Specify the current backup’s CPU for the switch.
1010: ZEMS-ERR-CPU-RANGE

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A switch CPU command was issued, but the collector received a command message with an invalid CPU number (that is, not in the range 0 through 15).

**Effect.** The switch does not take place.

**Recovery.** Specify the current backup’s CPU for the switch.

1013: ZEMS-ERR-CDIST-CPU

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A switch CPU command was issued for the compatibility distributor, but the CPU number was not equal to the backup CPU.

**Effect.** The switch does not take place.

**Recovery.** Specify the compatibility distributor’s current backup CPU for the switch.

1019: ZEMS-ERR-FLT-FORM

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZEMS-TKN-FAIL-REASON</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-FILTER-ERROR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-PROC-ERR**

Can have the values:

- ZEMS-VAL-ZFILOPEN 22
- ZEMS-VAL-ZFILWRITEREAD 26
**Cause.** The collector received a command message for processing a filter that is not in the expected format. The filter might be too big to fit in the supplied buffer, the filter format is invalid, or a version incompatibility was detected. The filter might have been compiled with a version newer than the version supported by the collector. If the specified filter is a burst filter or filter table edit file, it cannot be converted to an object form.

**Effect.** The filter is not added.

**Recovery.** Correct the filter source, or use an appropriate compiler.

### 1020: ZEMS-ERR-FLT-LOAD

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERROR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector could not load the filter because of one of these I/O problems:

- An incorrect file name is given.
- The file is not of the correct type (845 for filter objects, 101 for filter table source files).
- The filter contains destination profile data for routing events—a feature supported only by the distributor.
- The parameter tokens in the command (if any) are invalid.

**Effect.** The filter is not added.

**Recovery.** Specify a correct name, generate a correct filter file, or remove destination profiles and routing statements.
1034: ZEMS-ERR-OPEN-LOG

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** An ALTER collector (or CONTROL) command has been received to change the current logging subvolume. The volume is not accessible, and the log file cannot be created.

**Effect.** The collector continues to log into the current subvolume.

**Recovery.** Check the file system error returned and try to resolve the access problem.

1036: ZEMS-ERR-DEST-ACCESS

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-PROC-ERR**

can have the value:

**ZEMS-VAL-FINDDEV** 39

**Cause.** The primary collector cannot access a TEXTOUT destination device.

**Effect.** The change does not occur.

**Recovery.** Check the command you issued for any errors in syntax or values provided.
1040: ZEMS-ERR-ZFIL

**Cause.** An ADD FILTER or ALTER FILTER command was issued for the collector, and an I/O error occurred when attempting to read the specified filter file. This error could be a security access error, or a nonexisting file error. The file-system error number is reported.

**Effect.** The filter is not added or altered.

**Recovery.** Check the file name and access security of the filter.

**1053: ZEMS-ERR-INV-MODE**

**Cause.** The primary collector received a command message with an invalid mode for the compatibility distributor: the field ZCOL-CDIST-MODE-SET, in the token map ZEMS-MAP-COL-CONTROL-CDIST, contained an invalid value. Valid values are ZEMS-VAL-CDIST-CRITICAL-ONLY, -CDIST-EMSMODE, and -CDIST-STOP.

**Effect.** The command to change mode is not executed.

**Recovery.** Check the command you issued for any errors in syntax or values provided.
1054: ZEMS-ERR-CDIST-DOWN

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The primary collector received a CONTROL or ALTER command for the compatibility distributor, but the distributor is not running.

**Effect.** The command for the distributor cannot be carried out.

**Recovery.** Because the compatibility distributor cannot be restarted, use a distributor in its place.

1059: ZEMS-ERR-MAXFLT

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** An ADD FILTER command was issued causing the maximum number of filters (10) to be exceeded.

**Effect.** The filter is not added.

**Recovery.** Remove filters first to accommodate the new filter.

1060: ZEMS-ERR-FLT-ALLOC

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** An ADD FILTER command was issued, and the combined resources for event source, destination, and filter parameter buffers exceed the buffer pool size.

**Effect.** The filter is not added.

**Recovery.** Remove sources, destinations, and filters to reduce the buffer pool size.
1067: ZEMS-ERR-ZOPR-SEND

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** A communication error occurred between $0$ and $ZOPR$. This error might be the result of a command that involves file system IO, such as ADD FILTER.

**Effect.** The command fails.

**Recovery.** Retry the command. If not successful, contact your service provider.

1068: ZEMS-ERR-ZOPR-SYNC

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJNAME</td>
<td>token-type ZSPI-TYP-STRING</td>
</tr>
<tr>
<td>ZCOM-TKN-OBJTYPE</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-ZOPR-CMD</td>
<td>token-type ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZEMS-TKN-ZOPR-NUM-FILTERS</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-PCOLL-NUM-FILTERS</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** An internal error occurred where $0$ and $ZOPR$ are not synchronized with respect to filter configuration data. This error might be the result of an ADD or ALTER FILTER command.

**Effect.** The command fails.

**Recovery.** Delete all filters. If the error persists, contact your service provider.
ZEMS- Errors (Over Basic SPI Interface)

These errors can be returned by EMS collectors through the basic SPI-compliant interface in response to ZEMS- collector command.

Token and Data Type Definitions

Collector error lists include some tokens and values defined by SPI (prefixed by ZSPI) as well as those defined by EMS (prefixed by ZEMS). These tokens and values are defined here, rather than with each error list, because most of them occur in several error lists.

Note. To express SPI token types such as ZSPI-TYP-UINT, these listings omit ZSPI-TYP- and place the remainder in italicized, lowercase letters, as in (type uint).

SPI Token Codes

These SPI tokens (with prefix ZSPI) occur in collector error lists:

ZSPI-TKN-ENDLIST (type ssctl)

is the SPI token that ends a list—an error list in this section.

ZSPI-TKN-ERRLIST (type list)

is the SPI token that begins an error list.

ZSPI-TKN-ERROR (type error)

is the standard SPI error token. The DDL for ZSPI-TKN-ERROR is:

DEFINITION ZSPI-DDL-ERROR.
  02 Z-SSID      TYPE ZSPI-DDL-SSID.
  02 Z-ERROR     TYPE ZSPI-DDL-ENUM.
END

ZSPI-TKN-PARM-ERR

is the standard SPI error token. The DDL for ZSPI-TKN-PARM-ERR is:

DEF  ZSPI-DDL-PARM-ERR.
  02 Z-TOKENCODE TYPE ZSPI-DDL-TOKENCODE.
  02 Z-INDEX     TYPE ZSPI-DDL-UINT.
  02 Z-OFFSET    TYPE ZSPI-DDL-UINT.
END

The ZSPI-TKN-PARM-ERR token gives the token code and index (but not the value) of a command parameter token used in error.

ZSPI-TKN-PROC-ERR (type enum)

specifies a procedure associated with the error. (The exact association depends on the particular error and is described with the related ZSPI-TKN-ERROR value.)
ZSPI-TKN-PROC-ERR can have these values for EMS collectors using the basic SPI interface:

ZEMS-VAL-FILTER-EVAL  20
ZEMS-VAL-ZFILAWAITIO  21
ZEMS-VAL-ZFILOPEN  22
ZEMS-VAL-ZFILPOSITION  23
ZEMS-VAL-ZFILREAD  24
ZEMS-VAL-ZFILWRITE  25
ZEMS-VAL-ZFILWRITEREAD  26
ZEMS-VAL-FILTER-READ  27
ZEMS-VAL-FILTER-VERIFY  28
ZEMS-VAL-EMSADDBUFFER  29
ZEMS-VAL-EMSADDSUBJECT  30
ZEMS-VAL-EMSADDTOKENS  31
ZEMS-VAL-EMSGET  32
ZEMS-VAL-EMSINIT  33
ZEMS-VAL-EMSSSEND  34
ZEMS-VAL-CHECKOPEN  35
ZEMS-VAL-NEWPROCESS  36
ZEMS-VAL-CHECKPOINT  37
ZEMS-VAL-CHECKMONITOR  38
ZEMS-VAL-FINDDEV  39

ZSPI-TKN-SSID-ERR (type ssid)  
is the subsystem ID used in error.

EMS Token Codes

These EMS tokens (with prefix ZEMS) occur in collector error lists for ZEMS- collector errors returned using the basic SPI interface:

ZEMS-TKN-FAILFILENAME (type fname; nonshared)  
is the file name of a bad log file.

ZEMS-TKN-ZFILERR (type uint; nonshared)  
is the error-code associated with a distributor ZSPI-TKN-ERROR value.

1001: ZEMS-ERR-VERSION

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Cause. The collector does not support the version specified in the command message.

Effect. The command is not executed.
Collector Errors

1002: ZEMS-ERR-INV-CMD

**Recovery.** Check that you use versions of the collector and operating system that are compatible with each other.

**1002: ZEMS-ERR-INV-CMD**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector has received an invalid command message.

**Effect.** The command is not executed.

**Recovery.** Check your command for any syntax errors or invalid parameter entries.

1003: ZEMS-ERR-INV-SSID

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-SSID-ERR</td>
<td>token-type ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector received a command message with an invalid subsystem ID (SSID) from a message buffer not owned by EMS.

**Effect.** The command is not executed.

**Recovery.** Specify a valid subsystem ID (SSID) in your command.

1004: ZEMS-ERR-INV-TKN

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>[ ZSPI-TKN-SSID-ERR ]</td>
<td>token-type ZSPI-TYP-SSID</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector received a command message with an unneeded or unrecognized token. (The bracketed token, SSID-ERR, appears in the error message only if the unrecognized token is associated with a subsystem ID not associated with EMS.)

**Effect.** The command is not executed.

**Recovery.** Check that you specify only tokens that are needed and recognizable.
1005: ZEMS-ERR-INV-VALUE

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector received a command message with a bad token; the token value is invalid.

**Effect.** The command is not executed.

**Recovery.** Check that you specify valid token values.

1006: ZEMS-ERR-DUP-TKN

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector received a command message with duplicate tokens.

**Effect.** The command is not executed.

**Recovery.** Check that your command does not contain duplicate tokens if it is one where duplicate tokens are not allowed.

1008: ZEMS-ERR-INV-OBJECT

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector received a command message with an invalid object type. The object type must be zero for collector commands.

**Effect.** The command is not executed.

**Recovery.** Check that you specify valid object types (zero for collector commands).
1009: ZEMS-ERR-INV-CPU

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Cause. The ZCOL-PRIMARYCPU DDL definition contains an invalid CPU number. Because ZCOL-PRIMARYCPU directs the collector to operate in its backup CPU, ZCOL-PRIMARYCPU must contain the CPU number of the current backup CPU.

Effect. The command is not executed.

Recovery. Check that ZCOL-PRIMARYCPU contains the CPU number of the current backup CPU.

1010: ZEMS-ERR-CPU-RANGE

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Cause. The collector received a command message with an invalid CPU number (outside the range 0 through 15).

Effect. The command is not executed.

Recovery. Use a valid CPU number in the command (in the range 0 through 15).

1013: ZEMS-ERR-CDIST-CPU

<table>
<thead>
<tr>
<th>ZSPI-TKN-ERRLIST</th>
<th>token-type ZSPI-TYP-LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-PARM-ERR</td>
<td>token-type ZSPI-DDL-PARM-ERR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

Cause. The primary collector received a command message with an invalid CPU number. The CPU number in ZEMS-TKN-CDISTPRICPU or ZCOL-CDIST-PRICPU is not the backup CPU for the compatibility distributor.

Effect. The command is not executed.

Recovery. Use a valid CPU number in the command.
1015: ZEMS-ERR-REQ-TKN

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector received a collector CONTROL command message that is missing a required token. You must include at least one of these tokens: ZEMS-MAP-COL-CONTROL-CDIST, ZEMS-MAP-COL-CONTROL, or ZEMS-TKN-CDISTPRICPU. (ZEMS-MAP-COL-CONTROL-CDIST and ZEMS-TKN-CDISTPRICPU are mutually exclusive; do not include both.)

**Effect.** The command is not executed.

**Recovery.** Make sure your command has all the required tokens it needs.

1017: ZEMS-ERR-INV-OCCURS

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector received a command message request from an application whose message buffer is not large enough to contain the entire command message.

**Effect.** The command is not executed.

**Recovery.** Fix the byte length of the WRITEREAD operation sending the command message so that it fits within the buffer size, or increase the message buffer size.

1031: ZEMS-ERR-LOG-ACCESS

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRRLIST</td>
<td>token-type ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>token-type ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZEMS-TKN-ZFILERR</td>
<td>token-type ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>token-type ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>token-type ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** An I/O error occurred while the collector was accessing a log or ZZEVECONF file.

**Effect.** The command that caused the I/O error does not execute completely.

**Recovery.** Examine the log or ZZEVECONF files to determine the cause of the I/O problem.
1034: ZEMS-ERR-OPEN-LOG

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**Cause.** The collector has received a collector CONTROL command message that contains a ZEMS-TKN-LOGSUBVOL token, but the collector cannot access the specified volume or subvolume. The collector returns the token ZEMS-TKN-FAILFILENAME to identify the bad subvolume.

**Effect.** The command is not executed.

**Recovery.** Examine the specified volume or subvolume to determine the access problem.

1036: ZEMS-ERR-DEST-ACCESS

<table>
<thead>
<tr>
<th>Token</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSPI-TKN-ERRLIST</td>
<td>ZSPI-TYP-LIST</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ZFILERR</td>
<td>ZSPI-TYP-UINT</td>
</tr>
<tr>
<td>ZEMS-TKN-FAILFILENAME</td>
<td>ZSPI-TYP-FNAME</td>
</tr>
<tr>
<td>ZSPI-TKN-PROC-ERR</td>
<td>ZSPI-TYP-ENUM</td>
</tr>
<tr>
<td>ZSPI-TKN-ERROR</td>
<td>ZSPI-TYP-ERROR</td>
</tr>
<tr>
<td>ZSPI-TKN-ENDLIST</td>
<td>ZSPI-TYP-SSCTL</td>
</tr>
</tbody>
</table>

**ZSPI-TKN-PROC-ERR** can have the value:

<table>
<thead>
<tr>
<th>Token</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-VAL-FINDDEV</td>
<td>39</td>
</tr>
</tbody>
</table>

**Cause.** The primary collector cannot access a TEXTOUT destination device.

**Effect.** The change does not occur.

**Recovery.** Check the command you issued for any errors in syntax or values provided.
1041: ZEMS-ERR-ZSPI

ZSPI-TKN-ERRLIST  token-type ZSPI-TYP-LIST
ZSPI-TKN-ERROR    token-type ZSPI-TYP-ERROR
ZSPI-TKN-PARM-ERR token-type ZSPI-DDL-PARM-ERR
ZSPI-TKN-ERRLIST  token-type ZSPI-TYP-LIST
ZSPI-TKN-ERROR    token-type ZSPI-TYP-ERROR
ZSPI-TKN-PARM-ERR token-type ZSPI-DDL-PARM-ERR
ZSPI-TKN-PROC-ERR token-type ZSPI-TYP-ENUM
ZSPI-TKN-ENDLIST  token-type ZSPI-TYP-SSCTL
ZSPI-TKN-ENDLIST  token-type ZSPI-TYP-SSCTL

ZSPI-TKN-PROC-ERR (in the second (nested) ZEMS-ERR-ZSPI error list)
can have the values:

ZSPI-VAL-SSGET  2
ZSPI-VAL-SSGETTKN  3

Cause. A message-format error occurred when the collector tried to decode a command message. The error message comprises a ZSPI error list; the list includes the token ZSPI-TKN-PARM-ERR, if appropriate, and a second, nested ZSPI error list.

Effect. The command is not executed.

Recovery. Check the command message for any errors, as indicated in the message.

1048: ZEMS-ERR-ACC-VIOL

ZSPI-TKN-ERRLIST  token-type ZSPI-TYP-LIST
ZSPI-TKN-ERROR    token-type ZSPI-TYP-ERROR
ZSPI-TKN-ENDLIST  token-type ZSPI-TYP-SSCTL

Cause. The collector received a collector CONTROL command message from a management application that has insufficient access privileges; this is a security violation. The application must have super-group access privileges to control the primary collector. For an alternate collector, access is limited to the super-group users or to the user who created the alternate collector.

Effect. The command is not executed.

Recovery. A user with appropriate access privileges must process this command.
1053: ZEMS-ERR-INV-MODE

**Cause.** The primary collector received a command message with an invalid mode for the compatibility distributor: the field ZCOL-CDIST-MODE-SET, in the token map ZEMS-MAP-COL-CONTROL-CDIST, contained an invalid value. Valid values are ZEMS-VAL-CDIST-CRITICAL-ONLY, -CDIST-EMSMODE, and -CDIST-STOP.

**Effect.** The command to change mode is not executed.

**Recovery.** Check the command you issued for any errors in syntax or values provided.

1054: ZEMS-ERR-CDIST-DOWN

**Cause.** The primary collector received a CONTROL command for the compatibility distributor, but the distributor is not running.

**Effect.** The command for the distributor could not be carried out.

**Recovery.** Because the compatibility distributor cannot be restarted, use a distributor in its place.

1055: ZEMS-ERR-NO-BACKUP

**Cause.** The content of the ZCOL-PRIMARYCPU field was the alternate collector’s backup CPU, but the alternate collector does not currently have a backup process.

**Effect.** The command is not executed.

**Recovery.** Determine whether the backup was created or if it simply has not started. If necessary, stop the alternate collector and restart it so it automatically creates a new backup process.
1056: ZEMS-ERR-ALLOC-LOG

**Cause.** An alternate collector log file could not be allocated.

**Effect.** The command is not executed.

**Recovery.** Make more space in the LOGSUBVOL.

1057: ZEMS-ERR-LOGGING-STOPPED

**Cause.** The alternate collector is currently holding events in its event buffer pool and cannot write them to disk because logging has stopped.

**Effect.** The command is not executed.

**Recovery.** Relieve the logging-stopped situation by increasing MAXFILE, enabling ROTATEFILES, or switching to a new LOGSUBVOL.
Part VI: EMS Example Files

This part of the manual contains sample code and source files for many of the EMS procedures discussed throughout this manual:

- Appendix A, Example of Retrieving Event Messages
- Appendix B, Example of Reporting Events
- Appendix C, Standard Event Sample Files
Example of Retrieving Event Messages

This appendix presents a complete working example of a management application in four versions: a TAL version, a TACL version, a COBOL version, and a C version. The example includes a filter specification and a DDL file.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running the TAL Version</td>
<td>A-2</td>
</tr>
<tr>
<td>Running the TACL Version</td>
<td>A-2</td>
</tr>
<tr>
<td>Running the COBOL Version</td>
<td>A-2</td>
</tr>
<tr>
<td>Running the C Version</td>
<td>A-2</td>
</tr>
<tr>
<td>Overview of Application Logic</td>
<td>A-3</td>
</tr>
<tr>
<td>DDL Source File</td>
<td>A-4</td>
</tr>
<tr>
<td>Filter Source File</td>
<td>A-5</td>
</tr>
<tr>
<td>TAL Source File</td>
<td>A-5</td>
</tr>
<tr>
<td>TACL Source File</td>
<td>A-14</td>
</tr>
<tr>
<td>COBOL Source File</td>
<td>A-19</td>
</tr>
<tr>
<td>C Source File</td>
<td>A-24</td>
</tr>
<tr>
<td>Program Modifications</td>
<td>A-34</td>
</tr>
</tbody>
</table>

The four versions of the example have DDL and filter source files in common but require distinct main source code files.

When you run this application at your terminal, it displays the event messages that originated in a particular CPU of your system. The application prompts you for the CPU number and reads and displays event messages as they are logged by the collector. Execution of the application continues until you press the BREAK key at the terminal and stop the application with a STOP command.

**Note.** In this appendix, the name or prefix MYAP designates a user application, subsystem, and so on. Wherever you see MYAP, replace the complete name that includes it with the name of the application or subsystem you want used in event retrieval.
Running the TAL Version

To run the TAL version of this application:

1. Load the TACL macro LDQ (load quietly), which is defined as:

   ```
   #SECTION LDQ MACRO
   #PUSH:DUMMY
   #LOAD /KEEP 1, LOADED:DUMMY/ %1%
   #POP:DUMMY
   ```

2. Load the definition files for the programmatic interface, the EMS subsystem, and MYAP by entering these TACL commands at your terminal:

   ```
   DDL /IN MYAPDDL/
   LDQ $SYSTEM.ZSPIDEF.ZSPITACL
   LDQ $SYSTEM.ZSPIDEF.ZEMSTACL
   LDQ MYAPTACL
   EMF /IN MYAPFSRC/$SYSTEM.FILT.MYAPFOBJ
   TAL /IN MYAPMAIN/MYAP
   RUN MYAP
   ```

3. Compile the DDL file.

4. Load standard definition files for the SPI, EMS, and MYAP subsystems.

5. Compile the TAL procedures.

6. Issue the RUN command.

Running the TACL Version

To run the TACL version, proceed as in the TAL version, except replace the compilation step (TAL /...) and the RUN command with this TACL call:

```
MYAPMAIN
```}

Running the COBOL Version

To run the COBOL version, proceed as in the TAL version except replace the compilation step (TAL /...) with:

```
COBOL85 /IN MYAPMAIN/MYAP
```}

Running the C Version

To run the C version, proceed as in the TAL version except replace the compilation step (TAL /...) with:

```
C /IN MYAPMAIN/MYAP
```
Overview of Application Logic

The example application in this appendix:

1. Starts a consumer distributor
2. Prompts the user for a CPU number
3. Opens the distributor for SPI communication; sends a distributor CONTROL command message that:
   - Loads the compiled filter, which selects the event messages that originated on the specified CPU
   - Passes the CPU number to the filter as a parameter
   - Connects $0 to the distributor as a source collector
4. Enters a loop that:
   - Requests filter-specified event messages from the consumer distributor through use of a GETEVENT command message and response
   - Displays (at the terminal) the text that the EMSTEXT procedure produced, based on the event message.

The program example consists of source statements for:

- The DDL file that defines the subsystem ID and token name used by the filter parameter
- The event-message filter specification
- The TAL program, which can be divided roughly as:
  - Definitions common to the TAL procedures
  - The get^cpu^num procedure, which prompts the user for the CPU number
  - The spi^control procedure, which generates and sends one distributor CONTROL command message and processes the response
  - The displ^event procedure, which calls the EMSTEXT procedure to produce display text and sends the formatted text to the terminal
  - The getevent^loop procedure, which generates and sends one distributor GETEVENT command message and calls the displ^event procedure
  - The dist^intfc MAIN procedure
- The TACL main source module
- The COBOL main source module
- The C main source module
DDL Source File

A DDL file is necessary to event-message retrieval only for passing parameters to an event-message filter.

This file contains DDL source statements to pass a CPU parameter to the filter:

! File name: MYAPDDL

?DICT !
?NOLIST
?SOURCE $SYSTEM.ZSPIDEF.ZSPIDDL
?LIST

! Request TACL, TAL, COBOL, and C output
?TACL MYAPTACL !, NOTIMESTAMP
?TAL MYAPTAL !, NOTIMESTAMP
?COBOL MYAPCOB !, NOTIMESTAMP
?C MYAPC !, NOTIMESTAMP

?SETSECTION constants
!
! Components of a subsystem ID
!
CONSTANT myap-val-owner VALUE IS "MYAPLAB ".
CONSTANT myap-ssn-myap VALUE IS 4.
CONSTANT myap-val-version VALUE IS 2.

* Defines the structure for the MYAP SSID
DEFINITION myap-val-ssid TACL SSID.

02 Z-FILLER TYPE CHARACTER 8
   VALUE IS myap-val-owner.

02 Z-OWNER TYPE ZSPI-DDL-CHAR8
   REDEFINES Z-FILLER.

02 Z-NUMBER TYPE ZSPI-DDL-INT
   VALUE IS myap-ssn-myap.

02 Z-VERSION TYPE ZSPI-DDL-UINT
   VALUE IS myap-val-version.

END
!
! MYAP token codes
!
TOKEN-CODE myap-tkn-param-cpu VALUE IS 1
   TOKEN-TYPE IS ZSPI-TYP-UINT.
Filter Source File

This file contains filter source statements:

```
== File name: MYAPFSRC
[#SET myap^val^ssid
    [myap^val^owner].[myap^ssn^myap].[myap^val^version]]
FILTER cpu^param (SSID(myap^val^ssid, myap^tkn^param^cpu));
== FILTER cpu^param (myap^tkn^param^cpu);
BEGIN
    IF ZEMS^TKN^CPU = myap^tkn^param^cpu THEN PASS;
END;
```

TAL Source File

This file contains the TAL source statements:

```
! File name: MYAPMAIN ---------------------------------------
!
! The MYAP application does the following:
!
!    - Starts and opens a consumer distributor
!
!    - Prompts you at your terminal for a CPU number
!
!    - Issues CONTROL command messages to
!
!        connect the collector $0
!        load a filter file
!        load a filter param
!            (cpu-number from user)
!
!    - Performs the following steps in a loop:
!        -- Issues a GETEVENT command message
!        -- Displays an event message at your terminal
!
! MYAP waits for new events from the given CPU and displays
! them as they arrive.
!
! To terminate the program, press the BREAK key
! and type STOP.
!
! The filter file must be $SYSTEM.FILT.MYAPFOBJ.
!
?NOMAP,NOLMAP
?INSPECT,SYMBOLS
?EXTENDSTACK 4   ! To ensure enough stack space for EMSTEXT
?NOCODE
?
?NOLIST, SOURCE $SYSTEM.ZSPIDEF.ZSPITAL
?LIST
?NOLIST, SOURCE $SYSTEM.ZSPIDEF.ZEMSTAL
?LIST
```
NOLIST, SOURCE MYAPTAL
LIST

LITERAL true = 1;
LITERAL false = 0;

INT .term^name[0:11], term,
   .rcv^name[0:11] := ["$RECEIVE", 8*[" "]], rcv,
   ct^rd, error;
INT .distr^name[0:11] := [12*[" "]], distr,
   .distr^prog^file[0:11] := ["$SYSTEM SYSTEM EMSDIST "],
   cpu^num, .term^buf[0:39], text^len, input^len,
   spi^err, used^len, ems^err;

STRUCT startup^msg;
   BEGIN
   INT msgcode;
   STRUCT default;
      BEGIN
      INT volume[0:3],
         subvol[0:3];
      END; -- of STRUCT
   STRUCT infile;
      BEGIN
      INT volume[0:3],
         subvol[0:3],
         dname[0:3];
      END; -- of STRUCT
   STRUCT outfile;
      BEGIN
      INT volume[0:3],
         subvol[0:3],
         dname[0:3];
      END; -- of STRUCT
   INT param[0:30];
   END; -- of STRUCT

   STRUCT .spi^buf(zems^ddl^msg^buffer^def),
     .sav^buf(zems^ddl^msg^buffer^def);
   STRUCT .err^buf(zspi^ddl^error^def);

   STRING .s^distr^qual := @distr^name[4] '<<' 1,
      .s^term^buf := @term^buf '<<' 1, .end^of^text;

   INT ibuflen := ZEMS^VAL^BUFLEN;
   INT msgcount:= 0;
   INT msglimit:= 0;  ! 0 represents no limit, 100 is typical

   LITERAL evt^text^len = 78;
   LITERAL num^evt^lines = 2;
   INT .evt^text^buf[0:(evt^text^len/2)*num^evt^lines];
   INT .actual^len[0:num^evt^lines];
! *************** SPI Definitions **********************

! Declare SSIDs

STRUCT .zems^val^ssid(zems^val^ssid^def);
STRUCT .myap^val^ssid(myap^val^ssid^def);

INT .coll^name[0:11] := ["$0",11*" "],
.filt^name[0:11] := ["$SYSTEM FILT MYAPFOBJ"];

?NOLIST
?SOURCE $SYSTEM.SYSTEM.EXTDECS0 ( MYTERM, OPEN, DEBUG,
   READUPDATE, REPLY, CREATEPROCESSNAME, NEWPROCESS,
   WRITE, CLOSE, WRITEREAD, FILEINFO, DELAY, STOP, SSINIT,
   SSPUTTKN, SSGETTKN, EMSTEXT, NUMIN, SSMOVETKN,
   EMSGETTKN,
   JULIANTIMESTAMP,
   FILEERROR )
?LIST
?PAGE
PROC get^cpu^num;
BEGIN
INT got^it;

! This procedure prompts the user for a CPU number, which
! can be passed as a parameter to the filter

got^it := false;

WHILE ( NOT got^it ) DO
BEGIN

s^term^buf ':=' ["CPU number?: "] -> @end^of^text;
text^len := @end^of^text '-' @s^term^buf;

retry:
! Prompt for and read CPU number
CALL WRITEREAD ( term, term^buf, text^len, 14,
   input^len);

IF > THEN  ! EOF from terminal
   CALL STOP;

IF < THEN  ! Some unusual condition
   BEGIN  ! Check whether to retry
      IF FILEERROR ( term ) THEN GOTO retry;
      CALL DEBUG;
   END;

   ! Terminate input with a null
   s^term^buf[input^len] := 0;

   ! Translate number to internal integer
   CALL NUMIN ( S^TERM^BUF, CPU^NUM, 10, ERROR);
IF ( ( ERROR = 0 ) AND ( CPU^NUM >= 0 ) ) THEN
    got^it := true;
END;      ! WHILE
END;

PROC send^spi^cmd;
BEGIN

! This procedure accepts, in spi^buf, a command message
! prepared by another procedure, such as the
! spi^cmd^set^source procedure, below. The
! send^spi^cmd procedure sends the command message to
! the distributor and checks the response message.

! Determine how much of buffer has been used
spi^err := SSGETTKN(spi^buf, ZSPI^TKN^USEDLEN,
    used^len);
IF spi^err <> ZSPI^ERR^OK THEN
    CALL DEBUG;

! Send the used part to the distributor
CALL WRITEREAD(distr, spi^buf, used^len,
    ZEMS^VAL^BUFLEN);
IF <> THEN
    CALL DEBUG;

! Reset buffer length to what you declared for spi^buf
spi^err := SSPUTTKN(spi^buf,
    ZSPI^TKN^RESET^BUFFER,
    ibuflen);
IF spi^err <> ZSPI^ERR^OK THEN
    CALL DEBUG;

! Response is in buffer--check for return code
spi^err := SSGETTKN(spi^buf, ZSPI^TKN^RETCODE,
    ems^err, 1);
IF spi^err <> ZSPI^ERR^OK THEN
    CALL DEBUG;
IF (ems^err) THEN
    CALL DEBUG;
END;

PROC spi^cmd^set^source;
BEGIN

! This procedure builds a command message that directs
! the distributor to use a collector as a source of
! event messages.

! Initialize spi^buf for distributor CONTROL message
spi^err := SSINIT (spi^buf, ZEMS^VAL^BUFLEN,
    zems^val^ssid,
Example of Retrieving Event Messages

ZSPI^VAL^CMDHDR, ZEMS^CMD^CONTROL);
IF spi^err <> ZSPI^ERR^OK THEN
  CALL DEBUG;

! Place the connect-source-collector token in buffer
spi^err := SSPUTTKN(spi^buf, ZEMS^TKN^CONNECT^SRC^COLL,
coll^name);
IF spi^err <> ZSPI^ERR^OK THEN
  CALL DEBUG;

! Send the command message to the distributor
CALL send^spi^cmd;

END;

?PAGE

PROC spi^cmd^load^filter;
BEGIN

! This proc builds a command message that loads a filter
! into the distributor and passes one filter parameter

! Initialize spi^buf for distributor CONTROL message
spi^err := SSINIT (spi^buf, ZEMS^VAL^BUFLEN,
  zems^val^ssid,
  ZSPI^VAL^CMDHDR, ZEMS^CMD^CONTROL);
IF spi^err <> ZSPI^ERR^OK THEN
  CALL DEBUG;

! Place the load-filter token in the buffer
spi^err := SSPUTTKN(spi^buf, ZEMS^TKN^FILTERFILE,
  filt^name);
IF spi^err <> ZSPI^ERR^OK THEN
  CALL DEBUG;

! Place the filter-parameter token in the buffer
spi^err := SSPUTTKN(spi^buf, MYAP^TKN^PARAM^CPU,
  cpu^num, , myap^val^ssid);
IF spi^err <> ZSPI^ERR^OK THEN
  CALL DEBUG;

! Send the command message to the distributor
CALL send^spi^cmd;

END;

?PAGE

PROC displ^event(event^buf);
INT .EXT event^buf;
BEGIN
INT(32) etxt^stat;
INT i;

! This procedure displays at your terminal the event
! message just retrieved.
!
! Generate display text from the event message

EMS Manual—426909-005
A-9
etxt^stat :=
   EMSTEXT(event^buf,
      EVT^TEXT^BUF,
      evt^text^len,   ! Displayable line length
      num^evt^lines,  ! Number of display lines
      actual^len,     ! Line lens stored here
      ,               ! Reserved
      ,               ! Indent default
      1);             ! Console compatible

! Check for EMSTEXT calling errors
IF $HIGH (etxt^stat) = 0 and $INT (etxt^stat) <> 0
THEN CALL DEBUG;

! Display the text
FOR i := 0 TO num^evt^lines-1 DO
BEGIN
   IF (actual^len[i] <> -1) THEN
   BEGIN
      retry1:
      CALL WRITE (term,
         evt^text^buf[i*evt^text^len/2],
         actual^len[i]);
      IF <> THEN
      BEGIN
         IF FILEERROR(term) THEN
            GOTO retry1;
         CALL DEBUG;
      END;
   END;
END;
?PAGE
PROC getevent^loop;
BEGIN
   INT .EXT event^buf(zems^ddl^msg^buffer^def);
   INT(32) tkn;
   INT ivalue;

   ! This procedure consists of a loop to retrieve event
   ! messages. Each time through the loop, the procedure
   ! gets an event message and calls displ^event to display
   ! it at your terminal.

   ! Initialize spi^buf for GETEVENT command message
   spi^err := SSINIT(spi^buf, ZEMS^VAL^BUFLEN,
      zems^val^ssid, ZSPI^VAL^CMDHDR,
      ZEMS^CMD^GETEVENT);
   IF spi^err <> ZSPI^ERR^OK THEN
      CALL DEBUG;

   ! Save the original command message
   sav^buf ':=' spi^buf FOR $LEN(sav^buf) BYTES;

   ! Begin loop that gets and displays event messages
WHILE msgcount < msglimit OR msglimit = 0 DO
    BEGIN
        msgcount := msgcount + 1;

        ! Send GETEVENT command message to the distributor
        CALL send^spi^cmd;

        ! Find the event message within the GETEVENT response
        tkn := ZEMS^TKN^EVENT;
        spi^err := SSGETTKN ( spi^buf, ZSPI^TKN^ADDR,
            tkn, 1,
            @event^buf);
        IF spi^err <> ZSPI^ERR^OK THEN
            CALL DEBUG;

        ! Move past the length part (variable length token)
        @event^buf := @event^buf + 2D;

        ! Display the event message at your terminal
        CALL displ^event(event^buf);

        ! Save CONTEXT token from this GETEVENT response
        ! message for the next GETEVENT command message
        spi^err := SSMOVETKN(ZSPI^TKN^CONTEXT,
            spi^buf, 1, ! Source
            sav^buf, 1); ! Destination

        IF spi^err <> ZSPI^ERR^OK THEN
            CALL DEBUG;

        ! Move the updated command to spi^buf for next time
        ! through the loop
        spi^buf ':=' sav^buf FOR $LEN(spi^buf) BYTES;
    END;            ! WHILE
END;

PROC dist^intfc MAIN;
BEGIN
    ! This MAIN procedure starts a distributor and calls
    ! procedures that use command messages to connect a
    ! source collector and to load a filter. Then the
    ! procedure calls the getevent^loop procedure to retrieve
    ! and process event messages.

    ! Initialize subsystem IDs
    zems^val^ssid ':=' [ZSPI^VAL^TANDEM,
        ZSPI^SSN^ZEMS,
        ZEMS^VAL^VERSION];

    myap^val^ssid ':=' [MYAP^VAL^OWNER,
        MYAP^SSN^MYAP,
        MYAP^VAL^VERSION];

    ! Get terminal name and open terminal
CALL MYTERM ( term^name );
CALL OPEN ( term^name, term );

! OPEN $RECEIVE file
CALL OPEN ( rcv^name, rcv, , 1 );
IF <> THEN
    CALL DEBUG;

! Read startup message from $RECEIVE
CALL READUPDATE ( rcv, startup^msg, $LEN(startup^msg),
                    ct^rd );
IF <> THEN
    CALL DEBUG;

CALL REPLY ( , , , , 0 );
IF <> THEN
    CALL DEBUG;

! Create a name for the distributor process
CALL CREATEPROCESSNAME ( distr^name );
IF <> THEN
    CALL DEBUG;

! Set up the startup message - distributor must be passed
! a TYPE definition (C for Consumer, P for Printing, or
! F or Forwarding).
startup^msg.param ':'="["TYPE CONSUMER",0,0];

! Start the distributor process
CALL NEWPROCESS ( distr^prog^file, , , , , error, 
                   distr^name);
IF ( ERROR.<0:7> > 0 ) THEN
    CALL DEBUG;

! Open the distributor to write startup message
CALL OPEN ( distr^name, distr );
IF <> THEN
    CALL DEBUG;

! Write the startup message
CALL WRITE ( distr, startup^msg, $LEN(startup^msg));

! Check for errors--error 70 is continuation
! (ASSIGNS and PARAMS)
CALL FILEINFO ( distr, error );
IF ( NOT ( (error = 0) OR ( error = 70) ) ) THEN
    CALL DEBUG;
CALL CLOSE ( distr );    ! Don't send assigns & params

! Prompt user for the CPU number
CALL get^cpu^num;

! Reopen the distributor for command messages
s^distr^qual ':'=#ZSPI;
CALL OPEN ( distr^name, distr );
IF <> THEN
CALL DEBUG;

! Tell distributor to use collector log files as
! the source of event messages
CALL spi^cmd^set^source;

! Load the filter and filter parameter(s)
! into the distributor
CALL spi^cmd^load^filter;

! Retrieve and display event messages
CALL getevent^loop;

CALL STOP;
END;
### TACL Source File

This file contains the main TACL source:

```tcl
?TACL MACRO
== File name: MYAPMAIN

== MYAPMAIN
==    - Starts and opens a consumer distributor
==    - Gets a CPU number from the user
==    - Issues an SPI CONTROL command message to
==        connect collector $0
==    - load a filter file
==    - load a filter param
==        (cpu-number from user)
==    - Performs the following steps in a loop:
==       -- Issue SPI GETEVENT command message
==       -- Display event on home terminal

== The program will wait for new events from the given CPU
== and will display them as they arrive.

== To terminate the program, press the BREAK key.

== The filter file must be $SYSTEM.FILT.MYAPFOBJ

== NOTE: The error handling in this example is very basic.
== The error text causes TACL to stop executing the program
== and point to the error text.

== Load variables if they are not already.
#FRAME
#PUSH #HOME
#SET #HOME :   == Guarantee a 'writeable' home

== Load SPI and EMS defined variables
== Use the ATTACHSEG command to gain access to an existing
== TACL segment file.

ATTACHSEG SHARED $dsv.zspisegf.zspisegf :spidir
ATTACHSEG SHARED $dsv.zspisegf.zemssegf :emsdir

== Set USE so that TACL will find the variables in :spidir
== and :emsdir
== USE also makes sure that : :UTILS :UTILS:TACL are included
== in #USELIST
USE [#USELIST] :spidir :emsdir

== Initialize SSID for EMS
[IF [#MATCH [zems^val^ssid] 0.0.0] |THEN|
   #SET zems^val^ssid &
   [ZSPI^VAL^TANDEM].[ZSPI^SSN^ZEMS].[ZEMS^VAL^VERSION]
]
[IF NOT [#VARIABLEINFO/EXISTENCE/ myap^val^ssid] |THEN|
   #LOAD/LOADED vars/ MYAPTA
```
Example of Retrieving Event Messages

[46x786]EMS Manual—426909-005
[231x66]TACL Source File

] == Initialize SSID for MYAP
[#IF [#MATCH [myap^val^ssid] 0.0.0] |THEN|
      #SET myap^val^ssid &
      [MYAP^VAL^OWNER].[MYAP^SSN^MYAP].[MYAP^VAL^VERSION]
] == Initialize SSID for MYAP

#PUSH distr_request distr_reply distr_error
#DEF bell STRUCT BEGIN CHAR c; BYTE b REDEFINES c; END;
#SET bell:b 7 == ASCII value for BEL character.
#PUSH emphasis == Tells if event should be emphasized.

#PUSH distr^name
#PUSH cpu^num
#PUSH spi^err, used^len, ems^err

#DEF spi^buf STRUCT LIKE zems ddl msg buffer;
#DEF sav^buf STRUCT LIKE zems ddl msg buffer;

== This routine gets and checks the CPU number -------------
[#DEF get^cpu^num ROUTINE |BODY|
  #RESULT &
  [#CASE [#ARGUMENT/VALUE cpu^num/ NUMBER OTHERWISE]
    [1] [#IF cpu^num >= 0 and cpu^num <= 15 |THEN| -1 |ELSE| 0]
    [2] 0
  ]
} (get^cpu^num)

== End of get^cpu^num -------------------------------------

[#DEF spi^control TEXT |BODY|
  == Initialize spi^buf for distributor CONTROL command
  #SET spi^err [
      #SSINIT spi^buf [zems^val^ssid] ZEMS^CMD^CONTROL ]
  [#IF spi^err |THEN|
      *** [spi^err] on #SSINIT of CONTROL command
  ] == Place the connect-source-collector token in buffer.
  #SET spi^err [
      #SSPUT spi^buf ZEMS^TKN^CONNECT^SRC^COLL $0]
  [#IF spi^err |THEN|
      *** [spi^err] on #SSPUT of CONNECT^SRC^COLL
  ] == Place the load-filter-from-file token in the buffer.
  #SET spi^err [#SSPUT spi^buf ZEMS^TKN^FILTERFILE
      $system.filt.myapfobj]
  [#IF spi^err |THEN|
      *** [spi^err] on #SSPUT of FILTERFILE
  ] == Place the filter-parameter token in the buffer.
  #SET spi^err [#SSPUT /SSID [myap^val^ssid]/
spi^buf MYAP^TKN^PARAM^CPU [cpu^num]]

[IF spi^err |THEN|
  *** [spi^err] on #SSPUT of CONNECT^SRC^COLL]
== Send the entire buffer to the distributor.
#APPENDV distr_request spi^buf
#EXTRACTV distr_reply spi^buf
[IF distr_error |THEN|
  *** [distr_error] sending CONTROL command]

== Change buffer length to what was declared.
#SET spi^err [  
  #SSPUT spi^buf ZSPI^TKN^RESET^BUFFER ZEMS^VAL^BUFLEN]
[IF spi^err |THEN|
  *** [spi^err] on #SSPUT of RESET^BUFFER]
== Response is in buffer -- check for return code
#SETMANY spi^err _{count} ems^err & , [#SSGET /INDEX 1/ spi^buf ZSPI^TKN^RETCODE]  
[IF spi^err |THEN|
  *** [spi^err] on #SSGET of RETCODE]
[IF ems^err |THEN|
  *** [ems^err] RETCODE on CONTROL command]
}

== This routine displays the event message ---------------

[DEF displ^event ROUTINE |BODY| == <event^buf> is param 1
== Adds bell to output to indicate EMPHASIS.
  #OUTPUT     == Skip a line
  [#IF emphasis |THEN| #OUTPUT/HOLD/ [bell]]
== Generated display text from the event message
  [#OUTPUT [#EMSTEXT/WIDTH 78 == Displayable line length
  ,LINES 2/ == Number of display lines
  [#REST] ]]
}

== End of displ^event  -----------------------------

== This routine sends and processes GETEVENT requests -----

[DEF getevent^loop ROUTINE |BODY| #FRAME  
#DEF event^buf STRUCT LIKE zems^ddl^msg^buffer;  
#PUSH event_offset
== Initialize spi^buf for GETEVENT command message
#SET spi^err [  
  #SSINIT spi^buf [zems^val^ssid] ZEMS^CMD^GETEVENT]
[IF spi^err |THEN|
  *** [spi^err] on #SSINIT of GETEVENT command]
== Save the original command
#SETBYTES sav^buf spi^buf

== Begin loop that gets and displays event messages
[#LOOP |WHILE| -1 |DO|

== Send GETEVENT command message to the distributor.
#APPENDV distr_request spi^buf
#EXTRACTV distr_reply spi^buf
[#IF distr_error |THEN|
  *** [distr_error] sending GETEVENT]

== Protect against a longer buffer in the distributor
#SET spi^buf:z^BUFLEN zems^val^BUFLEN
#SET spi^err [#SSPUT spi^buf ZSPI^TKN^RESET^BUFFER
  ZEMS^VAL^BUFLEN]
[#IF spi^err |THEN|
  *** [spi^err] on #SSPUT of RESET^BUFFER]

== Process GETEVENT response message
#SETMANY spi^err (count) ems^err &
  , [#SSGET /INDEX 1/ spi^buf ZSPI^TKN^RETCODE]
[#IF spi^err |THEN|
  *** [spi^err] on #SSGET of RETCODE]
[#IF ems^err |THEN|
  *** [ems^err] RETCODE on GETEVENT command]

== Get the offset of the event-message token
#SETMANY spi^err event_offset &
  , [#SSGET /INDEX 1/ spi^buf ZSPI^TKN^OFFSET &
    ZEMS^TKN^EVENT]]
[#IF spi^err |THEN|
  *** [spi^err] on #SSGET of OFFSET of EVENT]

== Move past the length part (variable length token)
#SET event_offset [#COMPUTE event_offset + 2
  - [#VARIABLEINFO/OFFSET/event^buf:z^filler]]
#SETBYTES event^buf &
  spi^buf:z^filler([event_offset]:[#COMPUTE
    [spi^buf:z^occurs] - 1])

== Decide if event deserves emphasis as critical
#SETMANY spi^err (count) emphasis &
  , [#EMSGET event^buf ZEMS^TKN^EMPHASIS]
[#IF spi^err |THEN|
  *** [spi^err] on #EMSGET of EMPHASIS]

== Call procedure that displays the event
displ^event event^buf

== Move context token from reply into copy of
== original command
#SETMANY spi^err , [#SSMOVE /SINDEX 1, DINDEX 1/
  spi^buf == Source
  sav^buf == Destination]
Example of Retrieving Event Messages

ZSPI^TKN^CONTEXT]

== If no CONTEXT token, we are at end
[#IF spi^err = ZSPI^ERR^MISTKN |THEN|
  #UNFRAME
  #RETURN
]
[#IF spi^err |THEN|
  *** [spi^err] on #SSMOVE of CONTEXT]

== Move updated command to spi^buf for next time
== through loop
#SETBYTES spi^buf sav^buf

} == WHILE
  #UNFRAME
} {getevent^loop}
== End of getevent^loop -----------------------------------

== This is the main program ---------------------------------

== Start the distributor process
$SYSTEM.SYSTEM.EMSDIST/NAME, NOWAIT/ TYPE CONSUMER
#SET distr^name [#PROCESS]

== Open the distributor for command and response messages
#SET distr_error [#REQUESTER /WAIT [zems^val^buflen]/
  READ [distr^name].#ZSPI
  distr_error
  distr_reply
  distr_request]

[#IF distr_error |THEN|
  *** [distr_error] Opening distributor [distr^name]]

== Prompt user for the CPU number
[#LOOP |WHILE| NOT [get^cpu^num [#INPUT CPU number?:  ]]
  DO |
]

== Send CONTROL command message to manage the distributor
spi^control

== Send GETEVENT command messages, display event messages
getevent^loop

#UNFRAME {myap}
This file contains the main COBOL source:

* File name: MYAPMAIN -------------------------------------
* The MYAP application does the following:
*   - Starts and opens a consumer distributor
*   - Prompts you at your terminal for a CPU number
*   - Issues SPI CONTROL command messages to
*       connect the collector $0
*       load a filter file
*       load a filter param
*       (cpu-number from user)
*   - Performs the following steps in a loop:
*     -- Issues an SPI GETEVENT command message
*     -- Displays an event message at your terminal
* MYAP waits for new events from the given CPU and displays
* them as they arrive.
* To terminate the program, press the BREAK key
* and type STOP.
* The filter file must be $SYSTEM.FILT.MYAPFOBJ.
* ?SEARCH $system.system.cobollib
?CONSULT $system.system.cobolex0
?SAVE STARTUP
IDENTIFICATION DIVISION.
PROGRAM-ID. dist-initfc.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. Tandem.
OBJECT-COMPUTER. Tandem.
SPECIAL-NAMES.
   FILE "$system.system.cbl85utl" IS cobut1.
* constants
REPLACE
   ==evt-text-len== BY ==78==
   ==num-evt-lines== BY ==2==
.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT distr-process ASSIGN TO #DYNAMIC.
DATA DIVISION.
FILE SECTION.
FD distr-process
RECORD IS VARYING.
COPY zems-ddl-msg-buffer OF $system.zspidef.zemscob
  REPLACING ==ZEMS-DDL-MSG-BUFFER== BY ==distr-rec==.

WORKING-STORAGE SECTION.
01 actual-len-g.
  02 actual-len NATIVE-2 OCCURS num-evt-lines TIMES.
01 coll-name.
  02 PIC X(8) VALUE "$0".
  02 PIC X(8) VALUE SPACES.
  02 PIC X(8) VALUE SPACES.
01 cpu-num NATIVE-2.
01 distr-name PIC X(24) VALUE SPACES.
01 distr-name-qual PIC X(24) VALUE SPACES.
01 distr-prog-file PIC X(30)
    VALUE "$SYSTEM.SYSTEM.EMSDIST".
01 ems-err NATIVE-2.
  88 end-of-loop VALUE 1.
01 error-flag NATIVE-2.
01 evt-text-buf.
  02 evt-text-line PIC X(evt-text-len)
    OCCURS num-evt-lines TIMES.
01 filt-name.
  02 PIC X(8) VALUE "$SYSTEM".
  02 PIC X(8) VALUE "FILT".
  02 PIC X(8) VALUE "MYAPFOBJ".
01 ind PIC 99 COMP.
01 option NATIVE-2 VALUE 1.
01 text-stat NATIVE-4.
  02 text-stat-high NATIVE-2.
  02 text-stat-low NATIVE-2.
01 offset NATIVE-2.
01 cp-list PIC 9(9) COMP VALUE 0.
01 string-portion PIC X(6) VALUE "STRING".
01 string-value PIC X(13) VALUE "TYPE CONSUMER".

?NOLIST
COPY constants OF $system.zspidef.zspicob.
COPY constants OF $system.zspidef.zemscob.
COPY constants OF myapcob.

?LIST
EXTENDED-STORAGE SECTION.
COPY zems-ddl-msg-buffer OF $system.zspidef.zemscob
  REPLACING ==ZEMS-DDL-MSG-BUFFER== BY ==event-buf==.
COPY zems-ddl-msg-buffer OF $system.zspidef.zemscob
  REPLACING ==ZEMS-DDL-MSG-BUFFER== BY ==save-buf==.

PROCEDURE DIVISION.
startt.

* Get an unused process name
  ENTER "CREATEPROCESSNAME" USING distr-name
* Prepare the startup message with "TYPE" parameter for distributor (CONSUMER).
  ENTER "PUTSTARTUPTEXT" USING string-portion,
  string-value,
  cp-list
  GIVING error-flag
  
  IF error-flag < ZERO
    ENTER "DEBUG" END-IF
  
* Start the distributor
  ENTER "CREATEPROCESS" USING distr-prog-file, distr-name,
  option GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG" END-IF
  
* Construct the name to be used to open the distributor
  MOVE SPACES TO distr-name-qual
  STRING distr-name DELIMITED BY SPACES,
  ".#ZSPI" DELIMITED BY SIZE INTO distr-name-qual
  
  ENTER "COBOLASSIGN" USING distr-process, distr-name-qual
  GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG" END-IF
  
  OPEN I-O distr-process
  
  DISPLAY "Enter CPU number"
  ACCEPT cpu-num
  
* Initialize distr-rec for distributor CONTROL command
  ENTER "SSINIT" USING distr-rec, zems-val-buflen,
  zems-val-ssid, zspi-val-cmdhdr, zems-cmd-control
  GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG" END-IF
  
* Place the connect-source-collector token in buffer
  ENTER " SSPUT" USING distr-rec,
  zems-tkn-connect-src-coll, coll-name GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG" END-IF
  
* Place the load-filter-from-file token in the buffer
  ENTER " SSPUT" USING distr-rec, zems-tkn-filterfile,
  filt-name GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG" END-IF
  

* Place the filter-parameter token in the buffer
  ENTER "SSPUT" USING distr-rec, myap-tkn-param-cpu
cpu-num, OMITTED, myap-val-ssid GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG"
  END-IF

* Send command message to distributor and get response
  READ distr-process
  WITH PROMPT distr-rec AT END ENTER "DEBUG"
  END-READ

* Protect against a longer buffer in the distributor
  ENTER "SSPUT" USING distr-rec, zspi-tkn-reset-buffer,
zems-val-buflen GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG"
  END-IF

* Response is in buffer - check for return code
  ENTER "SSGET" USING distr-rec, zspi-tkn-retcode,
  ems-err, 1 GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG"
  END-IF
  IF ems-err NOT = ZERO
    ENTER "DEBUG"
  END-IF

* Send GETEVENT commands and display event messages
  PERFORM getevent-loop
  CLOSE distr-process
  STOP RUN
.

getevent-loop.

* Initialize distr-rec for GETEVENT command message
  ENTER "SSINIT" USING distr-rec, zems-val-buflen,
zems-val-ssid, zspi-val-cmdhdr, zems-cmd-getevent
  GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG"
  END-IF

* Save the original command
  MOVE distr-rec TO save-buf

  PERFORM WITH TEST AFTER UNTIL end-of-loop

* Send GETEVENT command to the distributor and get response
  READ distr-process WITH PROMPT distr-rec
  AT END ENTER "DEBUG"
  END-READ
* Protect against a longer buffer in the distributor
  ENTER "SSPUT" USING distr-rec,
    zspi-tkn-reset-buffer,
    zems-val-buflen GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG"
  END-IF

* Check return code in GETEVENT command response
  ENTER "SSGET"
    USING distr-rec, zspi-tkn-retcode, ems-err
    GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG"
  END-IF
  IF ems-err NOT = ZERO
    ENTER "DEBUG"
  END-IF

* Get the offset of the event-message
  in the response buffer

  ENTER "SSGET"
    USING distr-rec, zspi-tkn-offset,
      zems-tkn-event, 1, offset
    GIVING error-flag
  IF error-flag NOT = ZERO
    ENTER "DEBUG"
  END-IF

* Move the event message to a buffer of its own.
  (+2 reflects that event is a variable-length
  token; +1 adjusts for the COBOL counting offset
  from 1 rather than from 0)
  MOVE distr-rec(offset + 2 + 1:) TO event-buf

* Get the display text for the event
  ENTER "EMSTEXT" USING event-buf,
    evt-text-buf,
    evt-text-len,
    num evt-lines,
    actual-len-g,
    OMITTED,
    2
    GIVING text-stat
  IF text-stat-high = 0
    AND text-stat-low NOT = 0
    ENTER "DEBUG"
  END-IF

* Display the display text
  PERFORM VARYING ind FROM 1 BY 1
    UNTIL ind > num evt-lines
    OR actual-len (ind) = -1
DISPLAY evt-text-line (ind) (1: actual-len (ind))
END-PERFORM

* Move context token from the reply
* into the copy of the orig. command
ENTER "SSMOVE"
   USING zspi-tkn-context, distr-rec, 1, save-buf, 1
   GIVING error-flag
EVALUATE error-flag
   WHEN zspi-err-mistkn
*     Done if no context token
   SET end-of-loop TO TRUE
   WHEN 0
* Move the updated command to distr-rec
* for next time through the loop
   MOVE save-buf TO distr-rec
   WHEN OTHER
   ENTER "DEBUG"
END-EVALUATE
END-PERFORM
.

C Source File

This file contains the main C source:

/* File name: MYAPMAIN --------------------------------------------------------------
 * The MYAP application does the following:
 * - Starts and opens a consumer distributor
 * - Prompts you at your terminal for a CPU number
 * - Issues CONTROL command messages to
 *      connect the collector $0
 *      load a filter file
 *      load a filter param
 *      (cpu-number from user)
 * - Performs the following steps in a loop:
 *      -- Issues a GETEVENT command message
 *      -- Displays an event message at your terminal
 * MYAP waits for new events from the given CPU and displays them as they arrive.
 * To terminate the program, press the BREAK key and type STOP.
 * The filter file must be $SYSTEM.FILT.MYAPFOBJ.
*/
Example of Retrieving Event Messages

C Source File

#include <stdio.h>     nolist
#include <string.h>    nolist
#include <stdlib.h>    nolist
#include <memory.h>    nolist
#include <tal.h>       nolist   /* CCG, etc. */
#include <CEXTDECS(CLOSE,CREATEPROCESSNAME,DEBUG,FILEINFO)> nolist
#include <CEXTDECS(NEWPROCESS,OPEN,STOP,WRITE,WRITEREAD)> nolist
#include <CEXTDECS(EMSGETTKN,EMSTEXT)> nolist
#include <CEXTDECS(SSGETTKN,SSMOVETKN,SSPUTTKN,SSINIT,SSNULL)> nolist

/*  Change to refer to local DSV where ZSPIDEF subvol exists */
#include "$System.Zspidef.ZSPIC"  nolist
#include "$System.Zspidef.ZEMSC"  nolist

#pragma PAGE "Globals and defines"
#include "MYAPC"     /* Includes DDL generated from MYAPDDL */

/* These defines are used for zems structs to shorten names */
define EMSBUFDEF     zems_ddl_msg_buffer_def
#define SPIERRDEF     zspi_ddl_error_def
#define FALSE 0
#define TRUE  (!FALSE)

/* The following struct is used to pass a startup message to */
/* the process started by newprocess. The C initialization */
/* code conveys this information as argc/argv information */
/* and disposes of the original startup message. */
typedef struct startup_msg {
  int msgcode;
  int defaults[8];
  int infile[12];
  int outfile[12];
  char params[35];          /* leave room for some params */
} CISTART;

/* ---------- Globals ----------------- */
/*    recfn; is not used in the C version. C processes startup */
int   distr;         /* distributor file number */
error;

/* install loc * /
char distr_prog_file[25] = "$SYSTEM SYSTEM EMSDIST ";
/* starts blank */
char distr_proc_name[25] = " ";
char *distr_qual = &distr_proc_name[8]; /* for #zspi */

int cpu_num, 
used_len,
ems_err;
/* spi buffers and such */
int spi_err;
EMSBUFDEF *spi_buf;
EMSBUFDEF *sav_buf;
SPIERRDEF *err_buf;

int ibuflen = ZEMS_VAL_BUFLEN;
int msgcount = 0;
int msglimit = 0; /* 0 represents no limit. 100 is typical */
#define EVT_TEXT_LEN   78
#define NUM_EVT_LINES  2
char evt_text_buf[EVT_TEXT_LEN * NUM_EVT_LINES];
int actual_len[NUM_EVT_LINES];
/* Declare the ssids using the typedefs from the DDL output */
ze ms_val_ssid_def zems_val_ssid;
myap_val_ssid_def myssid;

char coll_name[25] = "$0 ";
char filt_name[25] = "$SYSTEM FILT MYAPFOBJ";

#pragma PAGE "init_startup()"
/* -------- init_startup --------------
 * Recreate the startup structure discarded by C.
 * C retains the information we need and it can be accessed
 * by GETENV(). Pass: nothing. Returns: a pointer to the
 * dynamic storage for ci_startup msg.
 */
CISTART *init_startup(void)
{
    CISTART *ci_startup;
    lowmem char ExternFile[34];
    lowmem int TempDefaults[9];

    ci_startup = (CISTART *) malloc( sizeof (CISTART) );
    if (ci_startup == NULL)
        DEBUG();

EMS Manual—426909-005
A-26
/* Note that the following code is order-dependent since using FNAMEEXPAND to directly set the 'defaults' field overwrites the front of 'infile' */
strcpy((char *)TempDefaults, "$SYSTEM SYSTEM ");
strcpy(ExternFile, (char *)getenv("DEFAULTS"));
strcat(ExternFile, ".X");
FNAMEEXPAND(ExternFile, ci_startup->defaults, TempDefaults);
strcpy(ExternFile, (char *)getenv("STDIN"));
FNAMEEXPAND(ExternFile, ci_startup->infile, TempDefaults);
strcpy(ExternFile, (char *)getenv("STDOUT"));
FNAMEEXPAND(ExternFile, ci_startup->outfile, TempDefaults);

/* Don't forget to set the msgcode to -1 (startup msg) */
  ci_startup->msgcode = -1;
/* Plant a NULL char at the beginning of the params */
  ci_startup->params[0] = 0;

  return(ci_startup);
}

#pragma PAGE "get_cpu_num()"
/* --------- get_cpu_num -----------
 * Prompts the user for a valid cpu number, which can be
 * passed as a parameter to the filter.
 */
void get_cpu_num(void)
{
  int errval;
  int got_it = FALSE;

  while ( !got_it ) {
    /* prompt the user for the cpu number to use */
    printf("\Cpu number?: ");
    errval = scanf("%d", &cpu_num);

    if (errval == EOF)
      STOP();

    if ( (errval != 0) && (cpu_num >= 0) )
      got_it = TRUE;
  }
}

#pragma PAGE "send_spi_cmd()"
/* ----------- send_spi_cmd -------------
 * Puts the finishing touches on an SPI command that has
 * been prepared by another procedure. Sends the command to
 * the distributor and checks the command response.
 */
void send_spi_cmd(void)
{
  int ccval;  /* for cc return from writeread */
Example of Retrieving Event Messages

C Source File

/* Determine how much buffer was used */
spi_err = SSGETTKN((int *)spi_buf, ZSPI_TKN_USEDLEN, 
(char *)&used_len);

if (spi_err != ZSPI_ERR_OK)
    DEBUG();

/* Send the used part to the distributor */
ccval = WRITEREAD(distr, (int *)spi_buf, used_len, 
ZEMS_VAL_BUFLEN);
if (ccval != CCE)
    DEBUG();

/* reset the buffer length to what was declared for 
spi_buf */
spi_err = SSPUTTKN((int *)spi_buf, ZSPI_TKN_RESET_BUFFER, 
(char *)ibuflen);
if (spi_err != ZSPI_ERR_OK)
    DEBUG();

/* Response is in the buffer--check for return code */
spi_err = SSGETTKN((int *)spi_buf, ZSPI_TKN_RETCODE, 
(char *)&ems_err, 1);
if (spi_err != ZSPI_ERR_OK)
    DEBUG();
if (ems_err != 0)
    DEBUG();
}

#pragma PAGE "spi_cmd_set_source()"

/* ---------- spi_cmd_set_source -------------------
* Builds an SPI command that directs the distributor to use 
* a collector as a source of event messages.
*/

void spi_cmd_set_source(void)
{
    /* Initialize the spi_buf for distributor CONTROL 
    command */
    spi_err = SSINIT((int *)spi_buf, ZEMS_VAL_BUFLEN, 
    (int *)&zems_val_ssid, 
    ZSPI_VAL_CMDHDR, ZEMS_CMD_CONTROL);
    if (spi_err != ZSPI_ERR_OK)
        DEBUG();

    /* place the connect-source-collector token in buffer */
    spi_err = SSPUTTKN((int *)spi_buf, 
    ZEMS_TKN_CONNECT_SRC_COLL, 
    coll_name);
    if (spi_err != ZSPI_ERR_OK)
        DEBUG();

    /* Send the command to the distributor */

Example of Retrieving Event Messages

C Source File

send_spi_cmd();

#pragma PAGE "spi_cmd_load_filter()"
/*  Builds an SPI command that loads a filter into the
  distributor and passes one filter parameter.  */
void spi_cmd_load_filter(void)
{
  /* Initialize the spi_buf for distributor CONTROL
     command */
  spi_err = SSINIT( (int *)spi_buf, ZEMS_VAL_BUFLEN,
                    (int *)&zems_val_ssid,
                    ZSPI_VAL_CMDHDR, ZEMS_CMD_CONTROL );
  if (spi_err != ZSPI_ERR_OK)
     DEBUG();

  /* place the load filter token in the buffer */
  spi_err = SSPUTTKN( (int *)spi_buf, ZEMS_TKN_FILTERFILE,
                      filt_name );
  if (spi_err != ZSPI_ERR_OK)
     DEBUG();

  /* place the filter parameter token in the buffer */
  spi_err = SSPUTTKN( (int *)spi_buf, MYAP_TKN_PARAM_CPU,
                      (char *)&cpu_num, , (int *)&myssid);
  if (spi_err != ZSPI_ERR_OK)
     DEBUG();

  /* Send the command to the distributor */
  send_spi_cmd();
}

#pragma PAGE "displ_event()"
/*  This function displays an event message on the home
     terminal.  Pass:  The event buffer pointer to convert to
     a message.  */
void displ_event(EMSBUFDEF *event_buf)
{
  unsigned long etxt_stat;
  # define high(x) (x > 16)
  # define low(x) (x & 0xFFFF)

  char       text[EVT_TEXT_LEN+1]; /* needed for printf() call */
  int        i;

  /* Generate display text from the event message */
  etxt_stat = EMSTEXT( (int *)event_buf,
                      evt_text_buf,
                      EVT_TEXT_LEN, /* displayable
                      line len */
                      NUM_EVT_LINES, /* no. of display
Example of Retrieving Event Messages

/* check for EMSTEXT calling errors */
if ( (high(etxt_stat) == 0) && (low(etxt_stat) != 0) )
    DEBUG();

for(i=0; i < NUM_EVT_LINES; i++) {
    if( actual_len[i] != -1 ) {
        movmem(&evt_text_buf[i*EVT_TEXT_LEN], text,
            actual_len[i]);
        /* add null to conform to C convention */
        text[ actual_len[i] ] = 0;
        printf("%s
", text);
    }
}

#pragma PAGE "getevent_loop()"
/* ---------- getevent_loop() ------------------------
* This procedure consists of a loop to retrieve event
* messages. Each time through the loop, procedure gets an
* event message and calls displ_event to display it at the
* terminal. The procedure returns in the following cases:
* (1) the GETEVENT response contains a ZEMS_TKN_PASSVAL of 0
*     (from a PASS 0 filter statement),
* (2) the GETEVENT response contains an EOF warning, or the
*     messages-processed limit (msglimit) is exceeded.
*/
int getevent_loop(void)
{
    EMSBUFDEF *event_buf;  /* pointer to storage in stack */
    /* Byte offset to event returned by SSGETTKN */
    int byteoffset;
    unsigned long tkn;

    /* Initialize spi_buf for GETEVENT SPI command */
    spi_err = SSINIT( (int *)spi_buf, ZEMS_VAL_BUFLEN,
        (int *)&zems_val_ssid,
        ZSPI_VAL_CMDHDR, ZEMS_CMD_GETEVENT );

    if (spi_err != ZSPI_ERR_OK)
        DEBUG();

    /* Save the original command */
    movmem( (char *)spi_buf, (char *)sav_buf, sizeof(EMSBUFDEF) );

    /* Begin loop that gets and displays event messages */
    while ( (msgcount < msglimit) || (msglimit == 0) ) {
        msgcount++;

        /* Send GETEVENT command to distributor */
Example of Retrieving Event Messages

send_spi_cmd();

/* Find the event message within the GETEVENT response */
tkn = ZEMS_TKN_EVENT;

/* Returns the offset of the event in the spi buffer via event_buf_loc. */
spi_err = SSGETTKN((int *)spi_buf, ZSPI_TKN_OFFSET, (char *)&tkn, 1, (int *)&byteoffset);
if (spi_err != ZSPI_ERR_OK)
    DEBUG();

event_buf = (EMSBUFDEF *)((char *)(spi_buf) + byteoffset + 2);

/* Display the event message at your terminal */
displ_event( event_buf );

/* Save CONTEXT token from this GETEVENT response for * /
/* the next GETEVENT request */
spi_err = SSMOVETKN(ZSPI_TKN_CONTEXT, (int *)spi_buf, 1, /* Source */
                   (int *)sav_buf, 1); /* Destination */
if (spi_err != ZSPI_ERR_OK)
    DEBUG();

/* Move the updated command to spi_buf for next time */
/* through the loop */
movmem((char *)sav_buf, (char *)spi_buf, sizeof(EMSBUFDEF));

} /* end while */
return(0);
} /* end getevent_loop() */

#pragma PAGE "main program code"
/* ---------- main: -------------------*/
/* Starts a distributor and calls procedures that use SPI */
/* commands to connect a source collector, to load a filter, */
/* and to position the distributor within the log files. */
/* Then the getevent_loop procedure is called to retrieve */
/* and process event messages. */

main(int argc, char **argv)
{
    /* local pointer to the startup message struct */
    CISTART *startup;
    char *cptr;
    int ccval = 0;
    static char zspi_name[] = "#ZSPI";

    /* Initialize subsystem IDs */
    cptr = strncpy(zems_val_ssid.u_z_filler.z_filler, ZSPI_VAL_TANDEM, 8);
    zems_val_ssid.z_number = ZSPI_SSN_ZEMS;
zems_val_ssid.z_version = ZEMS_VAL_VERSION;
cptr = strncpy(myssid.u_z_filler.z_filler, MYAP_VAL_OWNER, 8);
myssid.z_number = MYAP_SSN_MYAP;
myssid.z_version = MYAP_VAL_VERSION;

/* malloc some memory for spi buffers */
spi_buf = (EMSBUFDEF *) malloc( sizeof(EMSBUFDEF) );
if (spi_buf == NULL) DEBUG();
sav_buf = (EMSBUFDEF *) malloc( sizeof(EMSBUFDEF) );
if (sav_buf == NULL) DEBUG();
err_buf = (SPIERRDEF *) malloc( sizeof(SPIERRDEF) );
if (err_buf == NULL) DEBUG();

/* Not necessary to open home term in C. */
/* Not necessary to read startup message from C. Stored in env ptrs */

/* Initialize startup struct areas that will contain volume */
/* and subvol names to spaces for passing to new process */
startup = init_startup();

/* Create a name for the distributor process */
ccval = CREATEPROCESSNAME( (int *)distr_proc_name );
if (ccval != CCE) {
    printf("\n Unable to Create process name.\n");
    DEBUG();
}
NEWTASK( (int *)distr_prog_file, , , , , (int *)&error,
    (int *)distr_proc_name);
if ( (error > 8) != 0 ) {
    printf(" Newprocess returned %xh\n",error);
    DEBUG();
}

/* Open the newprocess to pass the startup message */
ccval = OPEN( (int *)distr_proc_name, (int *)&distr );
if (ccval != CCE) {
    printf(" Open returned non-zero ccval: %d.\n",ccval);
    DEBUG();
}

/* Add "TYPE CONSUMER" to startup msg to send to EMSDIST. */
strcat(startup->params, "TYPE CONSUMER");

/* Now write the startup struct */
ccval = WRITE( distr, (int *)startup, sizeof(CISTART) );
FILEINFO(distr, (int *)&error);
if ( !(error == 0 || error == 70) ) {
    printf("Write returned error: %d\n",error);
    DEBUG();
}

/* We can now close the "standard" interface to
   distributor */
CLOSE(distr);

/* Prompt user for the cpu number */
get_cpu_num();

/* prepare to re-open the distributor spi interface */
movmem( zspi_name, distr_qual, 5 );
ccval = OPEN( (int *)distr_proc_name, (int *)&distr );
if (ccval != CCE) {
    printf("Open of %s returned non-zero ccval: %d\n", 
           distr_proc_name, ccval);
    DEBUG();
}

/* Tell collector to use log files as event-message source */
spi_cmd_set_source();

/* Load filter and filter parameter(s) into the distributor */
spi_cmd_load_filter();
getevent_loop();

CLOSE(distr);
}
Program Modifications

This subsection suggests how you can modify this example application to demonstrate different aspects of the programmatic interface. You might need to change the example by easy stages into a practical application.

Modifications—shown in TAL—can be used literally if your program is in TAL, or as a guide to program logic if your program is in TACL, COBOL, or C.

Processing Retrieved Event Messages

Displaying event messages, as done by the displ^event procedure, is one of many possible ways to process messages retrieved by an application. You might replace displ^event with a procedure that prepares a summary of events or with a procedure that takes action to correct problems. For example, a program could bring down a line when serious problems occur.

Filter Parameters

To change the example’s existing parameter or to add additional parameters:

- Modify the statements in the spi^cmd^load^filter procedure below the comment:

  ! Place the filter-parameter token in the buffer.

- Add similar groups of statements to pass additional parameters.

- Modify the definitions of parameter tokens in the DDL file and recompile the DDL file.

- Modify the parameter definitions in the filter and recompile the filter.

- Modify the get^cpu^num procedure to prompt the user for the new parameters.

If your filter requires no parameters (and if you have no other reasons to prompt the user), you can delete the get^cpu^num procedure and the call to it. You can also delete the DDL file unless your application needs it for other reasons, such as reporting events.

Tracking Earlier Events

As originally shown, the example retrieves messages as the collector receives them. The modifications discussed here cause the distributor to begin retrieving event messages generated four hours earlier and subsequently logged. After retrieving the logged event messages, the application continues to process event messages as they arrive.

To provide the application with the means to specify a time (starting position) to the distributor, insert this procedure after the spi^cmd^load^filter procedure:

?PAGE
PROC spi^cmd^set^position (time);
Fixed time;
BEGIN

! This procedure builds a command message that sets the
! position within the log file at which the distributor
! will start to retrieve event messages. The calling
! procedure specifies the position by the time parameter,
! which is in Julian-time-stamp format.

! Initialize spi^buf for distributor CONTROL command
spi^err := SSINIT (spi^buf, ZEMS^VAL^BUFLEN,
                              zems^val^ssid,
                              ZSPI^VAL^CMDHDR, ZEMS^CMD^CONTROL);
IF spi^err <> ZSPI^ERR^OK THEN
  CALL DEBUG;

! Time token will direct the distributor to position
! the log file at messages generated four hours ago
spi^err := SSPUTTKN(spi^buf, ZEMS^TKN^GMTTIME,
                          time);
IF spi^err <> ZSPI^ERR^OK THEN
  CALL DEBUG;

! Send the command message to the distributor
CALL send^spi^cmd;
END;

The spi^cmd^set^position positions the distributor at the event messages generated at
the time specified by the procedure parameter.

To call spi^cmd^set^position from the example, insert these statements in the main
procedure, just before the call to getevent^loop:

! Specify the position within the log files where
! event-message retrieval will begin: four hours ago.
julian^time^stamp :=
                                  JULIANTIMESTAMP ( ) - 14400000000F;
CALL spi^cmd^set^position (julian^time^stamp);

Also, insert this declaration in the dist^intfc (main) procedure:

Fixed julian^time^stamp;

On most active systems, this modification gives you a substantial set of event
messages. Try changing the filter to screen out large numbers of them.

To limit the number of messages displayed at your terminal, change the msglimit
declaration to:

INT msglimit:= 100;  ! Set to 100 to represent no limit

This limit provides a reasonable number to page through at your terminal. After finding
100 event messages that pass the filter, the application stops.
Stopping at the End of the Logged Messages

In the modifications discussed so far, the distributor waits for more event messages when it has read all the messages logged so far. The distributor has an option to stop when it reaches the end of the logged messages. The application must do two things to select this option:

- Request the distributor to send an end-of-log-file warning.
- Check whether it has received the warning.

To request the warning, insert these statements just before the comment “Save the original command” in the getevent^loop procedure:

```fortran
! Include the EOFSTOP token, which will cause the distributor
! to send an ZEMS^WRN^EOF token when it has read all event
! messages currently on hand.
ivalue := ZSPI^VAL^TRUE;
spi^err := SSPUTTKN(spi^buf, ZEMS^TKN^EOFSTOP,
   ivalue );
IF spi^err <> ZSPI^ERR^OK THEN
   CALL DEBUG;
```

These statements add the appropriate token to the GETEVENT command message. To test whether the application has received the warning, insert this procedure after the get^cpu^num procedure:

```fortran
?PAGE
INT PROC get^warning;
BEGIN
   INT warning;
      ! This procedure returns true (1) or false (0), depending
      ! on whether the GETEVENT response contains an error list
      ! with the warning token ZEMS^WRN^EOF or not,
      ! respectively. True indicates that the distributor has
      ! read all messages up to the present time.

      ! Return right away if no error list
      spi^err := EMSGETTKN(spi^buf,
         ZSPI^TKN^ERRLIST, , 1 );
      IF spi^err <> ZSPI^ERR^OK
         AND spi^err <> ZSPI^ERR^MISTKN THEN
         CALL DEBUG;
      IF spi^err = ZSPI^ERR^MISTKN THEN
         RETURN(false);

      ! Error list found, now retrieve ERROR token and look
      ! for an EOF warning code as a STOP request
      spi^err := EMSGETTKN(spi^buf,
         ZSPI^TKN^ERROR, err^buf, 1 );
      IF spi^err <> ZSPI^ERR^OK THEN
         CALL DEBUG;
      warning := (err^buf.z^error = ZEMS^WRN^EOF);
```

EMS Manual—426909-005
A-36
! Read ENDLIST token to pop out of error list
spi^err := EMSGETTKN(spi^buf,
                 ZSPI^TKN^ENDLIST, , 1 );
IF spi^err <> ZSPI^ERR^OK THEN
  CALL DEBUG;
RETURN(warning);
END;

Insert a call to get^warning in the getevent^loop procedure, just before the comment “Find the event message within the GETEVENT response”:

! Check for EOF warning
IF get^warning THEN
  RETURN;
END;

You can use the techniques shown in get^warning whenever you must search an error list for a certain token.

Using PASS Values

The distributor sends the values on filter PASS statements (called PASS values) to the application in the GETEVENT command response. PASS values indicate to the application which filter statement passed the current event message. After you insert these modifications, the application stops if it receives a PASS value of zero (zero is chosen arbitrarily) but continues to run when it receives all other PASS values (or no PASS value).

To search for a PASS value of zero, insert these statements in the getevent^loop procedure, just before the comment “Find the event message within the GETEVENT response”:

! Look in GETEVENT response for a PASSVAL of 0
passval := -1;
spi^err := EMSGETTKN(spi^buf, ZEMS^TKN^PASSVAL,
            passval, 1, , ZEMS^VAL^SSID);
IF spi^err <> ZSPI^ERR^OK
AND spi^err <> ZSPI^ERR^MISTKN THEN
  CALL DEBUG;
This statement sets passval to the PASS value if present or leaves passval at -1.

If you insert this test after the call to displ^event (a few statements beyond the last insertion), the message passed by the PASS 0 statement is the last event message displayed at your terminal, because the application stops:

! RETURN after displaying the PASS 0 event message
IF passval = 0 THEN
  RETURN;

Finally, insert this declaration in the getevent^loop procedure:

INT passval;
Both the value zero and the decision to stop are arbitrary conventions used between filter and application.
This appendix presents a working example of a subsystem like one you might write. The programs in this example generate event messages when subsystem-defined events occur. This example outlines the programming structure that your subsystems will require to report events. The example demonstrates how to use EMS procedures to build event messages and how to send those messages to the collector ($0) through use of the OPEN, CLOSE, and WRITEREAD procedures.

There are five files in this example:

- The MAKE TACL macro file to compile the user subsystem DDL and the TAL program.
- CXOURDDL is a DDL compile control file.
- XOURDDL is a DDL source file.
- SOURTMPL is a source template file.
- CSRC is a C source file.
- TACLSRC is a TACL source file.

The DDL file contains statements to support event-message creation. The file defines a token with an enumerated type and an extensible structured token.

### Testing the Program

1. Run the MAKE TACL macro. This compiles and links the C program and compiles the EMS template file, SOURTMPL.

2. Create an edit file, called INFILE, with two lines. The first line specifies the location of the current EMS template file that's installed on the system. The second line specifies, XOURTMPL, which is the template object file created from SOURTMPL.

   ```
   file $system.sys01.template
   file xourtmpl
   ```

3. Run TEMPLI to generate a EMS template file you can use for testing.

   ```
   RUN TEMPLI /IN INFILE/ TEMPR, TEMPNR
   ```
4. Run the C test program, COBJ, entering input with pauses greater than 10
seconds and greater than 30 seconds between lines. Terminate the program by
entering "quit". See below:

    $NSKOS KMZEMSEX 31> cobj
    1stMessage
    2ndMessage
    3rdMessage
    quit
    $NSKOS KMZEMSEX 32>

5. Execute the following TACL statement. Adding the DEFINE, _EMS_TEMPLATES,
effectively "installs" in your TACL session the TEMPNR template you created in the
TEMPLI step above.

    add define =_ems_templates, class map, file tempnr

6. Execute the EMS Distributor to display the EMS logfile that was created by the
Alternate Collector the COBJ program launched. The last three events displayed
are the ones generated by the COBJ program.

    $NSKOS KMZEMSEX 36> emsd distributor, log0001, te [#myterm]
    12-09-27 18:46:26 \BLPROD.$ACOL    *TANDEM.EMS.H01        000514 EMS: COLLECTOR \BLPROD.$ACOL SWITCHED LOG FILES FROM FILE TO \BLPROD.$NSKOS.KMZEMSEX.ZZEV0000 BY OPERATOR
    12-09-27 18:46:26 \BLPROD.$ACOL    TANDEM.EMS.H01        000522 EMS: ALTERNATE COLLECTOR \BLPROD.$ACOL, DISK
    12-09-27 18:46:50 \BLPROD.1,893    OURCO.OURSUBSS.3      000001 More than 10 seconds since the last input.
    File: $RECEIVE
    Last Message: 1stMessage
    Time since last message (in usec): 11788900
    12-09-27 18:47:03 \BLPROD.1,893    OURCO.OURSUBSS.3      000001 More than 10 seconds since the last input.
    File: $RECEIVE
    Last Message: 2ndMessage
    Time since last message (in usec): 12752049
    12-09-27 18:47:35 \BLPROD.1,893    *OURCO.OURSUBSS.3      000001 More than 30 seconds since the last input.
    File: $RECEIVE
    Last Message: 3rdMessage
    Time since last message (in usec): 32384456
    $NSKOS KMZEMSEX 37>
The MAKE TACL Macro File

This MAKE TACL macro file compiles the DDL and the TAL source. The ERRORFILE option is included to help you make corrections if your TAL program does not compile successfully.

```tacl
?tacl MACRO
#FRAME
#PUSH tools
#SET  tools $BFS001.B9FSNJA0
== MAKE file
== Event Generator DDL and TAL Compile OBEY command file
== compile the DDL
[tools].DDL /IN cxourddl, OUT $s.#ddl/
[tools].TEMPL /IN sourtmpl, OUT $s.#tmpl/ xourtmpl
== Compile the C event generator program
[tools].CPPCOMP /IN csrc, OUT $S.#c/ cobj; extensions, inspect, symbols, &
nomap,nolmap,runnable,optimize 0
#UNFRAME
```

The DDL Compile-Control File

These statements are in the DDL compile-control file:

```plaintext
! FILE CxourDDL
?DICT !
?NOLIST
?SOURCE $system.ZSPIDEF.ZSPIDDL
?SOURCE $system.ZSPIDEF.ZEMSDDL
?LIST ! TACL is for filters, TAL & COBOL for event generation
?TACL xourtac1 !, NOTIMESTAMP
?TAL xourtal !, NOTIMESTAMP
?COBOL xourcob !, NOTIMESTAMP
?C xourc !, NOTIMESTAMP
?SETSECTION CONSTANTS
?SOURCE xourddl( structs )
?SETSECTION
?SOURCE xourddl( constants )
?SETSECTION
?SOURCE xourddl( xour-ddl-evt-buffer )
?SETSECTION
?SOURCE xourddl( xour-enm-evt )
```
The DDL Source File

These statements are in the DDL source file:

```
!FILE xourDDL
?SECTION structs
!--------------------------------------!
! Data Definitions for Token Types    !
!--------------------------------------!
DEF xour-ddl-too-long-stats.
  02 occurrences           TYPE ZSPI-DDL-UINT.
  02 subsys-process-type   TYPE ZSPI-DDL-ENUM.
  02 ios                   TYPE ZSPI-DDL-INT2.
END

?SECTION constants
!-------------------!
! OUR token numbers !
!-------------------!
CONSTANT xour-TNM-io-file     VALUE IS 1.
CONSTANT xour-TNM-io-msg      VALUE IS 2.
CONSTANT xour-TNM-io-time     VALUE IS 3.
CONSTANT xour-TNM-too-long-stats VALUE IS 4.
!-------------------!
! OUR token codes    !
!-------------------!
TOKEN-CODE
  xour-TKN-io-file   VALUE IS xour-TNM-io-file
  TOKEN-TYPE IS zspi-typ-fname.
TOKEN-CODE
  xour-TKN-io-msg    VALUE IS xour-TNM-io-msg
  TOKEN-TYPE IS zspi-typ-string.
TOKEN-CODE
  xour-TKN-io-time   VALUE IS xour-TNM-io-time
  TOKEN-TYPE IS zspi-typ-int4.

!-------------------!
! OUR values         !
!-------------------!
CONSTANT xour-VAL-ourco              VALUE IS "OURCO   ".
CONSTANT xour-SSN-xour              VALUE IS 1.

* Enumeration of SUBSYS-PROCESS-TYPE in TOO-LONG-STATS.

! Input bound process
CONSTANT xour-VAL-proc-input-bound VALUE IS 1.

! Output bound process
CONSTANT xour-VAL-proc-output-bound VALUE IS 2.

* OURCO EVENT NUMBER VALUES

CONSTANT xour-EVT-time-too-long    VALUE IS 1.
CONSTANT xour-EVT-other           VALUE IS 2.

* DEFINES OUR CURRENT SUBSYSTEM VERSION
CONSTANT xour-val-version         VALUE IS 3. ! Rev 3

* DEFINES THE STRUCTURE FOR OUR SSID
DEF xour-VAL-SSID TACL SSID.
  02 Z-FILLER TYPE CHARACTER 8
     VALUE IS xour-VAL-ourco.

  02 Z-OWNER REDEFINES Z-FILLER
     TYPE ZSPI-DDL-CHAR8.
```

Example of Reporting Events

The DDL Source File
The SOURTMPL File

This file contains the source template.

VERSION: "Our Subsystem - 24AUG12"
SSID: XOUR-VAL-SSID
SSNAME: "OURSUBSS"

DEF_ENUM : ZEMS-TKN-EVENTNUMBER AS XOUR-ENM-EVT
MSG: ZEMS-TKN-EVENTNUMBER, XOUR-EVT-TIME-TOO-LONG
"More than <*IF 1> 10 <*ELSE> 30 <*ENDIF> seconds "
"since the last input.<*CR>"
"File: <2><*CR>"
"Last Message: <3><*CR>"
"Time since last message (in usec): <4>"
1: ZEMS-TKN-EMPHASIS <> ZSPI-VAL-TRUE
The C Source File

This file contains the C source code. The program decides an event has occurred if you take more than 10 seconds to give it data. If you take more than a half a minute, the program decides the event is critical.

Assumption: These files should be available in the same subvolume or directory with this file in order to compile and to create the object for this file:

- The DDL source file “xourddl”
- The file “xourc” (generate from DDL)

/*
 * The C source file
 * The following file contains the C source code:
 */
#include <stdio.h>   nolist
#include <string.h>  nolist
#include <stdlib.h>  nolist
#include <memory.h>  nolist
#include <tal.h>     nolist
#include <CEXTDECS(EMSINIT, EMSADDTOKENS, EMSADDTOKENMAPS)> nolist
#include <CEXTDECS(SSNULL, SSGETTKN, FILE_OPEN_, FILE_CLOSE_)> nolist
#include <CEXTDECS(WRITEREADX, WRITEX, DEBUG)>    nolist
#include <CEXTDECS(PROCESS_LAUNCH_, PROCESS_STOP_, JULIANTIMESTAMP)> nolist
#include <dlaunch.h> nolist
#include "$system.zsysdefs.zsysc(zsys_ddl_phandle)"
#include "$system.zsysdefs.zsysc(zsys_ddl_smgs_proccreate, process_constant)"
#include "$system.zspidef.ZEMSC(constants)" nolist
#include "xourc"  /* Includes DDL generated from xourddl */
#pragma PAGE "Globals and defines"
/* Define is used for zspi struct to shorten names */
define SPIIDLDEF zspi_ddl_sshid_def

/*
 * SSID's used in generating event message
 */
SPIIDLDEF our_sshid;
SPIIDLDEF ems_sshid;

/*
 * Maps are effectively constants like SSIDs
 */
define YOUR_MAP_TOO_LONG sizeof(xour_map_too_long_stats)
short map_too_long_stats[YOUR_MAP_TOO_LONG];

/*
 * This is the buffer in which event message are built
 */
xour_ddl_evt_buffer_def *event_buf;

/*
 * Global counter for TOO_LONG proc
 */
short evt_calls = 0; /* number of event */
Example of Reporting Events

The C Source File

```c
/*
 * The EMS Collector name and file number when OPEN'ed
 */
char coll_name[] = "$ACOL"; /* Name of the collector process */
short coll_filenum;

char in_msg[132];
short in_msg_len = sizeof(in_msg);

/*
 * SS or EMS PUT of values can't be literals
 */
short true_val = ZSPI_VAL_TRUE;
short false_val = ZSPI_VAL_FALSE;

/*
 * PROCESS_LAUNCH_ variables
 */
process_launchParms_def paramList = P_L_DEFAULT_PARMS_;
zsys_ddl_smgr_proccreate_def outputList;
short error, errorDetail, outputListLen;

#pragma PAGE "too_long"

/*
 * This procedure accepts the values for the time-too-long event, places
 * the values with the appropriate tokens in the event buffer using the
 * EMS procedure, and sends the event message to the Alternate Collector,
 * $ACOL, by using the file system FILE_OPEN_, WRITEREADX, and
 * FILE_CLOSE_ procedures.
 * If the event message can't be generated, or an error sending the event
 * message to $ACOL occurs, DEBUG is called.
 */
void too_long(char *file_name, long long io_time,
             char *io_msg, short io_len, long num_ios)
{
    short spi_error = 0; /* to save last SPI error for debug */
    short emphasis = ZSPI_VAL_FALSE; /* true if io_time is over a minute */
    short size; /* length of buffer */
    xour_ddl_too_long_stats_def *too_long_stats;
    short i;
    int fs_error;

    too_long_stats = (xour_ddl_too_long_stats_def *)
                     malloc (sizeof(xour_ddl_too_long_stats_def));

    event_buf = (xour_ddl_evt_buffer_def *)
                malloc (sizeof(xour_ddl_evt_buffer_def));

    strncpy(our_ssid.z_owner.u_z_c.z_c, XOUR_VAL_OURCO, 8);
    our_ssid.z_number = XOUR_SSN_XOUR; /* value is 1 */
    our_ssid.z_version = XOUR_VAL_VERSION; /* value is 3 */

    strncpy(ems_ssid.z_owner.u_z_c.z_c, ZSPI_VAL_TANDEM, 8);
    ems_ssid.z_number = ZSPI_SSN_ZEMS;
    ems_ssid.z_version = ZEMS_VAL_VERSION;

    for (i = 0; i < XOUR_MAP_TOO_LONG; i++)
        map_too_long_stats[i] = xour_map_too_long_stats[i];

    /* Begin to executable code of too_long proc */
    evt_calls = evt_calls + 1;

    /* Initialize buffer */
```
if (spi_error = EMSINIT((short *)event_buf, /* Event build buffer */
    XOUR_VAL_EVT_BUFLEN, /* total buffer length in bytes*/
    (short *)&our_ssid, /* ssid of reporting subsystem*/
    XOUR_EVT_TIME_TOO_LONG, /* event number */
    XOUR_TKN_IO_FILE, /* subject token code */
    file_name)) /* event subject value */
{
    printf(*** EMSINIT error = %d\n", spi_error);
    DEBUG();
}
/* If more than 1/2 minute, then critical */
if (io_time > 30000000)
    emphasis = ZSPI_VAL_TRUE;
/* Add several tokens at once */
if (spi_error = EMSADDTOKENS((short *)event_buf, /* our SSID by default */,
    /* Token code        Value         Size */
    XOUR_TKN_IO_MSG, io_msg , io_len,
    XOUR_TKN_IO_TIME, (char*)&io_time , ,
    ZEMS_TKN_EMPHASIS, (char*)&emphasis , ,
    ZEMS_TKN_CONSOLE_PRINT, (char*)&true_val, )
{
    printf (*** EMSADDTOKENS error = %d\n", spi_error);
    DEBUG();
}
/* Add a single map */
if (SSNULL((short *)map_too_long_stats, (char *)too_long_stats))
    DEBUG();
    too_long_stats->subsys_process_type = XOUR_VAL_PROC_OUTPUT_BOUND;
    too_long_stats->ios = num_ios;
    too_long_stats->occurrences = evt_calls;
if (spi_error = EMSADDTOKENMAPS((short *)event_buf, /* our SSID by default */,
    (short*)map_too_long_stats, (char *)too_long_stats))
{
    printf("*** EMSADDTOKENMAPS error = %d\n", spi_error);
    DEBUG();
}
/* The next gets the size of the event that was built to use for */
/* sending the event to the collector. */
if (spi_error = SSGETTKN((short *)event_buf, /* event buffer */
    ZSPI_TKN_USEDLEN, /* special spi token code */
    (char *)&size)) /* size is return in size */
{
    printf("*** SSGETTKN error = %d\n", spi_error);
    DEBUG();
}
/* Open the EMS Collector */
if (fs_error = (FILE_OPEN_(outputList.u_z_data.z_procname, 
    outputList.z_procname_len, &coll_filenum))
{
    printf (" Unsuccessful open of the collector \n");
    DEBUG();
}
Example of Reporting Events

The C Source File

/* Send the event. Note 0 read count */
if ( WRITEREADX(coll_filenum, (char *)event_buf, size, 0))
    DEBUG();
FILE_CLOSE_(coll_filenum); /* close the collector */
} /* end too_long */

#pragma PAGE "main program code"

/* This procedure represents a subsystem doing work. 
   It generates an event for an exceptional condition. */
int main(void)
{
l long long start_io, end_io;
l long io_calls = 0;
l char in_fname[] = "$RECEIVE               ";
l short count_read, error;
l long long io_time;
l bool bQuit = false;
l short msg_length;
l startup_msg_type my_startup_msg;

/* Get startup message */
if (error = get_startup_msg(&my_startup_msg, &msg_length))
    DEBUG();

/* Launch an alternate collector process */
paramList.program_name = "$SYSTEM.SYSTEM.EMSACOLL";
paramList.program_name_len = sizeof("$SYSTEM.SYSTEM.EMSACOLL") - 1;
paramList.name_options = ZSYS_VAL_PCREATOPT_NAMEINCALL;
paramList.process_name = "$ACOL";
paramList.process_name_len = sizeof("$ACOL") - 1;
if (error = PROCESS_LAUNCH_( &paramList, 
                                              &errorDetail, 
                                              &outputList, sizeof(outputList), 
                                              &outputListLen))
    DEBUG();

/* Open the EMS Collector and send it the startup message */
if (error = (FILE_OPEN_(outputList.u_z_data.z_procname, 
                                 outputList.z_procname_len, &coll_filenum)))
{
    printf (" Unsuccessful open of the collector \n");
    DEBUG();
}
if (WRITEX(coll_filenum, (char *) &my_startup_msg, msg_length))
    DEBUG();
FILE_CLOSE_(coll_filenum); /* close the collector */
start_io = JULIANTIMESTAMP();
while (bQuit == false)
{
    scanf ("%s", in_msg);
    end_io = JULIANTIMESTAMP();
    io_calls = io_calls + 1;
    /* Generate event if more than 10 seconds since last input */
    if (io_time > 10000000)
The TACL Source File

This file contains the TACL source code:

?TAACL MACRO
== File name: TACLSRC
==
#FRAME
==================================================================
== Define variables and initialize variable values
==
#PUSH vars log^subvol
#PUSH coll^err coll^name coll^reply coll^req
#PUSH ems^err

#SET coll^name $acol
#SET log^subvol $data.acollog

== Load SPI and EMS defined variables
== Use the ATTACHSEG command to gain access to an existing TACL segment file.
ATTACHSEG SHARED $dsv.zspisegf.zspisegf :spidir
ATTACHSEG SHARED $dsv.zspisegf.zemssegf :emsdir

== Set USE so that TACL will find the variables in :spidir and :emsdir
== USE also makes sure that : :UTILS :UTILS:TACL are included in #USELIST
USE [#USELIST] :spidir :emsdir

== Load XOURTACL definitions
[#IF NOT [#VARIABLEINFO /EXISTENCE/ xour^val^ourco] |THEN|
  #LOAD /LOADED vars/ xourtacl ]

== Initialize EMS ssid variable
#PUSH ems^ssid
#SET EMS^SSID [ZSPI^VAL^TANDEM].[ZSPI^SSN^ZEMS].[ZEMS^VAL^VERSION]

== Initialize our SSD
#PUSH our^ssid
#SET our^ssid [XOUR^VAL^OURCO].[XOUR^SSN^XOUR].[XOUR^VAL^VERSION]

== Define event buffer
#DEF event^buf STRUCT LIKE zems^ddl^evt^buffer;

== Define a structure for STRING token value
 [#DEF stringdata STRUCT
 BEGIN
 UINT byte^len ; == string byte length
 CHAR data (0:131); == string data
 ]

 too_long((char *)in_fname, io_time, in_msg,
 strlen (in_msg), io_calls);
/* .... Do subsystem processing on input ..........*/
if (strcmp (in_msg, "quit") == 0)
  bQuit = true;
/* Get next input */
  start_io = end_io;
} PROCESS_STOP_( (short *) &outputList.z_phandle );
}
Example of Reporting Events

END;

== Define other variables
#PUSH in^msg start^io end^io io^time emphasis io^calls evt^calls evt^txt

============================================================================
== Build XOUR^EVT^TIME^TOO^LONG event and send it to the collector.
==
[#DEF too^long TEXT |BODY|

== Set up XOUR^TKN^IO^FILE token value. This token is the subject
== of the event number XOUR^EVT^TIME^TOO^LONG.
#DEF in^fname STRUCT LIKE zspi^ddl^fname;
#SET in^fname [#IN]

== Initialize buffer
#SET ems^err [#EMSINITV event^buf [OUR^SSID] XOUR^EVT^TIME^TOO^LONG
XOUR^TKN^IO^FILE in^fname]
[#IF ems^err |THEN|
  *** #EMSINITV error: [ems^err]
]

== Fill in values for XOUR^TKN^IO^MSG
== The format is the string length followed by the text string.
== Because the first character in in^msg may be numeric, we need
== to separate the string length and the in^msg by using
== stringdata STRUCT.
#SET stringdata [#CHARCOUNT in^msg] [in^msg]
#SET ems^err [#SSPUTV event^buf XOUR^TKN^IO^MSG stringdata]
[#IF ems^err |THEN|
  *** #SSPUTV error: [ems^err]
]

== Fill in values for XOUR^TKN^IO^TIME
#SET ems^err [#SSPUT event^buf XOUR^TKN^IO^TIME [io^time]]
[#IF ems^err |THEN|
  *** #SSPUT error: [ems^err]
]

== Fill in values for ZEMS^TKN^EMPHASIS
[#IF io^time > 30000000 |THEN| #SET emphasis [zspi^val^true]
  ELSE |SET emphasis [zspi^val^false]
]
#SET ems^err [#SSPUT event^buf ZEMS^TKN^EMPHASIS [emphasis]]
[#IF ems^err |THEN|
  *** #SSPUT error: [ems^err]
]

== Add a map token in the buffer.
[DEF too^long^stats STRUCT LIKE xour^ddl^too^long^stats;]
#SET ems^err [#SSNULL xour^map^too^long^stats too^long^stats]
[#IF ems^err |THEN|
  *** #SSNULL error: [ems^err]
]

#SET too^long^stats:subsys^process^type [xour^val^proc^output^bound]
#SET too^long^stats:ios [io^calls]
#SET evt^calls [#COMPUTE [evt^calls] + 1]
#SET too^long^stats:occurrences [evt^calls]
[#SET ems^err
  [#SSPUTV event^buf xour^map^too^long^stats too^long^stats]
]  
[#IF ems^err |THEN|
Example of Reporting Events

The TACL Source File

```tcl
*** #SSPUTV error: [ems^err]
"
== Add a "ZEMS^TKN^TEXT" token in the buffer.
== The format is the string length followed by the actual string.
== Since the first character in evt^txt is not numeric, we can use
== [#CHARCOUNT evt^txt][evt^txt] for the ZEMS^TKN^TEXT token value.
#SET evt^txt I/O on [#IN] took [#COMPUTE [IO^TIME]/1000000] &
seconds to complete!
#SET ems^err [#SSPUT /SSID [ems^ssid]/
event^buf ZEMS^TKN^TEXT [#CHARCOUNT evt^txt][evt^txt]]
[#IF ems^err |THEN|
*** #SSPUT error: [ems^err]
]
== Send the event to the collector
#APPENDV coll^req event^buf
#EXTRACTV coll^reply event^buf
[#IF coll^err |THEN|
*** Error on sending the event to [coll^name]: [coll^err]
]
#output "Time^too^long" event message sent to [coll^name]!
} == end of DEF too^long

==========================================================================
=== This is the main program ===

== Start an alternate collector
[#IF NOT [#PROCESSEXISTS [coll^name]] |THEN|
$system.system.emsacoll /name [coll^name], nowait/ logsubvol [log^subvol]
]

== Start a requester that will writeread (READ means WRITEREAD) events
== to a collector.
#SET coll^err [#REQUESTER /WAIT [zems^val^buflen]/ READ [coll^name]
coll^err coll^reply coll^req]
[#IF coll^err |THEN|
#OUTPUT *** Opening collector [coll^name] error: [coll^err]
]

#SET #INPUTEOF 0
[#LOOP |WHILE| NOT [#INPUTEOF] |DO|
#SET start^io [#JULIANTIMESTAMP]
== Accept input
#SET in^msg [#INPUT ?]
#SET end^io [#JULIANTIMESTAMP]
#SET io^time [#COMPUTE [end^io] - [start^io]]
#SET io^calls [#COMPUTE [io^calls] + 1]

== If more than 10 seconds were used since last input, then generate an
== event.
[#IF io^time > 10000000 |THEN| == io^time > 10 seconds
too^long
]
#UNFRAME
```

Standard Event Sample Files

This appendix provides three sample files, formerly available in InfoWay, that you can copy and use on your system when creating standard events:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event External Specification Sample File</td>
<td>C-1</td>
</tr>
<tr>
<td>DDL Definitions Sample File</td>
<td>C-20</td>
</tr>
<tr>
<td>EMS Templates Sample File</td>
<td>C-30</td>
</tr>
</tbody>
</table>

When viewing this appendix using the NonStop Technical Library (NTL), you can select and copy the text of any of these sample files and paste it into a text editor. You can then modify it for use on your system.

**Event External Specification Sample File**

You can use this sample file, formerly stored in InfoWay as stndevet.txt, as a basis for creating your own subsystem event external specifications.

<<*************** GENERAL INSTRUCTIONS ******************>

This is the sample Subsystem Event External Specification file. It should be used in conjunction with, from the EMS Manual, Section 10, Generating Standard Events. Guidelines for writing a subsystem event ES are also discussed in Section 10; instructions for editing this file are provided in each section below.

General instructions on the use of this file:

1. Copy this file to your subvolume and rename it to <ss>ES, where <ss> is your subsystem acronym like ZSAM. Edit this file to create your event ES.

2. Print a formatted copy of this document.

3. All instructions in this file are enclosed by "<< ... >>". All text, including TFORM commands, enclosed by "<< ... >>" should be deleted before publishing your document.

4. Do not change any of the TFORM macros or margins defined here. If you do, you may get formatting errors.

5. Information in this document is presented in "tables" whenever possible so that it is easy to scan and change the information. Use "tables" whenever you can to add your information. Information in tables are enclosed in a box. The box is defined by the custom TFORM macros "\my_box_on" and "\my_box_off." Text inside the box will appear as you...
entered them, that is, the TFORM commands \"\SET JOIN OFF;SET JUSTIFY OFF\" are specified inside the box. So use the text editor to explicitly format your text. Make sure you set the "join width" in your editor to 70 characters or less; otherwise you will get the TFORM error message "LINE WAS LONGER THAN ...".

7. Edit the following TFORM "assign" statements for your ES:

\assign ES_version    "1.0"
\assign ES_publication_date "July 1993"
\assign name_subsys   "subsys-name"
\assign name_author   "author-name"
\assign name_company  "Tandem Computers Inc."
\assign addr_street_company "19333 Vallco Parkway"
\assign addr_city_state_company "Cupertino, CA 95014"

Proprietary and Confidential
Copyright (c) 1993
Printed from \(\text{sys_system}\).\(\text{sys_filename}\) on \(\text{sys_date}\)

The following are trademarks or service marks of Tandem Computers Incorporated:

6AX, BINDER, CROSSREF, Cyclone, DDL, ENABLE, ENCOMPASS, ENCORE, ENFORM, ENSCRIBE, ENVOY, ENVOYACP/XF, EXCHANGE, EXPAND, FASTSORT, FAXLINK, FOX, FOXII, Guardian, Guardian 90, Guardian 90XF, INSPECT, Laser-LX, LXN, MEASURE, MULTILAN, NetBatch, NonStop, NonStop 1+, NonStop CLX, NonStop EXT, NonStop EXT10, NonStop EXT25, NonStop SQL, NonStop II, NonStop TXP, NonStop VLX, PATHMAKER, PCFORMAT, PC LINK, PERUSE, PS MAIL, PS TEXT EDIT, PS TEXT FORMAT, RDF, SAFE, SAFEGUARD, SAFE-T-NET, T-TEXT, TACL, TAL, Tandem, TGAL, THL, TIL, TMF, TRANSFER, ViewPoint, VLX, WPLINK, XL8.

All brand names and product names are trademarks or registered trademarks of their respective companies.

All rights reserved. No part of this document may be reproduced in any form, including photocopying or translation to another language, without the prior written consent of Tandem Computers Incorporated.

Copyright (c) 1993 by Tandem Computers Incorporated. Printed in U.S.A.

<<customize the text to your company's copyright requirements>>


<<add documents your ES referred to here>>
<<add documents related to your ES here -- like your subsystem spec>>
This specification describes the Event Management System (EMS) events generated by the <<provide your subsystem name here>> subsystem. The overview for the subsystem is described in [<<provide the reference notation of the document that describes your subsystem here>>].

This document contains the following sections:

Introduction. This section.
Description of subsystem environment. It briefly describes what the subsystem is and how it generates EMS events. It also defines the subsystem ID and the EMS-related files this subsystem generates or depends on.
Reporting strategy for EMS events. It describes the standard management functions (as defined in Section 9, Standard Events, in the EMS Manual) and any non-standard management functions supported by this subsystem. It lists the event subjects, event types -- both standard and private -- and the event messages that this subsystem defined to support these management functions.
Description of EMS event tokens. It lists the standard SPI tokens, the standard EMS tokens, and the tokens from other subsystems that are needed by this subsystem. It also describes, in detail, all the tokens defined by this subsystem.
Description of EMS event messages. This section lists the tokens, describes the display, and any cause, effect and recovery text of events generated by this subsystem.
Obsoleted EMS events and tokens section. It lists the tokens and event messages obsoleted by this subsystem.
Describe what your subsystem does, how it generates EMS events, the subsystems it depends on in the definition of its event messages, and the subsystem (or program code) it depends on in generating its event messages.

Refer to Section 10, Generating Standard Events, in the EMS Manual, for a detailed list of topics to be included here. Supply whatever information about your subsystem that you think may help operators or management application developers in managing your subsystem.

The "Subsystem Environment Table" below defines the subsystem ID and the EMS-related data files that are provided by this subsystem or depended on by this subsystem.

<<Replace "TANDEM " with the registered name of your company. Replace "8" with the subsys number assigned to you. Replace "ZSAM" with the 4-char subsys acronym assigned to you.>>

<table>
<thead>
<tr>
<th>Company Name</th>
<th>&quot;TANDEM &quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem Number</td>
<td>8</td>
</tr>
<tr>
<td>Subsystem acronym</td>
<td>ZSAM</td>
</tr>
</tbody>
</table>

<<Replace "ZSAM" & "SSAM" with your 4-char subsystem acronym. Replace "*" with a blank character for any of the file not provided by your subsystem. For Tandem-subsystems, all files must be provided except "ZSAMPAS" which is optional.>>

* means file provided (i.e. released) by subsystem

* ZSAMDDDL  event tokens definition in DDL
* ZSAMC     event tokens definition in C
* ZSAMCOB   event tokens definition in COBOL
* ZSAMPAS   event tokens definition in PASCAL
* ZSAMTACL  event tokens definition in TACL
* ZSAMTAL   event tokens definition in TAL
* SSAMTMPL  event formatting template source
* ZSAMTMPL  event formatting template object

<<Add to list below any DDL files that your subsystem depends on. For example, if your subsystem uses the DDL definitions provided by Guardian, add ZGRDDDL to the list; if your subsystem uses shared code that generates EMS events, like the Common Kernel, then include ZCMKDDL to the list.>>

* ZSPIDDL   standard SPI tokens definition file
* ZEMSDDL   standard EMS tokens definition file

This section provides a quick overview of the management functions and the EMS events supported by this subsystem, as follows:
It lists the standard management functions, as defined in Requirements for Standard Events on page 9-3, in the EMS Manual, supported by this subsystem. It describes in detail the private management functions, if any, supported by this subsystem. It lists the EMS event subjects -- in DDL names -- for all the events generated by this subsystem. It lists the EMS event types -- standard and private event types -- that are generated by this subsystem. It lists the EMS event messages -- with a brief description of the event message -- that are generated by this subsystem.

The "Standard Management Functions Table" below lists all the standard management functions supported by EMS events of this subsystem.

<<All events generated by your subsystem must be for the support of one of the management functions listed below. Replace "*" with a blank character for the management function that your subsystem does not support.>>

* means the management function is supported by subsystem events

* Object State Monitoring Functions on page 9-4
* Reactive Problem Management Functions on page 9-5
* Proactive Problem Management Functions on page 9-8
* Production Requests Requiring Operator Attention on page 9-9

The "Private Management Functions Table" below lists all the private (i.e., non-standard) management functions, if any, supported by EMS events of this subsystem.

<<All events generated by your subsystem must be for the support of one of the standard management functions listed earlier or for one of these private management functions listed here.>

If you are not supporting any private management function with EMS events, state "none" here. Do not delete this subsection.

If you are supporting private management functions with EMS events, list each function here.

>>

<<

For each function listed in the table above, provide a new subsection below describing this function in detail. The type of information that should be provided is described in Task 4. Define Private Event Types for Your Subsystem on page 10-9, in the EMS Manual.

Use these TFORM commands to define each of your subsections:
The "Event Subjects Definition Table" below lists all the objects that EMS events of this subsystem are generated for. These objects are the event subjects and they follow ZEMS-TKN-SUBJECT-MARK, the subject marker, in every event message.

The event subjects are listed in alphabetical order by their DDL names.

<table>
<thead>
<tr>
<th>Event subject name</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZPWY-TKN-FNAME</td>
<td>a PATHWAY file</td>
</tr>
<tr>
<td>ZPWY-TKN-LMNAME</td>
<td>a LINKMON process</td>
</tr>
<tr>
<td>ZPWY-TKN-PMNAME</td>
<td>the PATHMON process</td>
</tr>
<tr>
<td>ZPWY-TKN-PROGNAME</td>
<td>a PATHWAY program</td>
</tr>
<tr>
<td>ZPWY-TKN-SCNAME</td>
<td>a PATHWAY server</td>
</tr>
<tr>
<td>ZPWY-TKN-TCPNAME</td>
<td>a Terminal Control Process (TCP)</td>
</tr>
<tr>
<td>ZPWY-TKN-TELLNAME</td>
<td>a tell message</td>
</tr>
<tr>
<td>ZPWY-TKN-TERMNAME</td>
<td>output device for a SCREEN COBOL pgm</td>
</tr>
<tr>
<td>ZPWY-TKN-UNKNOWNNAME</td>
<td>subject unknown to PATHMON</td>
</tr>
</tbody>
</table>

If your subject is a group name, specify the operator instructions for obtaining the member names from the group name here.

The "Event Types Definition Table" below lists all the event types of EMS events generated by this subsystem. Event types with names beginning with "ZEMS-VAL" are standard event types defined by EMS; all others are private event types defined by subsystems.

The event types are listed by their DDL names in alphabetical order.

<<All events generated by your subsystem must be of one of the types below. Replace "*" with a blank character for the event type that your subsystem does not support. Do not delete any entry from this table.>>

* means that event type is generated by this subsystem
Standard event type
(ZEMS-TKN-CONTENT-STANDARD)

* ZEMS-VAL-ATTN-COMPLETED Operator attention completed
* ZEMS-VAL-ATTN-NEEDED Operator attention needed
* ZEMS-VAL-OBJECT-AVAILABLE Object available
* ZEMS-VAL-OBJECT-UNAVAILABLE Object unavailable
* ZEMS-VAL-OTHER-STATE-CHANGE State change other than available/unavailable
* ZEMS-VAL-TRANSIENT-FAULT Transient fault
* ZEMS-VAL-USAGE-THRESHOLD Usage threshold

Private event type
(ZEMS-TKN-CONTENT-USER)

* ZEMS-VAL-DATA-DEBUG Debug data
* ZEMS-VAL-DATA-DIAGNOSTIC Diagnostic data
* ZEMS-VAL-DATA-TRACE Trace data

<<Add your private event types -- <ss>-VAL-<typename> here. >>

The "Event Numbers Definition Table" below lists all the EMS event messages generated by this subsystem. Event numbers are defined by their DDL symbolic event names (<ss>-EVT-<eventname>.) They are listed in alphabetical order under the event subject -- the event subject is also listed in alphabetical order -- that these events are generated for. Each event subject defined in the "Event Subjects Definition Table" is listed here.

If an event is defined for more than one subjects, the event is listed under each subject. The comment "*** see subject-so-and-so" is indicated in the "Brief Description" heading of the "later" subject.

Listed with each event message is four attributes -- under the C, D, M and T headings -- defined as follows:

heading C
value meaning
"C" event is critical token value ZSPI-VAL-TRUE
"-" event is not critical token not present or token value ZSPI-VAL-FALSE

heading D
value meaning
"D" event is to be displayed on console like ViewPoint or NonStop NET/MASTER token not present or token value ZSPI-VAL-FALSE
"-" event not to be displayed token value ZSPI-VAL-TRUE

heading M
value meaning
"M" event is associated with a manager process token value is name of manager
"S" event is a standard event non-null value null value defined in Section 9, Standard Events, in the EMS Manual
"P" event is a private event null value non-null value defined by this or other subsystem
"B" event was a private event non-null value non-null value and now also a standard event

<<Replace "ZSAM" with your 4-char subsystem acronym.

List all the event subjects from the "Event Subjects Definition Table" here, in alphabetical order.

Under each event subject, list the names of all your events generated for that subject, also in alphabetical order. If an event has already been listed under another subject earlier, enter "*** see subject-so-and-so" in the "Description" column.

>>

<table>
<thead>
<tr>
<th>CDMT</th>
<th>Event Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(ZSAM-EVT-)</td>
<td></td>
</tr>
</tbody>
</table>

<<Below are examples of events defined for three of PATHWAY's event subjects. Replace them with your events.

-DM- PM-DELINKFAIL can't return link to server
-DM- PM-LINKFAIL server can't grant link
-DM- PM-TCPFAIL process stopped abnormally

-DM- PM-SHUTDOWN PATHWAY completed shutdown
CDM- PM-PMABENDED PATHMON aborted due to fatal err

PM-DELINKFAIL *** see ZPWY-TKN-LMNAME
PM-LINKFAIL *** see ZPWY-TKN-LMNAME
PM-TCPFAIL *** see ZPWY-TKN-LMNAME

-DM- PM-ABORTEDSTATUS acknowledge term been aborted
CDM- TCP-BADPCODE invalid pseudocode detected
-DM- TCP-DEPENDVARTOOBIG DEPENDING ON data item too big
CDM- TCP-BADTERMTYPE illegal terminal type specified

>>

This section describes all the EMS event tokens that are used by event messages of this subsystem. The tokens are described under the subsystem -- like SPI, EMS or this subsystem -- where the tokens are defined. Each subsection below defines the tokens of
one subsystem.

The tokens of a subsystem are listed, in alphabetical order of token name, under one of these headings in the table:

Tokens under "Common ... tokens" heading

Tokens listed under this heading are defined for all event messages of this subsystem. A token listed here does not mean that the token must always be present in an event message. If a token is not present, specific meaning -- as defined by the token -- is associated with an event.

Tokens under "Event-specific ... tokens" heading

Tokens listed under this heading are defined only for specific event messages generated by this subsystem. Refer to the description of the individual event to determine whether the token is used in an event (see "Description of EMS event messages" section.)

This section lists all the tokens that are still used by events from this subsystem. Refer to the "Obsoleted EMS events and tokens" section for the tokens and events that have been obsoleted by this subsystem.

The "Standard SPI Tokens Definition Table" below lists all the SPI defined tokens that are used by this subsystem. The standard SPI tokens are defined in the ZSPIDEF.ZSPIDDL definition file. The default tokens provided below are explained in Detailed Description of Standard Events on page 9-17, in the EMS Manual.

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX-FIELD-VERSION</td>
<td>max version of extensible token</td>
</tr>
<tr>
<td>SSID</td>
<td>subsystem ID</td>
</tr>
<tr>
<td>USEDLEN</td>
<td>bytes used in EMS buffer</td>
</tr>
</tbody>
</table>

None

The "Standard EMS Tokens Definition Table" below lists all the EMS defined tokens that are used by this subsystem. The standard EMS tokens are defined in the ZEMSDEF.ZEMSDDL definition file. The default tokens provided below are explained in Detailed Description of Standard Events on page 9-17, in the EMS Manual.

* means the token is supported by this subsystem

<<You must support all these tokens. Do not change anything below.>>

* BATCHJOB-ID Batch job ID of process
* CONTENT-STANDARD type of standard event
* CONTENT-USER type of user-defined event
* CPU                  CPU no. of event sender
* EMPHASIS             critical event?
* EVENTNUMBER          event number
* GENTIME              event generation time
* LOGTIME              event log time
* NAME-MANAGER         manager process name
* NODENUM              system no. of event sender
* PIN                  PIN of event sender
* SUBJECT-MARK         event subject marker
* SUPPRESS-DISPLAY      display event?
* USERID               user ID of event sender

<<Replace "*" with blank for tokens your subsystem does not support. Do not delete any of these tokens from the table here.>>

* ACTION-ID            action id
* ACTION-NEEDED        operator action needed?
* CHANGE-REASON         object state change reason
<<Add your private enumerations here.>>
* FAILURE-CAUSE         object unavailable cause
* INFO-CHANNEL-UNIT     I/O channel unit info
* NAME-CODEFILE         Guardian process codefile name
* NAME-PROCESS          Guardian process descriptor
* NAME-DUMPFILE         Guardian process dumpfile name
* PLATFORM-TYP          T/16 cpu type
* REGISTERS-T16         T/16 P/E registers
* STATE-CURRENT         current object state
<<Add your private enumerations here.>>
* STATE-PREVIOUS        previous object state
<<Add your private enumerations here.>>
* SYMPTOM-STRING        symptom string
* TRAP-CODE             Guardian 90 trap code
* TXFAULT-TYPE          type of transient fault
<<Add your private enumerations here.>>
* XSYSPID               PID of Guardian process
* UTIL-CONFIG-HI        configured resource level (high)
* UTIL-CONFIG-LOW       configured resource level (low)
* UTIL-LEVEL-CURR       current resource usage level
* UTIL-LEVEL-PREV       previous reported usage level
* UTIL-TIME-PREV        previous reported usage timestamp

<<If you do not use tokens defined by other subsystems, state "None" here. Do not delete this subsection.

If you use tokens defined by other subsystems, create a table for each subsystem on a new page. List the tokens from that subsystem that you use in the table. Follow the same table format that is used in the "Standard EMS tokens" subsection earlier. Use this outline for your subsection:

\new

EMS Manual—426909-005
C-10
"name of subsystem tokens like Standard INTERPOL Tokens"

State where one find the definitions listed below

List the common tokens

List the event-specific tokens

The example below illustrates the description of defining INTERPOL tokens here.

The "INTERPOL Tokens Definition Table" below lists all the INTERPOL defined tokens that are used by this subsystem. The standard INTERPOL tokens are defined in the ZSPIDEF.ZCOMDDL definition file.

<table>
<thead>
<tr>
<th>OBJNAME</th>
<th>name of an object (up to 256 char)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJTYP</td>
<td>type of object in ZCOM-TKN-OBJNAME</td>
</tr>
</tbody>
</table>

- ZCOM-OBJ-FILE = disk file
- ZCOM-OBJ-PROC = management process
- ZCOM-OBJ-LAN = LAN interface process
- ZCOM-OBJ-SERVER = server process

<table>
<thead>
<tr>
<th>OBJSTATE</th>
<th>summary state of process (current)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD-OBJSTATE</td>
<td>summary state of process (previous)</td>
</tr>
<tr>
<td>SUBJ-FILE</td>
<td>file involved in the event</td>
</tr>
<tr>
<td>SUBJ-LAN</td>
<td>LAN involved in the event</td>
</tr>
<tr>
<td>SUBJ-PROC</td>
<td>process involved in the event</td>
</tr>
<tr>
<td>SUBJ-SERVER</td>
<td>server involved in the event</td>
</tr>
<tr>
<td>SUBJ-SUBSYS</td>
<td>subsystem involved in the event</td>
</tr>
<tr>
<td>SUBJ-USER</td>
<td>user involved in the event</td>
</tr>
</tbody>
</table>

<<change "ZSPIDEF.ZSAMDDL" to where one can find your DDL definitions>>

The "Private Tokens Definition Table" below lists all the tokens defined by this subsystem. These tokens are defined in the ZSPIDEF.ZSAMDDL definition file.

<<Change "ZSAM" to your subsystem acronym>>

<<Describe all the common tokens defined by your subsystem. If none, state "None." Example below shows the kind of information that should be provided here. See "Detailed Description of Standard Events" in Section 9, Standard Events, in the EMS Manual, for other examples.
Critical ZEMS-TKN-EMPHASIS (ZSPI-TYP-BOOLEAN,C) Indicator
Indicates the critical nature of the condition reported in the event. Standard values are:

   ZSPI-VAL-TRUE   indicates event is critical
   ZSPI-VAL-FALSE  indicates event is not critical (default)

An event is critical if it reports ...

If an event is considered critical, this token must be present and have the value of ZSPI-VAL-TRUE; otherwise this should not be present or has the value ZSPI-VAL-FALSE.

Describe all the event-specific tokens used by your subsystem. If none, state "None."

This section describes the EMS event messages generated by this subsystem. Event messages are described in alphabetical order of their symbolic event names. Each subsection below describes one event message with the following information:

A brief description of the meaning of the event message.
List of unconditional tokens.
List of conditional tokens.

Unconditional tokens are tokens that are always present in the event message.

Conditional tokens are tokens that are present in the event message if the stated conditions are met. If a conditional token is not present, specific meaning, as defined by the token, is associated with the event. "None" means this event message has no conditional tokens.

Every event message contains all the common tokens listed under the heading "Common ... tokens" of the "Description of EMS event tokens" section earlier. These common tokens are not listed in the event message below unless (a) more specific values other than those where the tokens are defined apply to the event message, or (b) the event message text is to be customized and it needs to refer to the token.

These common tokens are listed with each event message:

   ZEMS-TKN-SUBJECT-MARK identifies the event subject or subjects
   ZEMS-TKN-CONTENT-STANDARD identifies the standard event type
   ZEMS-TKN-CONTENT-USER identifies the private event type

These common tokens are not listed here -- they are defined for
each event message under the "Event Numbers Definition Table":

- ZEMS-TKN-NAME-MANAGER identifies the manager process name
- ZEMS-TKN-SUPPRESS-DISPLAY indicates if event is to be displayed
- ZEMS-TKN-EMPHASIS indicates if event is critical

Every event message also contains one or more event-specific tokens listed under the heading "Event-specific ... tokens" of the "Description of EMS event tokens" section earlier. The event-specific tokens that apply to an event are listed here.

List of internal tokens.

Internal tokens are tokens that are used by this subsystem only, like for debugging a program problem. These tokens are not intended for management applications and could change at any time. "None" means this event message has no internal tokens.

Event-Message text.

This describes the display text of an event. "use standard display" means the display described in Description of Standard EMS Templates on page 9-34 in the EMS Manual, with no modification, is to be used to display the event; otherwise the "display" is included here with these notations:

"<nn>" refers to the token that has the same <nn> notation in the "Unconditional Tokens" or "Conditional Tokens" heading above.

"<*>" refers to a standard EMS token that is defined by the standard Template.

The event message text is specified by the EMS Template file of this subsystem.

Cause.

This describes what caused this subsystem to generate the event message.

Effect.

This explains the effect this event has on the system. "None" means the event has no effect on the system.

Recovery Procedures.

This describes how one can solve the problem reported by the event message. "Informative message only; no corrective action is needed" means the event only reports status information and no recovery action is needed.

Other considerations.

This describes the special considerations that are needed to interpret or use the event message. If this subsection is absent, it means no special consideration for the event message.

<<
A number of "templates" are provided to help you describe your events, as follows:

1) Create a new subsection to describe each of your event messages.
   The subsections should be ordered in alphabetical order by the symbolic event names.

   Each subsection must start on a new page and with this outline:
   \header_2 "symbolic event name like ZSAM-EVT-LINE-DOWN"
   \new
   \level 2 "symbolic event name like ZSAM-EVT-LINE-DOWN"
   Brief description of your event.
   Event descriptions -- use provided "templates".

2) Provide a brief description of your event.

3) Copy the "template" that applies to your event.

4) Edit the "template" you copied with the information of your event.

>>
use this "template" to describe OBJECT AVAILABLE type event

<<provide a brief description of your event here>>

ZEMS-TKN-SUBJECT-MARK  <1> <<list subjects for event>>
ZEMS-TKN-CONTENT-STANDARD  ZEMS-VAL-OBJECT-AVAILABLE
ZEMS-TKN-CONTENT-USER  <2> <<specific value or delete>>
<<Add your private unconditional tokens here>>
Conditional Tokens  Value
<<Add your private conditional tokens here or state "None">>
Internal Tokens  Value
<<Add your internal tokens here or state "None">>
Event-Message Text

<<state "use standard display" if using display below; otherwise specify your display (you must provide your Template in this case).

Object available <1>, event number: <>, reason: <>, previous state: <>, current state: <>, manager: <>, batch ID: <>, user content:<2>

>>

<<describe cause of event here>>
<<describe effect of event on system or state "None.">>

<<describe the recommended recovery action or state "Information only. No recovery action needed.">>

<<describe other considerations for event here or do not include this heading>>
use this "template" to describe OBJECT OTHER STATE CHANGE type event

<<provide a brief description of your event here>>

ZEMS-TKN-SUBJECT-MARK         <1> <<list subjects for event>>
ZEMS-TKN-CONTENT-STANDARD      ZEMS-VAL-OTHER-STATE-CHANGE
ZEMS-TKN-CONTENT-USER          <2> <<specific value or blank>>
<<add your private unconditional tokens here.>>
Conditional Tokens              Value
<<add your private conditional tokens here or state "None">>
Internal Tokens                 Value
<<add your internal tokens here or state "None">>
Event-Message Text

<<state "use standard display" if using display below; otherwise specify your display (you must provide your Template in this case).

(Other) state change <1>, event number: <*> , reason: <*> ,
previous state: <*> , current state: <*> , manager: <*> , batch ID: <*> , user content: <2>

>>

<<describe cause of event here>>

<<describe effect of event on system or state "None.">>

<<describe the recommended recovery action or state "Information only. action for operator No recovery action needed.">>

<<describe other considerations for event here or do not include this heading>>
use this "template" to describe OBJECT UNAVAILABLE type event

<<provide a brief description of your event here>>

ZEMS-TKN-SUBJECT-MARK         <1> <<list subjects for event>>
ZEMS-TKN-CONTENT-STANDARD      ZEMS-VAL-OBJECT-UNAVAILABLE
ZEMS-TKN-CONTENT-USER <2> <<specific value or blank>>
<<add your private unconditional tokens here.>>
Conditional Tokens Value

<<add your private conditional tokens here or state "None">>
Internal Tokens Value
<<add your internal tokens here or state "None">>
Event-Message Text

<<state "use standard display" if using display below; otherwise specify your display (you must provide your Template in this case).>

Object unavailable <1>, event number: <*>, cause: <*>, underlying object: <*>, manager: <*>, batch ID: <*>, user content: <2>
>>

<<describe cause of event here>>

<<describe effect of event on system here or state "None.">>

<<describe the recommended recovery action here or state "Information only. No recovery action needed.">>

<<describe other considerations for event here or do not include this heading>>
use this "template" to describe OPERATOR ATTN COMPLETED type event

<<provide a brief description of your event here>>

ZEMS-TKN-SUBJECT-MARK <1> <<list subjects for event>>
ZEMS-TKN-CONTENT-STANDARD ZEMS-VAL-ATTN-COMPLETED
ZEMS-TKN-CONTENT-USER <2> <<specific value or blank>>
<<add your private unconditional tokens here.>>
Conditional Tokens Value
<<add your private conditional tokens here or state "None">>
Internal Tokens Value
<<add your internal tokens here or state "None">>
Event-Message Text

<<state "use standard display" if using display below; otherwise specify your display (you must provide your Template in this case).>

Operator attention completed for <1> thank you, event number: <*>, action id: <*>, manager: <*>, batch ID: <*>, user content: <2>
>>
<<describe cause of event here>>

<<describe effect of event on system here or state "None.">>

<<describe the recommended recovery action here or state "Information only. No recovery action needed.">>

<<describe other considerations for event here or do not include this heading>>

use this "template" to describe OPERATOR ATTN NEEDED type event

<<provide a brief description of your event here>>

ZEMS-TKN-SUBJECT-MARK               <1> <<list subjects for event>>
ZEMS-TKN-CONTENT-STANDARD            ZEMS-VAL-ATTN-NEEDED
ZEMS-TKN-CONTENT-USER                <2> <<specific value or blank>>
<<add your private unconditional tokens here.>>
Conditional Tokens                   Value
<<add your private conditional tokens here or state "None">>
Internal Tokens                      Value
<<add your internal tokens here or state "None">>
Event-Message Text

<<state "use standard display" if using display below; otherwise specify your display (you must provide your Template in this case).>

Operator attention needed for <1> please, event number: <*>, action ID: <*>, manager: <*>, batch ID: <*>, user content: <2>

<<describe cause of event here>>

<<describe effect of event on system here or state "None.">>

<<describe the recommended recovery action here or state "Information only. No recovery action needed.">>

<<describe other considerations for event here or do not include this heading>>

use this "template" to describe TRANSIENT FAULT type event

<<provide a brief description of your event here>>

ZEMS-TKN-SUBJECT-MARK               <1> <<list subjects for event>>
Transient fault <1>, event number: <*>, fault type: <*>, manager: <*>, batch ID: <*> , user content: <2>

<<describe cause of event here>>

<<describe effect of event on system here or state "None.">>

<<describe the recommended recovery action here or state "Information only. No recovery action needed.">>

<<describe other considerations for event here or do not include this heading>>

use this "template" to describe USAGE THRESHOLD type event

<<provide a brief description of your event here>>

Usage threshold <1>, event number: <*> , usage level (current): <*>, usage level (last reported): <*>, usage unit: <*>, timestamp GMT (current): <*>, timestamp GMT (last reported): <*>, configured usage level (high): <*>, configured usage level (low): <*>, manager: <*>, batch ID: <*> , user content: <2>
<<describe cause of event here>>

<<describe effect of event on system here or state "None.">>

<<describe the recommended recovery action here or state "Information only. No recovery action needed.">>

<<describe other considerations for event here or do not include this heading>>

The "Obsoleted Events and Tokens Table" below lists the event messages and tokens that this subsystem has obsoleted. The events and tokens are listed under the version they are obsoleted.

<<Following are examples of obsoleted events and tokens for subsystem ZSAM. Replace with your events and tokens that you have obsolete. If none, state "None" under the appropriate heading.

<table>
<thead>
<tr>
<th>Version</th>
<th>Events</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T9999C10</td>
<td>ZSAM-EVT-LINE-TRACE</td>
<td>trace communication line</td>
</tr>
<tr>
<td>T9999C20</td>
<td>ZSAM-EVT-LINE-TEST</td>
<td>loopback test</td>
</tr>
<tr>
<td>Version</td>
<td>Tokens</td>
<td>Description</td>
</tr>
<tr>
<td>T9999C10</td>
<td>ZSAM-TKN-TRACE-BEG</td>
<td>beg trace location</td>
</tr>
<tr>
<td>T9999C10</td>
<td>ZSAM-TKN-TRACE-END</td>
<td>end trace location</td>
</tr>
<tr>
<td>T9999C20</td>
<td>ZSAM-TKN-TEST-RESULT</td>
<td>loopback test result</td>
</tr>
</tbody>
</table>
DDL Definitions Sample File

This file was stored in InfoWay as ddltxt.txt:

!=====================================================================
!This is the DDL definitions file for the fictitious subsystem "SAMPLER" described in the EMS Manual in Example of a Fictitious NonStop Kernel Subsystem on page 10-31.
!Purpose of this file is to illustrate some commonly used ways to define tokens and events in a subsystem. You should refer "Create and build your DDL definitions" section of the same document for more details on how to customize this file.
!In general, follow the layout of the sections here and use the naming conventions for the tokens and literals.
!=====================================================================

!=====================================================================
!This file defines the EMS event tokens for the SAMPLER subsystem.
!Definitions below follow this naming convention:
!<subs>-<type>-<name>
!where
!<subs>  identifier of this subsystem (ZSAM)
!<type>  is a three-character mnemonic indicating what <name> describes, as follows:
!DDL  DDL definitions for structures
!ENM  level 89 items in ENUM declarations
!EVT  literals for event numbers
!MAP  literals for extensible structured tokens
!TKN  literals for simple token codes
!TNM  literals for token numbers of token codes & maps
!TYP  literals for SPI token types
!VAL  literals for token values
!<name>  descriptive name for the literals
!=====================================================================

Product information of subsystem
******************************************************************************
| Product Name  | Product Number & Version | Release Date |
******************************************************************************
* SAMPLER T9999D99^9SEP99 (09SEP99)

******************************************************************************
! Subsystem ID (SSID) definition (ZSAM-VAL-SSID)

! ZSAM-VAL-VERSION reflects the release version of the subsystem, not the version of its SPI/EMS interface. It is changed whenever release version of the subsystem is changed.

******************************************************************************

! For Tandem:
! * ZSAM-VAL-VERSION: change "D10" to your product release version ID.
! * ZSPI-SSN-ZSAM: delete reference here as it will be defined in ZSPIDEF.

! For non-Tandem:
! * ZSPI-VAL-TANDEM: change to the literal that defines the registered name of your company (the literal is defined by your company)
! * ZSPI-SSN-ZSAM: change to the literal that defines the subsystem number assigned to you (the literal is defined by your company)
! * ZSAM-VAL-VERSION: change "D10" to reflect your release version.

CONSTANT ZSAM-VAL-VERSION VALUE VERSION "D10".
CONSTANT ZSPI-SSN-ZSAM VALUE 99.

DEFINITION ZSAM-VAL-SSID TACL SSID.
  02 z-filler TYPE character 8 VALUE IS ZSPI-VAL-TANDEM.
  02 z-owner TYPE ZSPI-DDL-CHAR8 REDEFINES z-filler.
  02 z-number TYPE ZSPI-DDL-INT VALUE IS ZSPI-SSN-ZSAM.
  02 z-version TYPE ZSPI-DDL-UINT VALUE IS ZSAM-VAL-VERSION.
END

******************************************************************************
! External SSID definition (ZSAM-VAL-EXTERNAL-SSID)

******************************************************************************

! For all:
! * change "99" to the subsystem number assigned to you.
! For non-Tandem:
! * change "TANDEM" to the registered name of your company.

CONSTANT ZSAM-VAL-EXTERNAL-SSID VALUE "TANDEM.99.0".

******************************************************************************
! Event Number definitions (ZEMS-TKN-EVENTNUMBER)

******************************************************************************

! Naming convention: ZSAM-EVT-eventname for event number
! ZSAM-ENM-eventname for 89 enumeration clause
! Valid range: 1 through ????
EVENT NUMBER

<table>
<thead>
<tr>
<th>Event Subject</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSAM-TKN-tp</td>
<td>001 - 100</td>
<td>application transport connection</td>
</tr>
<tr>
<td>ZSAM-TKN-netx25</td>
<td>101 - 200</td>
<td>underlying X.25 network service</td>
</tr>
<tr>
<td>ZSAM-TKN-tape</td>
<td>201 - 300</td>
<td>application profile tape</td>
</tr>
</tbody>
</table>

CONSTANT ZSAM-EVT-tp-disconnected VALUE IS 1. OBJECT
CONSTANT ZSAM-EVT-tp-connected VALUE IS 2. OBJECT
CONSTANT ZSAM-EVT-tp-x-conntime VALUE IS 3. OTHER
CONSTANT ZSAM-EVT-tp-x-resets VALUE IS 4. TRANSIENT FAULT
CONSTANT ZSAM-EVT-tp-usage-data VALUE IS 5. appl net usage data

CONSTANT ZSAM-EVT-netx25-down VALUE IS 101. OBJECT
CONSTANT ZSAM-EVT-netx25-up VALUE IS 102. OBJECT
CONSTANT ZSAM-EVT-netx25-dataloss VALUE IS 103. TRANSIENT FAULT
CONSTANT ZSAM-EVT-netx25-util-req VALUE IS 104. USAGE THRESHOLD

CONSTANT ZSAM-EVT-tape-mount-needed VALUE IS 201. ATTENTION NEEDED
CONSTANT ZSAM-EVT-tape-mount-done VALUE IS 202. ATTENTION COMPLETED

DEFINITION ZSAM-DDL-EVENTNUMBER-ENUM
TYPE ENUM TAACL ENUM
!*do not specify "AS clause"
BEGIN.
  89 ZSAM-ENM-tp-disconnected VALUE ZSAM-EVT-tp-disconnected
    AS "appln not connected to remote".
  89 ZSAM-ENM-tp-connected VALUE ZSAM-EVT-tp-connected
    AS "appln connected to remote".
  89 ZSAM-ENM-tp-x-conntime VALUE ZSAM-EVT-tp-x-conntime
    AS "appln taking too long to connect".
  89 ZSAM-ENM-tp-x-resets VALUE ZSAM-EVT-tp-x-resets
    AS "appln has too many resets".
  89 ZSAM-ENM-tp-usage-data VALUE ZSAM-EVT-tp-usage-data
    AS "appln transport usage".
data".
  89 ZSAM-ENM-netx25-down VALUE ZSAM-EVT-netx25-down
               AS "X25 net down".
  89 ZSAM-ENM-netx25-up VALUE ZSAM-EVT-netx25-up
               AS "X25 net up".
  89 ZSAM-ENM-netx25-dataloss VALUE ZSAM-EVT-netx25-
  dataloss
               AS "too much data loss for
  X25 net".
  89 ZSAM-ENM-netx25-util-req VALUE ZSAM-EVT-netx25-util-
  req
               AS "no. requests queued for
  X25 net".
  89 ZSAM-ENM-tape-mount-needed VALUE ZSAM-EVT-tape-mount-
  needed
               AS "mount user profile
tape".
  89 ZSAM-ENM-tape-mount-done VALUE ZSAM-EVT-tape-mount-
  done
               AS "user profile tape done".
END.

!***************************************************************
!   Private Event Type definitions (ZEMS-TKN-CONTENT-USER)     *
!                                                              *
!   Naming convention: ZSAM-VAL-typename                       *
!                      ZSAM-ENM-typename for 89 enumeration    *
!                      clause                                  *
!   Valid range:       ZEMS-VAL-MIN-USER-VALUE (1024) to ???  *
!***************************************************************

!   private
!   event types        description
!   (ZSAM-VAL-)
!
!   data-usage         network usage data for application
!   * network connection time (in minutes)
!   * no. data bytes sent
!   * no. data bytes received
!   * no. data packets send
!   * no. data packets received
CONSTANT ZSAM-VAL-data-usage VALUE IS 1024.

DEFINITION ZSAM-DDL-CONTENT-USER-ENUM
  TYPE ENUM TACL ENUM      ! do not specify the "AS
  clause"
BEGIN.
  89 ZSAM-ENM-data-usage VALUE ZSAM-VAL-data-usage
               AS "appl network usage
data".
END.

!******************************************************************
!   Private enumerations for these standard tokens  *
The following are valid state transitions and change reason as reported in the OBJECT AVAILABLE or OTHER STATE CHANGE events.

<table>
<thead>
<tr>
<th>valid state transition</th>
<th>change reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ZSAM-VAL-)</td>
<td>(ZSAM-VAL-cr-)</td>
</tr>
<tr>
<td>tp-disconnected -&gt; tp-connecting</td>
<td>tp-xconntime</td>
</tr>
<tr>
<td>tp-connecting -&gt; tp-connected</td>
<td>tp-applreq</td>
</tr>
<tr>
<td>netx25-down -&gt; netx25-up</td>
<td>netx25-online</td>
</tr>
</tbody>
</table>

CONSTANT ZSAM-VAL-tp-disconnected VALUE IS 1024.
CONSTANT ZSAM-VAL-tp-connecting VALUE IS 1025.
CONSTANT ZSAM-VAL-tp-connected VALUE IS 1026.
CONSTANT ZSAM-VAL-netx25-up VALUE IS 1027.
CONSTANT ZSAM-VAL-netx25-down VALUE IS 1028.

DEFINITION ZSAM-DDL-STATE-ENUM TYPE ENUM TACL ENUM
!*do not specify the "AS clause"
BEGIN.
89 ZSAM-ENM-tp-disconnected VALUE ZSAM-VAL-tp-
disconnected AS "appln not connected to
remote".
89 ZSAM-ENM-tp-connected VALUE ZSAM-VAL-tp-connected
AS "appln is connected to
remote".
89 ZSAM-VAL-tp-connecting VALUE ZSAM-VAL-tp-connecting
AS "appln is connecting to
remote".
89 ZSAM-VAL-netx25-up VALUE ZSAM-VAL-netx25-up
AS "underlying X.25 net up".
89 ZSAM-VAL-netx25-down VALUE ZSAM-VAL-netx25-down
AS "underlying X.25 net
down".
END.
CONSTANT ZSAM-VAL-cr-tp-applreq VALUE IS 1024.
CONSTANT ZSAM-VAL-cr-tp-xconntime VALUE IS 1025.
CONSTANT ZSAM-VAL-cr-netx25-online VALUE IS 1027.

DEFINITION ZSAM-DDL-CHANGEREASON-ENUM
    TYPE ENUM TACL ENUM      !*do not specify the "AS
clause"
    BEGIN.
        89 ZSAM-ENM-cr-tp-applreq VALUE ZSAM-VAL-cr-tp-applreq
        AS "requested by local
appln".
        89 ZSAM-ENM-cr-tp-xconntime VALUE ZSAM-VAL-cr-tp-
xconntime
        AS "excessive TP resets".
        89 ZSAM-ENM-cr-netx25-online VALUE ZSAM-VAL-cr-netx25-
online
        AS "x.25 net online".
    END.

CONSTANT ZSAM-VAL-tf-resets-tp VALUE IS 1024.
CONSTANT ZSAM-VAL-tf-resets-net VALUE IS 1025.

DEFINITION ZSAM-DDL-TXFAULT-TYPE-ENUM
    TYPE ENUM TACL ENUM      !*do not specify the "AS
clause"
    BEGIN.
        89 ZSAM-ENM-tf-resets-tp VALUE ZSAM-VAL-tf-resets-tp
        AS "too many TRANSPORT
resets".
        89 ZSAM-ENM-tf-resets-net VALUE ZSAM-VAL-tf-resets-net
        AS "too many NETWORK
resets".
    END.

!**************************************************************
!   Private enumerations of other subsystems' tokens           *
!                                                              *
!   Naming convention: ZSAM-VAL-enumname                       *
!                      ZSAM-ENM-enumname for 89 enumeration    *
!                      clause                                  *
!**************************************************************

! Enumerate values you added to other subsystems' token
Provide "89 enumeration clause" if token has "89" (see above)

Private enumerations of this subsystem's tokens
Naming convention: ZSAM-VAL-enunmname
ZSAM-ENM-enunmname for 89 enumeration clause

CONSTANT ZSAM-VAL-appl-remote-rej VALUE IS  1.
CONSTANT ZSAM-VAL-net-unavail     VALUE IS  2.
CONSTANT ZSAM-VAL-tp-remote-rej    VALUE IS  3.
CONSTANT ZSAM-VAL-tp-local-abort   VALUE IS  4.

Private token definitions -- common building blocks
1) Token-number definitions
Naming convention: ZSAM-TNM-tokenname
Valid range:       1 through 9998
"tokenname" corresponds to the token ZSAM-TKN-tokenname
or ZSAM-MAP-tokenname defined below

2) Common DDL structure definitions
Naming convention: ZSAM-DDL-structname
This section defines the common DDL structures for tokens below

3) Private token-type definitions
Naming convention: ZSAM-TYP-tokentypename
ZSPI-TYP-??? are standard SPI token-type definitions
ZSPI-TDT-??? are standard SPI token-data-type definitions
This section defines the private token-types for tokens below

------------------------
Token-number definitions
------------------------
Every token you define under TOKEN-CODE and TOKEN-MAP later must have a token number. The token numbers for the private tokens defined are:

CONSTANT ZSAM-TNM-subj-tp VALUE IS  1.
CONSTANT ZSAM-TNM-subj-netx25 VALUE IS  2.
CONSTANT ZSAM-TNM-subj-tape  VALUE IS  3.
CONSTANT ZSAM-TNM-tp-diagcode    VALUE IS    4.
CONSTANT ZSAM-TNM-data-usage     VALUE IS    5.
CONSTANT ZSAM-TNM-netx25-error   VALUE IS    6.
CONSTANT ZSAM-TNM-pdnx25-error   VALUE IS    7.

!   --------------------------------
!   Common DDL structure definitions
!   --------------------------------

DEFINITION ZSAM-DDL-tp-diagcode-ENUM
    TYPE ENUM TACL ENUM     !*do not specify the "AS
clause"
BEGIN.
   89 ZSAM-ENM-appl-remote-rej VALUE ZSAM-VAL-appl-remote-
       rej AS "rejected by remote
       appl".
   89 ZSAM-ENM-net-unavail   VALUE ZSAM-VAL-net-unavail
       AS "aborted by no net
       available".
   89 ZSAM-ENM-tp-remote-rej VALUE ZSAM-VAL-tp-remote-rej
       AS "rejected by remote
       transport".
   89 ZSAM-ENM-tp-local-abort VALUE ZSAM-VAL-tp-local-
       abort AS "aborted by local appl".
END.

DEFINITION ZSAM-DDL-data-usage.
    02 time-connect TYPE ZSPI-DDL-INT2
       HEADING "appl session connect time (min)".
    02 bytes-send  TYPE ZSPI-DDL-INT2
       HEADING "no. bytes sent to remote appl".
    02 bytes-recv  TYPE ZSPI-DDL-INT2
       HEADING "no. bytes recv from remote appl".
    02 pkts-send  TYPE ZSPI-DDL-INT2
       HEADING "no. packets sent to network".
    02 pkts-recv  TYPE ZSPI-DDL-INT2
       HEADING "no. packets recv from network".
END.

DEFINITION ZSAM-DDL-pdnx25-error.
    02 stdlevel   TYPE ZSPI-DDL-INT
       HEADING "PDN X.25 standard level".
    02 errcode    TYPE ZSPI-DDL-INT
       HEADING "PDN X.25 error code".
    02 diagcode   TYPE ZSPI-DDL-INT
       HEADING "PDN X.25 diagnostic code".
    02 info       TYPE ZSPI-DDL-CHAR50
       HEADING "PDN X.25 error description".
END.

!   ------------------------------
!   Private token-type definitions
!   ------------------------------
TOKEN-TYPE ZSAM-TYP-tp-diagcode
VALUE IS ZSPI-TDT-ENUM
DEFINITION IS ZSAM-DDL-tp-diagcode-ENUM.

TOKEN-TYPE ZSAM-TYP-data-usage
VALUE IS ZSPI-TDT-STRUCT
DEFINITION IS ZSAM-DDL-data-usage.

!***************************************************************
!   Event Subject (private token) definitions                  *
!   naming convention: ZSAM-TKN-SUBJ-subjectname (simple/fixed)*
!                      ZSAM-MAP-SUBJ-subjectname (extensible)  *
!***************************************************************

!   event subject
!   (ZSAM-TKN-) description of object
!
!   subj-tp application transport connection
!   subj-netx25 underlying X.25 network service
!   subj-tape user profile tape

TOKEN-CODE ZSAM-TKN-subj-tp
VALUE ZSAM-TNM-subj-tp
TOKEN-TYPE ZSPI-TYP-STRING
SSID ZSAM-VAL-EXTERNAL-SSID
HEADING "application transport connection".

TOKEN-CODE ZSAM-TKN-subj-netx25
VALUE ZSAM-TNM-subj-netx25
TOKEN-TYPE ZSPI-TYP-STRING
SSID ZSAM-VAL-EXTERNAL-SSID
HEADING "X.25 network service".

TOKEN-CODE ZSAM-TKN-subj-tape
VALUE ZSAM-TNM-subj-tape
TOKEN-TYPE ZSPI-TYP-STRING
SSID ZSAM-VAL-EXTERNAL-SSID
HEADING "appln profile tape".

!***************************************************************
!   Private token (non-event-subject) definitions              *
!   Naming convention: ZSAM-TKN-tokenname for simple/ixed      *
!                      tokens                                  *
!                      ZSAM-MAP-tokenname for extensible tokens*  *
!***************************************************************

!   token code
!   (ZSAM-TKN-) type description
!
!   tp-diagcode enum appl transport disconnect
!   data-usage struct appl transport usage data
! netx25-error int error (File Sys) returned from net
! (ZSAM-MAP-)
!
! netx25-info Xstruct error information returned by X.25 PDN

TOKEN-CODE ZSAM-TKN-tp-diagcode
VALUE IS ZSAM-TNM-tp-diagcode
TOKEN-TYPE ZSAM-TYP-tp-diagcode !allow
"89" DEF's
SSID ZSAM-VAL-EXTERNAL-SSID
HEADING "appl transport disconnect diagnostic code".

TOKEN-CODE ZSAM-TKN-data-usage
VALUE IS ZSAM-TNM-data-usage
TOKEN-TYPE ZSAM-TYP-data-usage !allow
def of struct
SSID ZSAM-VAL-EXTERNAL-SSID
HEADING "appl transport usage data".

TOKEN-CODE ZSAM-TKN-netx25-error
VALUE IS ZSAM-TNM-netx25-error
TOKEN-TYPE ZSPI-TYP-INT !File
SSID ZSAM-VAL-EXTERNAL-SSID
HEADING "X.25 net error".

TOKEN-MAP ZSAM-MAP-pdnx25-error
VALUE IS ZSAM-TNM-pdnx25-error
DEF IS ZSAM-DDL-pdnx25-error !allow
def of struct
SSID ZSAM-VAL-EXTERNAL-SSID
HEADING "X.25 PDN error diag info".
VERSION "C00" FOR stdlevel THRU info.
END.
**EMS Templates Sample File**

This file was stored in InfoWay as templat.txt:

```
****************************************************************
==   This is the sample EMS Templates file for the fictitious ==
==   subsystem "SAMPLER" described in the EMS Manual in Example  ==
==   of a Fictitious NonStop Kernel Subsystem on page 10-31.       ==
==   Purpose of this file is to illustrate some commonly used ==
==   ways to:                                                 ==
==   1) Define a minimal subsystem Template file which       ==
==      consists of the following sections:                   ==
==                                                            ==
==      * subsystem description                               ==
==      * assign your enumerations to standard SPI/EMS tokens,==
==        if any                                              ==
==      * assign your enumerations to other subsystems'       ==
==        tokens, if any                                      ==
==                                                            ==
==   2) Customize standard Templates for your standard       ==
==      events, if any                                        ==
==   3) Define Templates for your private events, if any      ==
==                                                            ==
==************************************************************==
================================================================
==                       subsystem description                ==
================================================================

VERSION:   "T9999D99 - 9SEP99"
== DICT:                                not used in this example
SSID:      ZSAM-VAL-SSID
SSNAME:    "SAMPLER","SAM"

== SHARED_SSID:                         not used in this example
== FORMAT:                              not used in this example

================================================================
==  assign enumerations to standard SPI/EMS tokens            ==
================================================================

DEF_ENUM:  ZEMS-TKN-CONTENT-USER      AS    ZSAM-DDL-CONTENT-
USER-ENUM
DEF_ENUM:  ZEMS-TKN-STATE-CURRENT     AS    ZSAM-DDL-STATE-ENUM
DEF_ENUM:  ZEMS-TKN-STATE-PREVIOUS    AS    ZSAM-DDL-STATE-ENUM
DEF_ENUM:  ZEMS-TKN-CHANGE-REASON     AS    ZSAM-DDL-CHANGE-
CHANGEREASON-ENUM
DEF_ENUM:  ZEMS-TKN-TXFAULT-TYPE      AS    ZSAM-DDL-TXFAULT-
TYPE-ENUM
DEF_ENUM:  ZEMS-TKN-EVENTNUMBER       AS    ZSAM-DDL-
EVENTNUMBER-ENUM

================================================================
==   assign enumerations to other subsystems' tokens          ==
```
== if you have defined your private enumerations to a token from == another subsystem, assign them here (see above)  
== (none in this example) 

== Templates for subsystem standard events ==

== The following two Object Unavailable events with these == event numbers are customized to display the private tokens == in the events: ==
== ZSAM-EVT-tp-disconnected ==
== ZSAM-EVT-netx25-down ==

== The Templates below are modified from the standard Object == Unavailable Template. The same format as the standard == Template is used so that a consistent and uniform display == is presented to the operator. Statement marked "changed" == indicates that statement is modified from the standard == Template; statement marked "added" indicates that statement == is added to the standard Template. ==

== Note that the standard Template is defined by the standard == type token "ZEMS-TKN-CONTENT-STANDARD". When you customize == one of these Templates, you must use "ZEMS-TKN- == EVENTNUMBER." The EMSTEXT program will use your event == number Template to display your event, if one is specified, == before using the standard Template provided by EMS. ==

MSG:  ZEMS-TKN-EVENTNUMBER, ZSAM-EVT-tp-disconnected == changed  "Object unavailable  "<31><32> - <31><33>"  
"", event number: <1>"  
"", cause: <2>"  
"<*IF 24>, underlying object: <14><*ENDIF>"  
"<*IF 21>, manager: <11><*ENDIF>"  
"<*IF 22>, batch ID: <12><*ENDIF>"  
"<*IF 23>, user content: <13><*ENDIF>"  
"", diagnostic: <3>" == added  
1:ZEMS-TKN-EVENTNUMBER, ENUM  
2:ZEMS-TKN-FAILURE-CAUSE, ENUM  
3:ZSAM-TKN-tp-diagcode, ENUM == added  
11:ZEMS-TKN-NAME-MANAGER  
12:ZEMS-TKN-BATCHJOB-ID  
13:ZEMS-TKN-CONTENT-USER, ENUM  
14:ZEMS-TKN-UNDERLYING-OBJ-NAME  
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)  
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)  
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)  
24:TOKENPRESENT(ZEMS-TKN-UNDERLYING-OBJ-NAME)  
31:ZEMS-TKN-SUBJECT-MARK  
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE

MSG:  ZEMS-TKN-EVENTNUMBER, ZSAM-EVT-netx25-down  == changed
"Object unavailable  <31><32> - <31><33>"
" event number:  <1>"
" cause:  <2>"
"<IF 24>, underlying object:  <14><ENDIF>"
"<IF 21>, manager:  <11><ENDIF>"
"<IF 22>, batch ID:  <12><ENDIF>"
"<IF 23>, user content:  <13><ENDIF>"
"<IF 25>, net svc err:  <15><ENDIF>"  == added
"<IF 26>"  == added
" PDN-level:  <16>"  == added
" PDN-err:  <17>"  == added
" PDN-diag:  <18>"  == added
"<ENDIF>"  == added
1:ZEMS-TKN-EVENTNUMBER, ENUM
2:ZEMS-TKN-FAILURE-CAUSE, ENUM
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
13:ZEMS-TKN-CONTENT-USER, ENUM
14:ZEMS-TKN-UNDERLYING-OBJ-NAME
15:ZSAM-TKN-netx25-error = = added
16:ZSAM-MAP-pdnx25-error.ZSAM-DDL-pdnx25-error.stdlevel = = added
17:ZSAM-MAP-pdnx25-error.ZSAM-DDL-pdnx25-error.errcode = = added
18:ZSAM-MAP-pdnx25-error.ZSAM-DDL-pdnx25-error.diagcode = = added
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
23:TOKENPRESENT(ZEMS-TKN-CONTENT-USER)
24:TOKENPRESENT(ZEMS-TKN-UNDERLYING-OBJ-NAME)
25:TOKENPRESENT(ZSAM-TKN-netx25-error) = = added
26:TOKENPRESENT(ZSAM-MAP-pdnx25-error) = = added
31:ZEMS-TKN-SUBJECT-MARK
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE

================================================================
==   Templates for subsystem private events                   ==
================================================================

== In our example, we only defined one private event type:  ==
== * ZSAM-VAL-DATA-USAGE                                  ==
== There is only one event of this type:                   ==
== * ZSAM-EVT-tp-usage-data                                ==
== The Template for this event is:                         ==

MSG:  ZEMS-TKN-EVENTNUMBER, ZSAM-EVT-tp-usage-data
"Usage data for  <31><32> - <31><33>"
"event number: <1>"
"# minutes conn time: <2>"
"# bytes sent: <3>"
"# bytes recv: <4>"
"# pkts sent: <5>"
"# pkts recv: <6>"
"<IF 21>, manager: <11><ENDIF>
"<IF 22>, batch ID: <12><ENDIF>

1:ZEMS-TKN-EVENTNUMBER, ENUM
2:ZSAM-TKN-DATA-USAGE.ZSAM-DDL-DATA-USAGE.time-connect
3:ZSAM-TKN-DATA-USAGE.ZSAM-DDL-DATA-USAGE.bytes-send
4:ZSAM-TKN-DATA-USAGE.ZSAM-DDL-DATA-USAGE.bytes-recv
5:ZSAM-TKN-DATA-USAGE.ZSAM-DDL-DATA-USAGE.pkts-send
6:ZSAM-TKN-DATA-USAGE.ZSAM-DDL-DATA-USAGE.pkts-recv
11:ZEMS-TKN-NAME-MANAGER
12:ZEMS-TKN-BATCHJOB-ID
21:TOKENPRESENT(ZEMS-TKN-NAME-MANAGER)
22:TOKENPRESENT(ZEMS-TKN-BATCHJOB-ID)
31:ZEMS-TKN-SUBJECT-MARK
32:ZSPI-TKN-NEXTTOKEN, TOKENHEADING
33:ZSPI-TKN-NEXTTOKEN, TOKENVALUE
Introduction

EMS (Event Management System) is a key component in the management of HP NonStop Systems. It is used by HP NonStop subsystems to create, manage, and display events that are used in the management of the software and hardware components of NonStop Systems. It is also used by third parties and customers in the management of their software and hardware components.

This document contains the best practices and recommendations when designing and implementing EMS events. It is highly recommended that you consider these guidelines so that your software makes efficient and effective use of EMS.

Event Requirements Guidelines

EMS-REQ-101: Create a documentation that describes what events are reported and how they will be used. This information will be useful to the audience in dealing with the events.

Considerations: The following should be considered when formulating EMS requirements.

Who is the audience for the events?

- **Operator**: Is it a human operator (e.g., data center operator)?
- **Application**: Are any of the events processed by an application and result in automated actions (e.g., an email message)?
- **Internal versus External Consumption**: Are the events meant for your organization’s internal usage or for your customers?

How will the events be used?

How will the operator or application use the events? Include the usage scenarios.

Event Content Guidelines

EMS-CON-101: An event should report a significant change in system or application condition. Examples of such changes are:

- Conditions that require an operator action; e.g., process stop.
- State changes of significant objects (devices, services, etc.); e.g., ATM needs service, application server down.
- System configuration changes; e.g., path switches, takeovers, capacity changes, threshold changes.

- Threshold crossings; e.g., usage of a resource crossed a 70% threshold, capacity of a device exceeded 1000 transfers per second.

- Other conditions that could affect the stability or performance of your system; e.g., security violations or attacks, unusual conditions that may require future action, frequent occurrence of recoverable errors.

- Occurrences and anomalies that may lead to more serious problems. These events could be useful in diagnosis should the more serious problem occur. You should report these events judiciously; i.e., don't flood the event log.

**EMS-CON-111**: Events should be atomic. An event should report a single occurrence of a change of an object or subsystem. A single occurrence should not be reported in more than one event. Examples:

- If a service goes down, this should be reported in a single event. Having every client of the service reporting the same event would flood the EMS log with redundant information. However, each client could report an event that describes how the service being down affected it.

- If a device is reconfigured, put all the information of this occurrence in a single event.

**EMS-CON-121**: Events that are not useful to the operator or support, that will clutter the event log, or displace severe events, should not be reported. Below are some examples:

- Status or informational events that require no action. Occasional status or informational events (say once a day) are permissible if they have an audience.

- Repetitive events; e.g., one event for a down device is enough. It should not be repeated until the device is up and goes down again. If rapid state change oscillations or threshold crossings are possible, event dampening strategies such as event burst suppression or threshold hysteresis should be employed in the event creating module itself, rather than being dependent on the EMS burst suppression.

- Trace or debug events. Instead, files should be used for tracing and debugging information.
Event Destination Guidelines

There are two primary event destinations: your EMS Collector and $0. The following guidelines should be used in deciding where the events should be reported:

**EMS-DST-101**: Events should never to logged to $ZLOG (the OSM Collector). This collector is reserved for events from NonStop subsystems.

**EMS-DST-111**: In general, avoid using $0 (the Primary Collector). Most HP System events go into that log. Events that are important to your Operations staff can be logged to $0.

**EMS-DST-121**: Most, if not all, of your events should be logged to your own EMS Collector.

Event Subject Guidelines

**EMS-SUB-101**: Event subject token should identify the real object for which the event is reported.

Examples: Service name in a service down event, device name in a device needs service event.

**EMS-SUB-111**: Event subject token should NOT be a generic process name or program file, etc., unless the event corresponds to that process or program file itself; it should be the object/resource managed by that process.

Example: An alarm created event by an application should have as the subject token the object on which the alarm is created, rather than the application process that created the alarm.

**EMS-SUB-121**: Multiple event subject tokens should be used if one of them is not sufficient to indicate the object for which the event is reported.

Event Severity Guidelines

**EMS-SEV-101**: All non-informational events should use the ZEMS-TKN-ISO-SEVERITY token to indicate the severity of the event.
EMS Usage Best Practices and Recommendations

Event Emphasis Guidelines

EMS-SEV-111: ZEMS-TKN-ISO-SEVERITY should be used according to the following guidelines:

<table>
<thead>
<tr>
<th>Token value</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEMS-VAL-SEVERITY-CRITICAL</td>
<td>Critical condition that indicates a service affecting condition has occurred, requiring an immediate corrective action (e.g., significant degradation of performance, loss of software availability, loss of data, loss of application stability, security breaches, crossing of thresholds into a “red” zone)</td>
</tr>
<tr>
<td>ZEMS-VAL-SEVERITY-MAJOR</td>
<td>Major condition that indicates that a service affecting condition has developed, requiring an urgent action (e.g., not significant performance degradation, loss of capacity, crossing of thresholds into a “yellow” zone)</td>
</tr>
<tr>
<td>ZEMS-VAL-SEVERITY-MINOR</td>
<td>Minor condition that indicates a non-service affecting fault, generally not requiring any action (e.g., proactive failures, still in the “green” zone)</td>
</tr>
<tr>
<td>ZEMS-VAL-SEVERITY-WARNING</td>
<td>Indicates that a potential Major or Critical condition could occur in the future</td>
</tr>
<tr>
<td>ZEMS-VAL-SEVERITY-CLEARED</td>
<td>Problem is fixed</td>
</tr>
</tbody>
</table>

EMS-SEV-121: When in doubt, a less severe value of the ZEMS-TKN-ISO-SEVERITY token should be used rather than a more severe value.

Event Emphasis Guidelines

EMS-EMP-101: All events that have the ZEMS-TKN-ISO-SEVERITY token value as ZEMS-VAL-SEVERITY-CRITICAL or ZEMS-VAL-SEVERITY-MAJOR should set the ZEMS-TKN-EMPHASIS token to true.

EMS-EMP-111: All events that have the ZEMS-TKN-ISO-SEVERITY token value as ZEMS-VAL-SEVERITY-MINOR, ZEMS-VAL-SEVERITY-WARNING, or ZEMS-VAL-SEVERITY-CLEARED, or do not have the ZEMS-TKN-ISO-SEVERITY token, should NOT use the ZEMS-TKN-EMPHASIS token.

Event Text Guidelines

EMS-TXT-101: Event text message should have sufficient information to understand the event, without looking at all the tokens in the event.
Event Frequency Guidelines

**EMS-FRQ-101**: Events should not be repeated if the same problem persists or the same condition exists. See [EMS-CON-121](#).

Event Specifications Guidelines

**EMS-SPC-101**: Your software specification should describe every event that your software can generate, including:

- Possible values and interpretations of all of its tokens (excluding the “infrastructure” tokens added by the EMS subsystem itself, such as timestamps).
- Whether each token is unconditional (always present in the event) or conditional (only present sometimes).
- Conditions in which a conditional event will be present.
- All possible severities of the event in all possible conditions.
- Emphasis token values.
- Destination logs.
- Operator text messages.
- Cause (all conditions that can cause the event), effect (effect on the application/service), and recovery (actions that can be taken to remedy the situation described in the event).

Event Sustaining Guidelines

The intent of these guidelines is to keep events the same across different versions of the software, so that applications and operators do not make incorrect decisions.

**EMS-SUS-101**: Events should not change in meaning. If an event is no longer relevant or appropriate, it should not be generated. Instead, a new event with a new event number should be defined.

**EMS-SUS-111**: Documented tokens in an event should not be deleted.

**EMS-SUS-121**: Event templates can be modified, but they should not be deleted.

**EMS-SUS-131**: The DDL definitions for tokens, token values, and events should not change.
Index

Numbers
89 enumeration clause  10-22

A
A  12-5
Action events  2-4, 8-3
Action tokens  8-9
ALLOCATE keyword
   EMSACOLL  13-6
   EMSCCTRL  13-10
ALLPROCESSORS section of
SYSGEN  12-22
Alternate collector  2-7
   See also EMSACOLL
   BACKUP  12-6
   configuring  12-2
   context  12-24
   DEFAULTSUBVOL  12-7
   definition of  1-2
   log files  12-4
   POOLPAGES  12-8
   REPLYAFTERWRITE  12-9
Application interface to EMS  8-12
Applications of EMS  1-8
AS clause  10-22
Assignments, filter language  5-30
AUTOSTOP keyword, EMSDIST  13-39

B
BACKUP keyword
   EMSACOLL  13-4
   EMSDIST  13-35
BACKUP, alternate collector attribute  12-6
Bad event message buffer  2-13
Basic components of filter language  5-9
BDS (burst detection and suppression)
   collector events not suppressed by  7-9
   configuration examples  7-5
   definition of similar events  7-5
   features of  7-2
   implemented from a distributor  7-7
   implemented from a primary collector  7-6
   implemented from an alternate collector  7-6
   introduction to  1-6
   methods for implementing  7-6
   parameters  7-3
   using FILTER keyword of
      EMSDIST  13-36
   using SUPPRESS keyword of
      EMSACOLL  13-3
   using SUPPRESS keyword of
      EMSCCTRL  13-15
BEL character  14-5, 20-6, 20-8
Block mode applications
   CONSOLE device incompatible with  13-16
   Console device incompatible with  12-15
   BLOCKING attribute  12-6
Both  6-1
Buffer
   display-text  15-20
   event message  2-13, 12-21, 15-20
   primary collector  12-21
   SPI command  19-1
   SPI response  19-1
BUFFERED keyword, EMSCCTRL  13-10
BUILD_Z0_PROCESS  12-22
Burst detection and suppression
   See BDS
Burst filters
   combined with other EMS filters  6-8
   configuration parameters  6-9
   defined  2-12
detecting and suppressing event bursts 6-8

directives 6-9

example of 6-11

function of 6-7

used for BDS 6-7

used with PLF 2-8

CDIST option of SWITCH keyword, EMSCCTRL 13-16

CDISTMODE keyword of EMSCCTRL 13-10

CDISTSTOP keyword of EMSCCTRL 13-11

CDISTSTOP keyword, EMSCCTRL 12-14

CDISTUSER keyword, EMSCCTRL 13-11

Changing the distributor environment 4-4

Clock reset 20-31

CLOSE procedure 15-3

Closing collector 8-14

COBOL85 15-3

Collector

alternate 2-7

closing 8-14

commands 19-1/19-61

common ZSPI-TKN-RETCODE values 19-27

summary of 19-2

connecting and disconnecting 4-6

errors 19-28, 22-1, 22-27

functions 2-7

interface 2-11

log file management 2-8

opening 8-12

primary 2-7

queue 12-21

relation to compatibility distributor 2-11

responses 19-1/19-61

selection of for internal events 16-3

subsystem support for 2-8

support for distributors 2-9

use of BDS with 7-2

COLLECTOR keyword, EMSDIST 13-35

Collector-generated messages 20-1/20-65

alternate collector messages 20-2

definition of 20-1

primary collector messages 20-1

summary of primary collector messages 20-2

tokenized operator-console messages 20-1, 20-4, 20-6

tokenized text messages 20-1

Command buffer, SPI 19-1

Commands

collector 19-1/19-61

distributor 17-1/17-41

object types supported 17-3

SPI 19-2

ZCOM-CMD-ADD, collector 19-30

ZCOM-CMD-ADD, distributor 17-12

ZCOM-CMD-ALTER, collector 19-31

ZCOM-CMD-ALTER, distributor 17-13

ZCOM-CMD-DELETE, collector 19-42

ZCOM-CMD-DELETE, distributor 17-24

ZCOM-CMD-GETVERSION, collector 19-43

ZCOM-CMD-STATUS, collector 19-51

ZCOM-CMD-STATUS, distributor 17-30

ZCOM-CMD-STOP, collector 19-59

ZEMS-CMD-CONTROL, collector 19-34

ZEMS-CMD-CONTROL, distributor 17-17

ZEMS-CMD-GETEVENT, distributor 17-25

ZEMS-CMD-GETVERSION, collector 19-45

ZEMS-CMD-GETVERSION, distributor 17-28
ZEMS-CMD-REPLACE, distributor 17-29
ZEMS-CMD-STATUS, collector 19-53
ZEMS-CMD-STATUS, distributor 17-41
ZEMS-CMD-STOP, collector 19-60
Comments, filter language 5-9
Common definitions
  ZCOM- commands
    collector 19-4
distributor 17-5
  ZEMS- commands
    collector 19-27
distributor 17-9
Compatibility distributor
  characteristics 2-11
default mode 12-15, 12-22
defined 2-9
EMSTEXT and 2-14
  mode
    changing 12-22
default 12-15, 12-22
    selecting 12-22
object file 12-22
primary process of, switching 12-14
purpose of 2-10
reasons to use 12-22
relation to primary collector 2-11
selection criterion 2-11
source of its event messages 2-11
starting 12-22
stopping 12-14
SYSGEN considerations 12-22
system-disk CPUs and 12-22
Compiled filters 2-11, 5-1
Compiling a filter specification 5-4
Compound statement, filter language 5-31
Conditional tokens 9-17
CONFAUX file 12-21, 12-22
Configuration file 12-21, 12-22
Configuration issues 12-16
Configuring EMS
  alternate collector 12-2
delivery-integrity issues of 12-19
event messages 12-19
for collector performance 12-19
for file security 12-20
for system reliability 12-20
for task support 12-16
log files 12-3, 12-5
network considerations 12-16, 12-17
primary collector 12-2
system resource use 12-20
trade-offs in 12-16
CONSOLE device
  block mode applications incompatible with 12-15, 13-16
command 12-15
set with ZEMS-CMD-CONTROL command 19-39
Console-compatible format
  EMSTEXT and 2-12, 15-19
Consumer distributor
  defined 2-9
  starting 4-2
CONSUMER keyword, EMSDIST 13-35
Context
  of alternate collector 12-24
  of primary collector 12-23
CONTROL command
  alternate collector 19-34
  attributes changed by 19-34
  compatibility distributor 19-34
distributor 17-23
Creating
  event messages 8-13
  filters 3-3, 5-3
Critical events 2-3, 8-3, 13-40

D
D30 token 15-16
Data definition file (DDL), creating 8-11
Declarations, filter language 5-26
DEFAULTSUBVOL attribute 12-7
Definition files 2-14
Definitions, EMS 14-1
DELAY keyword, EMSDIST 13-39
DESTINATION statement, filter language 5-33
Directives
  burst filters 6-9
  filter table 6-3
DISCACCESSID attribute 12-7
DISCARDEVENT statement, filter language 5-35
Display format
  EMSTEXT and 2-12, 15-19
  event-message headers in 15-19, 15-22
Display text
  buffer 15-20
  generating 2-12, 4-18, 15-19
Displaying event messages 3-2
Distributor
  changing environment of a 4-4
  commands 17-1/17-41
  error codes 21-6
  error lists 17-10
  error messages 21-6
  error numbers 17-11
  errors 17-10, 21-1
  event messages 18-1/18-57
  function of 12-12
  initiating a 2-10
  installing filters in a 17-20
  responses 17-1/17-41
  specifying options of a 2-10
  starting 12-13, 12-14
  startup error messages 13-40, 21-6
  support from collector 2-9
  types of 2-9
  use of 12-12
  using BDS with 7-2
Distributor object files
  compatibility distributor 12-22
Distributor, consumer 4-2
DSM display format
  EMSTEXT and 2-12, 15-19
  event-message headers in 15-19, 15-22
DSM environment 1-3
DUMP ON keyword, EMSDIST 13-40
E
EMF default subsystem ID 5-12
Emphasis token 8-9, 14-5
EMS
  applications of 1-8
  architecture 2-1/2-15
  basic capabilities of 1-6
  capabilities and features of 1-5
  components 2-1/2-15, 12-1
  definitions 14-1, 14-11
  event messages
    tokens in 1-7
  filters, types of 2-11
  in DSM environment 1-3
  interface in application 8-12
  interfaces 1-3, 1-5
  introduction to 1-1
  key features of 1-6
  procedures 15-1/15-25
    declarations required for 15-7
    definitions required for 15-7
    examples of use
      determining collector type 15-6
      issuing a text message 15-5
      opening a collector to send a command 15-5
      opening a collector to send an event 15-4
sending a command 15-6
sending an event 15-4
overview of 2-15
passing tokens by reference or value 15-2
summary of 15-2
programs 13-1/13-51
system environment of 1-2
EMS filter 5-3
EMSACOLL
RUN command 13-2
run-time parameters
ALLOCATE 13-6
BACKUP 13-4
BUFFERED 13-5
CPU 13-2
DEFAULTSUBVOL 13-4
EXT 13-5
FILTER 13-3
LOGPREFIX 13-4
LOGSUBVOL 13-4
MAXFILE 13-5
NAME 13-2
NOWAIT 13-2
POOLPAGES 13-6
PRI 13-2
REFRESH 13-5
REPLYAFTERWRITE 13-6
ROTFILES 13-4
SECURITY 13-5
SUPPRESS 13-3
startup error messages 13-6
startup warning messages 13-8
using 13-2
EMSADDSDSUBJECT procedure 15-8
EMSADDSDSUBJECTMAP procedure 15-8
EMSADDSTOKENMAPS procedure 15-10
EMSADDSTOKENS procedure 15-10
EMSCCTRL keywords
ALLOCATE 13-10
BLOCKING 13-10
BUFFERED 13-10
CDISTMODE 13-10
CDISTSTOP 12-14, 13-11
CDISTUSER 13-11
EXT 13-11
FILTER 13-11
HELP 13-9
LOGSUBVOL 13-12
LOGUSERID 13-13
MAXFILE 13-13
NEXTLOGFILE 13-13
REFRESH 13-14
ROTFILES 13-14
SECURITY 13-14
SUPPRESS 13-15
SWITCH 12-14, 13-16
TEXTOUT 13-16
EMSCCTRL program 13-9
using 13-9
EMSCINFO program
DETAIL display 13-27
SUPPRESS OFF, FILTER ON 13-28
SUPPRESS ON, FILTER OFF 13-22
SUPPRESS ON, FILTER ON 13-26
sample output
alternate collector 13-19
definition of terms 13-19
primary collector 13-18
EMSDINFO 13-29
EMSDIST keywords
AUTOSTOP 13-39
BACKUP 13-35
COLLECTOR 13-35
CONSUMER 13-35
DELAY 13-39
DUMP ON 13-40
FILTER 13-36
FORWARDING 13-35
GMT ON 13-40
INDENT 13-40
LOGFILE 13-36
PRINTING 13-35
SBUF 13-39
STOP 13-38
TARGET 13-36
TEXTOUT 13-36
TIME 13-37
TYPE 13-35
WAIT 13-37

EMSDIST program 13-34
EMSFLAGS parameter 12-22
EMSINIT procedure 15-17
EMSINITMAP procedure 15-17
EMSTEXT procedure
command 15-19
error messages 2-13
introduced 2-12
numbered console messages and 15-22
using 4-18

EMSTEXTMATCH function, filter language 5-39
EMSTEXTV command 15-19
EOFREFRESH attribute 12-7
Error codes, distributor 21-6
Error lists, distributor 17-10
Error messages
distributor 21-6
EMSTEXT 2-13

Error numbers, distributor 17-11
Errors, collector 19-28

ZCOM-ERR-

CMD-INV-SUMSTATE 22-4
CMD-MISMATCH 22-4
CMD-NOT-SUPP 22-5
CMD-NOT-SUPP-BY-OBJ 22-9
OBJ-ALRDY-DEF 22-5
OBJ-NOT-FOUND 22-5
SECUR-VIOL 22-6
SPI-ERROR 22-6
SSID-INV 22-6
SUB-NOT-FOUND 22-7
TKN-CODE-INV 22-7
TKN-DUP 22-7
TKN-LEN-INV 22-8
TKN-REQ 22-8
TKN-VAL-INVAL 22-8
VSN-INCOMP 22-9

ZEMS-ERR-

ACC-VIOL 22-26
ALLOC-LOG 22-28
CDIST-CPU 22-13, 22-23
CDIST-DOWN 22-17, 22-27
CPU-RANGE 22-13, 22-23
DEST-ACCESS 22-15, 22-25
DUP-TKN 22-22
INV-CMD 22-21
INV-CPU 22-12, 22-23
INV-MODE 22-16, 22-27
INV-OBJECT 22-22
INV-OCURS 22-24
INV-SSID 22-21
INV-TKN 22-21
INV-VALUE 22-22
LOGGING-STOPPED 22-28
LOG-ACCESS 22-24
NO-BACKUP 22-27
OPEN-LOG 22-15, 22-25
REQ-TKN 22-24
VERSION 22-20
ZOPR-SYNC 22-18
ZSPI 22-26

Errors, distributor 17-10

ZEMS-ERR-

BAD-EVENT 21-18
BAD-FILTER 21-16
COLLECTOR-EXISTS 21-11
COLL-ACCESS 21-9
COLL-DISCONNECT 21-19
COLL-NOT-FOUND 21-12
COLL-PROTOCOL 21-17
CONTEXT 21-15
DEST-ACCESS 21-14
DEST-CONFLICT 21-21
DEST-EXISTS 21-14
DEST-NOT-FOUND 21-15
DEVTYPE 21-17
DIST-ALLOC 21-21
DUP-TKN 21-7
EOF 21-13
FLT-ALLOC 21-20
FLT-FORM 21-9
FORWARD-SEARCH 21-13
HIST-MODE 21-11
INV-CMD 21-6
INV-HEADERTYPE 21-9
INV-OBJECT 21-8
INV-OP 21-8
INV-PROFILE 21-21
INV-SSID 21-6
INV-TKN 21-7
INV-VALUE 21-7
LOG-ACCESS 21-12
MAX-COLLECTOR 21-11
MAX-DEST 21-13
MODE-CONFLICT 21-8
NO-EVENT-SOURCE 21-16
NO-POOL 21-16
REQ-PARAM 21-10
REQ-TKN 21-8
STARTUP-FAILED 21-22
STAT-ONLY 21-20
VERSION 21-6
WRITE-FAILED 21-22

ZSPI 21-15
Errors, filter table 6-5
Escape character (~), filter language 5-9
Event message
buffer 2-13, 12-21, 15-20
collectors 2-7
distributors 2-9
filter 5-1
source 17-22
Event messages
basics of 2-2
creating 8-13
displaying current 3-2
extracting tokens from 4-16
flow of 2-5
formatting text of 2-12, 15-19
information in 2-2
numbered operator-console 20-4, 20-6
requirements and conventions of 14-2
retrieving with GETEVENT 4-12
sending 8-13
size restrictions of 14-3
sources of 4-6
special types of 2-3
token definitions of 14-3
tokens in 1-7
using PLF filters 2-5
Event messages, collector 20-1/20-65
ZEMS-EVT-
ACOL-ALLOCATE-ERR 20-38
ACOL-BACKUP-ABENDED 20-47
ACOL-BACKUP-CREATED 20-46
ACOL-BACKUP-DELETED 20-49
ACOL-CHECKOPEN-FAILED 20-40
ACOL-CHECKPOINT-ERR 20-50
ACOL-CREATEBACKUP-ERR 20-44
ACOL-EVENT-DISCARDS 20-28
ACOL-INTERNAL-ERR 20-35
ACOL-SHUTDOWN 20-36
ACOL-TAKEOVER 20-42
BURST-END 20-60
BURST-START 20-57
COLD-LOAD 20-9
COLL-DISC-FAILED 20-14
COL-EVENT-DISCARDS 20-18
COMPAT-DISTR-STOPPED 20-16
FILESWITCH 20-11
FILE-ROTATE-PURGE 20-22
INVALIDEVENT 20-33
LOGGING-STOPPED 20-24
LOGTIME-DECREASE 20-30
MSGR-EVENTS-DISCARDED 20-20
WRITE-TO-0 20-7

Event messages, distributor 18-1/18-57

ZEMS-EVT-
BACKUP-ABENDED 18-39
BACKUP-CREATED 18-37
BACKUP-DELETED 18-41
BAD-EVENT 18-25
BAD-FILTER 18-21
BAD-LOG 18-44
BURST-END 18-10
BURST-START 18-7
CHECKOPEN-FAILED 18-30
CHECKPOINT-FAILED 18-42
COLL-ACCESS 18-15
COLL-PROTOCOL 18-23
COL-DISCONNECT 18-49
CREATEBACKUP-FAILED 18-34
DEST-ACCESS 18-17
DEVTYPE 18-27
FILES-LOST 18-46
INTERNAL-ERROR 18-29
LOGFILE-EOF 18-19

LOG-ACCESS 18-13
STARTUP-FAILED 18-51
STARTUP-OK 18-54
TAKEOVER 18-32
WRITE-FAILED 18-56

Event number 8-7
Event reporting guidelines 8-2, 8-5
Event routing
capability 16-1
collectors, selecting for 16-3
destination processes, launching 16-1
distributor generated messages 16-4
expanding number of destinations with 16-1
expanding number of event sources with 16-1
multiple filters 16-2
programmatic interfaces
CONTROL command 17-13
STATUS command 17-33
selecting format 16-2
user interfaces
filter DESTINATION statement 16-3
filter PASS statement 16-3
multiple filters startup 16-3

Event suppression 7-1
Events

time 2-4, 8-3
critical 2-3, 8-3
reporting 8-2, 8-5

Extended programmatic interface commands supported by 17-2
for distributor 17-2

Extracting tokens
from event message 4-16
from GETEVENT response 4-16

F
FAIL statement, filter language 5-35
Fields, filter language 5-14
Files
  compatibility-distributor object 12-22
  CONFAUX 12-21, 12-22
  configuration 12-21, 12-22
  definition 2-14
  SYSGEN auxiliary 12-22
  template 12-23
    installed at SYSGEN 2-12
Filter
  collector, purpose of 2-11
  creating 3-3, 5-3
  distributor, purpose of 2-11
  EMS 5-3
    installing 17-20
    loading and replacing a 4-4
    multiple filters 6-6, 16-2
    operating environment 5-1
    sample specification 5-3
    specification, compiling for 5-4
    types of 2-11
      writing
        compatibility 5-6
        correctness 5-7
        efficiency 5-6
        selectivity 5-5
Filter compiler
  completion codes from 5-62
  directives 5-45
  error messages 5-50
  errors and warnings 5-47
  fatal-error messages 5-59
  input and output 5-44
  invoking 5-46
  LIST directive 5-45
  NOLIST directive 5-46
  running 5-46
  SOURCE directive 5-46
  TACL fatal-error messages 5-61
  warning messages 5-48
FILTER declaration, filter language 5-27
FILTER keyword
  EMSACOLL 13-3
  EMSCCTRL 13-11
  EMSDIST 13-36
Filter language
  assignment statement 5-30
  basic components of 5-9
  bit-extraction operator 5-19
  Boolean expressions 5-25
  comments 5-9
  comparing
    fields 5-24
    file names 5-23
    missing values 5-25
    signed values 5-23
    strings 5-22
    unsigned values 5-23
    values 5-22
  comparison syntax 5-19
  compound statement 5-31
  constant lists 5-18
  constants 5-15
  data types, token and EMF 5-20
  declarations 5-26
  defined 5-7
  destination statement 5-33
  EMF default subsystem ID 5-12
  EMSTEXTMATCH function 5-39
  Escape character (~) 5-9
  FAIL statement 5-35
  features of 5-2
  fields 5-14
  file names 5-17
  FILTER declaration 5-27
  filter operation and 5-8
  filter parameters and 5-29
  functions 5-38
IF statement 5-36
integers 5-16
LITERALLY function 5-41
MATCH function 5-42
names in 5-10
NULL subsystem ID 5-12
PASS statement 5-36
precedence of operators in 5-26
reserved words in 5-9
SSID function 5-42
statements 5-30
strings 5-16
subsystem IDs 5-16
subsystem ID, NULL 5-12
TACL environment of 5-8
TOKENPRESENT function 5-43
tokens, references to
qualified 5-12
unqualified 5-11
Filter parameters
filter language and 5-29
passing and replacing 4-4, 17-21
Filter tables 6-1
defined 2-12
directives 6-3
errors 6-5
format of 6-2
keywords for 6-3
logical connection AND 6-7
methods for loading 6-6
PASS or FAIL types 6-1
recovery actions 6-6
restrictions 6-7
Format templates
formatting event-message headers
with 15-22
identification by an event message 2-12, 15-22
nonexistent 2-13
Formatting event-message headers 15-19
Forwarding distributor 2-9
FORWARDING keyword, EMSDIST 13-35
Functions, filter language 5-38

G
Generating display text (EMSTEXT) 4-18
GMT ON keyword, EMSDIST 13-40
GMTtime token 4-10
Guidelines for reporting events 8-2

H
Header tokens, event-message 14-2
Header-template-key parameter of EMSTEXT 15-19
HELP keyword, EMSCCTRL 13-9

I
If 10-27
IF statement, filter language 5-36
INDENT keyword, EMSDIST 13-40
InfoWay, files formerly available in C-1
Installation considerations for EMS 12-21
Installing filters 17-20

L
LITERALLY function, filter language 5-41
Loading standard definitions 5-4
Log file attributes
alternate collector 2-9
changing 12-10
DISCACCESSID 12-7
EOFREFRESH 12-7
MAXFILE 12-7
NEXTLOGFILE 12-8
primary collector 2-9
PRIMARYEXTENT 12-8
PROTECTION 12-9
SECONDARYEXTENT 12-10
WRITETHRUCACHE 12-10
Log files
  alternate collector 12-4
  configuring 12-3, 12-5
  management 2-8
  operation of 12-23
  positioning the distributor 4-10
  positioning, effect of clock reset on 20-31
  primary collector 12-3
  switching 12-28
LOGFILE keyword, EMSDIST 13-36
Logical connection AND
  filter tables 6-7
LOGPREFIX keyword, EMSACOLL 13-4
LOGSUBVOL keyword, EMSCCTRL 13-13
Logtime token 4-10
LOGUSERID keyword, EMSCCTRL 13-13

M
Manager token 8-9
MATCH function, filter language 5-42
MAXFILE
  attribute 12-7
  keyword, EMSCCTRL 13-13
Multiple filters 6-6, 16-2

N
NEXTLOGFILE
  attribute 12-8
  keyword, EMSCCTRL 13-13
Numbered console messages, EMSTEXT and 15-22

O
Object programs 2-14
OPEN procedure 15-3
Opening the collector 8-12

P
Parameters, filter
  filter language and 5-29
  passing 17-21
PASS statement, filter language 5-36
PLF (pre-log filtration)
  collector events not suppressed by 7-9
  definition of 2-8
  introduction to 1-6
  using FILTER keyword of EMSACOLL 13-3
  using FILTER keyword of EMSCCTRL 13-11
POOLPAGES, alternate collector attribute 12-8
Positioning in log files
  effect of clock reset on 20-31
  with ZEMS-TKN-GMTTIME 4-10
  with ZEMS-TKN-LOGTIME 4-10
Positioning within event-message sources 17-22
Pre-log filtration
  See PLF
Primary collector
  buffer 12-21
  configuring 12-2
  context 12-23
  definition of 1-2
  event message buffer 12-21
  features of 2-7
  log files 12-3
    attributes 12-26
    names 12-25
  logging
    after a cold load 12-25
    after a subvolume switch 12-27
  queue 12-21
  SYSGEN considerations 12-21
  PRIMARYEXTENT attribute 12-8
  Printing distributor

EMS Manual—426909-005
Index-11
definition of 2-9
EMSTEXT and 2-14
PRINTING keyword, EMSDIST 13-35
Procedures, EMS
   declarations required for 15-7
   definitions required for 15-7
   summary of 15-2
Programs
   collector
      EMSACOLL process 13-2
      EMSCCTRL control utility 13-9
      EMSCINFO status utility 13-18
   distributor
      EMSDINFO status utility 13-29
      EMSDIST process 13-34
   utility, overview of 2-15
PROTECTION attribute 12-9
PUP commands 12-15

Q
Queue, collector 12-21

R
Recovery actions, filter table 6-6
REPLYAFTERWRITE attribute 12-9
Reporting events
   action tokens 8-9
   emphasis token and 8-9
   event number and 8-7
   example of B-1
   guidelines for 8-2, 8-5
   information provided by EMS 8-6
   information provided optionally 8-10
   information to include in messages 8-5
   information you provide 8-6
   introduction to 8-1
   manager token and 8-9
   required tokens 8-6
   subject token and 8-8
   subsystem ID and 8-7
   terminology 14-1
   tokens required for 8-6
   Reserved words in filter language 5-9
   Resetting system clock
      effect on time-positioning in log files 20-31
   Response buffer, SPI 19-1
   Responses
      collector 19-1/19-61
      distributor 17-1/17-41
   Restrictions, filter tables 6-7
   Retrieving event information 4-12
   Retrieving event messages 3-1, 4-12
      example of A-1
      programmatically 4-1
   ROTATEFILES keyword, EMSCCTRL 13-14

S
SBUF keyword, EMSDIST 13-39
SECONDARYEXTENT attribute 12-10
Security 12-20
SECURITY keyword, EMSCCTRL 13-14
Selection criterion, compatibility distributor 2-11, 12-15
SenderID token 15-15
Sending an event message 8-13
Some 15-2
Source of event messages
   for filter execution 5-2
   specifying 4-6, 17-22
   time-positioning within 17-22
   Special operations of EMSGET and EMSGETTKN 15-15
   Special-operation tokens, usage restrictions on 15-17
   Specific-event suppression
      definition of 7-1
      from a distributor 7-8
      from a primary collector 7-8
from an alternate collector implementation methods

Specify
SPI commands, sending
SSID function, filter language
Standard definitions, loading
Standard EMS templates
Standard event procedure calls
condition code settings

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMS_OBJ_AVAIL_EVT_BLD_11-1</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_OBJ_UNAVAIL_EVT_BLD_11-11</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_OPER_ATTN_COMPED_EVT_T_BLD_11-30</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_OPER_ATTN_NEEDED_EVT_BLD_11-26</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_OTHER_STATE_CHANGE_EVT_BLD_11-21</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_TRANSIENT_FAULT_EVT_BLD_11-5</td>
<td>11-1</td>
</tr>
</tbody>
</table>

considerations

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMS_OBJ_AVAIL_EVT_BLD_11-1</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_OBJ_UNAVAIL_EVT_BLD_11-11</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_OPER_ATTN_COMPED_EVT_T_BLD_11-30</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_OPER_ATTN_NEEDED_EVT_BLD_11-26</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_OTHER_STATE_CHANGE_EVT_BLD_11-21</td>
<td>11-1</td>
</tr>
<tr>
<td>EMS_TRANSIENT_FAULT_EVT_BLD_11-5</td>
<td>11-1</td>
</tr>
</tbody>
</table>

Standard events

89 enumeration clause
analyzing your subsystem environment
AS clause
building message buffer
collectors for your events

common-standard tokens provided by EMS
common-standard tokens provided by subsystems
compiling EMS code
compliance with
conditional tokens
critical or non-critical events
customizing events

DDL definitions

<table>
<thead>
<tr>
<th>Clause</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>89 enumeration clause</td>
<td>10-22</td>
</tr>
<tr>
<td>AS clause</td>
<td>10-22</td>
</tr>
<tr>
<td>compiling</td>
<td>10-26</td>
</tr>
<tr>
<td>creating and building</td>
<td>10-22</td>
</tr>
<tr>
<td>defining event subjects</td>
<td>10-19</td>
</tr>
<tr>
<td>defining private events for your subsystem</td>
<td>10-9</td>
</tr>
<tr>
<td>defining system data for reactive problem management example</td>
<td>10-11</td>
</tr>
<tr>
<td>defining templates for</td>
<td>10-27</td>
</tr>
<tr>
<td>defining the EMS collectors for</td>
<td>10-27</td>
</tr>
<tr>
<td>description</td>
<td>9-17</td>
</tr>
<tr>
<td>designing your event messages</td>
<td>10-15</td>
</tr>
<tr>
<td>determining the standard events for your subsystem</td>
<td>10-6</td>
</tr>
<tr>
<td>displayed on operator console</td>
<td>10-15</td>
</tr>
</tbody>
</table>

EMS

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>appropriate platform for</td>
<td>10-10</td>
</tr>
<tr>
<td>compiling code</td>
<td>10-29</td>
</tr>
<tr>
<td>compiling templates</td>
<td>10-27</td>
</tr>
<tr>
<td>releasing and distributing files</td>
<td>10-31</td>
</tr>
<tr>
<td>templates</td>
<td>10-26</td>
</tr>
<tr>
<td>creating and building</td>
<td>10-26</td>
</tr>
<tr>
<td>description</td>
<td>9-34</td>
</tr>
<tr>
<td>testing code</td>
<td>10-30</td>
</tr>
<tr>
<td>event generation code, coding and testing</td>
<td>10-27</td>
</tr>
<tr>
<td>event message buffer</td>
<td>10-29</td>
</tr>
</tbody>
</table>
event messages, describing 10-20
event number definitions, ZEMS-TKN-EVENTNUMBER 10-23
event numbers 9-12
and CDMT attributes 10-19
event subject DDL names 10-19
event subject definitions 10-24
event template
  object available 9-34
  object other state change 9-35
  object unavailable 9-35
  operator attention completed 9-36
  operator attention needed 9-36
  transient fault 9-37
  usage threshold 9-37
event tokens
  object available 9-25
  object other state change 9-26
  object unavailable 9-26
  operator attention completed 9-29
  operator attention needed 9-29
  transient fault 9-30
  usage threshold 9-31
event types
  defining 10-19
  standard 10-33
  ZSAM-VAL-data-usage 10-33
extensions to 9-38
external specification, writing 10-17
fictitious Compaq subsystem, example of 10-31
for your subsystem 10-5
for ZSAM-TKN-subj-netx25 10-33
for ZSAM-TKN-subj-tape 10-36
for ZSAM-TKN-subj-tp 10-35
generating
  example of 10-31
  overview 10-1
incorporating old events into new events 10-14
intent and assumptions for
  generating 10-1
introduction to 9-1
management functions
  data required for automation of 10-11
  groupings 10-10
  operations defined for 10-11
private 10-19
manager process relationship to events 10-16
MAP token 10-25
message buffer 10-28
message text 10-16
migrating existing events 10-13
migration rules 10-13
numbering 10-15
object name
  for a group of objects 9-16
  for event subject 9-13
  for underlying object 9-15
object state monitoring functions 9-4
objects
  characteristics of 10-4
  managed 10-32
  managing 10-3
  operational states 10-6
  state transitions 10-4
  using 10-9
obtaining a subsystem ID and acronym 10-2
operator assistance 10-9
other subsystem tokens
  assign enumerations to 10-27
philosophy 9-10
private event message buffer
  build 10-28
private event type definitions
  ZEMS-TKN-CONTENT-USER 10-23
private tokens 10-20, 10-24, 10-36
private values, enumerate to standard tokens 10-20
proactive problem management functions 9-8
problems
bypass and recovery 10-12
data for resolving 10-13
detection and isolation 10-12
diagnosing 10-12
identifying causes 9-6
rediscovery 9-7
tracing and control 10-13
procedure calls
for generating events 11-1
introduction to 11-2
procedures
cause, effect, and recovery 10-16
EMS_COMMON_TOKENS_EVT_BL D_ 11-44
EMS_OBJ_AVAIL_EVT_BLD_ 11-3
EMS_OBJ_UNAVAIL_EVT_BLD_ 11-7
EMS_OPER_ATTN_COMPED_EVT_BLD_ 11-27
EMS_OPER_ATTN_NEEDED_EVT_BLD_ 11-23
EMS_OTHER_STATE_CHANGE_EVT_BL D_ 11-18
EMS_TRANSIENT_FAULT_EVT_BLD_ 11-2
program example 11-48
sending events 11-48
product information of subsystem 10-22
production requests requiring operator attention 9-9
program example, procedures 11-48
reactive problem management functions 9-5
requirements for 9-3
revising old events 10-14
sample files
DDL file 10-1
EMS template 10-1
external specification 10-1
simple tokens 10-25
SSID 10-22
standard and private events 10-16
standard EMS tokens, assign enumerations to 10-26
standard tokens, private enumerations for 10-23
standard usage threshold event
EMS_USAGE_THRESHOLD_CHK 11-35
EMS_USAGE_THRESHOLD_EVT_BLD_ 11-36
EMS_UTCB_INIT_ 11-33
programming example 11-41
standard usage threshold event algorithm 11-31, 11-32
state transition diagram
for ZSAM-TKN-subj-netx25 10-33
for ZSAM-TKN-subj-tape 10-36
for ZSAM-TKN-subj-tp 10-35
subjects 10-33
subsystem
99(number) 10-32
description 10-26, 10-32
product information 10-22
SAMPLER (name) 10-32
ZSAM (acronym) 10-32
subsystem environment 10-17
subsystem ID 10-22
system data
event types 10-13
reporting 10-10
system resources, using 9-8
templates for 10-36
tokens
conditional and unconditional 10-16
internal 10-16
nature of 10-25
of other subsystems, private enumerations 10-24
of your subsystem, private enumerations 10-24

transient faults 9-8
transition faults, subsystem detection of 10-9
unconditional tokens 9-17
usage threshold programming example 11-41
ZEMS-TKN-CONTENT-USER, private event type definitions 10-23
ZEMS-TKN-EVENTNUMBER, event number definitions 10-23

Standard events usage threshold event condition code settings
EMS_COMMON_TOKENS_EVT_BLD_ 11-47
EMS_USAGE_THRESHOLD_CHK_ 11-35
EMS_USAGE_THRESHOLD_EVT_BLD_ 11-39
EMS_UTCB_INIT_ 11-34

considerations
EMS_COMMON_TOKENS_EVT_BLD_ 11-47
EMS_USAGE_THRESHOLD_CHK_ 11-36
EMS_USAGE_THRESHOLD_EVT_BLD_ 11-39
EMS_UTCB_INIT_ 11-35

Starting a consumer distributor 4-2
Startup error messages, distributor 13-40, 21-6
STOP EMSCDIST command 12-14
STOP keyword, EMSDIST 13-38
Subject token 8-8, 15-16

Subsystem ID 8-7
Subvolume switch 12-27
SUPPRESS keyword
EMSACOLL 13-3
EMSCCTRL 13-15
SUPPRESS-DISPLAY token 14-6
SWITCH keyword, EMSCCTRL 12-14

Switching
log files 12-28
subvolumes 12-27

SYSGEN
of compatibility distributor 12-22
of primary collector 12-21
SYSGEN auxiliary file 12-21, 12-22
SYSGEN considerations for EMS 12-21
SYSGEN specification, ALLPROCESSORS section 12-22

System clock reset 20-31
System-disk CPUs, $Z0 running on 12-22

SYSTEM_PROCESS_MODIFIERS paragraph
EMSFLAGS parameter 12-22
XPOOLPAGES parameter 12-21

T

TACL commands
EMSTEXT 15-19
EMSTEXTV 15-19
STOP EMSCDIST 12-14

TACL environment of filter language 5-8
TARGET keyword, EMSDIST 13-36

Template files 12-23
installed at SYSGEN 2-12
OPEN error 2-13
READ error 2-13

Templates, format
formatting event-message headers 15-22
identification by an event message 2-12, 15-22
nonexistent 2-13
Text display from event messages 2-12, 15-19

Text formats
- console-compatible and EMSTEXT 2-12, 15-19
- display format and EMSTEXT 2-12, 15-19
- DSM display format and EMSTEXT 2-12, 15-19

Text token 2-13
TEXTOUT keyword
- EMSCCTRL 13-16
- EMSDIST 13-36

Text-formating procedure 2-12, 15-19

The 12-20, 13-19
The MAKE TACL Macro File B-3
This 5-44

TIME keyword, EMSDIST 13-37

Time-positioning in log files
- effect of clock reset on 20-31
  with ZEMS-TKN-GMTTIME 4-10
  with ZEMS-TKN-LOGTIME 4-10

Token definitions, event-message
- EMS data-portion tokens 14-9
- EMS header tokens 14-4
- EMS special token codes 14-8
- SPI data-portion tokens 14-10
- SPI header tokens 14-3

TOKENPRESENT function, filter language 5-43
Tokens 9-17
TYPE keyword, EMSDIST 13-35

U

Unconditional tokens 9-17
Utility programs 2-15

V

ViewPoint application 2-4, 2-14, 8-3, 14-6

W

WAIT keyword EMSDIST 13-37

Warnings, distributor
- ZEMS-WRN-EOF 21-4
- TOO-EARLY 21-4
- TOO-LATE 21-5

WRITE procedure 15-3
WRITEREAD procedure 15-3
WRITETHRUCACHE attribute 12-10

X

XPOOLPAGES parameter 12-21
XSENDERID token 15-16
XSENDERID-PD token 15-16

Z

ZCOM-CMD-ADD command
  collector 19-30
  distributor 17-12

ZCOM-CMD-ALTER command
  collector 19-31
  distributor 17-13

ZCOM-CMD-DELETE command
  collector 19-42
  distributor 17-24

ZCOM-CMD-GETVERSION command,
  collector 19-43
  distributor 17-30

ZCOM-CMD-STATUS command
  collector 19-51
  distributor 17-30

ZCOM-CMD-STOP command,
  collector 19-59
  ZCOM-ERR-
    CMD-INV-IN-SUMSTATE 22-4
    CMD-MISMATCH 22-4
    CMD-NOT-SUPP 22-5
    CMD-NOT-SUPP-BY-OBJ 22-9
    OBJ-ALRDY-DEF 22-5

EMS Manual—426909-005
Index-17
OBJ-NOT-FOUND 22-5
SECUR-VIOL 22-6
SPI-ERROR 22-6
SSID-INV 22-6
SUB-NOT-FOUND 22-7
TKN-CODE-INV 22-7
TKN-DUP 22-7
TKN-LEN-INV 22-8
TKN-REQ 22-8
TKN-VAL-INVAL 22-8
VSN-INCOMP 22-9

ZEMS-CMD-CONTROL command
  collector 19-34
  distributor 17-17

ZEMS-CMD-GETEVENT command,
  distributor 17-25

ZEMS-CMD-GETVERSION command
  collector 19-45
  distributor 17-28

ZEMS-CMD-REPLACE command,
  distributor 17-29

ZEMS-CMD-STATUS command
  collector 19-53
  distributor 17-41

ZEMS-CMD-STOP command,
  collector 19-60

ZEMS-DDL-
  COL-CDIST-STATUS 19-54
  COL-CONTROL 19-35
  COL-CONTROL-CDIST 19-35
  COL-EVENT-DISCARDS 20-19
  COL-STATUS 19-54
  DIST-STATUS 17-34
    See also ZEMS-DDL-STATUS-DIST
  DIST-TARGET 17-35
    See also ZEMS-DDL-STATUS-TARGET
  STATUS-DIST 17-34, 18-4
  STATUS-SOURCE 17-38

STATUS-TARGET 17-35, 17-40
STATUS-TEXTOUT 17-39

ZEMS-ERR-
  ACC-VIOL 22-26
  ALLOC-LOG 22-28
  BAD-EVENT 21-18
  BAD-FILTER 21-16
  CDIST-CPU 22-13, 22-23
  CDIST-DOWN 22-17, 22-27
  COLLECTOR-EXISTS 21-11
  COLL-ACCESS 21-9
  COLL-DISCONNECT 21-19
  COLL-NOT-FOUND 21-12
  COLL-PROTOCOL 21-17
  CONTEXT 21-15
  CPU-RANGE 22-13, 22-23
  DEST-ACCESS 21-14, 22-15, 22-25
  DEST-CONFLICT 21-21
  DEST-EXISTS 21-14
  DEST-NOT-FOUND 21-15
  DEVTYPE 21-17
  DIST-ALLOC 21-21
  DUP-TKN 21-7, 22-22
  EOF 21-13
  FLT-ALLOC 21-20
  FLT-FORM 21-9
  FORWARD-SEARCH 21-13
  HIST-MODE 21-11
  INV-CMD 21-6, 22-21
  INV-HEADERTYPE 21-9
  INV-MODE 22-16, 22-27
  INV-OBJECT 21-8, 22-22
  INV-OCCURS 22-24
  INV-OP 21-8
  INV-PROFILE 21-21
  INV-SSID 21-6, 22-21
  INV-TKN 21-7, 22-21
  INV-VALUE 21-7, 22-22
  LOGGING-STOPPED 22-28
| BDS-STATS  | 19-14 |
| BURST-STATUS  | 19-15 |
| COL-CDIST-INFO  | 19-24 |
| COL-CDIST-STATUS  | 19-54, 19-58 |
| ZCOL-CDISTBKUPCPU  | field 19-54 |
| ZCOL-CDISTPRICPU  | field 19-54 |
| ZCOL-CDIST-DEF-TEXTOUT  | field 19-54 |
| ZCOL-CDIST-MODE  | field 19-54 |
| ZCOL-CDIST-TEXTOUT  | field 19-54 |
| ZCOL-CDIST-USER  | field 19-54 |
| ZCOL-DISTR-ERROR  | field 19-54 |
| ZCOL-OPRLOG-ERROR  | field 19-54 |
| ZCOL-STOPCOMPATDIST  | field 19-54 |
| COL-CONTROL  | 19-17, 19-34, 19-36, 19-38 |
| ZCOL-DISCACCESSID  | field 19-37 |
| ZCOL-EOFREFRESH  | field 19-37 |
| ZCOL-EVENTBLOCKING  | field 19-38 |
| ZCOL-LOGSUBFILE  | field 19-36 |
| ZCOL-MAXFILINGNN  | field 19-37 |
| ZCOL-NEXTLOGFILE  | field 19-36 |
| ZCOL-PRIMARYCPU  | field 19-36 |
| ZCOL-PRIMARYEXTENT  | field 19-37 |
| ZCOL-PROTECTION  | field 19-38 |
| ZCOL-ROTSFILES  | field 19-36 |
| ZCOL-SECONDARYEXTENT  | field 19-37 |
| ZCOL-WRITETHRUCAHE  | field 19-37 |
| COL-CONTROL-CDIST  | 19-34, 19-39 |
| ZCOL-CDIST-CONSOLE-OUT  | field 19-39 |
| ZCOL-CDIST-MODE-SET  | field 19-39 |
| ZCOL-CDIST-PRICPU  | field 19-39 |
| ZCOL-CDIST-TEXTOUT-SET  | field 19-39 |
| ZCOL-CDIST-USER-SET  | field 19-39 |
| COL-INFO  | 19-22 |
| ZCOL-BACKUPCPU  | field 19-54 |
| ZCOL-BUFFERFAILURES  | field 19-54 |
| ZCOL-CLOSESRECEIVED  | field 19-54 |
| ZCOL-CURRENTFILENAME  | field 19-54 |
| ZCOL-CURRENTRECORD  | field 19-54 |
| ZCOL-DEFAULTFILENAME  | field 19-54 |
| ZCOL-DISCERRORS  | field 19-54 |
| ZCOL-EOFREFRESH  | field 19-54 |
| ZCOL-EVENTBLOCKING  | field 19-54 |
| ZCOL-EVENTSDISCARDED  | field 19-54 |
| ZCOL-EVENTSLOGGED  | field 19-54 |
| ZCOL-EVENTSRECEIVED  | field 19-54 |
| ZCOL-FILESWITCHES  | field 19-54 |
| ZCOL-INVALIDEVENTS  | field 19-54 |
| ZCOL-LOGDISCERROR  | field 19-54 |
| ZCOL-PRIMARYCPU  | field 19-54 |
| ZCOL-MAXFILINGNN  | field 19-54 |
| ZCOL-OPSNSRECEIVED  | field 19-54 |
| ZCOL-PRIMARYCPU  | field 19-54 |
| ZCOL-PRIMARYEXTENT  | field 19-54 |
| ZCOL-PRIORITY  | field 19-54 |
| ZCOL-PROTECTION  | field 19-54 |
ZCOL-ROTFIE Files field 19-54
ZCOL-SECONDARY EXTENT field 19-54
ZCOL-WRITE THRU CACHE field 19-54
DIST-STATUS 17-34
See also ZEMS-MAP-STATUS-DIST
DIST-TARGET 17-35
See also ZEMS-MAP-STATUS-TARGET
EXIOADDR 14-10, 20-6
PLF-STATS 19-15
STATUS-FILTER 19-25
STATUS-SOURCE 17-38
STATUS-TARGET 17-40
ZDIST-TARGET-EVENTS-PER-BLOCK field 17-35
ZDIST-TARGET-NAME field 17-35
ZDIST-TARGET-STATE field 17-35
STATUS-TEXTOUT 17-39
ZEMS-TKN-
ACOL-EVENTS-DISCARDS 20-29
ACTION-ID 14-9, 20-13, 20-25
ACTION-NEEDED 14-9, 20-13, 20-25
ADD-TEXTOUT 17-19
BLOCKLENGTH 18-4, 21-3
CDEV-PRI CPU 19-34, 19-38
CODESEG 20-36
COLLECTOR 20-10
COLNAME 18-4, 18-16, 18-24, 18-47, 18-50, 21-3
COLNAME-ENUM 18-4, 21-3
COL-EVENT-DISCARDS 20-19
ZMESSENGER field 20-19
ZSENDOPMSG field 20-19
ZUSEREVENT field 20-19
ZUSERTEXT field 20-19
COMPATDISTCRTPID 20-17
CONNECT-LOG 17-19
CONNECT-SRC-COLL 17-18, 21-3
CONSOLE-PRINT 14-4
CONTENT-STANDARD 14-6
CONTENT-USER 14-7
CPU 14-4
CU 14-10
D00 15-16
DELETE-TEXTOUT 17-19
DEVICE-TYPE 18-5, 21-3
DEVTYPE-ENUM 18-4, 21-3
DISCONNECT-LOG 17-19
DISCONNECT-SRC-COLL 17-19, 21-3
DIST-NAME 18-4, 18-31, 18-33, 18-35, 18-38, 18-40, 18-42, 18-43
EMPHASIS 14-5
EOFDELAY 17-19
EOFSTOP 4-13, 17-25
EVENT 17-26
EVENTNUMBER 14-4
FAIL-REASON 18-5, 18-47
FILECODE 18-5, 21-3
FILTERFILE 17-17
FILTERNAME 18-5, 18-22
FILTER-ERROR 18-5, 21-4
GENTIME 14-4
GMTTIME 17-18
using 4-10
LASTLOGFILE 18-5, 18-47, 20-12
LCT-LOGTIME 20-13
LDEV 14-9
LDEVNAME 14-9
LOGNAME 18-5, 18-14, 18-20, 18-26, 18-45, 21-4
LOGSTOPREASON 20-25
LOGSWITCHREASON 20-12
LOGTIME 14-4, 17-18
using 4-10
MESSENGERCPU 20-21
MSGR-EVENTS-DISCARDED 20-21
NEWLOGFILE 18-5, 18-48, 20-12
NEWPROCESS-CPU 18-5, 18-38, 20-45, 20-47
NEWPROCESS-ERROR 18-5, 20-45, 20-46
NEWPROCESS-PRIORITY 18-6, 18-38, 20-45, 20-47
NEW-LOGTIME 20-31
NODENUM 14-4
OLD-LOGTIME 20-31
OPMSG 14-10, 20-19
PASSVAL 17-26
PIN 14-4
PREG 20-36
PROC-DESC 14-4
PROGRAMFILE 18-6, 18-30, 18-35, 18-38
PURGEDLOGFILE 20-23
RECORD-ADDRESS 18-6, 21-4
RECOVERY-ENUM 18-6, 18-47
REPLACE-PARAM 17-18
REPLACE-TGT-COLL 17-18
RESET-FILTER 17-18
SENDERID 14-8, 15-15
SEQ-BLOCKING 17-19
SUBJECT 14-8
using 15-16
SUBJECT-MARK 14-9
SUPPRESS-DISPLAY 14-6
SYSPID 20-6, 20-34
TAKEOVERREASON 20-43
TAKEOVER-REASON 18-6, 18-33
TEXT 2-12, 2-13, 14-9, 20-8
USERID 14-4
XSENDERID 15-16
XSENDERID-PD 15-16

ZEMS-VAL-
BLOCKLENGTH-BAD 21-3, 21-17
CHECKPOINT 18-33, 18-43
CHECKSWITCH 18-6
COLL-CONNECT 17-38
COLL-EVENTWAIT 17-38
COLL-GETEVENT 17-38
COLL-IDLE 17-38
COLL-POSITION 17-38
COLL-STATUS 17-38
COLNAME-NOTPRESENT 21-3
COLNAME-PRESENT 21-3
CONSUMER-DIST 17-35
DEVICE-TYPE-BAD 21-3
DEVTYPE-BAD 21-17
DISCACCESS 20-12
DISCFAILED 20-25
DUP-SOURCE-TARGET 21-3, 21-17
EMSCHECKOPEN 18-31
EMSGET 21-5, 21-18
EVT-BUFLEN 14-3
FILECODE-BAD 21-3, 21-17
FILEFULL 20-12
FILTER-EVAL 18-26, 21-18
FORWARD-DIST 17-35
LOGNAME-BAD 21-3, 21-17
NEWPROCESS 18-35, 18-38
NOROTATE 20-25
NO-COLL 17-38
NO-IO 17-38
NO-TARGET 17-40
NO-TEXTOUT 17-40
OPRSWITCH 20-12
PRIMARY-ABEND 18-6
PRIMARY-CPU-DOWN 18-6
PRIMARY-STOPPED 18-6
PRINTING-DIST 17-35
READLOG 17-38
SSID 14-9
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGT-IDLE</td>
<td>17-40</td>
</tr>
<tr>
<td>TGT-RETRY</td>
<td>17-40</td>
</tr>
<tr>
<td>TGT-WRITE-PENDING</td>
<td>17-40</td>
</tr>
<tr>
<td>TXT-IDLE</td>
<td>17-40</td>
</tr>
<tr>
<td>TXT-PRINTING</td>
<td>17-40</td>
</tr>
<tr>
<td>TXT-WAIT-ERROR</td>
<td>17-40</td>
</tr>
<tr>
<td>TXT-WAIT-TIMEDOUT</td>
<td>17-40</td>
</tr>
<tr>
<td>VERSION-INCOMPATIBLE</td>
<td>21-3, 21-17</td>
</tr>
<tr>
<td>ZFILOPEN</td>
<td>18-14, 18-16, 18-18, 18-22, 21-9, 21-10, 21-12, 21-14, 22-13</td>
</tr>
<tr>
<td>ZFILPOSITION</td>
<td>18-14, 18-22, 21-12</td>
</tr>
<tr>
<td>ZFILREAD</td>
<td>18-14, 18-22, 21-10, 21-12</td>
</tr>
<tr>
<td>ZFILWRITE</td>
<td>18-18, 21-14</td>
</tr>
<tr>
<td>ZFILWRITEREAD</td>
<td>18-16, 21-9, 22-13</td>
</tr>
<tr>
<td>ZEMS-WRN-EOF</td>
<td>21-4</td>
</tr>
<tr>
<td>TOO-EARLY</td>
<td>21-4</td>
</tr>
<tr>
<td>TOO-LATE</td>
<td>21-5</td>
</tr>
<tr>
<td>ZEM-MAP-STATUS-SOURCE</td>
<td>17-34</td>
</tr>
<tr>
<td>ZSPI-TKN-COMMAND</td>
<td>17-9, 19-27</td>
</tr>
<tr>
<td>CONTEXT</td>
<td>4-12, 17-25, 17-26, 21-1</td>
</tr>
<tr>
<td>ERROR</td>
<td>22-2, 22-10, 22-19</td>
</tr>
<tr>
<td>MANAGER</td>
<td>14-11</td>
</tr>
<tr>
<td>MAX-FIELD-VERSION</td>
<td>14-3</td>
</tr>
<tr>
<td>PARM-ERR</td>
<td>21-2, 22-3, 22-10, 22-19</td>
</tr>
<tr>
<td>PROC-ERR</td>
<td>18-3, 21-2, 22-3, 22-11, 22-19</td>
</tr>
<tr>
<td>SSID</td>
<td>14-3, 17-10</td>
</tr>
<tr>
<td>SSID-ERR</td>
<td>22-3, 22-11, 22-20</td>
</tr>
<tr>
<td>USEDLEN</td>
<td>14-3</td>
</tr>
<tr>
<td>ZZEVCONF files</td>
<td>12-4, 12-5</td>
</tr>
<tr>
<td>alternate collector</td>
<td>12-24</td>
</tr>
<tr>
<td>definition of</td>
<td>12-3</td>
</tr>
<tr>
<td>FILES switch event message in</td>
<td>20-13</td>
</tr>
<tr>
<td>primary collector</td>
<td>12-24</td>
</tr>
<tr>
<td>ZZEVnnnn files</td>
<td>12-4, 12-5</td>
</tr>
<tr>
<td>after a subvolume switch</td>
<td>12-27</td>
</tr>
<tr>
<td>and the alternate collector</td>
<td>12-5</td>
</tr>
</tbody>
</table>

**Special Characters**

$Z0$

EMSTEXT and starting 12-22

system-disk CPUs 12-22

Alternate collector 12-24

Definition of 12-3

FILESWITCH event message in 20-13

primary collector 12-24

ZZEVnnnn files

after a subvolume switch 12-27

and the alternate collector 12-5