VMS Software

VSI ACMS for OpenVMS Remote Systems Management Guide

Operating System and Version: VSI OpenVMS Alpha Version 8.4-2L1 or higher
VSI OpenVMS IA-64 Version 8.4-1H1 or higher

Software Version: ACMS for OpenVMS Version 5.3-3

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

VSI ACMS for OpenVMS Remote Systems Management Guide

Copyright © 2025 VMS Software, Inc. (VSI), Boston, Massachusetts, USA

Legal Notice

Confidential computer software. Valid license from VSI 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 VSI 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. VSI shall not be liable for technical or editorial errors or omissions contained herein.

All other trademarks and registered trademarks mentioned in this document are the property of their respective holders.

Table of Contents

Preface xi
1. About VSI xi
2. About this manual xi
3. Document Structure xi
4. ACMS Help xii
5. Related Documents xiv
6. OpenVMS Documentation xiv
7. VSI Encourages Your Comments xv
8. Conventions xv

Part I. Introduction

Chapter 1. Overview of Remote Management 3
1.1. Architecture and Implementation 3
1.2. Remote Management Capabilities 4

Chapter 2. Getting Started with the ACMS Remote Manager 5
2.1. Running the ACMS Remote Manager 5
2.1.1. Server Node Setup 5
2.1.1.1. Verify Portmapper (RPC) Setup 6
2.1.1.2. Run the ACMS Postinstallation Procedure 7
2.1.1.3. Define Process Logicals and Symbols 7
2.1.1.4. Prepare the ACMS Environment 7
2.1.1.5. Start the ACMS Remote Manager 7
2.1.2. Client Node Setup 8
2.1.2.1. Run ACMS_POST_INSTALL.COM 8
2.1.2.2. Copy Files and Define Symbols 9
2.1.3. Communicate with the Remote Manager 9
2.2. TCP/IP Setup 10
2.2.1. Set Up the Portmapper (RPC) 11
2.2.1.1. Determine the Current Portmapper Configuration 11
2.2.1.2. Remove the Existing Portmapper Configuration 11
2.2.1.3. Configure the Portmapper 12
2.2.2. Set Up SNMP 13
2.2.2.1. Determine the Current SNMP Configuration 14
2.2.2.2. Remove the Existing SNMP Configuration 14
2.2.2.3. Configure SNMP 14
2.2.2.4. Test SNMP 16
2.3. Remote Manager Setup 16
2.3.1. Run the Postinstallation Procedure 17
2.3.2. Define Process Logicals and Symbols 18
2.3.3. Review and Update the Configuration File 19
2.3.4. Start the Remote Manager 20
2.3.5. Communicate with the Remote Manager 20
2.3.5.1. Using ACMSMGR and Logging In Explicitly 20
2.3.5.2. Using ACMSMGR and a Proxy Account 20
2.4. Troubleshooting the ACMS Remote Manager Startup 21
2.4.1. Problems Starting ACMS 22
2.4.2. Problems Starting the ACMS Remote Manager 22
2.4.2.1. ACMS$MGMT_SERVER.OUT Messages 22

2.4.2.2. Remote Manager Log Entries 23
2.4.3. Problems with the ACMSMGR Utility 25
2.4.3.1. ACMSMGMT-W-NOCLNT_ATTACH Messages 26
2.4.3.2. ACMSMGR Hangs 26
Chapter 3. Using the Remote Manager to Manage ACMS 27
3.1. Overview of the Remote Manager Web Agent 27
3.2. Remote Manager Web Agent Setup 27
3.2.1. Install the Remote Manager Web Agent Software 28
3.2.2. Install the VSI Management Agents for OpenVMS Software 30
3.2.3. Assign Additional Rights Identifiers 31
3.2.4. Start the Remote Manager Web Agent Process 31
3.2.5. Enable Access to Remote Manager Hosts 31
3.2.6. Stop the Remote Manager Web Agent 31
3.3. Using the Remote Manager Web Agent 32
3.3.1. Accessing the ACMS Remote Management Web Page 32
3.3.2. Conventions 33
3.3.3. Customizing the Display 33
3.3.4. Selecting the Remote Manager Host 34
3.4. Issuing Remote Manager Commands 34
3.4.1. Using Show Commands 34
3.4.2. Using Set Commands 35
3.4.3. Using Start and Stop Commands 36
3.4.4. Using Add and Delete Commands 36
3.5. Troubleshooting the Remote Manager Web Agent 37
3.5.1. Reporting Problems 38
Chapter 4. Managing the Remote Manager 39
4.1. Overview 39
4.2. Configuring Remote Manager Startup 39
4.2.1. How to Run the ACMSCFG Utility 40
4.2.2. Displaying Current Values 40
4.2.3. Changing Values 41
4.3. Starting and Stopping the Remote Manager 42
4.3.1. Remote Manager Startup 42
4.3.2. Remote Manager Shutdown 42
4.4. Logging In to the Remote Manager 43
4.4.1. Authentication 43
4.4.1.1. Logging In 44
4.4.1.2. Proxy Accounts 44
4.4.2. Authorization 45
4.4.2.1. Read Access 45
4.4.2.2. Write Access 45
4.4.2.3. Operate Access 45
4.5. Starting and Stopping Interfaces 46
4.5.1. Using ACMSCFG to Enable or Disable Interfaces 46
4.5.2. Using ACMSMGR to Start or Stop Interfaces 46
4.6. Modifying Management Parameters 47
4.6.1. Using ACMSCFG to Modify Management Parameters 47
4.6.2. Using ACMSMGR to Modify Management Parameters 48
4.7. Managing Log Files 48
4.7.1. Setting Audit Levels 48
4.7.2. Displaying Audit Messages 50

4.7.3. Resetting the Audit Log 50
Chapter 5. Using the Remote Manager to Manage ACMS 51
5.1. Managing Data Collection 51
5.1.1. Entities, Classes, Names, and Collections 52
5.1.2. Starting and Stopping Collections 54
5.1.2.1. Using ACMSCFG to Start or Stop Collections 54
5.1.2.2. Using ACMSMGR to Start or Stop Collections 55
5.1.2.3. Using SNMP to Start or Stop Collections 56
5.2. Displaying Collected Data 56
5.2.1. Using ACMSMGR to Display Collected Data 56
5.3. Managing ACMS Using the Remote Manager 57
5.3.1. Types of Variables 57
5.3.1.1. Stored Variables 57
5.3.1.2. Active Variables 57
5.3.2. How the Remote Manager Makes Changes 58
5.3.3. Using ACMSMGR to Modify the ACMS Run-Time System 58
5.3.4. Using SNMP to Modify the ACMS Run-Time System 59
5.3.4.1. Starting and Stopping Processes Using SNMP 60
5.3.4.2. Adding and Deleting Rows Using SNMP 60
5.3.4.3. Replacing Application Procedure Servers Using SNMP 60
5.3.5. Using ONC RPC to Modify the ACMS Run-Time System 60
Chapter 6. Management Programming Using ONC RPC 63
6.1. ONC RPC Overview 63
6.2. Building Multithreaded Clients 65
6.3. API Overview 65
6.4. Initialization and Security 66
6.4.1. Initialization Example 66
6.5. Get Procedures 67
6.5.1. Get Example 68
6.6. List Procedures 68
6.6.1. Linked List Example 69
6.7. Set Procedures 72
6.7.1. Set Example 73
6.8. Delete Procedures 74
6.8.1. Delete Example 75
6.9. Add Procedures 76
6.9.1. Add Example 76
6.10. Start, Stop, and Replace Procedures 77
6.10.1. Start Example 78
Chapter 7. Management Programming Using SNMP 81
7.1. SNMP Overview 81
7.2. SNMP Security 82
7.3. Initializing the SNMP Interface 82
7.4. SNMP Tables 83
7.4.1. Data Type Mapping 84
7.4.2. Single-Row Tables 84
7.4.3. Static Tables 85
7.4.4. Dynamic Tables 86
7.4.5. Servers and Task Groups 86
7.5. SNMP GET Operations 87
7.6. SNMP SET Operations 88

7.7. Using SNMP to Start and Stop ACMS Entities 88
7.8. SNMP Traps 89
7.8.1. EXISTS Traps 90
7.8.2. EVENT_SEVERITY Traps 90
7.9. SNMP Debug Tracing 91
7.9.1. Starting SNMP Debug Tracing 91
7.9.2. Stopping SNMP Debug Tracing 91
7.10. Remote Manager eSNMP Return Codes 92

Part II. Reference Information

Chapter 8. Management APIs 95
8.1. Common RPC Fields 95
8.1.1. Collection Classes 95
8.1.2. Interface Types 95
8.1.3. Enable States 96
8.1.4. Entity Types 96
8.1.5. Facility Types 96
8.1.6. Running States 97
8.1.7. Severity Codes 97
8.1.8. Trap Parameters 98
8.2. Thread-Safe and Non-Thread Safe Clients 98
8.3. ACMSMGMT_ADD_COLLECTION_2 98
8.4. ACMSMGMT_ADD_ERR_FILTER_2 101
8.5. ACMSMGMT_ADD_TRAP_1 104
8.6. ACMSMGMT_DELETE_COLLECTION_1 107
8.7. ACMSMGMT_DELETE_ERR_FILTER_2 110
8.8. ACMSMGMT_DELETE_TRAP_1 112
8.9. ACMSMGMT_GET_ACC_2 114
8.10. ACMS$MGMT_GET_CREDS 116
8.11. ACMSMGMT_GET_ERR_FILTER_2 119
8.12. ACMSMGMT_GET_MGR_STATUS_1 121
8.13. ACMSMGMT_GET_PARAM_2 124
8.14. ACMSMGMT_GET_QTI_2 126
8.15. ACMSMGMT_GET_TSC_2 128
8.16. ACMSMGMT_GET_VERSION_2 130
8.17. ACMSMGMT_LIST_AGENT_2 132
8.18. ACMSMGMT_LIST_COLLECTIONS_2 135
8.19. ACMSMGMT_LIST_CP_2 138
8.20. ACMSMGMT_LIST_ERR_2 141
8.21. ACMSMGMT_LIST_EXC_2 146
8.22. ACMSMGMT_LIST_INTERFACES_1 150
8.23. ACMSMGMT_LIST_LOG_1 152
8.24. ACMSMGMT_LIST_PROC_1 158
8.25. ACMSMGMT_LIST_SERVER_1 162
8.26. ACMSMGMT_LIST_TG_2 166
8.27. ACMSMGMT_LIST_TRAP_1 169
8.28. ACMSMGMT_LIST_USERS_1 172
8.29. ACMSMGMT_REPLACE_SERVER_1 178
8.30. ACMSMGMT_RESET_LOG_1 182
8.31. ACMSMGMT_RESET_ERR_2 184
8.32. ACMSMGMT_SAVE_ERR_FILTER_2 185

8.33. ACMSMGMT_SET_ACC_2 188
8.34. ACMSMGMT_SET_AGENT_2 193
8.35. ACMSMGMT_SET_COLLECTION_2 198
8.36. ACMSMGMT_SET_CP_2 201
8.37. ACMSMGMT_SET_EXC_2 204
8.38. ACMSMGMT_SET_INTERFACE_1 207
8.39. ACMSMGMT_SET_PARAM_2 210
8.40. ACMSMGMT_SET_QTI_2 213
8.41. ACMSMGMT_SET_SERVER_1 219
8.42. ACMSMGMT_SET_TRAP_1 223
8.43. ACMSMGMT_SET_TSC_2 226
8.44. ACMSMGMT_START_ACC_1 231
8.45. ACMSMGMT_START_EXC_1 235
8.46. ACMSMGMT_START_QTI_1 239
8.47. ACMSMGMT_START_TRACE_MONITOR_1 242
8.48. ACMSMGMT_START_TSC_1 244
8.49. ACMSMGMT_STOP_1 248
8.50. ACMSMGMT_STOP_ACC_1 250
8.51. ACMSMGMT_STOP_EXC_1 253
8.52. ACMSMGMT_STOP_QTI_1 257
8.53. ACMSMGMT_STOP_TRACE_MONITOR_1 261
8.54. ACMSMGMT_STOP_TSC_1 263

Chapter 9. Remote Manager Reference Tables 267

9.1. Data Types 267
9.2. ACC Table 268
9.2.1. Field Descriptions 273
9.3. Agent Table 279
9.3.1. Field Descriptions 283
9.4. Collection Table 288
9.4.1. Field Descriptions 288
9.5. CP Table 290
9.5.1. Field Descriptions 293
9.6. Error Filter Table 298
9.6.1. Field Descriptions 299
9.7. EXC Table 299
9.7.1. Field Descriptions 303
9.8. Interfaces Table 310
9.8.1. Field Descriptions 310
9.9. Manager Status Table 311
9.9.1. Field Descriptions 312
9.10. Parameter Table 312
9.10.1. Field Descriptions 314
9.11. QTI Table 318
9.11.1. Field Descriptions 321
9.12. Server Table 325
9.12.1. Field Descriptions 326
9.13. Task Group Table 327
9.13.1. Field Descriptions 328
9.14. Trap Table 330
9.14.1. Field Descriptions 330
9.14.2. Valid Trap Minimums and Maximums 332
9.14.3. SNMP Trap Format 333

9.15. TSC Table 333
9.15.1. Field Descriptions 336
9.16. Users Table 339
9.16.1. Field Descriptions 340
Chapter 10. ACMSCFG Commands 343
10.1. ACMSCFG Overview 343
10.1.1. Command Format 343
10.1.2. Command Objects and Qualifiers 343
10.2. ACMSCFG ADD COLLECTION 345
10.3. ACMSCFG ADD TRAP 348
10.4. ACMSCFG DELETE COLLECTION 350
10.5. ACMSCFG DELETE TRAP 352
10.6. ACMSCFG SET COLLECTION 353
Chapter 11. ACMSMGR Commands 369
11.1. ACMSMGR Overview 369
11.1.1. Command Format 369
11.1.2. Command Objects and Qualifiers 370
11.2. ACMSMGR ADD COLLECTION 374
11.3. ACMSMGR ADD FILTER 377
11.4. ACMSMGR ADD TRAP 379
11.5. ACMSMGR DELETE COLLECTION 382
11.6. ACMSMGR DELETE FILTER 384
11.7. ACMSMGR DELETE TRAP 385
11.8. ACMSMGR HELP 387
11.9. ACMSMGR LOGIN 388
11.10. ACMSMGR LOGOUT 390
11.11. ACMSMGR REPLACE SERVER 391
11.12. ACMSMGR RESET ERROR 393
11.13. ACMSMGR RESET LOG 394
11.14. ACMSMGR SAVE FILTER 395
11.15. ACMSMGR SET ACC 396
11.16. ACMSMGR SET AGENT 402
11.17. ACMSMGR SET COLLECTION 404
11.18. ACMSMGR SET CP 408
11.19. ACMSMGR SET EXC 410
11.20. ACMSMGR SET INTERFACE 413
11.21. ACMSMGR SET PARAMETER 415
11.22. ACMSMGR SET QTI 418
11.23. ACMSMGR SET SERVER 422
11.24. ACMSMGR SET TRAP 424
11.25. ACMSMGR SET TSC 427
11.26. ACMSMGR SHOW ACC 431
11.27. ACMSMGR SHOW AGENT 436
11.28. ACMSMGR SHOW COLLECTION 440
11.29. ACMSMGR SHOW CP 442
11.30. ACMSMGR SHOW ERROR 445
11.31. ACMSMGR SHOW EXC 448
11.32. ACMSMGR SHOW FILTER 450
11.33. ACMSMGR SHOW GROUP 452
11.34. ACMSMGR SHOW INTERFACE 455
11.35. ACMSMGR SHOW LOG 456

11.36. ACMSMGR SHOW MANAGER 462
11.37. ACMSMGR SHOW PARAMETER 463
11.38. ACMSMGR SHOW PROCESS 466
11.39. ACMSMGR SHOW QTI 469
11.40. ACMSMGR SHOW SERVER 472
11.41. ACMSMGR SHOW TRAP 474
11.42. ACMSMGR SHOW TSC 476
11.43. ACMSMGR SHOW USER 480
11.44. ACMSMGR SHOW VERSION 483
11.45. ACMSMGR START EXC 485
11.46. ACMSMGR START QTI 486
11.47. ACMSMGR START SYSTEM 487
11.48. ACMSMGR START TERMINALS 489
11.49. ACMSMGR START TRACE_MONITOR 490
11.50. ACMSMGR STOP EXC 492
11.51. ACMSMGR STOP MANAGER 493
11.52. ACMSMGR STOP QTI 494
11.53. ACMSMGR STOP SYSTEM 496
11.54. ACMSMGR STOP TERMINALS 497
11.55. ACMSMGR STOP TRACE_MONITOR 498

Chapter 12. ACMSSNAP Commands 501

12.1. ACMSSNAP Overview 501
12.1.1. Command Format 501
12.1.2. Command Objects and Qualifiers 501
12.2. ACMSSNAP CLOSE Command 503
12.3. ACMSSNAP EXIT Command 503
12.4. ACMSSNAP HELP Command 504
12.5. ACMSSNAP NEXT Command 505
12.6. ACMSSNAP OPEN Command 505
12.7. ACMSSNAP PREV Command 508
12.8. ACMSSNAP QUIT Command 509
12.9. ACMSSNAP RESET Command 509
12.10. ACMSSNAP SHOW Command 510
12.11. ACMSSNAP TRACE Command 514

Appendix A. Remote Manager Logical Names 517

A.1. Remote Manager Server 517
A.2. Remote Manager Client (ACMSMGR Utility) 517

Appendix B. RPC Procedures and Corresponding Rights Identifiers 519

Appendix C. RPC Procedures and Corresponding Rights Identifiers 521

C.1. Server Messages 521
C.2. ACMSMGR Messages 530
C.3. ACMSCFG Messages 535
C.4. ACMSSNAP Messages 540

Preface

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. About this manual

This manual explains how to use the Remote Manager to manage VSI ACMS for OpenVMS (ACMS) software systems remotely. The manual describes the features of the Remote Manager, which is based on a client/server architecture, how to use the features, and how to manage the Remote Manager. It also provides reference information for the utilities and commands you use in working with the Remote Manager.

3. Document Structure

This manual contains the following chapters and appendixes. The chapters are grouped into two parts. The first part contains chapters concerning the use of remote management features of ACMS. The second part contains chapters concerning reference information for the remote management of ACMS systems. The appendixes follow Part II.

Part I, "Introduction"	
Chapter 1, "Overview of Remote Management"	Introduces the architecture, implementation, and capabilities of ACMS remote management.
Chapter 2, "Getting Started with the ACMS Remote Manager"	Describes how to get started using the Remote Manager including preparation and startup of the server and client nodes; setting up TCP/IP; setting up SNMP; and troubleshooting the Remote Manager.
Chapter 4, "Managing the Remote Manager"	Describes how to manage the ACMS Remote Manager including configuring startup; starting, stopping, and logging in to the Remote Manager; starting and stopping interfaces; and modifying management parameters and log files.
Chapter 5, "Using the Remote Manager to Manage ACMS"	Describes how to use the Remote Manager to manage ACMS, including managing data collection, displaying collected data, and modifying ACMS systems.
Chapter 6, "Management Programming Using ONC RPC"	Describes how programmers can use the Open Network Computing (ONC) remote procedure call (RPC) interface to the ACMS Remote Manager to develop their own programs for managing ACMS systems.
Chapter 7, "Management Programming Using SNMP"	Describes how programmers can use the Simple Network Management Protocol (SNMP) interface

	to the ACMS Remote Manager to develop their own programs for managing ACMS systems.
Part II, "Reference Information"	
Chapter 8, "Management APIs"	Provides reference information about the ACMS remote management APIs, which are procedures that are intended to be called from ONC RPC clients.
Chapter 9, "Remote Manager Reference Tables"	Provides reference information about data types and tables for the ACMS Remote Manager.
Chapter 10, "ACMSCFG Commands"	Provides reference information about the commands of the ACMSCFG utility for performing operations on the Remote Manager configuration file.
Chapter 11, "ACMSMGR Commands"	Provides reference information about the commands of the ACMSMGR utility for performing operations on running ACMS systems.
Appendixes	
Appendix A, "Remote Manager Logical Names"	Contains information about the logical names used by the Remote Manager server and the Remote Manager client (ACMSMGR utility).
Appendix B, "RPC Procedures and Corresponding Rights Identifiers"	Contains information providing cross-references of remote procedure call (RPC) procedures to rights identifiers.
Chapter 12, "ACMSSNAP Commands"	Contains a listing of information about Simple Network Management Protocol (SNMP) object identifiers (OIDs) for ACMS management information base (MIB) fields.

4. ACMS Help

ACMS and its components provide extensive online help.

- DCL level help

  Enter **HELP ACMS** at the DCL prompt for complete help about the **ACMS** command and qualifiers, and for other elements of ACMS for which independent help systems do not exist. DCL level help also provides brief help messages for elements of ACMS that contain independent help systems (such as the ACMS utilities) and for related products used by ACMS (such as DECforms or Oracle CDD/Repository).

- ACMS utilities help

  Each of the following ACMS utilities has an online help system:

  - ACMS Debugger ACMSGEN Utility
  - ACMS Queue Manager (ACMSQUEMGR)
  - Application Definition Utility (ADU)

- Application Authorization Utility (AAU)
- Device Definition Utility (DDU)
- User Definition Utility (UDU)
- Audit Trail Report Utility (ATR)
- Software Event Log Utility Program (SWLUP)

The two ways to get utility-specific help are:

- Run the utility and type **HELP** at the utility prompt.
- Use the DCL **HELP** command. At the "Topic?" prompt, type **@** followed by the name of the utility. Use the ACMS prefix, even if the utility does not have an ACMS prefix (except for SWLUP). For example:

```
Topic? @ACMSQUEMGR
Topic? @ACMSADU
```

However, do not use the ACMS prefix with SWLUP:

```
Topic? @SWLUP
```

Note

Note that if you run the ACMS Debugger Utility and then type **HELP**, you must specify a file. If you ask for help from the DCL level with **@**, you do not need to specify a file.

- ACMSPARAM.COM and ACMEXCPAR.COM help

  Help for the command procedures that set parameters and quotas is a subset of the DCL level help. You have access to this help from the DCL prompt, or from within the command procedures.

- LSE help

  ACMS provides ACMS-specific help within the LSE templates that assist in the creation of applications, tasks, task groups, and menus. The ACMS- specific LSE help is a subset of the ADU help system. Within the LSE templates, this help is context-sensitive. Type **HELP/IND (PF1–PF2)** at any placeholder for which you want help.

- Error help

  ACMS and each of its utilities provide error message help. Use **HELP ACMS ERRORS** from the DCL prompt for ACMS error message help. Use **HELP ERRORS** from the individual utility prompts for error message help for that utility.

- Terminal user help

  At each menu within an ACMS application, ACMS provides help about terminal user commands, special key mappings, and general information about menus and how to select tasks from menus.

- Forms help

  For complete help for DECforms or TDMS, use the help systems for these products.

5. Related Documents

The following table lists the documents in the VSI ACMS for OpenVMS documentation set.

ACMS Information	Description
VSI ACMS Version 5.0 for OpenVMS Installation Guide [https://docs.vmssoftware.com/vsi-acms-installation-guide/]	Description of installation requirements, the installation procedure, and postinstallation tasks.
VSI ACMS for OpenVMS Getting Started [https://docs.vmssoftware.com/vsi-acms-get-started-guide/]	Overview of ACMS software and documentation. Tutorial for developing a simple ACMS application. Description of the AVERTZ sample application.
VSI ACMS for OpenVMS Concepts and Design Guidelines [https://docs.vmssoftware.com/vsi-acms-concepts-and-design-guide/]	Description of how to design an ACMS application.
VSI ACMS for OpenVMS Writing Applications [https://docs.vmssoftware.com/vsi-acms-writing-apps/]	Description of how to write task, task group, application, and menu definitions using the Application Definition Utility. Description of how to write and migrate ACMS applications on an OpenVMS system.
VSI ACMS for OpenVMS Writing Server Procedures [https://docs.vmssoftware.com/vsi-acms-writing-server-proc/]	Description of how to write programs to use with tasks and how to debug tasks and programs.
VSI ACMS for OpenVMS Systems Interface Programming [https://docs.vmssoftware.com/vsi-acms-sys-interface-prog/]	Description of using Systems Interface (SI) Services to submit tasks to an ACMS system.
VSI ACMS for OpenVMS ADU Reference Manual [https://docs.vmssoftware.com/vsi-acms-adu-ref-manual/]	Reference information about the ADU commands, phrases, and clauses.
VSI ACMS for OpenVMS Quick Reference [https://docs.vmssoftware.com/vsi-acms-quick-ref/]	List of ACMS syntax with brief descriptions.
VSI ACMS for OpenVMS Managing Applications [https://docs.vmssoftware.com/vsi-acms-managing-applications/]	Description of authorizing, running, and managing ACMS applications, and controlling the ACMS system.
VSI ACMS for OpenVMS Remote Systems Management Guide [https://docs.vmssoftware.com/vsi-acms-remote-systems-management-guide/]	Description of the features of the Remote Manager for managing ACMS systems, how to use the features, and how to manage the Remote Manager.
Online help	Online help about ACMS and its utilities.

6. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

7. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: <docinfo@vmssoftware.com>. Users who have VSI OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product.

8. Conventions

The following conventions may be used in this manual:

Ctrl/*x*	A sequence such as **Ctrl/*x*** indicates that you must press and hold the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as **PF1 *x*** indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.
Return	In the HTML version of this document, this convention appears as brackets rather than a box.
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
⋮	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
`Monospace text`	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example. In the HMTL version of this document, this text style may appear as italics.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.
bold text	Bold text represents the introduction of a new term or the name of an argument, an attribute, or a reason. In the HMTL version of this document, this text style may appear as italics.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that aries in system output

	(Internal error *number*), in command lines (**/PRODUCER=*name***), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE	Uppercase text indicates the name of a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege. In command format descriptions, uppercase text is an optional keyword.
UPPERCASE	In command format descriptions, uppercase text that is underlined is required. You must include it in the statement if the clause is used.
lowercase	In command format descriptions, a lowercase word indicates a required element.
<lowercase>	In command format descriptions, lowercase text in angle brackets indicates a required clause or phrase.
( )	In command format descriptions, parentheses indicate that you must enclose the options in parentheses if you choose more than one.
[| |]	In command format descriptions, vertical bars within square brackets indicate that you can choose any combination of the enclosed options, but you can choose each option only once.
{ | | }	In command format descriptions, vertical bars within braces indicate that you must choose one of the options listed, but you can use each option only once.

Part I. Introduction

Part I, "Introduction" contains information about using the remote management features of ACMS. It contains an overview of the Remote Manager, as well as information on how it is managed and operates. It also contains information on how to manage data collection and how to use the Remote Manager to modify a running ACMS system. Finally, this part shows you how to write programs that perform remote management using RPC and SNMP.

Chapter 1. Overview of Remote Management

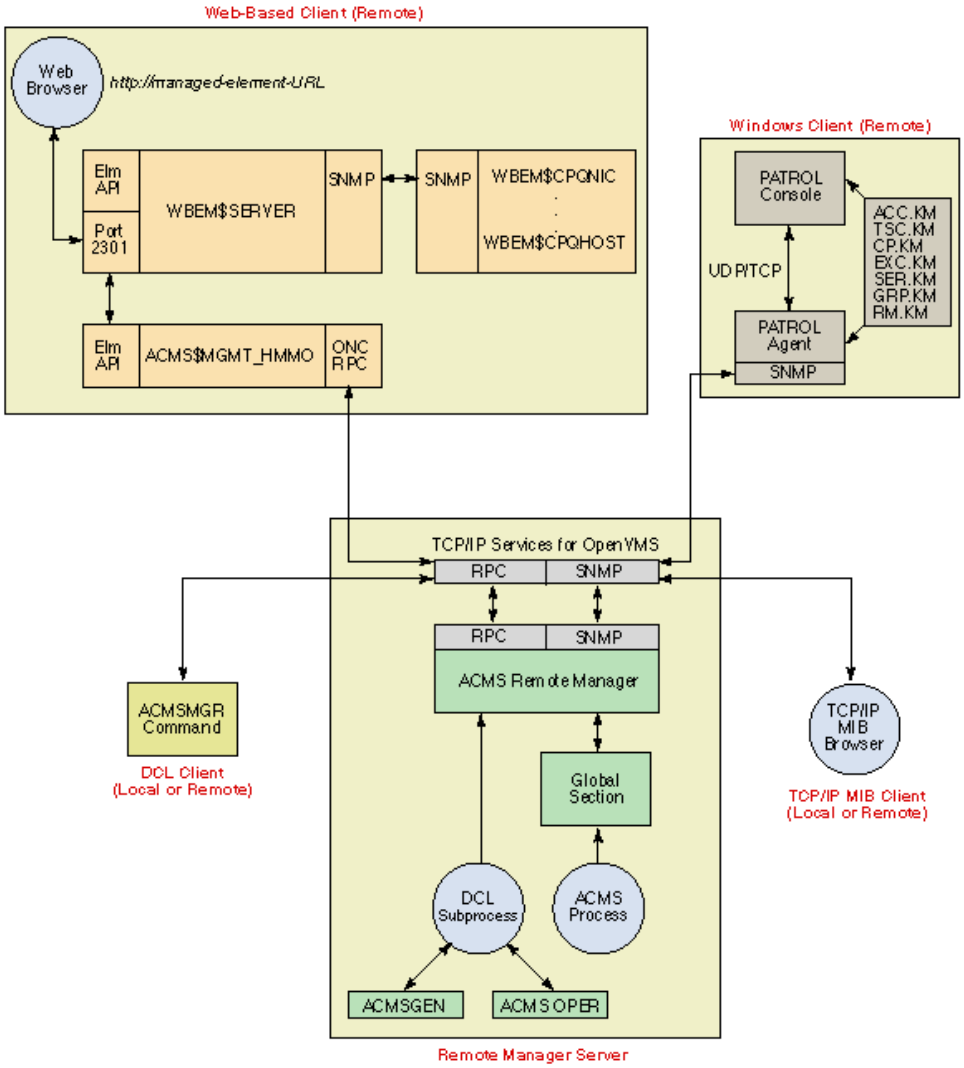
This chapter provides an overview of ACMS remote management.

1.1. Architecture and Implementation

The ACMS Remote Manager provides ACMS system managers with the capability of monitoring and managing their ACMS application environment across a network. The facilities that comprise the Remote Manager are based on a client/server architecture. Two protocols are supported for accessing the ACMS remote management server: Open Network Computing Remote Procedure Call (ONC RPC), which is used by command line utilities (provided with the remote management option) and can be called directly from user-written programs; and Simple Network Management Protocol (SNMP), for use with third-party management consoles.

Figure 1.1, "ACMS Remote Manager Architecture" shows the architecture of the ACMS Remote Manager.

Figure 1.1. ACMS Remote Manager Architecture



As *Figure 1.1, "ACMS Remote Manager Architecture"* shows, users communicate with the ACMS Remote Manager using one of the supported interfaces over a TCP/IP network. Communications between the ACMS Remote Manager and the ACMS run-time system are transparent. Users may be on the same or a different node than the Remote Manager, but the Remote Manager must be running on the same node as the ACMS run-time system it is monitoring or accessing.

The **command line utility** provides command line access to management information as well as control of the Remote Manager process. This utility uses the RPC interface and can be run from any OpenVMS node that has TCP/IP network connectivity to the ACMS node.

The **SNMP interface** provides network access to ACMS management information using the industry standard SNMP protocol. This protocol is supported by most leading system management packages.

The **RPC interface** provides local or remote access to ACMS management information and is used by user-written programs to access ACMS management information.

ACMS system managers control the data being collected as well as automatic variable monitoring and the management interfaces themselves through either SNMP or RPC commands.

The Remote Manager obtains initial configuration information during process startup from a user-maintained configuration file (described in *Chapter 4, "Managing the Remote Manager"*). Once started, the Remote Manager provides ACMS system managers remote access to their ACMS application environment through the interfaces.

1.2. Remote Management Capabilities

The Remote Manager provides ACMS system managers with the ability to:

- Remotely manage the Remote Manager (*Chapter 4, "Managing the Remote Manager"*)
- Remotely manage data collection (*Section 5.1, "Managing Data Collection"*), including:
  - Remotely configure SNMP traps
  - Remotely view ACMS Management Information on line
- Remotely modify ACMS run-time systems (*Section 5.3, "Managing ACMS Using the Remote Manager"*)
- Write programs that remotely access management information on line using ONC RPC (*Chapter 6, "Management Programming Using ONC RPC"*) and SNMP (*Chapter 7, "Management Programming Using SNMP"*)

Chapter 2. Getting Started with the ACMS Remote Manager

This chapter describes how to prepare and run the ACMS Remote Manager software on a node where ACMS has been installed. This chapter does not describe the actual installation. For information about installing ACMS, refer to the *VSI ACMS Version 5.0 for OpenVMS Installation Guide* [https://docs.vmssoftware.com/vsi-acms-installation-guide/].

Note

The procedures in this chapter assume VSI TCP/IP Services Version 5.0 for OpenVMS or higher is installed. The image and process names changed in Version 5.0 from UCX* to TCPIP*. If you are using a machine with an older version of TCP/IP Services installed, you should substitute UCX wherever you see TCPIP in the instructions in this chapter.

Terminology

The following terms are used in this chapter:

- **Server node**

  A node on which ACMS has been installed and on which the ACMS Remote Manager server will run. Server nodes can be either ACMS application or submitter nodes, and can be managed either locally or remotely using one of the supported interfaces (RPC or SNMP). Server nodes are automatically client nodes, but not all client nodes are server nodes.

- **Client node**

  A node on which ACMS may or may not be installed. Client nodes can get information from and perform operations on server nodes. However, users cannot obtain ACMS system management information from or perform system management functions on nodes that are client nodes only.

2.1. Running the ACMS Remote Manager

The following sections outline the steps required to get the ACMS Remote Manager running on an OpenVMS system. If you are an inexperienced user of ACMS, you should first read *Section 2.2, "TCP/IP Setup"* and *Section 2.3, "Remote Manager Setup"* for detailed information about how to set up a node for ACMS remote management.

This section describes setup for both client and server nodes. Server nodes automatically support all client functions; once a node is set up as a server, it can function as a client and a server without additional work. Client nodes can function only as clients.

When you complete the following procedures, the ACMS Remote Manager will be running on your system and you can access it using the ACMSMGR command line utility.

2.1.1. Server Node Setup

Before you begin, you must have already installed the ACMS kit on your system. Also ensure that you have the minimum supported version of TCP/IP installed on your node, and that it is operational. If

TCP/IP is not installed and operational, the ACMS Remote Manager will not run. For information about TCP/IP setup, see *Section 2.2, "TCP/IP Setup"*.

Once you have installed the ACMS and TCP/IP software, perform the following steps to set up a Remote Manager server node:

1. Verify Portmapper (RPC) setup (see *Section 2.1.1.1, "Verify Portmapper (RPC) Setup"*)
2. Run the ACMS postinstallation procedure (see *Section 2.1.1.2, "Run the ACMS Postinstallation Procedure"*)
3. Define process logicals and symbols (see *Section 2.1.1.3, "Define Process Logicals and Symbols"*)
4. Prepare the ACMS environment (see *Section 2.1.1.4, "Prepare the ACMS Environment"*)
5. Start the ACMS Remote Manager (see *Section 2.1.1.5, "Start the ACMS Remote Manager"*)

Server nodes are automatically client nodes. Therefore, you do not need to perform the tasks in *Section 2.1.2, "Client Node Setup"* for nodes that you set up as server nodes.

2.1.1.1. Verify Portmapper (RPC) Setup

Before you attempt to start the Remote Manager, ensure that the proper TCP/IP support is in place. This section provides an overview of the Portmapper (RPC) verification process. If you need more detailed information, or if you will be using third-party tools or writing your own SNMP management tools, see *Section 2.2, "TCP/IP Setup"*.

1. Look for the process TCPIP$PORTM (UCX$PORTM on older versions):

   ```
   $ SHOW SYSTEM/PROCESS=TCPIP*
   ```

   If you find the TCPIP$PORTM process, RPC is running and you can skip to *Section 2.1.1.2, "Run the ACMS Postinstallation Procedure"*. Otherwise, go to step 2.

2. See whether the Portmapper service is enabled:

   ```
   $ TCPIP
   TCPIP> SHOW SERVICE PORTMAPPER

   Service              Port  Proto    Process       Address    State

   PORTMAPPER            111  TCP,UDP  TCPIP$PORTM   0.0.0.0    Enabled
   ```

   The Portmapper should have both the TCP and UDP protocols defined. If it does not, you may need to configure the Portmapper (see *Section 2.2.1, "Set Up the Portmapper (RPC)"*). If the Portmapper state is Enabled, skip to *Section 2.1.1.2, "Run the ACMS Postinstallation Procedure"*. Otherwise, go to step 3.

3. To enable the Portmapper, enter the following commands:

   ```
   $ TCPIP
   TCPIP> ENABLE SERVICE PORTMAPPER
   TCPIP> SET CONFIGURATION ENABLE SERVICE PORTMAPPER
   TCPIP> EXIT
   ```

   Then, restart TCP/IP. The Portmapper process does not automatically start when TCP/IP starts, so you may not see the TCPIP$PORTM process. The process starts the first time the Portmapper is accessed.

2.1.1.2. Run the ACMS Postinstallation Procedure

If you did not run the postinstallation procedure when you installed the ACMS kit, do so now. For details, see *Section 2.3, "Remote Manager Setup"*.

1. Run the postinstallation procedure as follows:

   ```
   $ @SYS$STARTUP:ACMS_POST_INSTALL
   ```

2. When you are asked whether you want to configure the ACMS Remote Manager, answer YES:

   ```
   Do you want to SETUP and CONFIGURE the ACMS Remote System Manager [Y]?
    YES
   ```

3. Answer the questions according to the needs of your organization.

2.1.1.3. Define Process Logicals and Symbols

The ACMS$MGMT_ENV.COM command procedure is provided to define some symbols that make using the ACMSMGR utility simpler. For more information, see *Section 2.3.2, "Define Process Logicals and Symbols"*, or run the procedure now by entering the following command:

```
$ @SYS$STARTUP:ACMS$MGMT_ENV.COM
```

2.1.1.4. Prepare the ACMS Environment

You are now ready to start the Remote Manager. If you need more information about this procedure, see *Section 2.3.3, "Review and Update the Configuration File"*. Then follow these steps:

1. Ensure that the ACMSTART.COM procedure has been run by entering the following command:

   ```
   $ ACMS/SHOW SYSTEM
   ```

   If you get the following error, you must invoke the SYS$STARTUP:ACMSTART.COM procedure described in step 2:

   ```
   %DCL-W-ACTIMAGE, error activating image ACMSHR
   ```

   If you get a message indicating that the ACMS system is stopped, or if some information about the ACMS system is displayed, go to *Section 2.1.1.5, "Start the ACMS Remote Manager"*.

2. Invoke the ACMSTART command procedure:

   ```
   $ @SYS$STARTUP:ACMSTART
   ```

2.1.1.5. Start the ACMS Remote Manager

To start the ACMS Remote Manager, follow these steps:

1. Enter the following command:

   ```
   $ STARTMGR
   ```

2. Check that the ACMS$MGMT_SVR process started by entering the following command:

   ```
   $ SHOW SYSTEM/PROCESS=ACMS$MGMT_SVR
   ```

3. If the process is running, you should be able to communicate with it using ACMSMGR commands (see *Section 2.1.3, "Communicate with the Remote Manager"*).

   If the process is not running, you can look for information in either of two places:

   - Type out the SYS$ERRORLOG:ACMS$MGMT_SERVER.OUT text file:

     ```
     $ TYPE/PAGE SYS$ERRORLOG:ACMS$MGMT_SERVER.OUT
     ```

   - View the Remote Manager log file by using the following command:

     ```
     $ ACMSMGR SHOW LOG/LOCAL
     ```

   For more information about these sources, refer to *Section 2.4.2.1, "ACMS$MGMT_SERVER.OUT Messages"* and *Section 2.4.2.2, "Remote Manager Log Entries"*.

2.1.2. Client Node Setup

All ACMS Remote Manager client nodes require that TCP/IP be installed and operational. For information about TCP/IP setup, refer to *Section 2.2, "TCP/IP Setup"*. Other than TCP/IP connectivity to the server node, no additional TCP/IP setup is required. The Portmapper does not need to be running on the client node.

The following sections describe how to set up an ACMS Remote Manager client node. You can skip these sections if you are installing the ACMS Remote Mangement server; server nodes are automatically client nodes.

If the client node will not be used as an ACMS submitter node, the ACMS Remote Option kit does not need to be installed. How you set up the client node depends upon whether the ACMS Remote Option kit has been installed.

- If the ACMS Remote Option kit has been installed, simply run the ACMS_POST_INSTALL.COM command procedure (see *Section 2.1.2.1, "Run ACMS_POST_INSTALL.COM"*).

- If the ACMS Remote Option kit has not been installed, you must copy some files and define several symbols before you can use the ACMSMGR utility on a client node (see *Section 2.1.2.2, "Copy Files and Define Symbols"*).

Once you have completed these tasks, you can try to communicate with a Remote Manager on a server node using the procedure in *Section 2.1.3, "Communicate with the Remote Manager"*.

Note that you cannot obtain ACMS system management information or perform system management functions on nodes that are client nodes only. Client nodes can get information from and perform operations on server nodes only.

2.1.2.1. Run ACMS_POST_INSTALL.COM

Follow these steps to run the ACMS_POST_INSTALL.COM command procedure:

1. Run the postinstallation procedure as follows:

   ```
   $ @SYS$STARTUP:ACMS_POST_INSTALL
   ```

2. When you are asked whether you want to configure the ACMS Remote System Manager, answer YES:

```
Do you want to SETUP and CONFIGURE the ACMS Remote System Manager [Y]?
 YES
```

3. Answer the questions according to the needs of your organization.

4. Now, execute the ACMS$MGMT_ENV.COM command procedure to define some symbols that make using the ACMSMGR utility simpler:

```
$ @SYS$STARTUP:ACMS$MGMT_ENV.COM
```

2.1.2.2. Copy Files and Define Symbols

If you did not install the ACMS Remote Option kit (that is, if this node will not be an ACMS submitter node), follow this procedure. You will need access to a node with one of the ACMS Run-Time kits installed.

1. Copy the ACMSMGR executable to your node from SYS$SYSTEM on the node that has ACMS installed. Which executable to copy depends on the version of VSI TCP/IP Services for OpenVMS (TCP/IP) you have installed:

   - If you are running Version 4.2 of VSI TCP/IP, copy the ACMS$MGMT_CMD_UCX.EXE file to SYS$SYSTEM on your node.

   - If you are running TCP/IP Version 5.0 or higher, copy the ACMS$MGMT_CMD_TCPIP.EXE file to SYS$SYSTEM on your node.

2. Copy ACMS$MGMT_ENV.COM to your node and run it. This file is located in SYS$STARTUP of a node where ACMS is installed. ACMS$MGMT_ENV.COM defines some symbols that make using the ACMSMGR utility simpler. Execute the command procedure as follows:

```
$ @SYS$STARTUP:ACMS$MGMT_ENV.COM
```

2.1.3. Communicate with the Remote Manager

Before you issue any ACMSMGR commands, you must either log in to the Remote Manager (see step 1) or use an ACMS proxy (see step 2). For detailed information, see *Section 2.3.5, "Communicate with the Remote Manager"*.

1. To log in to the Remote Manager, you must have a valid user account and password on the node on which the Remote Manager is running. The following example commands log in to the Remote Manager on node SERVER, using account MYACCT and password MYPASS. For more details, see *Section 2.3.5.1, "Using ACMSMGR and Logging In Explicitly"*.

```
$ DEFINE ACMS$MGMT_SERVER_NODE SERVER
$ DEFINE ACMS$MGMT_USER MYACCT
$ ACMSMGR LOGIN

ACMS Remote Management -- Command line utility
Password:MYPASS
```

   If the login succeeds, no messages are displayed. Go to step 3.

   If the login fails, check the following possible reasons:

   - You typed in an invalid user name or password.

- You defined the ACMS$MGMT_SERVER_NODE logical incorrectly (wrong or misspelled node name).
- You defined the ACMS$MGMT_USER logical incorrectly (wrong or misspelled node name).
- The Remote Manager is not running on the node you specified.

Refer to *Section 2.4, "Troubleshooting the ACMS Remote Manager Startup"* for more help.

2. If you will be using ACMS proxies to access the Remote Manager, and you already know that you have a valid proxy account, go to step 3. If you have not set up proxies but would like to use them, create a proxy file on the node on which the Remote Manager will run. For more information, see *Section 2.3.5.2, "Using ACMSMGR and a Proxy Account"*.

```
$ SET DEFAULT SYS$SYSTEM $ MCR ACMSUDU UDU> CREATE/PROXY
```

Now,you can add a proxy. To add a proxy, you need to know the following information:

- The nodes and accounts from which you will access the Remote Manager
- The account on the Remote Manager node you will use

For example, assume you will be on node CLIENT using account MYACCT, and you will be accessing node SERVER using account SRVACCT. Enter the following command on node SERVER:

```
UDU> ADD/PROXY CLIENT::MYACCT SRVACCT
```

3. You can now enter any of the ACMSMGR commands. For example:

```
$ ACMSMGR SHOW INTERFACES
```

This command results in output similar to the following:

```
ACMS Remote Management -- Command line utility

ACMS V4.4-0  Interfaces Display                 Time: 18-APR-2001
 13:59:15.51

                 Enabled Running Get      Set      Alarms Time Last
Node   Interface State   State   Requests Requests Sent   Alarm Sent
------ --------- ------- ------- -------- -------- ------
 ------------------------
SERVER rpc       enabled started 987      0        0      17-NOV-1858
 00:00:00.00
SERVER snmp      enabled started 0        0        0      17-NOV-1858
 00:00:00.00
```

If you get error messages instead, refer to *Section 2.4, "Troubleshooting the ACMS Remote Manager Startup"*.

2.2. TCP/IP Setup

There are two components to the TCP/IP setup for the ACMS Remote Manager:

- Portmapper (RPC) setup (see *Section 2.2.1, "Set Up the Portmapper (RPC)"*)

Portmapper setup is required if you will be using the DCL command line utility ACMSMGR for remote management, or if you intend to write your own programs using the RPC API.

- SNMP setup (see *Section 2.2.2, "Set Up SNMP "*)

  SNMP setup is required if you will be using third-party tools (such as PATROL from BMC) for remote system management, or if you will be writing your own SNMP management tools.

The information in the following sections applies only to nodes on which the ACMS Remote Manager will run. It is not relevant for ACMS Remote Manager client nodes.

2.2.1. Set Up the Portmapper (RPC)

Perform this task if the Portmapper has not previously been set up on the node you are using, or if it has been set up incorrectly.

The procedure described here may require a restart of TCP/IP on the node you are using.

Note

When you configure RPC, you are providing network access to the node. This may have significant security implications. Be sure you understand these implications before you configure SNMP. If you are in doubt, consult your network or security administrator.

2.2.1.1. Determine the Current Portmapper Configuration

To determine whether the Portmapper is configured, use the following commands:

```
$ TCPIP TCPIP> SHOW SERVICE PORTMAPPER
```

If the Portmapper is configured, you will see a display similar to the following:

```
Service              Port  Proto    Process       Address   State

PORTMAPPER            111  TCP,UDP  TCPIP$PORTM   0.0.0.0   Enabled
```

If you get an error message indicating that the record is not found, or if both protocols are shown but the state is not Enabled, go to *Section 2.2.1.3, "Configure the Portmapper"*.

If the service is displayed, make sure that both TCP and UDP are shown in the "Proto" column and that the state is Enabled. If both protocols are not shown or if you suspect that the Portmapper is not working correctly, go to *Section 2.2.1.2, "Remove the Existing Portmapper Configuration"*.

If both protocols are shown and the state is Enabled, then the Portmapper is configured on this node and no additional work must be performed.

2.2.1.2. Remove the Existing Portmapper Configuration

Perform this task if you suspect the Portmapper is not working correctly, or if you were directed here from *Section 2.2.1.1, "Determine the Current Portmapper Configuration"*.

Enter the following commands:

```
$ TCPIP TCPIP> SET NOSERVICE PORTMAPPER
```

Enter **Y** at the `Remove? [N]:` prompt, and then exit the utility.

Now,shut down and restart TCP/IP on this node:

```
$ @SYS$STARTUP:TCPIP$SHUTDOWN
$ @SYS$STARTUP:TCPIP$STARTUP
```

Note

If you logged in to this node using TCP/IP, you will lose connectivity after the first command executes. You may have to reboot the machine in order to log in and complete the procedure. To avoid this problem, put the shutdown and startup commands into a command procedure, and submit the procedure to a batch queue that is guaranteed to run on this node.

2.2.1.3. Configure the Portmapper

To configure the Portmapper, run the SYS$MANAGER:TCPIP$CONFIG command procedure. Select option 3 (Server components) and then option 8 (PORTMAPPER). Select the option to "`Enable service on this node.`" For example:

```
$ @SYS$MANAGER:TCPIP$CONFIG

  VSI TCP/IP Services for OpenVMS Configuration Menu

  Configuration options:

           1  -  Core environment
           2  -  Client components
           3  -  Server components
           4  -  Optional components
           5  -  Shutdown VSI TCP/IP Services for OpenVMS
           6  -  Startup VSI TCP/IP Services for OpenVMS
           7  -  Run tests

           A  -  Configure options 1 - 4
          [E] -  Exit configuration procedure

  Enter configuration option: 3


  VSI TCP/IP Services for OpenVMS SERVER Components Configuration Menu

  Configuration options:
           1  -  BIND           Disabled
           2  -  BOOTP          Disabled
           3  -  TFTP           Disabled
           4  -  FTP            Enabled
           5  -  LPR/LPD        Disabled
           6  -  NFS            Disabled
           7  -  PC-NFS         Disabled
           8  -  PORTMAPPER     Enabled
           9  -  TELNET/RLOGIN  Enabled
          10  -  SNMP           Enabled
          11  -  NTP            Disabled
          12  -  METRIC         Disabled
```

```
          13  -  POP            Disabled
          14  -  FINGER         Disabled
          15  -  RMT            Disabled
          16  -  LBROKER        Disabled
          17  -  DHCP           Disabled
           A  -  Configure options 1 - 17
          [E] -  Exit menu

  Enter configuration option: 8


  PORTMAPPER SERVER configuration options:

         1 - Enable service on all nodes
         2 - Enable service on this node

         E - Exit PORTMAPPER configuration

  Enter configuration option: 2
```

To exit from the command procedure, enter **E** twice.

Now, shut down and restart TCP/IP on this node:

```
$ @SYS$STARTUP:TCPIP$SHUTDOWN
$ @SYS$STARTUP:TCPIP$STARTUP
```

Note

If you logged in to this node using TCP/IP, you will lose connectivity after the first command executes. You may have to reboot the machine in order to log in and complete the procedure. To avoid this problem, put the shutdown and startup commands into a command procedure, and submit the procedure to a batch queue that is guaranteed to run on this node.

After TCP/IP starts up, the Portmapper should be ready to use. The Portmapper process itself does not start until it is needed, but you should make sure it is defined as described in *Section 2.2.1.1, "Determine the Current Portmapper Configuration"*.

You can test RPC access to the Remote Manager by using ACMSMGR commands. But you will need to get the ACMSMGR running first (see *Section 2.3, "Remote Manager Setup"*).

2.2.2. Set Up SNMP

Perform this task if SNMP is not set up on the node you are using, or if SNMP is set up incorrectly.

This procedure may require that you restart TCP/IP on the node you are using.

Note

When you configure SNMP, you must configure the SNMP communities to which the node will belong. SNMP communities govern SNMP network access to the node, which may have significant security implications. Be sure you understand these implications before you configure SNMP. If you are in doubt, consult your network or security administrator. If the SNMP communities are not configured properly, you may be unable to access the ACMS Remote Manager.

2.2.2.1. Determine the Current SNMP Configuration

To determine whether SNMP is configured, enter the following commands:

```
$ TCPIP TCPIP> SHOW SERVICES
```

If SNMP is configured, you will see a display similar to the following:

```
Service   Port  Proto   Process      Address    State

...
ESNMP      242  UDP     ESNMP        0.0.0.0    Disabled
SNMP       161  UDP     TCPIP$SNMP   0.0.0.0    Enabled
...
```

If you do not see both of these services, proceed to *Section 2.2.2.3, "Configure SNMP"*. If both services are displayed, SNMP is configured on this node. If you suspect that SNMP is not working correctly, you can proceed to *Section 2.2.2.2, "Remove the Existing SNMP Configuration"*. Otherwise, there is no additional work to be performed. Note: It is fine if ESNMP has a state of Disabled.

2.2.2.2. Remove the Existing SNMP Configuration

Perform this step if you suspect SNMP is not working correctly or if you were directed here from *Section 2.2.2.1, "Determine the Current SNMP Configuration"*.

Enter the following commands:

```
$ TCPIP TCPIP> SET NOSERVICE SNMP
```

Enter **Y** at the `Remove? [N]:` prompt, and then enter:

```
TCPIP> SET NOSERVICE ESNMP
```

Enter **Y** again at the `Remove? [N]:` prompt, and then exit the utility.

Now, shut down and restart TCP/IP on this node:

```
$ @SYS$STARTUP:TCPIP$SNMP_SHUTDOWN
$ @SYS$STARTUP:TCPIP$SNMP_STARTUP
```

2.2.2.3. Configure SNMP

To configure SNMP, run the SYS$MANAGER:TCPIP$CONFIG command procedure. Select option 3 (Server components) and then option 10 (SNMP Configuration). Select the option to "`Enable service on this node`", and respond to the prompts as shown in the following example.

Note

Configuring SNMP communities must be coordinated among all nodes that will participate. If you are unsure which SNMP communities to configure, contact your network administrator.

```
$ @SYS$MANAGER:TCPIP$CONFIG

  VSI TCP/IP Services for OpenVMS Configuration Menu

  Configuration options:
```

```
          1  -  Core environment
          2  -  Client components
          3  -  Server components
          4  -  Optional components
          5  -  Shutdown VSI TCP/IP Services for OpenVMS
          6  -  Startup VSI TCP/IP Services for OpenVMS
          7  -  Run tests

          A  -  Configure options 1 - 4
         [E] -  Exit configuration procedure

  Enter configuration option: 3


  VSI TCP/IP Services for OpenVMS SERVER Components Configuration Menu

  Configuration options:
          1  -  BIND           Disabled
          2  -  BOOTP          Disabled
          3  -  TFTP           Disabled
          4  -  FTP            Enabled
          5  -  LPR/LPD        Disabled
          6  -  NFS            Disabled
          7  -  PC-NFS         Disabled
          8  -  PORTMAPPER     Enabled
          9  -  TELNET/RLOGIN  Enabled
         10  -  SNMP           Enabled
         11  -  NTP            Disabled
         12  -  METRIC         Disabled
         13  -  POP            Disabled
         14  -  FINGER         Disabled
         15  -  RMT            Disabled
         16  -  LBROKER        Disabled
         17  -  DHCP           Disabled
          A  -  Configure options 1 - 17
         [E] -  Exit menu

  Enter configuration option: 10


  SNMP SERVER configuration options:

        1 - Enable service on all nodes
        2 - Enable service on this node

        E - Exit PORTMAPPER configuration

  Enter configuration option: 2

  Do you want to provide the public community [Y]: <site dependent>
  Do you want to provide another community [N]: <site dependent>
  Enter contact person(s): <site administrator>
  Enter the location of the system: <site location>
```

To exit from the command procedure, enter **E** twice.

After exiting from the procedure, you may need to modify the public communities you just specified to allow SNMP reads, writes, or traps. The following example shows how to do so. Note that community

names are case sensitive. Also, note the use of double quotes. To allow SNMP writes to occur on the node, you also need to enable the set flag, as follows:

```
$ TCPIP
TCPIP> SET CONFIG SNMP/COMMUNITY="PUBLIC"/TYPE=WRITE
TCPIP> SET CONFIG SNMP/COMMUNITY="PUBLIC"/TYPE=TRAP
TCPIP> SET CONFIG SNMP/FLAGS=SETS
```

Now, exit the TCP/IP utility and restart TCP/IP on this node:

```
$ @SYS$STARTUP:TCPIP$SHUTDOWN
$ @SYS$STARTUP:TCPIP$STARTUP
```

Note

If you logged in to this node using TCP/IP, you will lose connectivity after the first command executes. You may have to reboot the machine in order to log in and complete the procedure. To avoid this problem, put the shutdown and startup commands into a command procedure, and submit the procedure to a batch queue that is guaranteed to run on this node.

After TCP/IP starts, SNMP should be ready to use. The following SNMP processes should be running:

TCPIP$ESNMP
TCPIP$OS_MIBS

2.2.2.4. Test SNMP

TCP/IP includes a DCL command line utility that can be used to issue SNMP commands to SNMP agents on OpenVMS. To use this utility, define the following foreign commands:

```
 $ SNMPGET :== $SYS$SYSTEM:TCPIP$SNMP_REQUEST
<your node name> PUBLIC GET -W 20 $ SNMPSET :== $SYS$SYSTEM:TCPIP
$SNMP_REQUEST
<your node name> PUBLIC SET -W 20
```

Then, after starting the ACMS Remote Manager (see *Section 2.3, "Remote Manager Setup"*), test access to SNMP:

```
$ SNMPGET 1.3.6.1.4.1.36.2.18.48.5.1.10.1
1.3.6.1.4.1.36.2.18.48.5.1.10 = 14

$ SNMPSET 1.3.6.1.4.1.36.2.18.48.5.1.10.1 -I 15
1.3.6.1.4.1.36.2.18.48.5.1.10 = 15
```

In this example, the first command issues an SNMPGET to get the value of the parameter `mgr_audit_level` (the audit level of the main thread). The second command sets the value of the `mgr_audit_level` parameter to 15 (log all messages). Following each command, the current value of the field is returned.

If these commands fail to return the expected results, refer to *Section 2.4, "Troubleshooting the ACMS Remote Manager Startup"*.

2.3. Remote Manager Setup

Setting up the Remote Manager primarily involves preparing the OpenVMS environment to start the Remote Manager. While many of the steps in this procedure can be performed without having previously

configured TCP/IP, it is strongly suggested that you perform TCP/IP setup tasks described in *Section 2.2, "TCP/IP Setup"* before you attempt to start and access the Remote Manager.

Most of what you need to know to set up the ACMS Remote Manager is covered in *Chapter 4, "Managing the Remote Manager"*. Please read that chapter before you set up the ACMS Remote Manager.

2.3.1. Run the Postinstallation Procedure

The postinstallation procedure creates two important command procedures:

- ACMS$MGMT_SETUP.COM
- ACMS$MGMT_ENV.COM

Both of these procedures are required to start and run the ACMS Remote Manager successfully.

In addition, the postinstallation procedure modifies ACMSTART.COM to execute ACMS$MGMT_SETUP.COM to ensure that important logicals are defined whenever the ACMS run-time system is started.

Run the ACMS_POST_INSTALL.COM command procedure as follows:

```
$ @SYS$STARTUP:ACMS_POST_INSTALL
```

Respond appropriately to all prompts until you reach the following prompt:

```
Do you want to SETUP and CONFIGURE the ACMS Remote System Manager [Y]?
```

Be sure to respond **YES** (the default) to this prompt. Several more questions are posed. The procedure continues with the following questions. Your responses are stored in the ACMS$MGMT_SETUP.COM file.

```
Do you want to allow Proxy Authorization [Y]?
```

All clients must be authenticated and authorized to access the ACMS Remote Manager. Proxy access allows ACMS proxies to be used for this purpose. Proxy access is described in detail in *Section 4.4.1.2, "Proxy Accounts"*.

Enter **Y** to enable proxy authentication and authorization when the Remote Manager is started.

```
(ACMS$MGMT_CONFIG) Enter the file specification for the configuration
   file used by the ACMS Remote Manager
Equivalence string [ SYS$SPECIFIC:[SYSEXE]ACMS$MGMT_CONFIG.ACM ]:
```

The configuration file contains the default startup configuration for both ACMS data collections and the Remote Manager. *Section 4.2, "Configuring Remote Manager Startup"* describes how to use the ACMSCFG utility to manage this file. The default location is SYS$SYSROOT:[SYSEXE]ACMS$MGMT_CONFIG.ACM. The information in this file is not node dependent; however, you may choose to configure the nodes in your cluster differently. If you configure all nodes in the cluster the same, you can put this file in the cluster common root. Otherwise, the default value places it in the node-specific root.

Either press **Return** to accept the default, or type the file specification you want to use.

```
(ACMS$MGMT_TEMP) Enter the directory where the temp command procedures
   will be created
```

```
Equivalence string [ SYS$SPECIFIC:[SYSMGR] ]:
```

The Remote Manager uses temporary command procedures (see *Section 5.3.2, "How the Remote Manager Makes Changes"* to update the ACMS run-time system. The default location of the command procedures is SYS$MANAGER. This directory should not be a cluster common directory.

Either press **Return** to accept the default, or type the directory specification you want to use. If the directory does not exist, the command procedure creates it for you.

```
(ACMS$MGMT_LOG) Enter the directory for the ACMS Remote Manager's Log file
Equivalence string [ SYS$SPECIFIC:[ACMS_RM.LOG] ]:
```

The Remote Manager log file (described in *Section 4.7, "Managing Log Files"* contains a variety of messages generated by the Remote Manager at run time. The default location of the audit log is SYS$SYSROOT:[ACMS_RM.LOG]ACMS$MGMT_LOG.LOG. If you choose to place this log in a cluster common directory, be sure that the file name is different for each node.

Either press **Return** to accept the default, or type the file specification you want to use.

```
(ACMS$MGMT_CREDS_DIR) Enter the directory for the ACMS Remote Manager
Credential's Equivalence string [ SYS$SPECIFIC:[ACMS_RM.CREDS] ]:
```

Client credential files (described in *Section 4.4.1.1, "Logging In"* contain encrypted client identity information used for client authorization. The default location for these files is SYS$SYSROOT:[ACMS_RM.CREDS]. Credential files are created with unique names and can be safely placed in a cluster common directory.

Either press **Return** to accept the default, or type the directory specification you want to use. If the directory does not exist, the command procedure creates it for you.

```
 Please enter the UIC for the ACMS$SNMP account, in the form [ggggg,nnnnnn]
  UIC:
```

This account is used to control SNMP access to ACMS system management information and functions. *Section 4.4.1, "Authentication"* and *Section 7.2, "SNMP Security"* describe the uses of this account. In general, if you will be using an SNMP-based management console to access ACMS, you should create this account.

```
Please enter a password of at least 8 characters, using only
the following characters: ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789$_

Password:
```

The password for this account is never used. Enter any combination of the characters shown. However, keep in mind that you will be prompted to verify whatever you type.

After you run the postinstallation procedure, you should rerun SYS$STARTUP:ACMSTART.COM to ensure that the newly created ACMS$MGMT_SETUP.COM is run.

2.3.2. Define Process Logicals and Symbols

Four symbols are defined in the ACMS$MGMT_ENV.COM procedure:

- ACMSMGR: Used to invoke the ACMSMGR utility, which provides remote access to the ACMS Remote Manager. The ACMSMGR utility is described in detail in *Chapter 11, "ACMSMGR Commands"*.

- ACMSCFG: Used to invoke the ACMSCFG utility, which allows the Remote Manager configuration file to be managed. The ACMSCFG utility is described in detail in *Chapter 10, "ACMSCFG Commands"*.
- ACMSSNAP (and ACMSNAP): Used to invoke the ACMSSNAP utility, which enables users to view ACMS Remote Manager data snapshot files. This utility and its use is described in detail in Chapter 12.
- STARTMGR: Used to invoke the Remote Manager startup procedure.
- SNMPGET and SNMPSET: Used to issue SNMP get and set commands to the Remote Manager. Requires knowledge of ACMS MIB OIDs, which are listed in the file MIB_OID.LIS available from the directory ACMS$RM_EXAMPLES.

Before you attempt to run any of these utilities, run the ACMS$MGMT_ENV.COM procedure:

```
$ @SYS$STARTUP:ACMS$MGMT_ENV.COM
```

2.3.3. Review and Update the Configuration File

The ACMS$MGMT_CONFIG system logical points to the configuration file. This logical is defined in the ACMS$MGMT_SETUP.COM procedure, which is executed by the ACMSTART.COM procedure. If this logical is not defined, the ACMSCFG utility will not be able to locate the file. If you have not already run ACMSTART.COM, do so before issuing any ACMSCFG commands.

The ACMS_POST_INSTALL.COM procedure creates a configuration file with default values in SYS$SYSROOT:[SYSEXE]ACMS$MGMT_CONFIG.ACM.

The configuration file contains the startup configuration for ACMS data collections and provides many defaults for the Remote Manager. *Section 4.2, "Configuring Remote Manager Startup"* describes how to use the ACMSCFG utility to manage this file.

In particular, consider the following:

- Interfaces

  By default, both RPC and SNMP interfaces are enabled. The RPC interface must be enabled if you intend to use the ACMSMGR command line utility, or if you will be writing programs that use the RPC API. The SNMP interface is required only if you will use a third-party SNMP management tool to manage ACMS. The following command disables the SNMP interface:

  ```
  $ ACMSCFG SET INTERFACE/INTERFACE=SNMP/STATE=DISABLED
  ```

- Data collections

  By default, only ID and CONFIG class data is collected by all ACMS processes. If you intend to use the Remote Manager to monitor run-time or pool data, you must enable data collection for those classes. The following commands enable run-time and pool data collection for all processes:

  ```
  $ ACMSCFG ADD COLL/ENT=*/CLASS=RUNTIME/COLL_STATE=ENABLED
  $ ACMSCFG ADD COLL/ENT=*/CLASS=POOL/COLL_STATE=ENABLED
  ```

- Traps

  Configuring traps is optional. Traps are used only if your SNMP management console listens for traps. *Section 7.8, "SNMP Traps"* discusses traps in more detail. If you are unsure about whether you need to configure traps, use the defaults.

- Parameters

  The parameters in the configuration file control various aspects of the Remote Manager. In general, use the default values unless you have a particular reason to modify them. Refer to *Section 9.10, "Parameter Table"* includes a table with descriptions of each parameter.

2.3.4. Start the Remote Manager

At this point, you can start the Remote Manager. You can start the Remote Manager before or after you start the ACMS run-time system. Start the Remote Manager by entering the following command:

```
$ STARTMGR
```

If you prefer, you can run the startup procedure directly:

```
$ @SYS$STARTUP:ACMS$MGMT_STARTUP
```

Once this command completes, you should be able to see the Remote Manager process running by issuing the following command:

```
$ SHOW SYSTEM/PROCESS=ACMS$MGMT_SVR
```

If this process is not running, refer to *Section 2.4, "Troubleshooting the ACMS Remote Manager Startup"*.

2.3.5. Communicate with the Remote Manager

If the ACMS$MGMT_SVR process is running and you have enabled both the RPC interface and proxy access, you should be able to communicate with the Remote Manager. The exact commands you will use depends on the interfaces you have enabled and the mode of authentication you want to use. This section shows two examples of communicating with the Remote Manager:

- Using ACMSMGR and logging in explicitly
- Using ACMSMGR and a proxy account

2.3.5.1. Using ACMSMGR and Logging In Explicitly

If you will not be using proxy accounts, or if you have not set them up yet, you can log in directly to the Remote Manager and communicate with it. To reduce typing, define the process logicals ACMS$MGMT_USER to be the user account you will log in with, and ACMS$MGMT_SERVER_NODE to be the node on which the Remote Manager is running:

```
$ DEFINE ACMS$MGMT_USER MYNAME
$ DEFINE ACMS$MGMT_SERVER_NODE NODE_SERVER_RUNS_ON
```

Then you can log in as follows (the ACMSMGR utility will prompt you for your password):

```
$ ACMSMGR LOGIN Password: MYPASSWORD
```

If no error messages are returned, you have successfully logged in to the Remote Manager. You can now issue ACMSMGR commands from this process. Try the following command:

```
$ ACMSMGR SHOW USERS
```

2.3.5.2. Using ACMSMGR and a Proxy Account

If you will be using proxy accounts, you must set them up prior to issuing any ACMSMGR commands. If you have already set them up, you can skip to the example ACMSMGR command.

If you have not set up your proxies, you start by running the ACMSUDU utility. It's best to run this from the SYS$SYSTEM directory, since that is where ACMSUDU expects to find the file in which it stores proxies.

Start ACMSUDU as follows:

```
$ SET DEFAULT SYS$SYSTEM
$ MCR ACMSUDU UDU>
```

If you have never set up an ACMS proxy before, create the proxy file now. Use the following command:

```
UDU> CREATE/PROXY
```

Now,you need to define the proxy accounts. Proxy accounts have three components: the remote node, the remote account, and the local account.

The remote node is the node from which you will be accessing this node. You can either specify a node name or use the asterisk wildcard (*). Be aware that the Remote Manager treats every access as a remote access. This means that even if you access the Remote Manager only from the same node it runs on, you must create a proxy. In that case, the remote and local nodes are the same.

The remote account is the account on the remote node that will be accessing the Remote Manager. This is the user name on the remote node.

The local account is the account on the local node that will be used for authorization. It must be a valid account on the local node.

To add the proxy record, use the following command:

```
 UDU> ADD/PROXY remote-node::remote-account local-account
```

Once the proxy record has been added, you can attempt to access the Remote Manager. Using a proxy does not require a separate login; you just issue the command. Also, do not define the ACMS$MGMT_USER logical. If it is defined, the ACMSMGR utility will look for login information and will not attempt proxy access.

Try this command:

```
 $ AMCMSGR SHOW USERS/NODE=remote-manager-node
```

If no error messages are returned, a list of users logged in to the Remote Manager will be displayed. To reduce typing when issuing more commands, define the process logical ACMS$MGMT_SERVER_NODE to be the name of the node you want to access; this eliminates the need for using the **/NODE** qualifier in ACMSMGR commands.

If an error is returned, refer to *Section 2.4, "Troubleshooting the ACMS Remote Manager Startup"*.

2.4. Troubleshooting the ACMS Remote Manager Startup

The following sections provide troubleshooting information for the following problems:

- Problems starting ACMS (*Section 2.4.1, "Problems Starting ACMS"*)

- Problems starting the ACMS Remote Manager (*Section 2.4.2, "Problems Starting the ACMS Remote Manager"*)
- Problems with ACMSMGR (*Section 2.4.3, "Problems with the ACMSMGR Utility"*)

2.4.1. Problems Starting ACMS

The following message is displayed when the ACMS run-time system is being started and the ACMS Central Controller (ACC) cannot open the Remote Manager configuration file:

```
%ACMSMGMT-I-CFGNOTOPEN, Unable to open management config file, using
 defaults
```

Possible reasons for this message include:

- The logical name ACMS$MGMT_CONFIG is not defined.

  Solution: This logical is typically defined in the file SYS$STARTUP:ACMS$MGMT_SETUP.COM, which is created by the SYS$STARTUP:ACMS_POST_INSTALL.COM command procedure. If the ACMS$MGMT_SETUP.COM file does not exist, run ACMS_POST_INSTALL.COM. If it does exist, edit it and add the definition of ACMS$MGMT_CONFIG. In either case make sure to run ACMS$MGMT_SETUP.COM, and then run the ACMSCFG utility to create a new, default file.

- The logical name ACMS$MGMT_CONFIG does not point to the configuration file, or the file has not been created.

  Solution: Ensure that the logical is defined properly (see the first bullet). If it is, you can create a new file by running the ACMSCFG utility. ACMSCFG will ask whether you want to create a new file. Answer `YES`, and then review the default settings.

- The ACC process does not have read access to the file pointed to by the logical name ACMS $MGMT_CONFIG.

  Solution: Modify the permissions on the file and restart ACMS.

- The ACMS_POST_INSTALL.COM procedure has not been run.

  Solution: Run this procedure to prepare your system to run the ACMS Remote Manager. See the first bullet for more information.

2.4.2. Problems Starting the ACMS Remote Manager

The Remote Manager writes error messages to two locations. If you are experiencing problems with the Remote Manager, check both locations for messages.

- SYS$ERRORLOG:ACMS$MGMT_SERVER.OUT (see *Section 2.4.2.1, "ACMS $MGMT_SERVER.OUT Messages"*)
- Remote Manager log, pointed to by logical ACMS$MGMT_LOG (see *Section 2.4.2.2, "Remote Manager Log Entries"*)

2.4.2.1. ACMS$MGMT_SERVER.OUT Messages

This is an ASCII text file that contains the redirected SYS$OUTPUT from the Remote Manager process. In general, messages appear in this log only if the Remote Manager is unable to write to its log file. The following conditions are exceptions:

- The literal "`log_to_sysout`" is passed to the Remote Manager startup procedure as P1 (for example, **`@SYS$STARTUP:ACMS$MGMT_STARTUP.COM log_to_sysout`**). Except for rare debugging circumstances, the `log_to_sysout` literal should not be passed to the Remote Manager startup procedure as P1.

- The Remote Manager experiences an access violation or other nontrapped fatal error.

Under these circumstances, OpenVMS exception output is written to ACMS$MGMT_SERVER.OUT.

If you experience problems with SNMP, refer to *Section 7.9, "SNMP Debug Tracing"* for information about obtaining SNMP debug output.

LOG: Could not open file ACMS$MGMT_LOG

This message indicates that the Remote Manager could not open the file pointed to by the logical ACMS$MGMT_LOG. Possible reasons for this include:

- The logical is not defined, or is improperly defined.

  Solution: This logical is typically defined in the file SYS$STARTUP:ACMS$MGMT_SETUP.COM, which is created by the SYS$STARTUP:ACMS_POST_INSTALL.COM command procedure. If the ACMS$MGMT_SETUP.COM file does not exist, run ACMS_POST_INSTALL.COM. If it does exist, edit it and add the definition of ACMS$MGMT_LOG. In either case, make sure to run ACMS$MGMT_SETUP.COM, and then start the Remote Manager again.

- The device is full.

  Solution: If there is insufficient space for the log file, either redefine the logical to point to another device or make room on the device.

- The Remote Manager does not have write access to the file.

  Solution: Modify the permissions on the file or directory to which the ACMS$MGMT_LOG logical points.

- The ACMS_POST_INSTALL.COM procedure has not been run.

  Solution: Run this procedure to prepare your system to run the ACMS Remote Manager. See the first bullet for more information.

2.4.2.2. Remote Manager Log Entries

The messages written to the Remote Manager are determined by Remote Manager parameter settings (for example, `mgr_audit_level`, `rpc_audit_level`, and so on). Changing the parameter values results in either more or fewer messages appearing in the Remote Manager log. By default, messages with a severity of warning (w), error (e), or fatal (f) are written to the Remote Manager log. The log is pointed to by logical ACMS$MGMT_LOG.

You can use the ACMSMGR **`SHOW LOG`** command to display messages in the Remote Manager log. If the Remote Manager is not running, use the **`/LOCAL`** qualifier to read the log file directly. You must be logged in to a node with direct access to the log file in order to use the **`/LOCAL`** switch. For instance:

```
$ ACMSMGR SHOW LOG/LOCAL
```

See *Section 4.7, "Managing Log Files"* for detailed information about the log file maintained by the ACMS Remote Manager.

mgr: f : Failure opening config file

The Remote Manager could not open the configuration file. See the discussion in *Section 2.4.2.1, "ACMS $MGMT_SERVER.OUT Messages"*.

mgr: f : No Interfaces were enabled. Process will shutdown

At least one interface must be enabled when the Remote Manager is started. Otherwise, it is impossible to communicate with the Remote Manager. If both interfaces are disabled, the Remote Manager will not start.

Solution: Issue the following command to see the current interface settings in the configuration file:

```
$ ACMSCFG SHOW INTERFACE
```

Enable at least one of the interfaces as follows (substitute SNMP for RPC if you want to enable the SNMP interface instead of the RPC interface):

```
$ ACMSCFG SET INTERFACE/INTERFACE=RPC/STATE=ENABLED
```

Now, restart the Remote Manager.

procmon: e : Failure obtaining current collection states. Bypassingqti

This message can safely be ignored. It is generated when an ACMS entity is not started and the Remote Manager is parsing the collection table.

procmon: f : Failure waiting on mgmt$x_proc_mon_cond_var

This message can safely be ignored. It is generated when the process monitor thread is unexpectedly interrupted, generally during Remote Manager shutdown.

Remote Manager hangs during process startup

Most Remote Manager hangs during process startup are due to problems with the Portmapper. Verify that the Portmapper is functioning properly, and restart the Remote Manager.

rpc: f : Unable to initialize security. Aborting

The Remote Manager was unable to find a rights identifier in the UAF.

Solution: Create the rights identifier.

sec: e : Failure obtaining uaf info for ACMS$SNMP

If the SNMP interface is enabled, the ACMS$SNMP account must exits. Otherwise, it can perform no operations. If the account exists, it must be granted at least one of the following rights identifiers: ACMS$MGMT_READ, ACMS$MGMT_WRITE, ACMS$MGMT_OPER.

Solution: Either disable the SNMP interface (**`$ [ACMSCFG,ACMSMGR] SET INTERFACE/INTERFACE=SNMP/STATE=DISABLED`**), or create the ACMS$SNMP account and grant it one of the rights.

sec: e : MGMTL_ACMSMGMT_READ Rights identifier not found in rights db!

The Remote Manager was unable to find the rights identifier in the UAF.

Solution: Create the rights identifier.

sec: f : ACMS$SNMP user has been granted no rights.

If the SNMP interface is enabled, the ACMS$SNMP account must be granted at least one of the following rights identifiers: ACMS$MGMT_READ, ACMS$MGMT_WRITE, ACMS$MGMT_OPER. Otherwise, the account cannot perform any operations. If it is not granted any rights identifiers, the thread will not start.

Solution: Either disable the SNMP interface (**`$ [ACMSCFG,ACMSMGR] SET INTERFACE/INTERFACE=SNMP/STATE=DISABLED`**), or grant one of the rights to the ACMS$SNMP account.

snmp: e : Terminating....

This is a general error that simply reports that the thread is exiting. Look in the log file for the reason the thread is exiting. If there are no other error messages, look in SYS$ERRORLOG:ACMS$MGMT_SERVER.OUT.

snmp: f : Internal Initialization failed, exiting...

This is a general error that simply reports that the thread is exiting. Look in the log file for the reason the thread is exiting. If there are no other error messages, look in SYS$ERRORLOG:ACMS$MGMT_SERVER.OUT.

snmp: w : An esnmp error has occurred: -1

This message, if followed by termination of the SNMP thread, usually indicates that SNMP has not been set up properly on the node.

Solution: Configure and enable the SNMP interface. Restart TCP/IP and then restart the Remote Manager.

If this message is received, but is not followed by termination of the SNMP thread, the SNMP interface was able to recover from this error and there is no action that must be taken.

snmp: w : An esnmp error has occurred: -5

This is a warning message that refers to a problem communicating with the SNMP master agent. These errors usually are recoverable and the SNMP interface continues to work. In general, you can ignore this message.

However, frequent occurrences of this error may be attributable to a busy system and may indicate a need to modify one or more of the following parameters: `snmp_agent_time_out`, `snmp_are_you_there`, `snmp_sel_time_out`.

2.4.3. Problems with the ACMSMGR Utility

ACMSMGR problems typically fall into two categories:

- ACMSMGMT-W-NOCLNT_ATTACH messages (see *Section 2.4.3.1, "ACMSMGMT-W-NOCLNT_ATTACH Messages"*)
- ACMSMGR hangs (see *Section 2.4.3.2, "ACMSMGR Hangs"*)

2.4.3.1. ACMSMGMT-W-NOCLNT_ATTACH Messages

ACMSMGR can display the following message:

```
%ACMSMGMT-W-NOCLNT_ATTACH, Cannot create client for node NODE\NOCLNT_ATTACH
```

This message usually is followed by these messages:

```
%ACMSMGMT-E-NOCLIENTS, No clients created, cannot continue
%ACMSMGMT-E-FAIL, Operation failed
```

These messages usually are returned when the Remote Manager is not running on the target node. Possible reasons for this include:

- The Remote Manager is not started.

  Solution: Start the Remote Manager as follows:

  ```
  $ @SYS$STARTUP:ACMS$MGMT_STARTUP
  ```

- The Remote Manager is not fully initialized. Complete initialization of the Remote Manager may take several seconds.

  Solution: Wait several seconds and then reissue the command that resulted in this error.

- The node name is incorrect.

  Solution: Double-check the spelling of the node name in the **/NODE** qualifier or in the ACMS$MGMT_SERVER_NODE logical.

2.4.3.2. ACMSMGR Hangs

ACMSMGR hangs are generally the result of a problem with the Portmapper or the Remote Manager. To verify that the Remote Manager has connected to the Portmapper, issue the following commands on the node on which the Remote Manager is running:

```
$ TCPIP
TCPIP> SHOW PORTMAPPER
```

If the Remote Manager has connected, you will see a display similar to the following:

```
Program Number     Version Protocol Port-number Process  Service-name
---------------------- ------- -------- ----------- -------- ------------
000186A0 (     100000)       2 TCP              111 20407E5E PORTMAPPER
000186A0 (     100000)       2 UDP              111 20407E5E PORTMAPPER
20000099 (  536871065)       1 UDP             1023 20408675
20000099 (  536871065)       1 TCP             1023 20408675
```

If the bottom two lines are missing (program number 20000099, version 1), then the Remote Manager is not connected to the Portmapper. Either the Remote Manager is not started or has terminated, or the RPC interface is not enabled.

If no lines are displayed (that is, if a "record not found" message is displayed), the Portmapper is not started. Refer to *Section 2.2, "TCP/IP Setup"* for more information.

Solution: Correct the problem with the Remote Manager or the Portmapper.

Chapter 3. Using the Remote Manager to Manage ACMS

This chapter describes how to prepare and run the ACMS Remote Manager web agent.

3.1. Overview of the Remote Manager Web Agent

With the Remote Manager web agent, system managers can use their web browser to monitor and tune remote ACMS systems. The ACMS for OpenVMS Alpha Development and Run-time kits include the Remote Manager Hyper- Media Management Object (HMMO), which is integrated into the VSI web-based enterprise management (WBEM) environment. Known as the Remote Manager web agent, this object functions as a Remote Manager client through the ONC RPC interface.

Note

ACMS HMMO will work only with Insight Management Agents using the ELM HTTP/HTTPS server. It will not work with versions of Insight Management Agent using the System Management Homepage as the HTTP/HTTPS server.

The Remote Manager web agent environment consists of the following host systems:

- **Web client –** One or more local systems running a web browser that supports Java plug-ins, JavaScript, and Cascading Style Sheets (CSS).
- **Web server –** An OpenVMS Alpha system where the web agent (ACMS$MGMT_HMMO) and WBEM management server (WBEM$SERVER) processes are running. This system serves the ACMS Remote Management web page and handles all communication between the web client and Remote Manager server systems.
- **Remote Manager server –** One or more OpenVMS systems where Remote Manager server processes (ACMS$MGMT_SVR) are running. The ACMS information displayed on the web agent home page is a result of executing ACMSMGR commands on the Remote Manager servers.

As shown in *Figure 1.1, "ACMS Remote Manager Architecture"*, the Remote Manager web agent (ACMS$MGMT_HMMO) relies on the WBEM management server (WBEM$SERVER) to relay data to and from the web browser. The web agent uses its internal web server to connect to the ACMS Remote Management page. All command input is then relayed to Remote Manager server through the HMMO.

3.2. Remote Manager Web Agent Setup

Before you begin, you must have already installed OpenVMS Alpha Version 8.2 on the web server system. Also, ensure that all web client systems are running one of the currently supported web browsers.

Once the OpenVMS Alpha software is installed, perform the following steps to set up the Remote Manager web agent on the web server system:

- Install the Remote Manager web agent software (*Section 3.2.1, "Install the Remote Manager Web Agent Software"*)

- Install the VSI Management Agents for OpenVMS software (*Section 3.2.2, "Install the VSI Management Agents for OpenVMS Software"*)
- Assign additional rights identifiers (*Section 3.2.3, "Assign Additional Rights Identifiers"*)
- Start the web agent (*Section 3.2.4, "Start the Remote Manager Web Agent Process"*)
- Enable access to Remote Manager hosts (*Section 3.2.5, "Enable Access to Remote Manager Hosts"*)

3.2.1. Install the Remote Manager Web Agent Software

The Remote Manager web agent software is bundled with the ACMS for OpenVMS Alpha Run-time and Development kits. To install the web agent software, choose to install the WBEM-related files component of either kit.

This section contains excerpts from an ACMS Development kit installation. Refer to the *VSI ACMS Version 5.0 for OpenVMS Installation Guide* [https://docs.vmssoftware.com/vsi-acms-installation-guide/] for detailed information about the entire ACMS installation procedure.

1. Run the ACMS for OpenVMS Alpha 5.0 installation procedure for either the ACMS Run-time or Development kit, in as described in *VSI ACMS Version 5.0 for OpenVMS Installation Guide* [https://docs.vmssoftware.com/vsi-acms-installation-guide/]. For example:

   ```
   $ @SYS$UPDATE:VMSINSTAL ACMSDEVA_050 MTA0: OPTIONS N,AWD=DISK1
   OpenVMS AXP Software Product Installation Procedure V8.2

   It is 22-JUN-2001 at 11:00.

   Enter a question mark (?) at any time for help.
   ```

2. A series of product-specific questions are displayed that prompt you to choose the appropriate installation options. Answer the following prompts accordingly:

   ```
   * Do you want the full ACMS installation [NO]? N
   * Do you want to install the ACMS component software [YES]? N
   * Do you want to install the WBEM-related files for ACMS [YES]? Y
   * Do you want to update the LSE environment for ACMS [YES]? N
   ```

   The installation procedure then checks for prerequisite software and adequate disk space and lists a summary of the components to be installed, as follows:

   ```
   CHECKING INSTALLATION PREREQUISITES
   -----------------------------------
        (required and optional software checked)
        (product licenses checked)
        (disk space checked)

   ACMS PREVIOUS INSTALLATION
   --------------------------
        (previous installation of ACMS is compatible with current
         installation)

   ACMS WBEM CHECK
   ---------------
   SUMMARY OF THIS ACMS INSTALLATION
   ---------------------------------
   ```

```
The following steps will be taken to complete this installation:
o WBEM environment will be updated for ACMS
The rest of the installation will take approximately 7 minutes.
Note that this time is heavily dependent your system load, hardware
and kit media. The time mentioned was measured on a stand-alone
DEC 3000 (Alpha) system with a disk-resident kit. Your time may
vary.
```

3. When prompted to continue the installation, answer `YES`. The procedure enters the ACMS WBEM Setup phase.

4. The WBEM setup procedure (SYS$STARTUP:ACMS$WBEM_SETUP.COM) is then invoked, which creates or updates the ACMS$WBEM account and creates the necessary directories and web agent files.

   Do one of the following:

   - If the account does not exist, you are prompted to supply a UIC and password for the account, as follows:

     ```
     The ACMS$WBEM account used to execute ACMSMGR WBEM commands is not
      available.
     You must supply a UIC and password for this account so that it can be
      created.
     Please enter the UIC for the ACMS$WBEM account, in the form
      [ggggg,nnnnnn]
     UIC: [320,525]

     Please enter a password of at least 8 characters, using only
     the following characters: ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789$_
     Password:
     Verification:
     ```

     Enter the appropriate UIC and password. The ACMS$WBEM account is then created and assigned the ACMS$MGMT_READ rights identifier.

   - If the account already exists, a list of the rights identifiers currently assigned to the account are displayed, as follows:

     ```
     ****************************************************************
     *                          ACMS WBEM Setup                     *
     ****************************************************************
          Checking for user account ACMS$WBEM
          Identifier for ACMS$MGMT_READ exists in RIGHTSLIST
          Identifier for ACMS$MGMT_WRITE exists in RIGHTSLIST
          Identifier for ACMS$MGMT_OPER exists in RIGHTSLIST
          Identifier for ACMS$MGMT_SYSUPD exists in RIGHTSLIST

         The account ACMS$WBEM exists.

     The identifiers on the account ACMS$WBEM are

     Identifier                      Value         Attributes
     ACMS$MGMT_READ                  %X8001012E
     ACMS$MGMT_WRITE                 %X8001012F
     ACMS$MGMT_OPER                  %X80010130
     ACMS$MGMT_SYSUPD                %X800101DB
     ```

```
   Do you wish to reset the account ACMS$WBEM to the default values
    [N] ? y
```

You can choose to reset the identifiers at this time by answering `YES`.

The setup procedure then completes by creating the following directories and files on the web server system:

```
The ACMS WBEM setup has completed.
The following files were copied:

 To SYS$SYSROOT:[WBEM]
      ACMS$MGMT_HMMO.EXE
      RUN_ACMS_HMMO.COM
      STOP_ACMS_HMMO.COM

 To SYS$SYSROOT:[WBEM.WEB.IM.ACMSHMMO.ENG]
      ACMS.CSS
      ACMSHMMO.JS
      ACMSMENUTREE.JAR
      ACMS_BANNER.HTML
      ACMS_INDEX.HTML
      ACMS_MENU.HTML
      ACMS_OUTPUT.HTML
 To SYS$SYSROOT:[WBEM.WEB.IM.ACMSHMMO.IMAGES]
      ACMSHMMO.GIF
      HPLOGO.GIF
      WEBBUM.GIF
 To SYS$COMMON:[SYSLIB]
      ACMS$TRACE_SHR.EXE
 You may wish to purge these directories.
```

Once the ACMS installation is complete, download and install the VSI Management Agent for OpenVMS software, as described in the next section.

3.2.2. Install the VSI Management Agents for OpenVMS Software

If you have not already installed the VSI Management Agents for OpenVMS software, do so now. You can download this software from the web page listed in *Section 3.1, "Overview of the Remote Manager Web Agent"*.

Follow the associated instructions to copy, unpack, and install the appropriate VSI Management Agents for OpenVMS PCSI kit. Once the software is installed, issue the following command to start the management agent process:

```
$ SET DEFAULT SYS$SPECIFIC:[WBEM]
$ @WBEM$RUN_WEBSERVER.COM
```

Note that the WBEM server (WBEM$SERVER) is the only process started by this procedure. None of the other Management Information Base (MIB) processes included in the WBEM kit (such as WBEM$CPQHOST) are used by the Remote Manager agent. If you plan to use software on this system that relies on the MIB processes, run the WBEM$STARTUP.COM procedure, as described in the WBEM installation material.

3.2.3. Assign Additional Rights Identifiers

The installation procedure automatically grants the ACMS$MGMT_READ rights identifier to the ACMS$WBEM account when it is created. This enables all **SHOW** commands to be executed from the web agent. In order to enable all other non-read operations (such as **SAVE**, **SET**, **START**, **STOP**, **RESET**, **ADD**, and **DELETE**), grant one or more of the following rights identifiers to the ACMS$WBEM account:

- ACMS$MGMT_WRITE
- ACMS$MGMT_OPER
- ACMS$MGMT_SYSUPD

See *Section 4.4.2, "Authorization"* for more information on the use of rights identifiers.

3.2.4. Start the Remote Manager Web Agent Process

Enter the following command to start the Remote Manager web agent process where *queue-name* is a valid OpenVMS batch queue:

```
$ SUBMIT/NOTIFY/LOG=SYS$SYSROOT:[WBEM]/QUEUE=queue-name -
_$ /USER=ACMS$WBEM SYS$SPECIFIC:[WBEM]RUN_ACMS_HMMO.COM
```

If the process is already running, this command restarts the process.

3.2.5. Enable Access to Remote Manager Hosts

In order for the Remote Manager web agent to access a Remote Manager server system, the logical ACMS$MGMT_ALLOW_PROXY_ACCESS on the host system must be set to a value of 1, which enables proxy access.

Also, an ACMS proxy entry for the ACMS$WBEM account is required. For example, the following proxy entry grants the user ACMS$WBEM access to the local host from any known system:

```
$ MCR ACMSUDU
UDU>SHOW /PROXY *::ACMS$WBEM
Remote User: *::ACMS$WBEM                    Local User: ACMS$WBEM
```

The rights identifiers on the local ACMS$WBEM account control the level of access allowed on the host system.

Note

Even if the Remote Manager is running on the same node as the web agent, it is still considered a remote host and the requirements above still apply.

3.2.6. Stop the Remote Manager Web Agent

Use the following command to stop the web agent process:

```
$ @SYS$SPECIFIC:[WBEM]STOP_ACMS_HMMO.COM
```

If you also want to stop the WBEM server process, enter the following command:

```
$ @SYS$SPECIFIC:[WBEM]WBEM$STOP_WEBSERVER.COM
```

3.3. Using the Remote Manager Web Agent

The following sections describe how to access and use the Remote Manager web agent interface.

3.3.1. Accessing the ACMS Remote Management Web Page

From a browser on the web client system, enter the following URL to connect to the web server system where *server-host* represents the address of the OpenVMS Alpha system on which the web agent software is running:

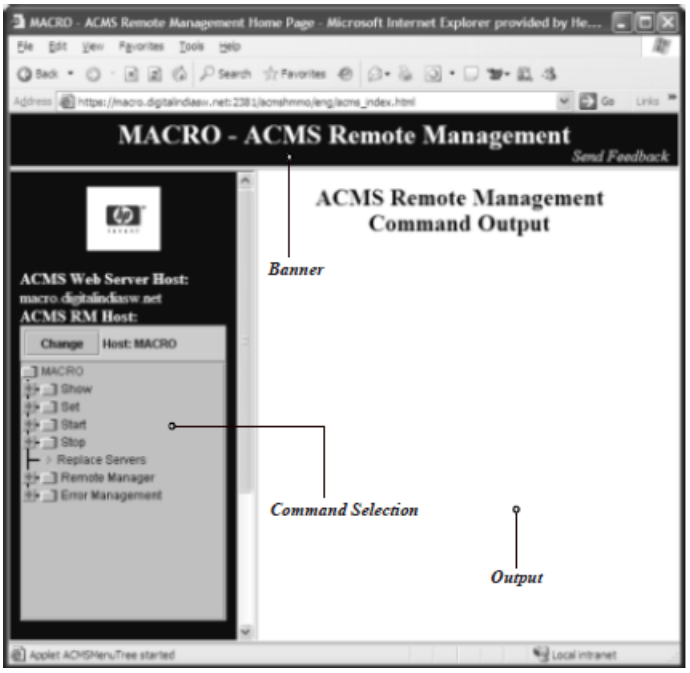
```
http://server-host:2301/acmshmmo/eng/acms_index.html
```

This address can be expressed in any of the following forms:

```
node (node name)
node.company.com (URL)
165.112.94.78 (IP address)
```

The ACMS Remote Management page is displayed, similar to the figure below.

Figure 3.1. Remote Manager Web Agent Page



This page consists of the following frames:

Banner frame

Displays the application name as well as the name of the web server system. This frame also contains a link that you can use to send feedback about the web agent directly to VSI.

Command Selection frame

Displays a tree that contains selections representing the various ACMSMGR commands. The items in this tree are grouped by common command verbs (such as **SHOW** and **SET**) or by object (such as Remote Manager). This frame also contains a series of links to pertinent VSI WBEM and ACMS information, such as to the HTML version of this guide.

Output frame

Displays the results of the selected command. Brief instructions on how to interact with the data in this frame are displayed along with the related output and status messages (if any).

3.3.2. Conventions

The web agent uses color and font highlighting to indicate the different states and types of data displayed in the output frame. The default conventions are described in the table below. Note, however, that you can change these conventions as described in the following section.

Table 3.1. Remote Manager Web Agent Conventions

Text appearing in	Indicates
White with teal or blue background	Active and stored values that can be changed. To set a value, single click on the item. (Set commands)
Teal italics	Dynamic configuration fields. (Show commands)
Gray	Inactive data; old process data that is still available will be displayed. The node name is also prefixed with an asterisk, similar to ACMSMGR displays. (Show commands)
Red	Warning or error messages. (All)
Blue	Disabled collection state. Data displayed for the related class may not be current. (Show commands)

3.3.3. Customizing the Display

The Remote Management web page relies on a cascading style sheet (CSS) to manage its formatting. Based on the CSS level 2 specification (CSS2) from the World Wide Web Consortium (W3C), the ACMS.CSS file functions as a template for information displayed in the output frame.

If you are familiar with CSS files, you can customize the formatting of information in the output frame by editing the file ACMS.CSS located in SYS$SYSROOT:[WBEM.WEB.IM.ACMSHMMO.ENG].

For example, to remove the background image in the output frame, open the CSS file and search for the following statement:

```
BODY {background-color: white; background-image:
url(/acmshmmo/images/webbum.gif); color: black;}
```

Replace this statement with the following:

```
BODY {background-color: white; color: black;}
```

To learn more about CSS files or the CSS2 specification, visit the W3C web site for the latest information and resource listings:

```
http://www.w3.org/Style/CSS
```

Note

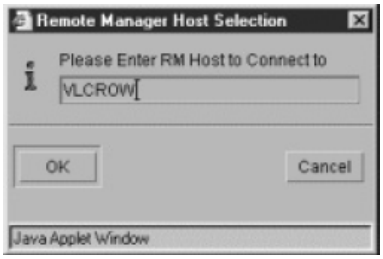
Each browser may interpret style sheet properties differently. Be aware that slight variations in format may occur depending upon the browser that you use.

3.3.4. Selecting the Remote Manager Host

When you first access the ACMS Remote Management web page, the name of the web server is displayed as the Remote Manager host.

To choose a different Remote Manager host, click the Change button in the command selection frame. The Select Host popup window is displayed, similar to the figure below.

Figure 3.2. Select Host



Enter the name of the Remote Manager host, and click **OK**. Note that if you enter a URL or IP address, only the short form of the name is displayed in the command selection frame.

3.4. Issuing Remote Manager Commands

The Remote Manager web agent interface provides detailed usage instructions on each page displayed within the output frame. Therefore, the following sections are only intended to provide a brief overview of issuing the most common Remote Manager commands with the web agent.

The Remote Manager web agent interface provides you with much of the same capability as ACMSMGR in managing ACMS systems. The main functional differences are that with the web agent:

- You cannot view TRAP information.
- There is no equivalent to the **`SHOW LOG/LOCAL`** and **`SHOW ERROR/LOCAL`** commands.
- You can only connect to one Remote Manager host system per window.

You can quickly reissue any web agent command using the **Refresh** (or **Reload**) option of your browser to reload the page in the output frame. To save frequently issued commands, bookmark the page in the output frame.

3.4.1. Using Show Commands

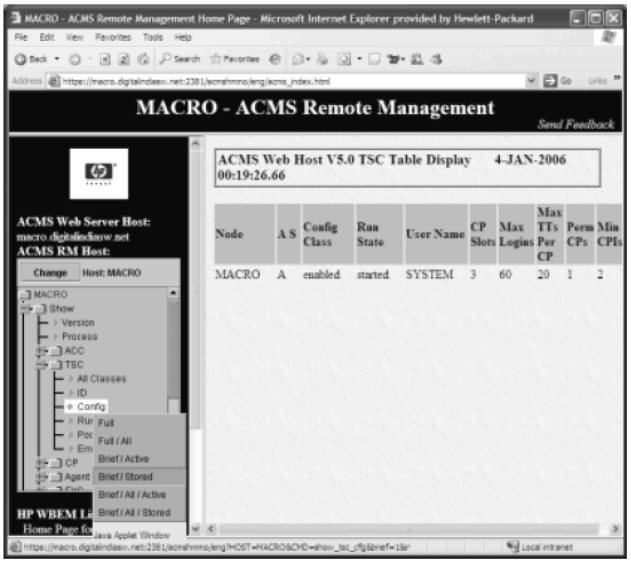
To display information about an ACMS entity or object:

1. In the command selection frame, click **Show** to expand the list of valid entities and Remote Manager objects.

2. Click on the appropriate entity (such as **TSC**) or object (such as **Process**). If you selected an entity, click on the appropriate type of information to display (such as **Config**), and choose the scope of display (such as **Brief/Stored**).

   The results of the command are displayed in the output frame, similar to the figure below. Note that all dynamic data is displayed in italics.

Figure 3.3. Show TSC



3.4.2. Using Set Commands

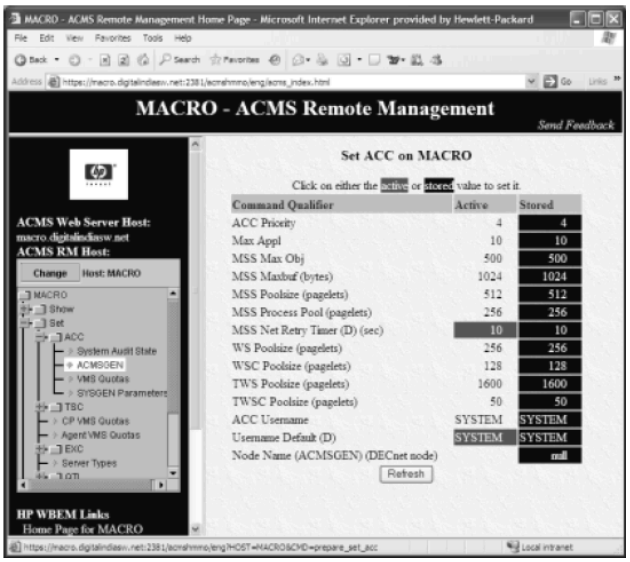
To change information related to an ACMS entity or Remote Manager object:

1. In the command selection frame, click **Set** to expand the list of valid entities and Remote Manager objects.

2. Click on the appropriate entity (such as **ACC**) or object (such as **Remote Manager > Collection**). If you selected an entity, click on the appropriate type of information you want to change (such as **ACMSGEN**).

   The available values are displayed in the output frame, similar to the figure below. Note that any active and stored data that can be changed is displayed reverse highlight.

Figure 3.4. Set ACC



3. Move the cursor over the value you want to change until the link cursor appears, and then click on the value. A popup window is displayed prompting you for a new value.

4. Enter the new value in the popup window, and click **OK**.

Note that you can update the values displayed in the output frame at any time by clicking the **Refresh** button.

3.4.3. Using Start and Stop Commands

To start or stop an ACMS object, such as an application (EXC):

1. In the command selection frame, click **Start** or **Stop** to expand the list of objects.

2. Click on the appropriate object (such as **Remote Manager > Collection**).

   Except for System, the command is executed as soon as it is selected. If you chose **Start** or **Stop System**, additional choices are displayed in the output frame.

3. Click on the appropriate check boxes to set or unset one or more values, and click the **Start SYSTEM** or **Stop SYSTEM** button.

3.4.4. Using Add and Delete Commands

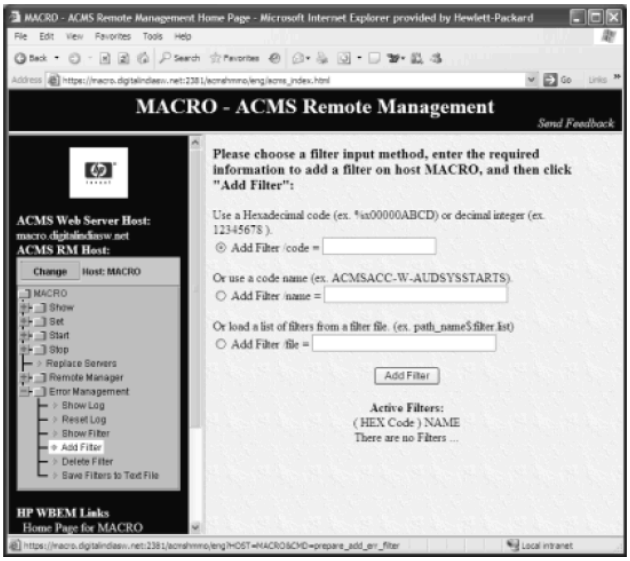
To add or delete a Remote Manager object, such as an error filter:

1. In the command selection frame, click on the appropriate entry (either **Error Management** or **Remote Manager**).

2. Click on the subentries until you reach the item you want to add or delete (such as **Add Filter**).

   A form with related parameter information is displayed, similar to the figure below.

Figure 3.5. Add Error Filter



3. Complete form and click **Add** or **Delete**.

3.5. Troubleshooting the Remote Manager Web Agent

- **WBEM Home Page does not display the ACMS Icon**

  ACMS HMMO is not registered with the WBEM$SERVER process. After starting the WBEM server with the following command:

  ```
  $ @SYS$SPECIFIC:[WBEM]WBEM$RUN_WEBSERVER.COM
  ```

  You may need to delay starting the ACMS HMMO until the WBEM$SERVER process is in the HIB state. If the ACMS HMMO is started too soon it may not register with the WBEM$SERVER. The SYS$SPECIFIC:[WBEM]WBEM$RUN_WEBSERVER.COM must be run prior to running SYS$SPECIFIC:[WBEM]RUN_ACMS_HMMO.COM.

- **WBEM Home Page does not display**

  If you access the WBEM Home Page at `http://host_name:2301/` and the page does not display, it may be that the WBEM$SERVER is not started. Another possibility is that the ACMS$MGMT_HMMO process was started prior to the WBEM$SERVER process. To ensure proper startup, stop both processes and then restart them in the correct order.

- **Remote Manager web page displays, but remote commands fail.**

  This indicates that the Remote Manager web agent cannot connect to the specified Remote Manager server host. If all commands fail, ensure that the Remote Manager server process is running on the host system and that access to it has been properly setup (as described in *Section 3.2.5, "Enable Access to Remote Manager Hosts"*). If some commands work and others fail, the ACMS$WBEM account may not have the required rights identifier; see *Section 3.2.3, "Assign Additional Rights Identifiers"*.

- **Page Refresh or Reload does not update the output frame.**

This behavior is browser dependent. To refresh the information displayed in the output frame, move the cursor inside the frame to specifically refresh or reload the information within it.

3.5.1. Reporting Problems

If the ACMS$MGMT_HMMO process crashes, the following files will contain any error information that was available: SYS$SPECIFIC:[WBEM]ACMS$MGMT_ HMMO.LOG;* SYS$SPECIFIC:[WBEM]ACMS$MGMT_HMMO.ERR;*.

If there are any new dump files, you may want to examine the file to locate the problem source: SYS$SPECIFIC:[WBEM]*.DMP;*.

If the problem is with WBEM$SERVER process, send the dump file to your VSI support representative. If the problem is with the ACMS HMMO process, please have the following files ready for analysis in addition to a procedure that reproduces the situation:

SYS$SPECIFIC:[WBEM]ACMS$MGMT_HMMO.LOG;*

SYS$SPECIFIC:[WBEM]ACMS$MGMT_HMMO.ERR;*

SYS$SPECIFIC:[WBEM]*.html;*

SYS$SPECIFIC:[WBEM]*.txt;*

SYS$SPECIFIC:[WBEM]SYS$OUTPUT.*;

SYS$SPECIFIC:[WBEM]*.DMP;*

Chapter 4. Managing the Remote Manager

This chapter describes how to manage the ACMS Remote Manager.

4.1. Overview

The ACMS Remote Manager runs on the same node as the ACMS run-time system but runs independently of it. The Remote Manager may be started and stopped at any time without affecting the ACMS run-time system. Similarly, the ACMS system can be started and stopped at any time without affecting the Remote Manager process. Remote management can be performed only on nodes where the Remote Manager has been started.

ACMS system managers configure the Remote Manager process (for example, which interfaces are enabled, what alarms to send) using a combination of the ACMSCFG utility (which provides initial configuration settings at process startup) and the ACMSMGR utility (to change settings once the process has started). Management consoles that support SNMP can also be used to configure and manage the Remote Manager.

Before the Remote Manager process can communicate with external entities, either SNMP or RPC must be configured and running on the appropriate nodes. See the *VSI ACMS Version 5.0 for OpenVMS Installation Guide* [https://docs.vmssoftware.com/vsi-acms-installation-guide/] for information about configuring and starting SNMP and RPC.

4.2. Configuring Remote Manager Startup

Before the Remote Manager is started, the **configuration file** should contain the appropriate settings. Both the ACMS run-time system and the Remote Manager read the configuration file during startup. If the ACMS Central Controller (ACC) process cannot read the configuration file when starting up, it uses default values. If the Remote Manager cannot read the configuration file when starting up, it logs an error and exits. By default, the configuration file is stored in SYS$SYSTEM:ACMS$MGMT_CONFIG.ACM. This location can be changed using the system-wide logical ACMS$MGMT_CONFIG. Use the ACMSCFG utility to change values in this file. The ACMSCFG utility allows ACMS system managers to set:

- The interfaces to be started
- Data-collection options
- Remote Manager run-time parameters
- SNMP traps

The configuration file is created during postinstallation with a set of default values. ACMS system managers should review these settings prior to starting the Remote Manager to determine whether the settings are appropriate for the node on which the process will run. Use the ACMSCFG **SHOW** commands as follows to display the settings:

```
$  ACMSCFG SHOW INTERFACE
$  ACMSCFG SHOW COLLECTION
```

```
$  ACMSCFG SHOW PARAMETER
$  ACMSCFG SHOW TRAP
```

Note

Changes made to the ACMSCFG file are not automatically reflected in the running system. The ACMSCFG file is read during Remote Manager and ACMS system startup only. The Remote Manager process must be restarted in order for configuration file changes to the Parameter, Interface, and Trap tables to become active. The ACMS run-time system must be restarted in order for configuration file changes to the Collection table to become active. After the Remote Manager process has been started, you can use the ACMSMGR utility to make dynamic changes to the active system.

4.2.1. How to Run the ACMSCFG Utility

The ACMSCFG utility is a DCL command line tool that is invoked using a foreign command. The ACMSCFG utility accepts a number of command line arguments that determine what operations it should perform. The basic syntax for running the ACMSCFG utility is as follows:

```
ACMSCFG verb object qualifier
```

For example, to display the current data collection settings, you would use the following command:

```
$ ACMSCFG SHOW COLLECTION
```

You can get help on the available ACMSCFG commands and their syntax using the following command:

```
$ ACMSCFG HELP
```

You can define your own foreign command by using the following DCL command:

```
$ MYCOMMAND :== $SYS$SYSTEM:ACMS$MGMT_CONFIG_CMD
```

If you do this, you would substitute **MYCOMMAND** for ACMSCFG in the preceding examples.

When the ACMSCFG utility is started, it attempts to locate the ACMS$MGMT_CONFIG.ACM file by translating the logical name ACMS$MGMT_CONFIG. If that attempt fails, it looks in the default location, SYS$SYSTEM:ACMS$MGMT_CONFIG. If that lookup fails, ACMSCFG asks the user whether to create a new file. New files are created with default values in the directory that the logical name ACMS$MGMT_CONFIG translates to. If the logical name is not defined or does not include a directory specification, the default directory location is the current directory.

4.2.2. Displaying Current Values

Current ACMSCFG values can be displayed using the **SHOW** command, as follows:

```
ACMSCFG SHOW object
```

Valid **SHOW** objects are:

- Collection
- Control
- Interface

- Parameter
- Trap

The values for each object type correspond directly to fields in management configuration tables. These tables are discussed in *Chapter 9, "Remote Manager Reference Tables"*.

The following is an example **SHOW** command and its output:

```
SPARKS> ACMSCFG SHOW COLLECTION

Entity                Collect Collect                      Storage  Storage
Type    Entity Name   Class   State     Storage Location   State
 Interval
------- ------------- ------- --------- ------------------ --------
 ---------
 *      *             id       enabled  acms$mgmt_snapshot enabled  3600
 *      *             config   enabled  acms$mgmt_snapshot disabled 3600
 *      *             error    enabled  acms$mgmt_snapshot disabled 300
```

4.2.3. Changing Values

ACMSCFG values can be changed using one of three verbs:

- The **ADD** verb is used to add rows for the following objects:
  - Collection
  - Trap

  Example:

  ```
  $ ACMSCFG ADD COLLECTION/ENTITY=*/NAME=*/CLASS=RUNTIME
  ```

- The **DELETE** verb is used to delete rows for the following objects:
  - Collection
  - Trap

  Example:

  ```
  $ ACMSCFG DELETE COLLECTION/ENTITY=*/NAME=*/CLASS=RUNTIME
  ```

- The **SET** verb is used to add rows for the following objects:
  - Collection
  - Interface
  - Parameter
  - Trap

  Example:

  ```
  $ ACMSCFG SET COLLECTION/ENTITY=*/NAME=*/CLASS=RUNTIME/
  COLL_STATE=ENABLED
  ```

Each object has unique qualifiers that determine which values are to change. Qualifiers are either **mandatory** or **optional**. **Mandatory** qualifiers have no default and must be specified by the user. **Optional** qualifiers have default values and do not have to be specified. See *Chapter 10, "ACMSCFG Commands"* for a complete description of the syntax for each command and the qualifiers they support.

4.3. Starting and Stopping the Remote Manager

The following information discusses starting and stopping the ACMS Remote Manager.

4.3.1. Remote Manager Startup

The Remote Manager is started as a detached process using the command procedure SYS$STARTUP:ACMS$MGMT_STARTUP as follows:

```
$ @SYS$STARTUP:ACMS$MGMT_STARTUP
```

You should run this file from the SYSTEM account during system startup. You can run the file either before or after the ACMS run-time system has been started. Alternatively, you can run it at any time from a privileged account.

During process startup, the Remote Manager reads the ACMSCFG file (located in SYS$SYSTEM:ACMS$MGMT_CONFIG.ACM or wherever the ACMS$MGMT_CONFIG logical points). If the file cannot be found and opened, the Remote Manager will not start.

The Remote Manager writes errors to the ACMS$MGMT_LOG file. This is a binary file that can be displayed using the ACMSMGR utility, as follows:

```
$ ACMSMGR SHOW LOG
```

The ACMSMGR utility generally performs operations on remote nodes. If the Remote Manager fails to start, it will not be accessible remotely. You will need to log in to the node on which it failed to start, and issue the following command:

```
$ ACMSMGR SHOW LOG/LOCAL
```

This command instructs the ACMSMGR utility to read the log file directly, bypassing the Remote Manager. See *Chapter 11, "ACMSMGR Commands"* for a complete description of the ACMSMGR utility, commands, and command syntax.

In addition to writing messages to the ACMS$MGMT_LOG file, the Remote Manager writes messages to SYS$OUTPUT if it cannot access the log file. You can have all messages written to SYS$OUTPUT by invoking the startup procedure with the LOG_TO_SYSOUT parameter, as follows:

```
$ @SYS$STARTUP:ACMS$MGMT_STARTUP LOG_TO_SYSOUT
```

The ACMS$MGMT_STARTUP procedure redirects SYS$OUTPUT for the Remote Manager to a file called ACMS$MGMT_SERVER.OUT in the SYS$ERRORLOG directory.

4.3.2. Remote Manager Shutdown

The Remote Manager is stopped using the ACMSMGR **STOP MANAGER** command, which has the following syntax:

```
ACMSMGR STOP MANAGER /NODE=node-name
```

The **/NODE** qualifier can be omitted if the ACMS$MGMT_SERVER_NODE logical is defined. If the **/NODE** qualifier is provided, it overrides the ACMS$MGMT_SERVER_NODE logical.

The Remote Manager can be stopped independently of the ACMS run-time system. Stopping the Remote Manager has no effect on the running ACMS system. Note, however, that simply stopping the Remote Manager does not stop any active data collections. Data collections can be stopped only by using ACMSMGR commands, or from an SNMP management console that has access to the Remote Manager.

Note also that prior to issuing this command, the user must either have logged in to the Remote Manager, or the user must have a valid proxy (and proxy access must have been enabled). Regardless of how access is gained, the user must hold the ACMS$MGMT_OPER rights identifier on the node the Remote Manager is running in order to stop it. See *Section 4.4, "Logging In to the Remote Manager"* for a description of how to log in to the Remote Manager.

The ACMSMGR **STOP MANAGER** command executes asynchronously of the actual shutdown. That is, the command will complete (control will return to the user) before the shutdown has completed.

If the Remote Manager fails to shut down, it can be stopped by using the DCL command **STOP/ID**, which has the following syntax:

```
STOP/ID=pid
```

Determine the PID of the Remote Manager using the DCL command **SHOW SYSTEM**, and then look for the process named ACMS$MGMT_SVR.

4.4. Logging In to the Remote Manager

The Remote Manager requires that each client is authenticated and that each access attempt is authorized.

4.4.1. Authentication

Authentication can be performed in one of two ways: either through an explicit login (using a valid OpenVMS user name and password) or through a valid ACMS proxy account.

The exception to this rule is SNMP access, which is controlled by the presence of the ACMS$SNMP account in the local rights database. Authentication for external entities that communicate with the Remote Manager through the SNMP protocol is allowed only when a valid OpenVMS account exists for the user ACMS$SNMP. If this account exists and is not disusered, the user is considered to be an authentic user. Authorization for SNMP users is treated the same as for any other user – by OpenVMS rights identifier. See *Section 4.4.2, "Authorization"* for more information about authorization.

All access for an interface can be disabled by disabling the interface itself, either through the ACMSCFG utility prior to management startup, or through the ACMSMGR utility after Remote Manager startup.

The total number of users that can be simultaneously logged in to the Remote Manager (regardless of authentication mechanism) is controlled by the Remote Manager parameter MAX_LOGINS, which can be modified by the Remote Manager. This parameter is not the same as the MAX_LOGINS ACMS system parameter in ACMSGEN. When the number of users currently logged in is equal to the value of this parameter, new logins are rejected until some users have logged out, or until their credentials have expired. You can set the initial value of MAX_LOGINS with the ACMSCFG utility. You can change the value of MAX_LOGINS dynamically (but nondurably) with the ACMSMGR utility.

Attempts to log in to the Remote Manager are recorded in the Remote Manager log file if the `security_audit_level` parameter is set for informational level logging (any odd value, up to and including F). By default, informational messages are not logged. See *Section 4.7.1, "Setting Audit Levels"* for more information.

Use the **SHOW USER** command of the ACMSMGR utility to display a list of users currently logged in to the Remote Manager. You must be authenticated in order to issue this command.

```
$ ACMSMGR SHOW USER
```

4.4.1.1. Logging In

Login is performed using the ACMSMGR **LOGIN** command, which has the following syntax:

```
ACMSMGR LOGIN /USER=user-name /PASSWORD=password /NODE=node-name
```

The **/USER** qualifier can be omitted if the ACMS$MGMT_USER logical is defined. If the qualifier is provided, it overrides the ACMS$MGMT_USER logical. If neither the logical nor the qualifier is present, the ACMSMGR utility prompts the user for the user name.

If the **/PASSWORD** qualifier is not present, the ACMSMGR utility prompts the user for the password. There is no logical name for the password.

The **/NODE** qualifier can be omitted if the ACMS$MGMT_SERVER_NODE logical is defined. If it is provided, it overrides the ACMS$MGMT_SERVER_NODE logical. If neither the qualifier nor the logical name is provided, no login is attempted.

For each node to which a user logs in, a **credentials file** is created, either in the current directory or in the directory pointed to by the logical name ACMS$MGMT_CREDS_DIR. The credentials file contains encrypted security information (password is not stored in the file) and can be used by subsequent executions of the ACMSMGR utility. Credentials are specific to the process that created them and cannot be used by other processes. Prior to creating a new credentials file, any old credential files for the process are deleted.

Once a user has logged in to the Remote Manager, the user's credentials are valid for the duration of the credentials lifetime period, as specified by the parameter LOGIN_CREDS_LIFETIME. You can set the initial value of LOGIN_CREDS_LIFETIME with the ACMSCFG utility. You can change the value of LOGIN_CREDS_LIFETIME dynamically (but nondurably) with the ACMSMGR utility.

Once a user's credentials have expired, the user must log in to the server again.

4.4.1.2. Proxy Accounts

Proxy access to the management server is supported if the logical name ACMS$MGMT_ALLOW_PROXY_ACCESS is defined on the Remote Manager node. The valid values for this logical name are: 1, T, t, Y, y, TRUE, and true. If the name is defined to be any other value or if the logical name is not defined, proxy access is disabled.

When proxy access is allowed, users do not need to explicitly log in to the Remote Manager with a user name and password, and no credentials file is created. See *Section 4.4.1.1, "Logging In"* for a description of how to log in with user name and password.

In order for a user to be granted proxy access, there must be an entry in the ACMSPROXY.DAT for the combination of node and user attempting access. See *VSI ACMS for OpenVMS Managing Applications*

[https://docs.vmssoftware.com/vsi-acms-managing-applications/] for more information. The first time a user attempts to access a management function without having first logged in using user name and password, the Remote Manager looks for a valid ACMS proxy. If one is found, the OpenVMS account specified by the proxy is used for authorization.

The Remote Manager maintains a cache of users who have been logged in by proxy. Records remain in the cache for the duration of the proxy credentials' lifetime, as specified by the PROXY_CREDS_LIFETIME parameter. You can set the initial value of PROXY_CREDS_LIFETIME with the ACMSCFG utility. You can change the value of PROXY_CREDS_LIFETIME dynamically (but nondurably) with the ACMSMGR utility. Proxy credentials are automatically refreshed when they expire.

4.4.2. Authorization

Authorization consists of ensuring that the user attempting access holds the appropriate rights identifier on the node they are attempting to access. There are three levels of access, each with its own identifier, as shown in *Table 4.1, "Node Access Types and Rights Identifiers"*.

Table 4.1. Node Access Types and Rights Identifiers

Access Type	Rights Identifier
Operate	ACMS$MGMT_OPER
Read	ACMS$MGMT_READ
Write	ACMS$MGMT_WRITE

4.4.2.1. Read Access

Read access allows users to perform the following functions:

- Log in
- Log out
- Issue **SHOW** commands

4.4.2.2. Write Access

Write access allows users to issue the following commands:

- **ADD**
- **DELETE**
- **SET**

4.4.2.3. Operate Access

Operate access allows users to issue the following commands:

- **REPLACE**
- **RESET**

- **START**
- **STOP**

4.5. Starting and Stopping Interfaces

You can control which interfaces are started or stopped by using either the ACMSCFG utility prior to Remote Manager startup or the ACMSMGR utility after Remote Manager startup. The Remote Manager supports two interfaces:

- RPC

  The RPC interface is used by the ACMSMGR utility, and also by any user-written programs based on the MGMT API. Most users will enable the RPC interface.

- SNMP

  The SNMP interface is used by third-party system management packages to access ACMS management information. If no SNMP enabled packages are being used, this interface can safely be disabled.

Either the RPC or SNMP interface should always be enabled. If both are disabled, there is no way to communicate with the Remote Manager.

For a more complete discussion of the available interfaces and their attributes, see *Section 9.8, "Interfaces Table"*.

4.5.1. Using ACMSCFG to Enable or Disable Interfaces

Use the ACMSCFG utility to configure which interfaces should be enabled or disabled when the Remote Manager starts up. Either the SNMP or RPC interface should always be enabled. If both are disabled, there is no way to communicate with the Remote Manager.

Use the ACMSCFG **SET INTERFACE** command to enable or disable an interface. This command has the following syntax:

```
ACMSCFG SET INTERFACE /INTERFACE=interface-name /STATE=state
```

In this format:

- *interface-name* is one of the supported interfaces (SNMP or RPC).
- *state* is one of the following states: ENABLED or DISABLED.

Use the ACMSCFG **SHOW INTERFACE** command to determine the state of an interface in the configuration file:

```
$ ACMSCFG SHOW INTERFACE
```

4.5.2. Using ACMSMGR to Start or Stop Interfaces

Use the ACMSMGR utility to dynamically enable or disable an interface after the Remote Manager has already been started. As noted previously, at least one of either the SNMP or RPC interfaces should always be enabled. If both are disabled, there is no way to communicate with the Remote Manager (for

example, to shut it down or to enable an interface). Changes made with the ACMSMGR interface are not stored in the ACMSCFG file and are lost when the Remote Manager is stopped. Use the ACMSCFG utility to save changes to the ACMSCFG file.

An interface cannot disable itself. Since the ACMSMGR utility uses the RPC interface, it cannot be used to disable the RPC interface. To disable the RPC interface, either use the ACMSCFG utility and restart the Remote Manager, or use the SNMP interface.

Use the ACMSMGR **SET INTERFACE** command to disable the SNMP interface. The command has the following syntax:

```
ACMSMGR SET INTERFACE /INTERFACE=interface-name /STATE=state
```

In this format:

- *interface-name* must be SNMP.
- *state* is one of the following states: ENABLED or DISABLED.

Use the ACMSMGR **SHOW INTERFACE** command to determine the state of an interface:

```
$ ACMSMGR SHOW INTERFACE
```

4.6. Modifying Management Parameters

There are a large number of parameters that affect the internal processing of the ACMS Remote Manager. In general, most of these parameters will not need to be changed. However, you may need to alter some of these parameters in order to make the ACMS Remote Manager operate more efficiently or to meet your computing needs. You can modify these parameters using both the ACMSCFG and the ACMSMGR utilities.

For a more complete discussion of the available management parameters and their functions, see *Section 9.10, "Parameter Table"*.

4.6.1. Using ACMSCFG to Modify Management Parameters

Use the ACMSCFG utility to set the values of management parameters when the Remote Manager starts up.

Use the ACMSCFG **SET PARAMETER** command to modify the value of a parameter. The command has the following syntax:

```
ACMSCFG SET PARAMETER /parameter-name=value
```

In this format:

- *parameter-name* is one of the management parameters listed in *Section 9.10, "Parameter Table"*.
- *value* is the new value for the parameter.

Use the ACMSCFG **SHOW PARAMETER** command to determine the current value of the parameter in the configuration file:

```
$ ACMSCFG SHOW PARAMETER
```

4.6.2. Using ACMSMGR to Modify Management Parameters

Use the ACMSMGR utility to dynamically modify a management parameter after the Remote Manager has already been started. Not all parameters can be modified dynamically. Also, changes made with the ACMSMGR interface are not stored in the ACMSCFG file and are lost when the Remote Manager is stopped.

Use the ACMSMGR **SET PARAMETER** command to modify the value of a parameter. The command has the following syntax:

```
ACMSMGR SET PARAMETER /parameter-name=value
```

In this format:

- *parameter-name* is one of the dynamic management parameters listed in *Section 9.10, "Parameter Table"*.
- *value* is the new value for the parameter.

Use the ACMSMGR **SHOW PARAMETER** command to determine the current value of the parameter in the configuration file:

```
$ ACMSMGR SHOW PARAMETER
```

4.7. Managing Log Files

The ACMS Remote Manager maintains an audit log of internally generated messages. The log is stored in a location determined by the logical name ACMS$MGMT_LOG. If the logical is not defined, the default location is in the default directory for the account under which the Remote Manager process runs.

Depending on the tracing levels specified, the size of this file will vary. It is strongly suggested that ACMS system managers monitor this file to ensure that it does not grow too large.

If the Remote Manager is unable to write to the audit log, it prints a message to file SYS$ERRORLOG:ACMS$MGMT_SERVER.OUT and terminates. This can occur if logical name ACMS$MGMT_LOG is incorrectly defined, if the output device is full, or if the Remote Manager does not have sufficient privilege to write to the file.

4.7.1. Setting Audit Levels

Facilities within the Remote Manager write audit log messages based on the parameter settings, as shown in *Table 4.2, "Audit Level Parameters"*.

Table 4.2. Audit Level Parameters

Parameter	Function
DCL_AUDIT_LEVEL	Controls auditing for the DCL subprocess (used internally to modify the ACMS run-time system).
MGR_AUDIT_LEVEL	Controls auditing for the main Remote Manager process.

Parameter	Function
MSG_PROC_AUDIT_LEVEL	Controls auditing for the message processing thread (used internally to handle communications from ACMS processes).
PROC_MON_AUDIT_LEVEL	Controls auditing for the process monitor.
RPC_AUDIT_LEVEL	Controls auditing for the RPC interface.
SECURITY_AUDIT_LEVEL	Controls auditing for security access (authorization and authentication).
SNMP_AUDIT_LEVEL	Controls auditing for the SNMP interface.
TIMER_AUDIT_LEVEL	Controls auditing for the timer thread.

The value of each parameter determines what level of information is stored in the Remote Manager audit log. *Table 4.3, "Auditing Levels and Their Values"* shows the four levels of auditing and the integer value for each.

Table 4.3. Auditing Levels and Their Values

Auditing Level	Integer Value
Informational	1
Warning	2
Error	4
Fatal	8

Auditing values can be combined by logically ORing the integer values in order to have multiple levels of auditing in effect for a given facility. *Table 4.4, "Auditing Level Combinations and Their Values"* shows the valid auditing values.

Table 4.4. Auditing Level Combinations and Their Values

Auditing Level	Value
None	0
Info	1
Warn	2
Info, Warn	3
Error	4
Info, Error	5
Warn, Error	6
Info, Warn, Error	7
Fatal	8
Info, Fatal	9
Warn, Fatal	A
Info, Warn, Fatal	B

Auditing Level	Value
Error, Fatal	C
Info, Error, Fatal	D
Warn, Error, Fatal	E
All	F

Parameter settings are stored in the ACMSCFG file and can also be modified dynamically using the ACMSMGR utility. For example, in order to specify that all messages and events generated by the security routines should be stored in the audit log, use the following command:

```
$ ACMSCFG SET PARAMETER/SECURITY_AUDIT_LEVEL=F
```

Alternatively, to dynamically modify an auditing level, use the following ACMSMGR utility command:

```
$ ACMSMGR SET PARAMETER/SECURITY_AUDIT_LEVEL=F
```

4.7.2. Displaying Audit Messages

Use the **SHOW LOG** command in the ACMSMGR utility to display Remote Manager audit messages. This command accepts a number of qualifiers, including a qualifier that identifies the node from which to get audit messages (**/NODE**) and a qualifier that specifies the beginning time of messages to display (**/SINCE**).

The following example shows how to display audit messages from the node SPARKS:

```
$ ACMSMGR SHOW LOG/NODE=SPARKS
```

You can display audit messages from a node other than the current node only if the Remote Manager is running on the target node. If the Remote Manager is not running on the target node, you must first log in to the target node, and then issue the **SHOW LOG** command using the **/LOCAL** qualifier.

The following example shows how to display audit messages on the current node when the Remote Manager process is not running:

```
$ ACMSMGR SHOW LOG/LOCAL
```

For a complete description of the ACMSMGR commands and qualifiers, see *Chapter 11, "ACMSMGR Commands"*.

4.7.3. Resetting the Audit Log

Use the ACMSMGR **RESET LOG** command to close the current audit log file and open a new version. You may want to reset the log if it has grown too large.

The following example shows how to reset the log on node SPARKS:

```
$ ACMSMGR RESET LOG/NODE=SPARKS
```

Chapter 5. Using the Remote Manager to Manage ACMS

5.1. Managing Data Collection

Data collection is the mechanism by which ACMS run-time data is made available to the ACMS Remote Manager and, consequently, to other processes. Data collections do not involve disk or network read or write operations. All data collection is performed in memory on the local node.

ACMS systems managers control what data is collected by manipulating entries in the Collection table. In the Collection table, the data to be collected is specified by a combination of entity, class, and name.

- **Entity** refers to an ACMS run-time process type, such as ACC, EXC, or CP.
- **Class** refers to the class of data to be collected (see *Section 5.1.1, "Entities, Classes, Names, and Collections"*).
- **Name** refers to a process or application name that uniquely identifies a particular ACMS run-time process.

Using the combination of entity, class, and name gives ACMS system managers a great deal of flexibility in configuring the data to be collected.

Data collection can be managed either statically, through the ACMSCFG file, or dynamically, using one of the supported interfaces. For example, the ACMSMGR **`SET COLLECTION`** command can be used to dynamically enable or disable data collection on a local or remote node. Similarly, SNMP management tools can issue SNMP **`SET`** commands to dynamically modify entries in the Collection table. Users can also write their own programs and use the remote procedure call ACMSMGMT_SET_COLLECTION_1 (see *Chapter 8, "Management APIs"*) to dynamically manage data collection.

In general, management information is not collected unless an ACMS system manager has specifically enabled it. The exceptions are identification and configuration information. By default, these two classes of data are enabled for all ACMS entities. Having these classes enabled by default is an optimization that imposes little run-time overhead and ensures that process startup information is available. VSI recommends that you leave these classes enabled.

Data collection for other entities and classes is not enabled by default. When the ACMS system is started, the ACMS processes read either the configuration file (if the Remote Manager is not already running) or the Collection table to determine which classes of data to collect. Thereafter, external processes use the SNMP or RPC management interfaces to enable or disable data collection for a given entity, class, and name.

For each entity and class for which collection is enabled, a table of data values is populated by the appropriate ACMS processes (determined by name) and can be accessed by external entities using one of the data access interfaces (SNMP or RPC).

ACMS entities that collect data do so continuously when collection has been enabled for that entity/class/name combination. With the exception of event notifications (generated as the result of ACMS process startup or shutdown), and POOL information (which is updated based on timer intervals), collection data is modified when it changes.

5.1.1. Entities, Classes, Names, and Collections

ACMS system managers control data collection by modifying entries in the Collection table. The Collection table is keyed to entity, class, and name.

An **entity** is an ACMS run-time process or object. The valid ACMS entities are:

- ACC
- CP
- EXC
- QTI
- SERVER
- TASK GROUP
- TSC
- * or ALL
- * (all)

The wildcard value `*' is valid, and specifies all entities. When specifying an entity, you are specifying the type of process that it is. The asterisk (*) wildcard value is valid and specifies all entities. When specifying an entity, you are specifying its process type. The asterisk (*) wildcard value is valid and specifies all entities. When specifying an entity, you are specifying its process type.

A *class* is a set of run-time data values that entities set. Referring to data by class is a convenient method of referring to a set of related data values. However, the actual values contained in a class are entity specific. The following are valid classes:

- Configuration

  This class is a set of values that can be changed for the process and that controls some fundamental aspects of the execution. Configuration values are entity specific. An example of a Configuration class value for ACC is the maximum number of applications that may be running. An example value for a Server is the maximum number of instances.

- Identification

  This class is a set of values that do not change for the process as long as it is running and that help identify the process. Examples of Identification class values are process name, PID, and version.

- Pool

  This class is a set of values related to the current or historical MSS or workspace pool processing for the process. MSS pool values are the same for all entities except ACC. An example of a Pool class value for ACC is the current free amount in the MSS shared pool. An example value for other processes is the current free amount in the MSS process pool.

- Run-time

This class is a set of values that reflect either current or historical run-time processing for the process. Run-time values are also entity specific. An example of a Runtime class value for ACC is current number of applications. An example value for an EXC is the current number of executing tasks.

- *

  The asterisk is a wildcard value that specifies all classes.

A **name** specifies one or more specific processes of an entity type. The name field is entity specific. An example name for EXCs is the application name. An example name for CPs is process name. The wildcard value `*' is also supported, and for CPs is process name. The asterisk (*) wildcard value is also supported and matches all names.

Entity, class, and name are used in combination to determine which processes will collect which values. Duplicate rows (that is, rows with the same entity, class, and name) are not allowed, but it is possible to have overlapping entries in the Collection table if the wildcard value `* is used. Consider the example in table if the asterisk wildcard value is used. Consider the example in *Table 5.1, "Example 1: Collection with Wildcards"*.

Table 5.1. Example 1: Collection with Wildcards

Name	Entity	Class
*	ACC	*
*	ACC	Runtime

In this example, the entries overlap but are not duplicates. This is allowed because the attributes of each collection may be different. But users should be cautious when using the wildcard to avoid redundant processing.

When more than one row applies, the most specific row will be used, based on the column precedence of name, then entity, and then class. Within a particular column, wildcards are the least specific. In *Table 5.1, "Example 1: Collection with Wildcards"*, both rows are equivalent in name and entity, but the second row is more specific in class. In this case, the values from the first row will be used for all classes except the Runtime class. The values from the second row will be used for the Runtime class.

Consider the example in *Table 5.2, "Example 2: Collection with Wildcards"*.

Table 5.2. Example 2: Collection with Wildcards

Name	Entity	Class	Collection State
*	*	Runtime	Enabled
*	EXC	Runtime	Disabled
VR_APPL	EXC	Runtime	Enabled

In this example, the first row enables run-time data collection for all entities. The second row disables it for all EXCs. The third row enables it for the VR_APPL. As a result, among applications, only the VR_APPL will collect run-time data.

As an aide to identifying which row is the most specific and therefore will apply to a given process, the ACMSMGR command **`SHOW COLLECTIONS`** includes a column that represents the weight of a given row. A row with higher weight overrides a row with lower weight when they apply to the same class and

process. Consider the following example, which is the same as the example in *Table 5.2, "Example 2: Collection with Wildcards"* but includes the weights (in the column labelled "Wt") of each row.

```
SPARKS> ACMSMGR SHOW COLLECTION
ACMS Remote Management -- Command line utility

ACMS V4.4-0   Entity/Collection Table Display   Time:  19-APR-2001
 11:46:36.49

 Node         Wt Entity                Collect  Collect
 Storage  Storage
                 Type   Entity Name    Class    State    Storage Location
 State    Interval
 ------------ -- ------ -------------- -------  -------- ------------------
 -------- --------
 SPARKS       2  *        *            runtime  enabled  acms$mgmt_snapshot
 enabled  3600
 SPARKS       4  exc      *            runtime  disabled acms$mgmt_snapshot
 disabled 10
 SPARKS       8  exc      VR_APPL      runtime  enabled  acms$mgmt_snapshot
 disabled 10
```

In this example, the last row has the highest weight, and will override the other two rows for the RUNTIME class for the VR_APPL.

5.1.2. Starting and Stopping Collections

Users start and stop data collections by modifying the **data collection state** field in the Collection table. The Collection table is accessed through either the ACMSCFG utility prior to management startup, or through the ACMSMGR utility after Remote Manager startup.

By default, the ACMSCFG file includes entries to enable collection for the Identification and Configuration classes for all processes. Unless specific action has been taken to disable these collections, identification and configuration information is always available for all running processes.

Before a collection can be modified, it must be added to the entity collection table. By default, if the collection state is not specified when a collection is added, the collection state is DISABLED. Otherwise, the collection state is whatever was specified.

When the data collection state is set to ENABLED, the Remote Manager sends messages to the appropriate ACMS processes (based on the entity and name fields in the Collection table row) to begin collection for the class. When the data collection state is set to DISABLED, a similar message is sent to stop collection for the class. Once collection has started, it continues until the data collection state is set to DISABLED.

The requesting user must have ACMS$MGMT_WRITE privilege in order to start or stop a collection.

5.1.2.1. Using ACMSCFG to Start or Stop Collections

Use the ACMSCFG utility to set the state for a collection when the Remote Manager starts up. Some ACMSCFG commands are described here; for details on all ACMSCFG commands, see *Chapter 10, "ACMSCFG Commands"*.

Use the ACMSCFG **`ADD COLLECTION`** command to create a new collection record. The command has the following syntax:

```
ACMSCFG ADD COLLECTION /ENTITY=entity /CLASS=class /NAME=name /
COLL_STATE=state
```

Use the ACMSCFG **SET COLLECTION** command to modify the state of an existing collection record in the configuration file. The command has the following syntax:

```
ACMSCFG SET COLLECTION /ENTITY=entity /CLASS=class /NAME=name /
COLL_STATE=state
```

Use the ACMSCFG **DELETE COLLECTION** command to delete a collection. The command has the following syntax:

```
ACMSCFG DELETE COLLECTION /ENTITY=entity /CLASS=class /NAME=name
```

Deleting a collection can cause the Remote Manager to disable the class for a process because collections are disabled by default. The collection state for a process becomes disabled when no collections remain to specifically enable the class.

Use the ACMSCFG **SHOW COLLECTION** command to determine which collections already exist and their collection states. The command has the following syntax:

```
ACMSCFG SHOW COLLECTION
```

Note

You cannot use the ACMSCFG utility to add, delete, or modify Collection and Identification class records.

5.1.2.2. Using ACMSMGR to Start or Stop Collections

Use the ACMSMGR utility to dynamically modify the state of a collection after the Remote Manager has already been started. Note that changes made with the ACMSMGR interface are not automatically stored in the ACMSCFG file and are lost when the Remote Manager is stopped.

Use the ACMSMGR **ADD COLLECTION** command to create a new collection record. The command has the following syntax:

```
ACMSMGR ADD COLLECTION /ENTITY=entity /CLASS=class /NAME=name /
COLL_STATE=state
```

Use the ACMSMGR **SET COLLECTION** command to modify the state of an existing collection. The command has the following syntax:

```
ACMSMGR SET COLLECTION /ENTITY=entity /CLASS=class /NAME=name /
COLL_STATE=state
```

Use the ACMSMGR **DELETE COLLECTION** command to delete a collection. The command has the following syntax:

```
ACMSMGR DELETE COLLECTION /ENTITY=entity /CLASS=class /NAME=name
```

Deleting a collection can cause the Remote Manager to disable the class for a process because collections are disabled by default. The collection state for a process becomes disabled when no collections remain to specifically enable the class.

Use the ACMSMGR **SHOW COLLECTION** command to determine which collections already exist and their collection states. The command has the following syntax:

```
ACMSMGR SHOW COLLECTION
```

5.1.2.3. Using SNMP to Start or Stop Collections

Use the SNMP interface to dynamically modify the state of a collection after the Remote Manager has already been started. Note that changes made with the SNMP interface are not stored in the ACMSCFG file and are lost when the remote Remote Manager is stopped.

The SNMP interface responds to SNMP commands issued by SNMP consoles. An SNMP console issues an SNMP **SET** command to the Remote Manager to modify the Collection table.

The SNMP OID (object ID) for the collection state columns are listed in in the file MIB_OID.LIS in ACMS$RM_EXAMPLES. The data type for the field is INTEGER. Possible settings for this field have the following meanings:

- 0 = Collection is disabled.
- 1 = Collection is enabled.
- 9 = Collection record is deleted.

You cannot add a collection record using the SNMP interface.

5.2. Displaying Collected Data

Management data can be displayed using either the ACMSMGR utility or one of the programming interfaces (SNMP or ONC RPC). Data is displayed by entity and, optionally, by class.

5.2.1. Using ACMSMGR to Display Collected Data

Use the ACMSMGR **SHOW** command to display collected data. See *Chapter 11, "ACMSMGR Commands"* for a description of each command.

The following ACMSMGR command displays ACC data:

```
$ ACMSMGR SHOW ACC /NODE=SPARKS /ID
```

The following example shows output from this command:

```
ACMS Remote Management Option -- Command line utility
ACMS V4.4-0   ACC Table Display                    Time:  19-APR-2001
 11:59:09.56


                ID
   Node        Class     PID      Process Name          Start Time
 User Name      Version
 ------------ -------- -------- --------------- ------------------------
 ------------ ------------
 sparks        enabled  2020C8BB ACMS01ACC001000  18-APR-2001 14:44:47.29
 SYSTEM        V4.4-0
```

5.3. Managing ACMS Using the Remote Manager

The ACMS Remote Manager provides the ability to modify the running ACMS system using either the SNMP or the RPC interface. In general, only Configuration class variables can be modified at run time. However, not all Configuration class variables can be modified. *Chapter 9, "Remote Manager Reference Tables"* lists all Configuration class variables by entity and indicates which ones can be modified.

5.3.1. Types of Variables

Many Configuration class variables can have the following two forms:

- Stored variable (see *Section 5.3.1.1, "Stored Variables"*)
- Active variable (see *Section 5.3.1.2, "Active Variables"*)

The programming interfaces expose stored and active values as separate variables.

5.3.1.1. Stored Variables

Stored variables are maintained by the ACMS run-time system on disk, either in the ACMSGEN file or as part of an ADB or TDB file. For example, mss_maxobj is a run-time variable that is stored in the ACMSGEN file. The auditing state for a particular application is a run-time variable that is stored in the application database (ADB).

As you might expect, the ACMS Remote Manager allows ACMSGEN stored values to be modified, but it does not allow modifications to values that are stored in application executables.

Changes to stored values are durable but not dynamic. That is, if the stored value of a variable is modified, the value survives the restart of the ACMS run-time system. However, changes to stored values do not take effect immediately. Some or all of the ACMS run-time system needs to be restarted before the new value takes effect.

For example, to change the value of the `mss_net_retry_timer` parameter in the ACMSGEN file using ACMSMGR, use the following command:

```
$ ACMSMGR SET ACC/MSS_NET_RETRY_TIMER=50/STORED
```

To change the value in ACMSGEN file using the RPC interface, set the `mss_net_retry_timer_stored` field in the `acc_config_rec` using the ACMSMGMT_SET_ACC procedure. To change the same value using an SNMP console, set the `acc_mss_net_retry_timer_stored` field in the ACC Table.

Note that none of these changes would effect the running system. To effect the running system, you must change the active value (see *Section 5.3.1.2, "Active Variables"*.)

5.3.1.2. Active Variables

Active variables are maintained in memory by the ACMS run-time system. All Configuration class variables are active because they have an in-memory value. Although the ACMS Remote Manager allows most active values to be modified, not all changes to active values are dynamic. Refer to *Chapter 9, "Remote Manager Reference Tables"* to determine whether a particular active value is dynamic. Changes to nondynamic active variables are essentially useless.

Changes to active values are never durable; that is, they never survive a restart of the system.

For example, to change the active value of the `mss_net_retry_timer` using ACMSMGR, use the following command:

```
$ ACMSMGR SET ACC/MSS_NET_RETRY_TIMER=50/ACTIVE
```

To change the value using the RPC interface, set the `mss_net_retry_timer_`active field in the `acc_config_rec` using the ACMSMGMT_SET_ACC procedure. To change the same value using an SNMP console, set the `acc_mss_net_retry_timer_active` field in the ACC table.

Note that none of these changes would survive a system restart. To change a value and have it survive a system restart, you have to change the stored value (see *Section 5.3.1.1, "Stored Variables"*).

5.3.2. How the Remote Manager Makes Changes

The ACMS Remote Manager applies changes to the ACMS run-time system either by using the ACMSGEN parameter file and utility, or through the ACMSOPER utility. In either case, the ACMS Remote Manager server applies updates to the running system by creating temporary command procedures that are executed by a spawned DCL subprocess (process name ACMS$MGMT_DCL).

The temporary command procedures are written to and read from the directory pointed to by the logical name ACMS$MGMT_TEMP. If this logical is not defined when the Remote Manager starts, it will define the logical to point to SYS$MANAGER.

Temporary command procedures are given names unique to the procedure instance that creates them, but the names are not unique across nodes. These names are deleted after they have been executed.

If the Remote Manager server does not have access to the directory pointed to by ACMS$MGMT_TEMP, all update attempts fail. However, the definition of the logical can be changed without restarting the Remote Manager. Changing the definition at run time should be done cautiously. One or more updates could fail if the logical is changed in the middle of an update operation.

If the ACMSMGR or RPC interface is used, any errors that occur during the system update are returned to the user and are written to the Remote Manager log file. Depending on the current setting of the `dcl_audit_level` parameter, some messages may not be written to the log.

User accounts (including proxy accounts and the ACMS$SNMP account, if SNMP is being used) must be granted the ACMS$MGMT_WRITE or ACMS$MGMT_OPERATE rights identifier in order to modify Configuration class values. See *Section 4.4.2, "Authorization"* for a list of functions and the rights identifier required for each.

5.3.3. Using ACMSMGR to Modify the ACMS Run-Time System

The ACMSMGR utility can be used to dynamically modify Configuration class parameters for ACMS run-time entities. More than one value can be modified at once, on one or more nodes. The command executes synchronously; that is, it does not complete until an attempt has been made to update all parameters. Multiple node updates are processed serially; all updates are performed on one node before any updates are attempted on subsequent nodes.

Use the ACMSMGR **`SET`** command to modify a Configuration class variable. The syntax of the command is as follows:

```
ACMSMGR SET entity [/parameter=value,...]
```

For example, the following command disables ACMS auditing on the node specified by ACC:

```
$ ACMSMGR SET ACC /AUDIT_STATE=DISABLED
```

Two qualifiers are provided to control whether the active (**/ACTIVE**) or stored (**/STORED**) value of the variable is to be modified. The default is **/STORED**. Both qualifiers can be specified in a single command to update both values. For example, the following command modifies both the active and stored values of the ACC Configuration class variable *node_name*:

```
$ ACMSMGR SET ACC/NODE_NAME=SPARKS/ACTIVE/STORED
```

If a specified qualifier does not apply (for example, **/ACTIVE** is specified for a nondynamic variable), the qualifier is ignored.

For a complete list of Configuration class variables, see *Chapter 9, "Remote Manager Reference Tables"*.

The ACMSMGR **START** and **STOP** commands can be used to dynamically start and stop the following processes:

- ACC (starts or stops the entire ACMS run-time system)
- EXC
- MANAGER (Remote Manager; stop only)
- QTI
- TRACE_MONITOR
- TSC (starts or stops the TSC and any CPs)

In addition, ACMS procedure servers can be replaced (stopped and restarted) using the ACMSMGR **REPLACE** command.

Different qualifiers are available for each command and process.

For more information about ACMSMGR commands, refer to *Chapter 11, "ACMSMGR Commands"*.

5.3.4. Using SNMP to Modify the ACMS Run-Time System

The SNMP interface can be used to dynamically modify Configuration class parameters for ACMS run-time entities. Updates to Configuration class parameters are synchronous; the SNMP command does not complete until an attempt has been made to update the parameter.

The SNMP interface responds to SNMP commands issued by SNMP consoles. An SNMP console issues an SNMP **SET** command to the Remote Manager to modify Configuration class parameters.

There are both active and stored values for many of the Configuration class variables. In the ACMS MIB, each value is given a separate variable (OID).

Because the SNMP protocol offers only **GET** and **SET** commands, the SNMP interface handles the following operations differently from the RPC interface in order to perform the full range of management activities:

- Starting and stopping processes (see *Section 5.3.4.1, "Starting and Stopping Processes Using SNMP"*)
- Adding and deleting table rows (*Section 5.3.4.2, "Adding and Deleting Rows Using SNMP"*)
- Replacing servers (*Section 5.3.4.3, "Replacing Application Procedure Servers Using SNMP"*)

Not all operations that can be performed by the RPC interface can be performed by the SNMP interface. The following sections indicate which operations are not available in the SNMP interface.

5.3.4.1. Starting and Stopping Processes Using SNMP

To start or stop the following ACMS processes, issue an SNMP **SET** command on the Configuration class variable *acms_state*, and specify the state as either 1 (to start the process) or 0 (to stop the process).

- ACC
- QTI
- TSC

You cannot start or stop CP processes.

To start an ACMS application, issue an SNMP **SET** command on the `exc-appl-name` field in the excTable, specifying a row that is not currently in use and that is less than the value of the `acc-max-appl-active` field in the accTable.

To stop an ACMS application, issue an SNMP **SET** command on the `exc-acms-state` field, specifying a value of 0.

You cannot start or stop application procedure servers or task groups.

5.3.4.2. Adding and Deleting Rows Using SNMP

Currently, no tables allow rows to be added using SNMP.

The Collection and Trap tables allow rows to be deleted using SNMP.

- To delete rows from the Collection table, set the collection-state field to 9. A value of 1 enables the collection; a value of 0 disables the collection; a value of 9 deletes the collection.
- To delete rows from the Trap table, set the `trap-delete` field to 1. This is the only value allowed for this field.

5.3.4.3. Replacing Application Procedure Servers Using SNMP

To replace an ACMS application procedure server, issue an SNMP **SET** command on the `set-replace-flag` field in the Server table, specifying a nonzero value.

5.3.5. Using ONC RPC to Modify the ACMS Run-Time System

The RPC interface can be used to dynamically modify Configuration class parameters for ACMS run-time entities. Configuration class parameter updates are synchronous; the RPC command does not complete until an attempt has been made to update the parameter.

There are both active and stored values for many of the Configuration class variables. In the ACMSMGMT_RPC.X IDL file, each value is given a separate variable.

Separate RPC commands for each entity type are provided for modifying Configuration class variables. In addition, RPC commands are provided to perform start, stop, add, delete, replace, and reset functions. *Chapter 8, "Management APIs"* provides details about all of the RPC commands.

Chapter 6. Management Programming Using ONC RPC

Programmers who want to access and maintain the ACMS Remote Manager from their own programs can use the following two interfaces:

- Simple Network Management Protocol (SNMP)

  The SNMP interface is provided for integration with enterprise management packages. For more information, see *Chapter 7, "Management Programming Using SNMP"*.

- Open Network Computing (ONC) Remote Procedure Call (RPC)

  The ONC RPC interface is for system managers and system programmers who want to write custom tools and applications that access the ACMS Remote Manager.

This chapter describes the ONC RPC interface. Programmers who are familiar with the C programming language and RPC mechanisms can use this information when coding and building their own client programs. For a more complete discussion of ONC RPC programming, see *Power Programming with RPC* by John Bloomer, published by O'Reilly & Associates, Inc., Sebastopol, CA ADD VMS DOC and *VSI TCP/IP Services for OpenVMS ONC RPC Programming* [https://docs.vmssoftware.com/vsi-tcpip-services-for-openvms-onc-rpc-programming/#VSI_TCPIP_SERVICES_ONC_RPC_REF] .

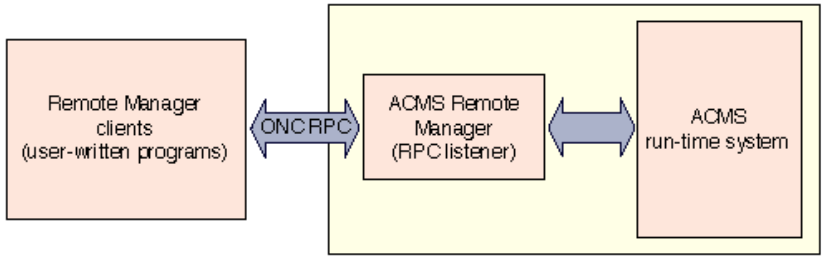
6.1. ONC RPC Overview

ONC RPC is a widely used and supported remote procedure call (RPC) mechanism. Similar to other RPC mechanisms, the ONC RPC protocol supports a request/response model, in which client applications make requests of servers and receive responses. Clients typically make synchronous calls to remote servers over a network. The RPC mechanism hides the network operations from the programmer, making each remote procedure call appear to be a local function invocation.

Unlike the SNMP interface, which connects to the ACMS Remote Manager using the SNMP master agent, access through ONC RPC is directly to the ACMS Remote Manager.

Figure 6.1, "ONC RPC Interface Overview" provides a graphical overview of the ONC RPC interface.

Figure 6.1. ONC RPC Interface Overview

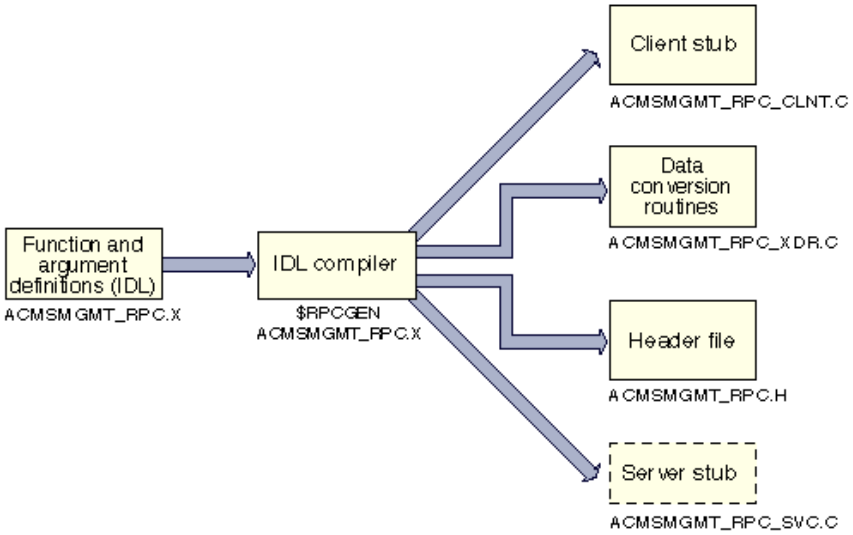


Programming for ONC RPC is based on interface definitions coded in Interface Definition Language (IDL). Functions and their arguments are described in IDL source files, which are precompiled using an IDL compiler. The outputs from the IDL compiler are a set of C source and header files that are then

compiled and linked with client and server programs to form run-time executables. For Remote Manger client development, server stub files are not needed and can be discarded.

Figure 6.2, "ONC RPC Programming Overview" provides a graphical overview of programming for ONC RPC.

Figure 6.2. ONC RPC Programming Overview



The IDL that describes the procedures supported by the ACMS Remote Manager is provided with the ACMS Remote Manager installation and provides the basis for ACMS management programming. Users write their own client programs, calling the functions described in the ACMS Remote Manager IDL file (ACMSMGMT_RPC.X). They precompile the IDL file with the precompiler provided by their TCP/IP package, and then compile and link their client programs. No compilation or linking is required for the Remote Manager; it contains all the support required by ONC RPC client programs.

The ACMS Remote Manager provides several types of procedures that are callable through the ONC RPC interface. These procedures provide read and write access to each table maintained by the Remote Manager, as well as command routines (such as Start and Stop). *Table 6.1, "Procedures for Accessing Remote Manager Functions"* summarizes the types of procedures available.

Table 6.1. Procedures for Accessing Remote Manager Functions

Procedure Type	Table or Object	Description
Add	Collection, Trap	Allows entries to be added to configuration tables.
Delete	Collection, Trap	Allows entries to be removed from configuration tables.
Get	ACC, MGR_STATUS, PARAM, QTI, TSC	Returns all columns in the table.
List	Collection, CP, EXC, Interfaces, Log, Process, Trap, Server, Task Group, Users	Returns a linked list of records based on selection criteria. All columns in the table are returned with each row.
Replace	Server	Allows an application server to be replaced.

Procedure Type	Table or Object	Description
Reset	Log	Allows the current version of the Remote Manager log to be closed and a new version to be opened.
Set	ACC, CP, Collection, EXC, Interfaces, Trap, Param, QTI, SERVER, TSC	Allows modifications to the table. For configuration tables, set functions allow rows to be added to tables. Entity rows can only be added by starting the appropriate process.
Start	ACC, TSC, QTI, EXC, Trace Monitor	Allows ACMS processes to be started.
Stop	Manager, ACC, TSC, QTI, EXC, Trace Monitor	Allows ACMS processes to be stopped.

The procedure names and arguments for each procedure type are similar – all Get calls have similar names and arguments; set calls have similar names and arguments, and so on.

The sections that follow describe in more detail how to write programs that access these functions.

6.2. Building Multithreaded Clients

The file ACMS$MGMT_EXAMPLES_BUILD.COM describes how to use the client stub provided with the Remote Manager when compiling and linking your application. The sample procedures provided in ACMS$MGMT_EXAMPLES.C are designed to use the thread-safe client stub, and as a result, each procedure contains one or more "free" calls (to prevent memory leaks).

To implement a non-thread safe client using the RPC-generated stub, omit the "free" calls. See ACMS$MGMT_EXAMPLES_BUILD.COM for detailed build instructions.

6.3. API Overview

Remote management client programs follow a typical programming model that involves the following phases:

- Initialization

  During the **initialization** phase, client programs establish connections with the Remote Managers they will be calling. As part of this phase, the programs select a security mode (explicit or implicit). Once this phase is complete, the Remote Managers have been verified to be available, and the client authentication has been verified. This phase involves using a combination of ONC RPC function calls and an ACMS Remote Manager function call (if explicit authentication is being used).

- Processing

  During the **processing** phase, client programs make procedure calls to the Remote Managers. During this phase, clients obtain or modify management information. This phase involves the use of the functions defined in the ACMS$RM_EXAMPLES:ACMSMGMT_RPC.X IDL file.

- Termination

During the **termination** phase, clients halt execution. There is no API support or programming requirement for this phase.

6.4. Initialization and Security

In order to perform initialization, ACMS remote client programs must first determine the type of authentication (explicit or implicit) they will use. The type of authentication determines whether or not the client program must obtain credentials.

The Remote Manager performs authentication either **explicitly**, using a valid OpenVMS account name and password, or **implicitly**, using ACMS proxies. Implicit authentication is allowed only if it has been enabled on the Remote Manager node, and does not require the use of credentials. Explicit authentication requires the use of credentials and also requires that the client process execute a separate login using the ACMSMGR utility.

See *Section 4.4, "Logging In to the Remote Manager"* for a discussion of the various security modes and how to log in using ACMSMGR.

Once the authentication mode has been determined, remote management clients perform the following tasks:

- Establish an RPC connection with the Remote Manager on the target node.

  The **clnt_create** function call establishes RPC client connections.

- Establish the security context and, optionally, populate it with credentials information.

  The security context is established by calling the **authunix_create_default** function. As a result of this call, client process-identity information is passed to the server on each procedure call. The Remote Manager uses this information to authorize the user for each function.

  The default security context is not sufficient if explicit authentication is being used. Clients that need to support explicit authentication call the ACMS$MGMT_GET_CREDENTIALS function to obtain a client ID, which was previously issued for the client process by executing a login through the ACMSMGR utility. This client ID is used on subsequent RPC calls.

Note

In order for credentials information to be created, the client process must first execute the **`LOGIN`** command of the ACMSMGR utility. The only way to create credentials files is by using the ACMSMGR utility.

6.4.1. Initialization Example

The following example code shows a client program that establishes an RPC connection with the Remote Manager, establishes the security context, and then populates it with credentials information if a logical name (ACMS$MGMT_USER) has been defined.

```
#include <rpc/rpc.h>
#include string
#include "acmsmgmt_rpc.h"
```

```
CLIENT *cl;
char sname[] = "sparks";
char *username_p, username[13] = "";
int client_id;
int status;

int acms$mgmt_get_creds();

int main ()
{

    /* if the logical is defined, credential information will be used */
    username_p = getenv("ACMS$MGMT_USER");
    if (username_p)
          strcpy(username,username_p);

    /* establish an rpc connection to the server */
    cl = clnt_create(sname, ACMSMGMT_RPC, ACMSMGMT_VERSION, "tcp");

    /* if the connection was established */
    if (cl != NULL) {

 /* create a security context */
        cl->cl_auth = authunix_create_default();
        client_id = 0;

 /* optionally, get credentials for this user & server */
        if (strlen(username))
           status = acms$mgmt_get_creds(sname,username,&client_id);

      }

return(1);
}
```

6.5. Get Procedures

Get procedures are available for all ACMS Remote Manager tables. Get procedures return all columns from a single table row.

As *Table 6.2, "Get Procedures"* shows, a separate get procedure is available for each entity and table.

Input arguments to get procedures are **client_id**. See *Chapter 8, "Management APIs"* for details about each call.

Table 6.2. Get Procedures

Procedure	Description
acmsmgmt_get_acc_1	No keys; only 1 ACC per node.
acmsmgmt_get_mgr_status_1	No keys; only one row in the Manager Status table.
acmsmgmt_get_param_1	No keys; only one row in the Parameter table.
acmsmgmt_get_qti_1	No keys; only 1 QTI per node.
acmsmgmt_get_tsc_1	No keys; only 1 TSC per node.

6.5.1. Get Example

The following example code shows how a client program calls the `acmsmgmt_get_param_1` procedure and displays the current value of a parameter.

```
int get_param_data(int client_id,CLIENT *cl)
 {
   int x = 0;
   int y = 0;

   param_rec2       *params;
   param_rec_out2   *param_rec;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;

   param_rec = acmsmgmt_get_param_2(&sub_rec,cl);

   if (!param_rec) {
      printf("\n RPC Call to get Parameter data failed");
      return(MGMT_FAIL);
   }

   if (param_rec->status != MGMT_SUCCESS) {
      printf("\n Call to get Parameter data failed, returning status code
 %d",
             param_rec->status);
      status = param_rec->status;
        xdr_free(xdr_param_rec_out2, param_rec);
        free(param_rec);
      return(status);
   }

   params = &param_rec->param_rec_out2_u.data;

      printf("\n Maximum logins allowed is %d",params->max_logins);
        xdr_free(xdr_param_rec_out2, param_rec);
        free(param_rec);
      return(0);
}
```

6.6. List Procedures

List procedures operate on all rows in a table. Procedures are available for each entity and each configuration table with more than one row. There are no list procedures for the following tables, since they contain only one row:

- ACC table
- TSC table
- QTI table
- Parameter table

As *Table 6.3, "List Procedures"* shows, separate list procedures are provided for the remainder of the management information and configuration tables. Input to a list procedure is a selection criteria record, which varies depending on the table being accessed. Some key values in the selection criteria records will support wildcards (*, %). support wildcards (*, %).

Table 6.3. List Procedures

Procedure	Description
acmsmgmt_list_collections_1	Key value is table index.
acmsmgmt_list_cp_1	No keys.
acmsmgmt_list_exc_1	Key value is application name or table index.
acmsmgmt_list_interfaces_1	No keys.
acmsmgmt_list_log_1	No keys; selection criteria is before time, since time, file name, facility, severity.
acmsmgmt_list_proc_1	No keys.
acmsmgmt_list_server_1	Key value is application name, server name, or table index.
acmsmgmt_list_tg_1	Key value is application name, task group name, or table index.
acmsmgmt_list_trap_1	No keys.
acmsmgmt_list_users_1	No keys.

For all list procedures, only entire rows (that is, all columns in the row) are returned. Data is returned in a linked list. The number of nodes in the list is determined by the system-wide parameter table field `max_rpc_return_recs`. When the number of rows to be returned exceeds the value of `max_rpc_return_recs`, the caller must reissue the call, providing the appropriate key values to fetch the next set of rows. The call returns status MGMT_NO_MORE_ROWS if there are no more rows available. Procedures with no keys return all rows in the table on the first call, regardless of the value of the `max_rpc_return_recs` field.

6.6.1. Linked List Example

Data from list calls is returned in a linked list. The example in this section uses the `acmsmgmt_list_log_1` procedure to illustrate how linked list processing works.

The call to the `acmsmgmt_list_log_1` procedure requires the following input structure:

```
struct  log_sel_struct {
    int          client_id;
    string       before_time<TIME_SIZE_A>;
    string       since_time<TIME_SIZE_A>;
    string       file_name<STORAGE_LOC_SIZE>;
    int          dup_count;
    int          facility;
    int          severity;
    };
```

In the code example that follows, the lines of code beginning with `log_rec` initialize the fields in this structure as follows:

- Client_id is set to 0 to select proxy authentication.
- Before_time is set to a NULL string to specify no end date for viewing log entries. Note that you cannot provide a NULL pointer.
- Since_time is set to the 1st of January 1998. Log entries from this date and later will be viewed.
- File_name is set to an empty string, which causes the active log file to be used.
- Dup_count is set to -1. This field is used to uniquely identify log records with identical times.
- Facility is set to -1, which causes entries for all facilities to be returned.
- Severity is set to -1, which causes entries of all severity levels to be returned.

The following example code shows the initialization of the client and the call to the `acmsmgmt_list_log_1` procedure:

```
#include <rpc/rpc.h>
#include <stdio.h>
#include string
#include "acmsmgmt_rpc.h"
CLIENT *cl;
int main ()
{
int skip_rec = 0;
char null_time_str[24] = "";
char first_of_jan[24] = "01-JAN-1998 00:00:00.00";
char file_spec[] = ""; /* use default, i.e. active log file */
char time_cache[MGMT_S_TIME_A+1];
static struct log_sel_struct log_rec;
log_data_list *log;
log_link *nl;
/* Initialize client connection; if that fails, exit*/
cl = clnt_create(sname, ACMSMGMT_RPC, ACMSMGMT_VERSION, "tcp");
if (!cl)
return(MGMT_FAIL);
/* Create a default security context */
cl->cl_auth = authunix_create_default();
/* So far so good. Initialize log selection data */
log_rec.client_id = 0;
log_rec.before_time = null_time_str;
log_rec.since_time = first_of_jan;
log_rec.file_name = file_spec;
log_rec.dup_count = -1;
log_rec.facility = -1; /* don't match on facility */
log_rec.severity = -1; /* don't match on severity */
top:
/* Now make RPC */
log = acmsmgmt_list_log_1(&log_rec,cl);
```

The return value from the calls to all list procedures (including `acmsmgmt_list_log_1`) is a pointer to a union. If the pointer returned is NULL, the call has failed. RPC error checking must be used to determine the cause of the error. If a valid pointer has been returned, it will point to a structure containing a union with the following structure:

```
struct log_data_list {
int status;
```

```
union {
log_list list;
int rc;
} log_data_list_u;
};
```

The status field determines which structure is being returned. If the status is equal to MGMT_FAIL, the `rc` field is returned. The rc field contains a status code indicating the reason for failure.

If the status field is not equal to MGMT_FAIL, a pointer to a linked list has been returned.

The `log_list` field is defined as a pointer to linked list node, as follows:

```
typedef struct log_link *log_list;
```

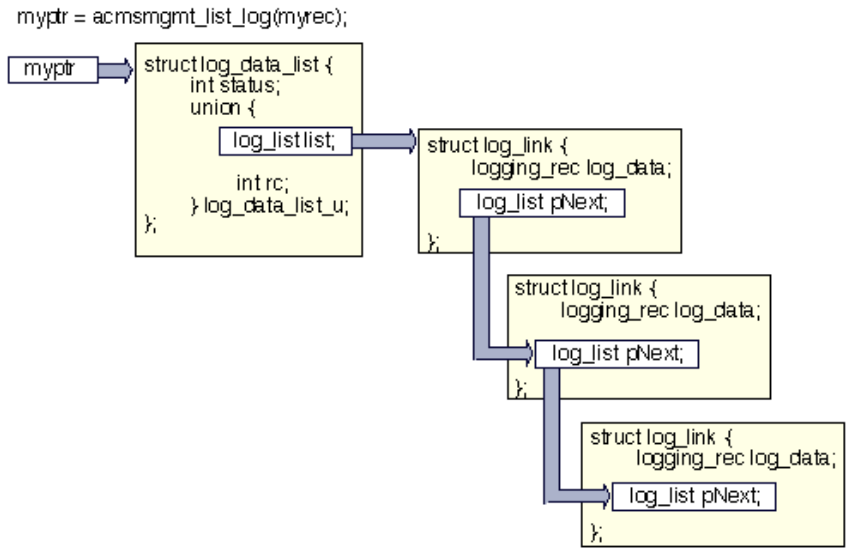
The linked list node has the following structure:

```
struct log_link {
        logging_rec log_data;
        log_list pNext;
};
```

In this structure, `log_data` is of type `logging_rec`, which is a record structure containing the log data. The `pNext` field is a pointer to the next node in the linked list (which is of type `log_link`).

Figure 6.3, "Linked List: Return Structure and Construction" illustrates the return structure and how the linked list is constructed.

Figure 6.3. Linked List: Return Structure and Construction



The following example code shows how to check whether the call completed successfully, and how to traverse the linked list to display the data:

```
/* if a NULL pointer was returned, the RPC failed */
if (!log)
       return(MGMT_FAIL);

/* if bad status was returned, something failed in our call.
        log->log_data_list_u.rc contains the status */
```

```
if (log->status == MGMT_FAIL)
        return(log->log_data_list_u.rc);

/* while more data in the list, display the data */
for (nl = log->log_data_list_u.list; nl != NULL; nl = nl->pNext) {
        if (skip_rec)
                skip_rec = 0;
        else
                printf("\n %-12s\t%-s",sname,nl->log_data.log_msg);

        /* save last time received to use as next time to read forward from
 */
        memcpy(&time_cache[0],nl->log_data.log_msg,23);
        log_rec.dup_count  = nl->log_data.dup_count;
        log_rec.since_time = time_cache;

}
if (log->status == MGMT_NOMORE_DATA)
        printf("\n *** End of data **");
else {
        skip_rec = 1;
        goto top;
}

return(1);
}
```

In this example, the returned pointer is checked for whether data has been returned (log is not NULL). Then the status code is checked for whether the call completed successfully.

If the call completed successfully, the code drops into a FOR loop and starts printing the data. For this particular call, the client prints all the records the very first time the RPC is called; on subsequent calls, the first record is a duplicate of the last one from the previous call and is not printed.

After printing a record, the key data is saved to be used again on a subsequent call. Remember that only `max_rpc_return_recs` is returned in each call to the `acmsmgmt_list_log_data_1` procedure. There may be more log records than can be sent at once. It is the responsibility of the client to initialize the call properly to get the next set of records.

Once all the returned records have been returned, the code will call the `acmsmgmt_list_log_data_1` procedure again if the status code from the call was not MGMT_NOMORE_DATA. In this way, all the records are retrieved.

6.7. Set Procedures

Set procedures are available for many of the ACMS Remote Manager tables. Set procedures allow you to modify ACMS entity and Remote Manager configuration information. As *Table 6.4, "Set Procedures"* shows, a separate set procedure is available for each entity and table.

Table 6.4. Set Procedures

Procedure	Description
acmsmgmt_set_acc_1	No keys; only 1 ACC per node.
acmsmgmt_set_coll_1	Key value is entity, ID, and class.

Procedure	Description
acmsmgmt_set_exc_1	Key value is application name.
acmsmgmt_set_interface_1	Key value is interface name.
acmsmgmt_set_param_1	No keys; only one row in the parameter table.
acmsmgmt_set_qti_1	No keys; only 1 QTI per node.
acmsmgmt_set_server_1	Key value is application name and server name.
acmsmgmt_set_trap_1	Key value is entity, ID, and parameter.
acmsmgmt_set_tsc_1	No keys; only 1 TSC per node.

For Entity tables, set procedures allow fields to be modified for a particular entry. A unique key value must be provided to identify the particular table row to be updated for tables with more than one row. Only configuration class fields can be modified in entity tables.

For the Trap and Collection tables, add and delete procedures (described in *Section 6.8, "Delete Procedures"* and *Section 6.9, "Add Procedures"*) are available along with set procedures. Each procedure requires a unique key value.

For all tables, some or all fields in a row can be modified in a single call. The Remote Manager scans the input record for uninitialized fields (that is, fields that are not set to the default value of -1); if a field contains an initialized value, the Remote Manager attempts to apply the update. The corresponding field in the return record is updated with the completion status of the update. Updates are applied serially, but the Remote Manager attempts to update all initialized fields regardless of the outcome of any individual update. The exception to this processing is if an internal error occurs, in which case processing is aborted.

All calls are synchronous.

See *Chapter 8, "Management APIs"* for details about each call.

6.7.1. Set Example

The following example code shows how a client program calls the `acmsmgmt_set_param_1` procedure to change the values of the `proc_mon_interval` and `mss_coll_interval` parameters.

This example assumes client initialization has been performed as described in *Section 6.4, "Initialization and Security"*.

```
int set_param_data(int client_id,CLIENT *cl)
 {
   int x = 0;
   int y = 0;

   static param_config_rec2 set_struct;
   param_status_rec2 *ret_struct;
   static int *status;

   /* initialize input argument; values < 0 are not processed
      by the server */
   memset(&set_struct,-1,sizeof(set_struct));
```

```
   /* establish the client id */
   set_struct.client_id = client_id;
   set_struct.params.proc_mon_interval = 60;
   set_struct.params.mss_coll_interval = 60;

   ret_struct = acmsmgmt_set_param_2(&set_struct,cl);

   if (!ret_struct) {
         printf("\nCall to modify parameters failed");
         return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

      if (ret_struct->status != MGMT_WARN) {
         printf("\nCall to modify parameters failed, returning %d",
                ret_struct->status);
         status=ret_struct->status;
           xdr_free(xdr_param_status_rec2, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
      }

      if (ret_struct->param_status_rec_u.data.proc_mon_interval !=
 MGMT_SUCCESS)
         printf("\n Call to modify proc_mon_interval failed");
      if (ret_struct->param_status_rec_u.data.mss_coll_interval !=
 MGMT_SUCCESS)
         printf("\n Call to modify mss_coll_interval failed");
           xdr_free(xdr_param_status_rec2, ret_struct);
           free(ret_struct);
      return(MGMT_FAIL);
   }
   else
      printf("\n Call to update parameters successful");
        xdr_free(xdr_param_status_rec2, ret_struct);
        free(ret_struct);
      return(0);
}
```

In this example, note that the input argument (**set_struct**) is initialized to negative values prior to the call. The Remote Manager will attempt to apply updates for any positive values found; negative values are ignored.

Following the call to the update routine, the return record pointer is tested to ensure that it is not NULL (that is, that the call completed). Then individual return codes are tested to determine the status of the updates. The first status check (`ret_rec->status`) determines the overall call status. For instance, security violations will be recorded in this field. If that status field contains a failure code, no updates were attempted. If that status field contains MGMT_SUCCESS, updates were attempted for the two fields. The subsequent status checks in the return record determine the outcome of those updates.

6.8. Delete Procedures

Delete procedures are available for the Collection and Trap tables. Delete procedures allow you to remove rows from the corresponding table. As *Table 6.5, "Delete Procedures"* shows, a separate delete procedure is available for each of these tables.

The delete procedures require an input record with key data to be passed by the caller. A simple status code is returned indicating the success or failure of the operation.

All calls are synchronous.

See *Chapter 8, "Management APIs"* for details about each call.

Table 6.5. Delete Procedures

Procedure	Description
acmsmgmt_delete_collection_1	Key value is entity, ID, and class.
acmsmgmt_delete_trap_1	Key value is entity, ID, and parameter.

6.8.1. Delete Example

The following example code shows how a client program calls the `acmsmgmt_delete_collection_1` procedure to remove a collection row.

This example assumes that client initialization has been performed as described in *Section 6.4, "Initialization and Security"*.

```
int del_coll_data(int client_id,CLIENT *cl)
 {

  static int *status;
  static coll_del_rec set_struct;
  static char ent_name[MGMT_S_ENTITY_NAME];

  set_struct.client_id = client_id;
  set_struct.entity_type            = MGMT_ACC;
  strcpy(ent_name,"*");
  set_struct.entity_name            = ent_name;
  set_struct.collection_class       = MGMT_CLASS_ALL;

  status = acmsmgmt_delete_collection_1(&set_struct,cl);

  if (!status) {
     printf("\n Call to delete collection failed");
     return(MGMT_FAIL);
  }

  if (*status != MGMT_SUCCESS) {
        printf("\nCall to delete collection failed with status
 %d",*status);
        return(MGMT_FAIL);
  }
  else
        printf("\nCall to delete collection was executed");
          free(status)
        return(0);
}
```

In this example, the input record is prepared with key information, and then the call to delete the row is performed. Following the call to the delete routine, the value pointed by status is checked for success or failure. In either event, a message is printed out indicating the completion status of the call.

6.9. Add Procedures

Add procedures are available for the Collection and Trap tables. Add procedures provide the ability to add rows to the corresponding table. As shown in *Table 6.6, "Add Procedures"*, a separate add procedure is available for each of these tables.

The add procedures require an input record with an entire table row, including unique key data to be passed by the caller. The Remote Manager validates the input fields before adding the record, including checking for duplicate keys. A record is returned with an overall status code indicating the success or failure of the operation, and with individual status codes for each field indicating which fields are invalid.

All calls are synchronous.

See *Chapter 8, "Management APIs"* for details about each call.

Table 6.6. Add Procedures

Procedure	Description
acmsmgmt_add_collection_1	Key value is entity, ID, and class.
acmsmgmt_add_trap_1	Key value is entity, ID, and parameter.

6.9.1. Add Example

The following example code shows how a client program calls the `acmsmgmt_add_collection_1` procedure to add a collection row.

This example assumes client initialization has been performed as described in *Section 6.4, "Initialization and Security"*.

```
int add_collection_data(int client_id,CLIENT *cl)
 {
  static char c_name_all[2]       = "*";
  static coll_config_rec_2 set_struct;
  struct coll_status_rec_2 *status_rec;

  set_struct.client_id              = client_id;
  set_struct.coll.entity_type       = MGMT_ACC;
  set_struct.coll.entity_name       = c_name_all;
  set_struct.coll.collection_class  = MGMT_CLASS_ALL;
  set_struct.coll.collection_state  = MGMT_STATE_ENABLED;

  status_rec = acmsmgmt_add_collection_2(&set_struct,cl);

   if (!status_rec) {
      printf("\n Call to add collection record failed");
      return(MGMT_FAIL);
   }

   if (status_rec->status == MGMT_WARN) {
        printf("\nThe following fields are invalid: ");
        if (status_rec->coll_status_rec_2_u.data_warn.entity_type ==
MGMT_FAIL)
           printf("\n     entity_type");
```

```
        if (status_rec->coll_status_rec_2_u.data_warn.collection_class
            == MGMT_FAIL)
           printf("\n     collection_class");
        if (status_rec->coll_status_rec_2_u.data_warn.collection_state
            == MGMT_FAIL)
           printf("\n     coll_state");
        return(0);
   }
   else if (status_rec->status != MGMT_SUCCESS) {
        printf("\nCall to add collection failed with status",
               status_rec->coll_status_rec_2_u.rc);
          xdr_free(xdr_coll_status_rec_2,status_rec);
          free(status_rec);
        return(0);
   }
   else
        printf("\nCall to add collection was executed");
          xdr_free(xdr_coll_status_rec_2,status_rec);
          free(status_rec);
        return(1);
}
```

In this example, the input record is prepared with key and data values, and then the call to add the row is performed.

Following the call to the add routine, the return record pointer is tested to ensure that it is not NULL (that is, that the call completed). Then the overall status code (`status_rec->status`) is checked to determine whether the add was performed.

A status value of MGMT_WARN indicates that some fields were in error, so individual return codes are tested to determine which fields were invalid.

A status value other than MGMT_WARN or MGMT_SUCCESS means a general error occurred. A value of MGMT_SUCCESS means the record was added.

6.10. Start, Stop, and Replace Procedures

These three types of procedures are similar in the way they are called and in the data that is returned to them, even though they do very different operations. Start and stop procedures are used to start or stop various ACMS processes; the replace procedure is used to replace a running procedure server in an application.

An exception is the call to the `acmsmgmt_stop_1` procedure, which requests the Remote Manager to shut down. For more information about the `acmsmgmt_stop_1` procedure, see *Chapter 8, "Management APIs"*.

For the rest of the start, stop, and replace procedures, an input record, which contains key data or startup or shutdown qualifier flags, is provided by the caller; the return data contains a status code and a linked list of status messages. Status messages are generated by ACMSOPER and are returned in their entirety. Linked-list processing is illustrated in *Section 6.6.1, "Linked List Example"*.

All calls are synchronous.

See *Chapter 8, "Management APIs"* for details about each call.

Table 6.7. Start, Stop, and Replace Procedures

Procedure	Description
acmsmgmt_replace_server_1	Key is application name and server name.
acmsmgmt_start_acc_1	No keys; specify auditing, QTI, and terminal disposition.
acmsmgmt_start_exc_1	Key is application name; no startup qualifiers.
acmsmgmt_start_qti_1	No keys or qualifiers.
acmsmgmt_start_tsc_1	No keys or qualifiers.
acmsmgmt_stop_acc_1	No keys; specify cancel disposition.
acmsmgmt_stop_exc_1	Key is application name; specify cancel disposition.
acmsmgmt_stop_qti_1	No keys or qualifiers.
acmsmgmt_stop_tsc_1	No keys or qualifiers.

6.10.1. Start Example

The following example code shows how a client program calls the `acmsmgmt_start_acc_1` procedure to start ACMS on a remote node. In this example, the QTI and TSC are started along with the system, and system auditing is enabled.

This example assumes client initialization has been performed as described in *Section 6.4, "Initialization and Security"*.

```
int start_acc(int client_id,CLIENT *cl)
 {
   dcl_link    *nl;
   static acc_startup_rec start_struct;
   static cmd_output_rec *ret_struct;

   start_struct.client_id = client_id;
   start_struct.audit_sw = 1;
   start_struct.qti_sw  = 1;
   start_struct.terminals_sw  = 1;

   ret_struct = acmsmgmt_start_acc_1(&start_struct,cl);

   if (!ret_struct) {
        printf ("\n Call to start ACMS system failed");
        return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

        if (ret_struct->status != MGMT_WARN) {
            printf("\nCall to start ACMS system failed with status %d",
                   ret_struct->status);
              xdr_free(xdr_cmd_output_rec, ret_struct);
              free(ret_struct);
            return(0);
        }
```

```
        printf("\n Call to start ACMS system completed with warnings or
               errors");

        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
            printf("\n %s",nl->dcl_msg);
              xdr_free(xdr_cmd_output_rec, ret_struct);
              free(ret_struct);
        return(MGMT_FAIL);
   }
   else {
        printf("\nCall to start ACMS system was executed");
        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
            printf("\n %s",nl->dcl_msg);
   }
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
  return(0);
}
```

In this example, the input record is prepared with qualifier data, and then the call to start the system is performed. Auditing is enabled, and QTI and TSC will be started with the system.

The return value from the calls to the start, stop (except `acmsmgmt_stop_1`), and replace procedures is a pointer to a union. If the pointer returned is NULL, the call has failed. RPC error checking must be used to determine the cause of the error. If a valid pointer is returned, it points to a structure containing a union with the following structure:

```
union cmd_output_rec switch (int status) {
    case MGMT_WARN:
        cmd_rec data_warn;
    case MGMT_SUCCESS:
        cmd_rec data;
    case MGMT_FAIL:
        int rc;
    default:
        void;
        };
```

The status field determines which structure is being returned. If the status is equal to MGMT_FAIL, the `rc` field is returned. The rc field contains a status code indicating the reason for failure.

If the status field is not equal to MGMT_WARN or MGMT_SUCCESS, a pointer to a linked list has been returned. The linked list contains a text field and a forward pointer. By following the forward pointers, all the records in the list can be retrieved. *Section 6.6.1, "Linked List Example"* illustrates how to follow the linked list.

In either case, the example code prints out the contents of all the strings in the linked list. These strings are status messages returned by ACMSOPER.

Chapter 7. Management Programming Using SNMP

Programmers who want to access and maintain the ACMS Remote Manager from their own programs can use the following two interfaces:

- Open Network Computing (ONC) Remote Procedure Call (RPC)

  The ONC RPC interface is for system managers and system programmers who want to write custom tools and applications that access the ACMS Remote Manager. For more information, see *Chapter 6, "Management Programming Using ONC RPC"*.

- Simple Network Management Protocol (SNMP)

  The SNMP interface is provided for integration with enterprise management packages.

This chapter discusses the SNMP interface. Programmers who are familiar with SNMP console programming can use this information when writing routines that interact with the ACMS Remote Manager using the SNMP protocol. The information in this chapter is also useful for programmers who are integrating the ACMS Remote Manager with other enterprise management packages through the SNMP protocol.

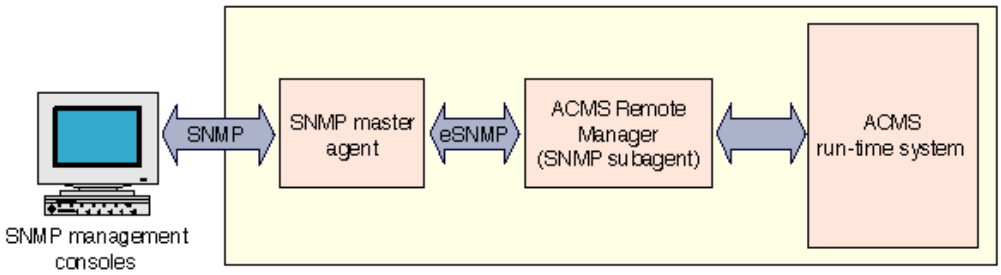
The ACMS Remote Manager implements the management information base (MIB) for SNMP. To access ACMS MIB information through SNMP, you must have an SNMP-enabled console or you can use an SNMP MIB browser such as the one provided by VSI TCP/IP Services for OpenVMS, which includes the TCPIP$SNMP_REQUEST.EXE utility.

Alternatively, you can write your own SNMP interface. For more information about programming SNMP, refer to *Windows NT SNMP* by James D. Murray, published by O'Reilly & Associates, Inc., Sebastopol, CA.

7.1. SNMP Overview

The ACMS Remote Manager implements a MIB for ACMS. When the SNMP interface is enabled, either during or after Remote Manager process startup, it registers the ACMS subtree with the local SNMP master agent. SNMP console requests go first to the SNMP master agent (provided by the installed TCP/IP software, such as VSI TCP/IP Services for OpenVMS), which in turn delivers them to the ACMS Remote Manager. *Figure 7.1, "SNMP Program Interface with Remote Manager"* illustrates the SNMP interface with the ACMS Remote Manager.

Figure 7.1. SNMP Program Interface with Remote Manager



Communications between the SNMP interface and the master agent use the eSNMP protocol. This protocol is transparent to SNMP consoles.

The ACMS Remote Manager provides management information to SNMP management platforms in response to snmp_get and snmp_getnext messages. Management platforms can modify many management data elements by sending the appropriate snmp_set message. If any traps have been configured, the ACMS Remote Manager will generate SNMP traps when the Remote Manager detects a trap condition (for example, when an ACMS process starts or stops).

All Management table fields are available to SNMP management applications through get operations, but not all fields can be set. In general, the fields that can be set are Configuration class fields (in ACMS entity tables) and nearly all Manager configuration table fields. See *Chapter 9, "Remote Manager Reference Tables"* for a list of all tables and fields.

7.2. SNMP Security

Security for the SNMP interface is enforced first by the SNMP master agent (not the ACMS MIB). SNMP supports the concept of communities, which are essentially node inclusion lists. Whoever installs and configures the SNMP software package (typically the network manager) sets up SNMP communities. Nodes that are part of the SNMP community to which the subagent belongs can connect to the master agent; any node that can connect to the master agent can connect and interact with the subagent. All SNMP communities are allowed any combination of read, write, and trap access. Nodes that are not part of the community do not have access to the master agent.

Note that communities work at the node level only. It is not possible to restrict the access of individual user accounts on the node, although it may be possible to restrict access to the SNMP console software on a per-user basis. Note also that node authentication itself is relatively weak and provides no safeguards against masquerades or other forms of network deception.

As a second level of security, the ACMS Remote Manager requires that a special OpenVMS account (ACMS$SNMP) be created for the SNMP interface on nodes on which the Remote Manager runs. The account must be granted OpenVMS rights for read, write, or operate access (or some combination of these) to Remote Manager data and functions. This allows ACMS system managers to grant read access, for instance, through the SNMP interface, but to prevent write or operate access. See *Section 4.4, "Logging In to the Remote Manager"* for a discussion of how to configure Remote Manager authentication and authorization for the SNMP interface.

7.3. Initializing the SNMP Interface

In order for SNMP consoles to communicate with the ACMS Remote Manager through SNMP, the Remote Manager SNMP interface must have been started. The SNMP interface runs as a separate thread in the Remote Manager and can be started or stopped at any time without restarting the Remote Manager.

The SNMP interface is started using the **SET INTERFACE** command. The current state of the interface can be determined using the **SHOW INTERFACE** command. Refer to *Section 4.5, "Starting and Stopping Interfaces"* for more information about using ACMSCFG and ACMSMGR to start and stop interfaces.

During startup, the SNMP interface first performs some housekeeping tasks and then attempts to register with the SNMP master agent.

In order for the SNMP interface to initialize successfully, the following conditions must be met:

- The ACMS$SNMP account on the Remote Manager node must exist.
- The ACMS$MGMT_READ, ACMS$MGMT_WRITE, and ACMS$MGMT_OPER rights identifiers must exist. At least one of these identifiers must be granted to the ACMS$SNMP account.
- The SNMP master agent must be running on the Remote Manager node.

If any of the initialization tasks fail, or if registration fails, the SNMP interface writes error messages to the Remote Manager log and the thread exits. In this case, users should check the Remote Manager log for messages, correct the problem, and restart the interface.

During initialization, the Remote Manager establishes a timeout that the master agent will use when communicating with it. The timeout is based on the value of the Remote Manager parameter SNMP_AGENT_TIME_OUT.

If initialization is successful, the SNMP interface thread waits for incoming SNMP requests. The wait times out periodically (based on the Remote Manager parameters SNMP_SEL_TIME_OUT and SNMP_ARE_YOU_THERE stored in the Parameter table), and checks to make sure the SNMP master agent is still running by sending an "are you there" message to the master agent. If the master agent responds, the Remote Manager continues to wait for incoming messages. If the master agent does not respond, the SNMP interface thread attempts to restart the connection. If the restart fails, the SNMP thread exits.

7.4. SNMP Tables

The tables in *Chapter 9, "Remote Manager Reference Tables"* and the tables defined in the ACMS MIB map to each other on a one-to-one basis. However, data types are slightly different between SNMP and RPC, most significantly in the use of the gauge structure type. *Section 7.4.1, "Data Type Mapping"* describes data type mapping.

When accessing any of the ACMS MIB tables, it is important to keep in mind the dynamic nature of the ACMS run-time system. ACMS entities may be stopped and restarted; collection states for the entities may change dynamically; new processes (especially EXC and CPs) may be created. It is also important to understand that the size of some ACMS MIB tables may change when either the ACMS run-time system is restarted, or even as certain processes are started and stopped.

If the proper access strategies are not used when getting or setting ACMS MIB data, unpredictable and erroneous results can occur.

Different access strategies must be used for different types of tables. In the ACMS MIB, there are three types of tables. Specific access strategies for each table type are discussed in separate sections, as follows:

- Single-row tables (see *Section 7.4.2, "Single-Row Tables"*)
- Static tables (see *Section 7.4.3, "Static Tables"*)
- Dynamic tables (see *Section 7.4.4, "Dynamic Tables"*)

Also refer to *Section 7.4.5, "Servers and Task Groups"* for a discussion of how the Server and Task Group tables are indexed.

Regardless of the type of table, identity and state validation should be performed for all ACMS entity tables (ACC, TSC, CP, QTI, EXC, server, task group).

Identity validation is performed by storing the PID field of the process occupying the row the first time the row is accessed. Then, when revisiting the table, get the PID along with the data values. Then check that the PID has not changed. If it has, the data refers to a new process.

Note that the process name is not a good means of identifying a process, because process names can be reused between entity executions.

Also note PID is not an ID class field for servers and task groups. For these two entity types, the EXC PID should be used.

State validation is performed by checking the collection state for the class that contains the field. For instance, if the `exc-current-waiting-tasks-num` (in the EXC run-time class) is being monitored, ensure that the `exc-rt-coll-state` is enabled (equal to 1). Otherwise, the value in that field is no longer being updated by the EXC, and is no longer accurate.

7.4.1. Data Type Mapping

The ACMS Remote Manager implements three data types:

- Integer
- String
- Gauge

The *integer* and *string* data types map directly to the SNMP INTEGER and DisplayString data types.

The *gauge* data type defined for the Remote Manager is not the same as the SNMP Gauge type. In order to avoid confusion, the Remote Manager SNMP interface maps the Remote Manager gauge fields to SNMP INTEGER and DisplayString data types. So for each Remote Manager *gauge* data type, three fields are defined in the MIB: the current field value, the maximum (or minimum) field value, and the maximum (or minimum) field value time.

For example, consider the ACC run-time field current_appls. This is defined as a Remote Manager *gauge* data type in *Section 9.2, "ACC Table"*. In the MIB, three fields are defined:

```
acc-current-appls-num            INTEGER,
acc-current-appls-max            INTEGER,
acc-current-appls-time           DisplayString
```

This is the case for all Remote Manager *gauge* data types. For Remote Manager *min gauge* data types, there is a `-min` field instead of a `-max` field. For both *gauge* data types, time is expressed in the form *DD-MMM-YYYY HH:MM:SS.hh*.

7.4.2. Single-Row Tables

Access to single-row tables is straightforward, because only a single row is ever accessed. The following are single-row tables:

- ACC table
- QTI table
- TSC table

- Parameter table
- Remote Manager table

Bounds checking need not be performed. However, for Entity tables (ACC, QTI, TSC), both identity and state validation must be performed.

7.4.3. Static Tables

Static tables are sized when the parent process starts and do not change as long as the parent process is running. For each static table, there is a field in the table of the parent process that indicates the upper bound of the static table.

Table 7.1, "Static Tables" shows the static tables, their parent process, and the field that indicates the upper bound of the table.

Table 7.1. Static Tables

Table	Parent Process	Upper Bound (field and table)
CP	TSC	`tsc-cp-slots-active` in the TSC table
EXC	ACC	`acc-max-appl-active` in the ACC table
Server	EXC	`exc-server-types` in the EXC table
Task Group	EXC	`exc-task-groups` in the EXC table
Interfaces	Remote Manager	`rmIfCt` in the Remote Manager table
Collection	Remote Manager	`totl-entity-slots` in the Parameter table

In static tables, table data is not always contiguous and table rows can be reused. The PID field should be used to establish process identity.

For example, consider the following CP table. Assume that the first CP is permanent, and the second two are not.

```
Table Row   CP process name   CP PID
1           ACMS01CP001000    2040013D
2           ACMS01CP002000    2040013E
3           ACMS01CP003000    2040013F
```

An SNMP console searching this table sequentially would find all three CP instances; access to table row 4 would return an error. However, if the users attached to the CP in table row 2 log out, the CP terminates and the table now looks like this:

```
Table Row   CP process name   CP PID
1           ACMS01CP001000    2040013D
2
3           ACMS01CP003000    2040013F
```

An SNMP console searching this table sequentially and stopping when the first error is returned would find only the first CP. Access to the second row would return an error. Therefore, when scanning static tables, it is important to examine all rows of the table before terminating the scan; that is, perform a loop based on the `tsc-cp-slots-active` field in the TSC table.

Finally, consider what happens if a new CP now starts. The table would look as follows:

```
Table Row   CP process name   CP PID
1           ACMS01CP001000    2040013D
2           ACMS01CP002000    20400140
3           ACMS01CP003000    2040013F
```

Table row 2 is now valid again, but a different process occupies it. Therefore, any cached information for table row 2 is invalid and must be refreshed with the data from the new process.

7.4.4. Dynamic Tables

Dynamic tables do not have a fixed upper bound; they grow and shrink as entries are added and removed. However, data in dynamic tables is always contiguous, so there are never invalid rows stored between valid rows. When a row becomes invalid because it is empty or unoccupied, it is removed from the table and the remaining rows are renumbered.

The following are dynamic tables:

- User table
- Log table
- Trap table

To see how a dynamic table changes when a table row is removed, assume that a user table has the following contents:

```
Table row   User Name   Client Id
1           User1       1
2           User2       2
3           User3       3
4           User4       4
```

If User2 logs out, the contents of the table would change as follows:

```
Table row   User Name   Client Id
1           User1       1
2           User3       3
3           User4       4
```

As with static tables, you must ensure that the table row being accessed has not been reused or renumbered. Among dynamic tables, only the Trap table allows updates. Note that entries are never deleted or modified in the Log table; new entries are always appended to the end.

7.4.5. Servers and Task Groups

The Servers and Task Group tables are indexed by a compound index. For both tables, the first key value is the table row of the owning EXC; the second key value is the Server or Task Group row. When fetching or setting Server or Task Group rows, you must first determine the EXC (application) they belong to, and then determine the particular server or task group.

For example, assume the EXC table has a total of four rows. Application Appl1 occupies row 1, and has two servers (ServerA and ServerB) and one task group (TaskGroupA). Application Appl2 occupies row 3 and has two servers (ServerC and ServerD) and two task groups (TaskGroupB, TaskGroupC). EXC table rows 2 and 4 are unused. *Table 7.2, "EXC Table (OID 1.3.6.1.4.1.36.2.18.48.13)"* and *Table 7.3, "Server Table (OID 1.3.6.1.4.1.36.2.18.48.13)"* list the contents of each table.

Table 7.2. EXC Table (OID 1.3.6.1.4.1.36.2.18.48.13)

Row	Contents
1	Appl1
2	(unused)
3	Appl2
4	(unused)

Table 7.3. Server Table (OID 1.3.6.1.4.1.36.2.18.48.13)

Key 1 (EXC)	Key 2 (Server)	Contents
1	1	ServerA
1	2	ServerB
3	1	ServerC
3	2	ServerD

Table 7.4. Task Group Table (OID 1.3.6.1.4.1.36.2.18.48.13)

Key 1 (EXC)	Key 2 (Task Group)	Contents
1	1	TaskGroupA
3	1	TaskGroupB
3	2	TaskGroupC

In order to access the ser-server-name field for ServerA in application Appl1, the OID would be 1.3.6.1.4.1.36.2.18.48.14.1.3.1.1 To access the same field for ServerD in Appl2, the OID would be 1.3.6.1.4.1.36.2.18.48.14.1.3.3.2.

There is no way to determine to which task group a server belongs. In contrast, you can always determine from the OID which application a server belongs to because ACMS requires that each server be given a unique name within the application.

7.5. SNMP GET Operations

SNMP get requests are satisfied at the time they are received by the subagent. Get requests can take one of three forms: get, get next, and get bulk.

- Get operations are simple requests for single data items.
- Get next requests are iterative requests for logically sequential information.
- Get bulk requests obtain a logical sequence of information in a single request.

The SNMP subagent for ACMS supports only the first operation. SNMP "walks", if performed, return unpredictable results.

OIDs for ACMS Management tables are documented in the file MIB_OID.LIS available from the directory ACMS$RM_EXAMPLES. The MIB definition for the ACMS subtree is also provided in ACMS$RM_EXAMPLES, in the file MGMTMIB.MY.

In order for SNMP get requests to complete successfully, the following conditions must be met:

- The ACMS$SNMP account on the Remote Manager node must be granted the ACMS$MGMT_READ identifier.
- The SNMP interface must already be started.
- The ACMS run-time system must already be started (to access ACMS entity information).

General eSNMP return codes are returned from the Remote Manager for SNMP get requests (see *Section 7.10, "Remote Manager eSNMP Return Codes"*). For details about a specific error, refer to the Remote Manager log.

7.6. SNMP SET Operations

SNMP set requests are executed at the time they are received by the subagent and are applied to the running system. However, not all fields that can be set are dynamic; the actual implementation of modification may not occur until the affected entities are restarted.

For more discussion about updates that modify the ACMS run-time system, see *Section 5.3, "Managing ACMS Using the Remote Manager"*.

OIDs for ACMS Management tables are documented in the file MIB_OID.LIS available from the directory ACMS$RM_EXAMPLES. The MIB definition for the ACMS subtree is provided in the ACMS$RM_EXAMPLES directory in file MGMTMIB.MY.

In order for SNMP set requests to complete successfully, the following conditions must be met:

- The ACMS$SNMP account on the Remote Manager node must be granted the ACMS$MGMT_WRITE or ACMS$MGMT_OPER identifier, depending on the operation being performed (see *Appendix B, "RPC Procedures and Corresponding Rights Identifiers"*).
- The SNMP interface must already be started.
- The ACMS run-time system must already be started (to update ACMS entity information).

General eSNMP return codes for SNMP get requests are returned from the Remote Manager (see *Section 7.10, "Remote Manager eSNMP Return Codes"*). For details about a specific error, refer to the Remote Manager log.

7.7. Using SNMP to Start and Stop ACMS Entities

To start and stop ACMS entities (or to stop the Remote Manager), the Remote Manager allows SNMP users to modify the ID class field `running_state`. In general, ID class fields are read only. However, since SNMP does not support a **`START`** or **`STOP`** command, the **`SET`** command must be used.

Modifications to the `running_state` fields are not performed directly by the Remote Manager. Instead, the Remote Manager uses ACMSOPER commands to request the shutdown or startup of the ACMS entity. The ACMS entities update the `running_state` field when they start or stop.

For instance, to start the ACMS run-time system, an SNMP console program issues an SNMP **`SET`** command for the ACC `running_state` OID, specifying the value "started". The Remote Manager

interprets this message as an attempt to start the system and issues the appropriate ACMSOPER command.

The SNMP set call is synchronous. That is, it does not complete until the ACMS operation has completed.

Failure messages related to start or stop requests are written to the Remote Manager log.

7.8. SNMP Traps

SNMP traps provide a means of automatically notifying the system support team when a warning or error condition exists. Users configure SNMP traps in the SNMP trap table; however, note that traps are not generated unless a corresponding entry exists in the entity/collection table.

At runtime, SNMP traps can be generated either as the result of an error event occurring (i.e. a new entry being made in the Class 6 table), or if a monitored parameter exceeds (or falls below) a user defined threshold. The SNMP interface will monitor both sources of data, and will generate traps accordingly. Note that SNMP traps themselves will result in error records being written, but are not themselves trappable.

Error events are inspected at the time they are detected by the error collection routines to determine if a trap should be generated or not. Thresholds are checked on a user defined interval to determine if a trap should be generated or not. For both sources of traps, a gating factor is used to prevent a persistent error from flooding the system. After *n* number of consecutive alarms (where *n* is the gating factor), the monitoring interval is continuously doubled, until the error condition does not re-occur. As a matter of implementation, monitoring of the value is not suspended when the interval is doubled. Instead, traps are discarded for the duration of the inflated monitoring interval.

It is possible to have SNMP traps generated for the following events: SNMP trap table; when a matching condition or event occurs, an SNMP trap is generated. SNMP management consoles listen for SNMP traps and then respond in a console-dependent (and usually user-configurable) manner.

See *Section 9.14, "Trap Table"* for a discussion of the Trap table and the format of trap messages.

At run time, SNMP traps can be generated as the result of either an ACMS process starting or stopping, or an event that occurred within the Remote Manager (for example, a failure in communications with ACMS).

ACMS system managers configure traps by modifying the Trap table, either by using the ACMSCFG utility prior to Remote Manager startup or by using the ACMSMGR utility after the Remote Manager has been started. Changes made using ACMSCFG do not affect the running system until the Remote Manager is restarted; changes made using ACMSMGR are not saved when the Remote Manager stops.

The configuration process is the same with either utility. You use the **ADD TRAP** command to add new traps, use the **DELETE TRAP** command to remove traps, and use the **SET TRAP** command to modify traps.

Keep in mind that although you can add, delete, or modify entries in the trap table at almost any time, traps will not be generated unless the SNMP interface is started. In addition, traps are not queued if the SNMP interface is disabled.

The combination of entity, name, and parameter uniquely identify a trap in the Trap table. For each trap, a minimum and a maximum value can be specified, along with a severity. Minimum and maximum trap

values specify thresholds that trigger traps when the associated parameter is either greater than or less than the threshold. Minimum and maximum trap values are parameter-specific.

A special value of -1 is used as a placeholder when creating a trap for which a minimum or maximum does not apply. In many situations, only the minimum or maximum value setting is meaningful. In this instance, set the desired field (minimum or maximum) to the threshold value, and set the other to -1.

Two trap parameters are supported:

- EXISTS (see *Section 7.8.1, "EXISTS Traps"*)
- EVENT_SEVERITY (see *Section 7.8.2, "EVENT_SEVERITY Traps"*)

7.8.1. EXISTS Traps

The trap parameter EXISTS allows traps to be generated based on whether an ACMS process starts or stops.

Specifying a minimum trap value of 1 for a process specifies, in effect, that a trap should be generated whenever the process stops – that is, when the process existence is less than 1.

Specifying a maximum value of 0 specifies that a trap should be generated whenever the process starts – that is, when the processes existence is greater than 0.

A minimum value of 0 or a maximum value of 1, while valid, is basically useless, since the EXISTS parameter is never greater than 1 or less than 0.

7.8.2. EVENT_SEVERITY Traps

The trap parameter EVENT_SEVERITY allows traps to be generated based on the facility and severity of events being logged to the Remote Manager log. For example, an EVENT_SEVERITY trap can be configured for Remote Manager SNMP events with severity higher than WARNING (such as ERROR or FATAL). Any time a Remote Manager SNMP operation fails with a severity higher than WARNING, an SNMP trap is generated.

Other facilities that can be monitored are:

- * (all)
- MGR (Remote Manager main process)
- PROCMON (process monitor thread)
- RPC (RPC interface thread)
- SNMP (SNMP interface thread)
- SEC (security routines)
- LOG (event logging thread)
- TIMER (internal timer thread)
- DCL (DCL subprocess management thread)

- MSG_PROC (processes incoming ACMS errors)
- TRAP (trap sender thread)

Use care when you configure traps so that you do not create unnecessary traps. In general, traps are intended to be used to signal significant events. For instance, specifying a minimum severity of FATAL or ERROR causes all informational and warning messages to generate traps. This is probably not a good use of network or console resources.

7.9. SNMP Debug Tracing

In addition to the normal logging the Remote Manager performs, it is possible to enable debug-level SNMP tracing. This level of tracing is performed by the eSNMP TCP/IP code layer and may not be available for all TCP/IP products. The VSI TCP/IP Services for OpenVMS product supports debug-level SNMP tracing. If you use a third-party TCP/IP product, check with that vendor regarding support for this option.

Debug-level tracing of the Remote Manager SNMP interface can be valuable for developing SNMP console applications or for trying to debug a particular SNMP environmental problem. However, it is relatively resource intensive and should be performed in a controlled environment for short durations.

To enable debug-level SNMP tracing, the Remote Manager must be started with the command line argument LOG_TO_SYSOUT, as follows:

```
$ @SYS$STARTUP:ACMS$MGMT_STARTUP LOG_TO_SYSOUT
```

The SNMP_AUDIT_LEVEL parameter must be greater than 0. When the SNMP interface is started, it will enable debug-level tracing in the eSNMP code layer. All output is directed to SYS$OUTPUT for the Remote Manager process, which is redirected by the startup command procedure to SYS$ERRRORLOG:ACMS$MGMT_SERVER.OUT.

7.9.1. Starting SNMP Debug Tracing

To start the Remote Manager with debug-level SNMP tracing, run the startup command procedure SYS$STARTUP:ACMS$MGMT_STARTUP, specifying LOG_TO_SYSOUT as the only parameter to the command procedure, as follows:

```
$  @SYS$STARTUP:ACMS$MGMT_STARTUP LOG_TO_SYSOUT
```

Once the Remote Manager has been started and the SNMP interface has been enabled, make sure that the SNMP_AUDIT_LEVEL parameter is greater than 0. To do this, use the following ACMSMGR command:

```
$  ACMSMGR SET PARAM/SNMP_AUDIT_LEVEL=F
```

The SNMP debug output is written to SYS$ERRORLOG:ACMS$MGMT_SERVER.OUT, which is an ASCII file that can be typed or edited.

7.9.2. Stopping SNMP Debug Tracing

To stop debug-level SNMP tracing, either restart the Remote Manager (without the LOG_TO_SYSOUT parameter), or use the following command to set the SNMP_AUDIT_LEVEL parameter to 0:

```
$  ACMSMGR SET PARAM/SNMP_AUDIT_LEVEL=0
```

7.10. Remote Manager eSNMP Return Codes

Table 7.5, "Remote Manager eSNMP Routines Return Codes" describes the return codes returned by the Remote Manager eSNMP routines.

Table 7.5. Remote Manager eSNMP Routines Return Codes

Return Code	Description
ESNMP_MTHD_commitFailed	An attempt to apply an update failed. This is also returned from a start or stop attempt that fails. Refer to the Remote Manager log for details.
ESNMP_MTHD_genErr	An internal error occurred. This could be due to security violations, a failure updating a particular field, or an internal processing error. Refer to the Remote Manager log for details.
ESNMP_MTHD_noCreation	The table does not allow new rows to be created. The OID specified for the set operation indicates a table row that does not exist, and the table does not allow new rows to be created.
ESNMP_MTHD_noError	The set operation was successful.
ESNMP_MTHD_noSuchInstance	A request was made for a variable that does not exist. Either the OID is invalid, or the particular table row does not exist (is out of bounds).
ESNMP_MTHD_noSuchObject	The column specified does not exist.
ESNMP_MTHD_notWritable	An attempt was made to set a variable that is read only.
ESNMP_MTHD_resourceUnavailable	The table row exists (is within the bounds of the table) but is currently unused (empty).
ESNMP_MTHD_wrongValue	An attempt was made to update a field with an invalid value.

Part II. Reference Information

Part II, "Reference Information" contains reference information for the ACMS Remote Manager.

Chapter 8. Management APIs

The Management APIs are intended to be called from Open Network Computing (ONC) Remote Procedure Call (RPC) clients. ONC RPC Interface Definition Language (IDL) for all procedures is contained in the file ACMS$RM_EXAMPLES:ACMSMGMT_RPC.X.

Programmers who write client programs are strongly urged to become familiar with the contents of this file. Many programming questions can be answered by looking at the actual RPC definitions. All structure definitions, for example, are contained within this file.

The procedures documented in this chapter are based on the most current record and field data available with Version 4.4 of the Remote Manager. Procedures names with `_2` indicate new or modified API functions that take advantage of this new data. These procedures should only be called from a Remote Manager Version 4.4 server system. To ensure backwards compatibility, the `_1` version of all calls is still available and provided with the Remote Manager Version 4.4 software.

Note

The ACMS$MGMT_GET_CREDS procedure is not included in the ACMSMGMT_RPC.X IDL because it is not a remote procedure call. It is a statically linked, locally executed function for those clients performing explicit authentication. The ACMS$MGMT_GET_CREDENTIALS.OBJ object module is located in the ACMS$RM_EXAMPLES directory.

The ACMS$MGMT_GET_CREDS procedure is for use by ONC RPC clients only.

8.1. Common RPC Fields

The tables in this section list commonly used fields and their values.

8.1.1. Collection Classes

Table 8.1, "Collection Classes" shows the symbolic names for Remote Manager collection classes.

Table 8.1. Collection Classes

Symbolic Name	Description
MGMT_CLASS_ALL	All classes
MGMT_CLASS_CFG	Config class
MGMT_CLASS_ID	ID class
MGMT_CLASS_POOL	Pool class
MGMT_CLASS_RT	Runtime class

8.1.2. Interface Types

Table 8.2, "Interface Types" shows the symbolic names for Remote Manager interfaces.

Table 8.2. Interface Types

Symbolic Name	Description
MGMT_IF_RPC	Remote Procedure Call (RPC) interface

Symbolic Name	Description
MGMT_IF_SNMP	Simple Network Management Protocol (SNMP) interface

8.1.3. Enable States

Table 8.3, "Enable States" shows the symbolic names for Remote Manager enable states.

Table 8.3. Enable States

Symbolic Name	Description
MGMT_STATE_DISABLED	Disabled
MGMT_STATE_ENABLED	Enabled

8.1.4. Entity Types

Table 8.4, "Entity Types" shows the symbolic names for Remote Manager entity types.

Table 8.4. Entity Types

Symbolic Name	Description
MGMT_ACC	Application Central Controller (ACC) process
MGMT_AGENT	User-written agents
MGMT_ALL	All entities
MGMT_CP	Command Process (CP) process
MGMT_EXC	Application Execution Controller (EXC) process
MGMT_MGR	Remote Manager process
MGMT_QTI	Queued Task Initiator (QTI) process
MGMT_SER	Procedure server types
MGMT_TG	Task groups
MGMT_TSC	Terminal Subsystem Controller (TSC) process
MGMT_UNSUPPORTED	Null value

8.1.5. Facility Types

Table 8.5, "Facility Types" shows the symbolic names for Remote Manager facility types.

Table 8.5. Facility Types

Symbolic Name	Description
MGMT_FAC_ALL	Any facility type.
MGMT_FAC_DCL	A thread that manages a spawned DCL process. The DCL process is used to execute ACMSOPER commands.

Symbolic Name	Description
MGMT_FAC_LOG	The event log writer thread.
MGMT_FAC_MGR	The mainline Remote Manager process.
MGMT_FAC_MSGPROC	A thread that handles messages coming in from ACMS processes.
MGMT_FAC_PROCMON	A thread dedicated to monitoring processes.
MGMT_FAC_RPC	The RPC interface thread (listener and procedures).
MGMT_FAC_SEC	Security routines in the Remote Manager.
MGMT_FAC_SNAP	A thread dedicated to performing data snapshots.
MGMT_FAC_SNMP	The SNMP interface thread (message loop and procedures).
MGMT_FAC_TIMER	A thread that controls timers for the Remote Manager.
MGMT_FAC_TRAP	A thread that sends out SNMP traps.

8.1.6. Running States

Table 8.6, "Running States" shows the symbolic names for Remote Manager running states.

Table 8.6. Running States

Symbolic Name	Description
MGMT_STATE_INITED	Process or object has initialized.
MGMT_STATE_INITING	Process or object is initializing.
MGMT_STATE_LOAD_DONE	Process or object has finished loading.
MGMT_STATE_LOADING	Process or object is loading itself.
MGMT_STATE_STARTED	Process or object has started and is ready to run.
MGMT_STATE_STARTING	Process or object is starting the mainline.
MGMT_STATE_STOPPED	Process or object is stopped.

8.1.7. Severity Codes

Table 8.7, "Severity Codes" shows the symbolic names for Remote Manager severities.

Severities are generally reported as simple severities (informational, warning, error, fatal) but may be combined by logically ORing the values when used as selection criteria (such as for selecting log records).

Table 8.7. Severity Codes

Symbolic Name	Description
MGMT_SEV_ERR	Error

Symbolic Name	Description
MGMT_SEV_FATAL	Fatal
MGMT_SEV_INFO	Informational
MGMT_SEV_NONE	Null value
MGMT_SEV_WARN	Warning

8.1.8. Trap Parameters

The table below shows the symbolic names for Remote Manager trap parameters.

Table 8.8. Trap Parameters

Symbolic Name	Description
MGMT_EXISTS	Existence traps
MGMT_SEVERITY	Remote Manager severity traps

8.2. Thread-Safe and Non-Thread Safe Clients

Each of the procedures documented in this chapter (and those in ACMS$MGMT_EXAMPLES.C) are designed to use the thread-safe client stub provided with the Remote Manager, as described in the file ACMS$MGMT_EXAMPLES_BUILD.COM. As a result, each procedure contains one or more "free" calls that prevent memory leaks in multithreaded client implementations.

If you intend to build a multithreaded client, you must modify any existing, customized API functions to include these calls, then recompile them along with the thread-safe client stub.

If you want to implement a non-thread safe client using the RPC-generated stub, omit the "free" calls. See ACMS$MGMT_EXAMPLES_BUILD.COM for detailed build instructions.

8.3. ACMSMGMT_ADD_COLLECTION_2

ACMSMGMT_ADD_COLLECTION_2

ACMSMGMT_ADD_COLLECTION_2 — This procedure adds entries to the Remote Manager Collection table. Collection table entries can also be modified and deleted.

Format

```
coll_status_rec_2 *acmsmgmt_add_collection_2(coll_config_rec_2
 *set_struct,CLIENT *cl)
```

Parameters

set struct

Type	Coll_config_rec_2
Access	Read

Mechanism	By reference	
Usage	Structure that contains the following client identification and Collection table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	coll	
	Type	Coll_update_rec_r_2
	Access	Read
	Mechanism	By value
	Usage	Structure containing a Collection table record. Collection table fields are described in *Section 9.4, "Collection Table"*. See the Description section for information on how to initialize this record.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Coll_status_rec_2	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting of either a failure code or a structure of type `coll_update_rec_r_2`, which contains status codes for each field. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write

	Mechanism	By value
	Usage	Failure return code.
	data_warn	
	Type	Coll_output_rec_r_2
	Access	Write
	Mechanism	By value
	Usage	Structure containing a Collection table record. The entries in this field contain status codes corresponding to the fields in the `coll` structure. See the Description section for a discussion of how to determine the update status for any field.

Description

This procedure adds a row to the Collection table (see *Section 9.4, "Collection Table"*).

Additions to this table are not durable; that is, they do not survive a restart of the Remote Manager. To make nondynamic, permanent updates to the Collection table, use the ACMSCFG utility.

Calls to this procedure must specify `entity_type`, `entity_name`, and `collection_class`. The combination of these fields must be unique within the collection table for the row to be added. The tables above contain symbolic values used to populate the `entity_type` and `collection_class` fields; `entity_name` is specified as a null-terminated string.

ID and Config class rows cannot be added. By default, these classes are always enabled for all ACMS processes.

The Collection table contains a fixed number of rows, which is determined by the Remote Manager parameter `total_entity_slots`. This is a nondynamic parameter and requires a restart of the ACMS system in order to be changed. The default is 20 rows.

Additions to the Collection table are processed immediately and may affect more than one ACMS process.

Example

```
int add_collection_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[] = "*";
   static coll_config_rec_2    set_struct;
   struct coll_status_rec_2    *status_rec;

   set_struct.client_id              = client_id;
   set_struct.coll.entity_type       = MGMT_ALL;
   set_struct.coll.entity_name       = c_name_all;
   set_struct.coll.collection_class  = MGMT_CLASS_RT;
   set_struct.coll.collection_state  = MGMT_STATE_DISABLED;
```

```
   status_rec = acmsmgmt_add_collection_2(&set_struct,cl);

   if (!status_rec) {
      printf("\n Call to add collection failed");
      return(MGMT_FAIL);
   }

   if (status_rec->status == MGMT_WARN) {
      printf("\nThe following updates failed: ");
      if (status_rec->coll_status_rec_2_u.data_warn.entity_type  ==
MGMT_FAIL)
         printf("\n     entity type invalid");
      if (status_rec->coll_status_rec_2_u.data_warn.collection_state
          == MGMT_FAIL)
         printf("\n     coll_state invalid");
      if (status_rec->coll_status_rec_2_u.data_warn.storage_state ==
MGMT_FAIL)
         printf("\n     storage_state invalid");
      if (status_rec->coll_status_rec_2_u.data_warn.storage_interval
          == MGMT_FAIL)
         printf("\n     storage_interval invalid");
     }
   else if (status_rec->status != MGMT_SUCCESS) {
      printf("\nCall to add collection with status %d",
             status_rec->coll_status_rec_2_u.rc);
        xdr_free(xdr_coll_status_rec_2, status_rec);
        free(status_rec);
      return(MGMT_FAIL);
   }
   else
      printf("\nCall to add collection was executed");
        xdr_free(xdr_coll_status_rec_2, status_rec);
        free(status_rec);
    return(0);
}
```

In the preceding example, the ACMSMGMT_ADD_COLLECTION_2 procedure is called to add a row to the Collection table. The row added is for all entity types (*), all entity names (*), and the RUNTIME collection class. The collection state is set to DISABLED. If the call succeeds, a Collection table row is added, and the RUNTIME collection state for some processes may be disabled. Otherwise, an error message is displayed.

8.4. ACMSMGMT_ADD_ERR_FILTER_2

ACMSMGMT_ADD_ERR_FILTER_2

ACMSMGMT_ADD_ERR_FILTER_2 — This procedure adds entries to the ACMS Error Filter table. Error Filter table entries can also be deleted.

Format

```
error_filter_config_rec_r_2
 *acmsmgmt_add_err_filter_2(err_filter_config_rec_r_2
 *err_filter_cfg_rec,CLIENT *cl2)
```

Parameters

err_filter_cfg_rec

Type	Err_filter_config_rec_r_2	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Error Filter table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value of `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	error_code	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Structure containing an Error Filter table record. Error Filter table fields are described in *Section 9.4, "Collection Table"*. See the Description section for information on how to initialize this record.

cl2

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Err_filter_status_rec

Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting of either a failure code or a structure of type `err_filter_update_rec_r`, which contains status codes for each field. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data_warn	
	Type	Err_filter_update_rec_r
	Access	Write
	Mechanism	By value
	Usage	Structure containing an Error Filter table record. The entries in this field contain status codes corresponding to the fields in the `err_filter_entry` structure. See the Description section for a discussion of how to determine the update status for any field.

Description

This procedure adds a row to the Error Filter table.

Additions to this table are durable; that is, they do survive a restart of the Remote Manager.

Calls to this procedure must specify a valid `message_code` for the row to be added.

The Error Filter table is dynamic and does not have a fixed upper boundary. The size of the table fluctuates as entries are added and deleted. When a row becomes empty or unoccupied, it is removed and the remaining rows are renumbered.

Additions to the Error Filter table are processed immediately and may affect more than one ACMS process.

Example

```
int add_err_filter(int client_id,CLIENT *cl2)
 {

   int *status;
   err_filter_config_rec_r_2 set struct;
```

```
   set_struct.client_id                  = client_id;
   set_struct.err_code                   = 16637820;

   status = acmsmgmt_add_err_filter_2(&set_struct,cl2);

   if (!status) {
      printf("\n Call to add filter failed");
      return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\nCall to add error filter failed with status %d", *status);
        free(status);
      return(MGMT_FAIL);
   }
   else {
      printf("\nCall to add error filter was executed");
   }
     free(status);
   return(0);
}
```

In the preceding example, the `acmsmgmt_add_err_filter_2` procedure is called to add a row to the Error Filter table. If the call succeeds, the filter is added to the Error Filter table. Otherwise, an error message is displayed.

8.5. ACMSMGMT_ADD_TRAP_1

ACMSMGMT_ADD_TRAP_1

ACMSMGMT_ADD_TRAP_1 — This procedure adds entries to the Remote Manager Trap table. Trap table entries can also be modified and deleted.

Format

```
trap_status_rec *acmsmgmt_add_trap_1(trap_config_rec *set_struct,CLIENT
 *cl)
```

Parameters

set_struct

Type	Trap_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Trap table fields.	
	client_id	
	Type	Integer
	Access	Read

	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value of `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	trap_entry	
	Type	Trap_update_rec_r
	Access	Read
	Mechanism	By value
	Usage	Structure containing a Trap table record. See the Description section for information on how to initialize this record.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Trap_status_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting of either a failure code or a structure of type `trap_update_rec_r`, which contains status codes for each field. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data_warn	

	Type	Trap_update_rec_r
	Access	Write
	Mechanism	By value
	Usage	Structure containing a Trap table record. The entries in this field contain status codes corresponding to the fields in the `trap_entry` structure. See the Description section for a discussion of how to determine the update status for any field.

Description

This procedure adds a row to the Trap table.

Additions to this table are not durable; that is, they do not survive a restart of the Remote Manager. To make nondynamic, permanent updates to the Trap table, use the ACMSCFG utility.

Calls to this procedure must specify `entity_type`, `entity_name`, and `param_to_trap`. The combination of these fields must be unique within the Trap table for the row to be added. *Table 8.1, "Collection Classes"* and *Table 8.4, "Entity Types"* contain symbolic values used to populate the `collection_class` and `entity_type` fields.

Setting fields `trap_min`, `trap_max` and/or severity to -1 causes them to be ignored when trap conditions are evaluated at run time. Otherwise, they must contain valid values for the row to be added (`trap_min` and `trap_max` must be position numbers; severity must be one of the valid severities).

Additions to the Trap table are processed immediately and may affect more than one ACMS process.

The size of the Trap table is unbounded.

Example

```
int add_trap_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[2] = "*";
   static trap_config_rec        set_struct;
   struct trap_status_rec        *status_rec;

   set_struct.client_id                  = client_id;
   set_struct.trap_entry.entity_type     = MGMT_ACC;
   set_struct.trap_entry.entity_name     = c_name_all;
   set_struct.trap_entry.param_to_trap   = MGMT_EXISTS;
   set_struct.trap_entry.min             = -1;
   set_struct.trap_entry.max             = 0;
   set_struct.trap_entry.severity        = MGMT_SEV_ERR;

   status_rec = acmsmgmt_add_trap_1(&set_struct,cl);

   if (!status_rec) {
      printf("\n Call to add trap failed");
      return(MGMT_FAIL);
```

```
   }

   if (status_rec->status == MGMT_WARN) {
      printf("\nThe following fields are invalid: ");
      if (status_rec->trap_status_rec_u.data_warn.entity_type  ==
MGMT_FAIL)
          printf("\n     entity_type not found or invalid");
      if (status_rec->trap_status_rec_u.data_warn.param_to_trap ==
MGMT_FAIL)
          printf("\n     param not found or invalid");
      if (status_rec->trap_status_rec_u.data_warn.min   == MGMT_FAIL)
          printf("\n     min invalid");
      if (status_rec->trap_status_rec_u.data_warn.max   == MGMT_FAIL)
          printf("\n     max invalid");
      if (status_rec->trap_status_rec_u.data_warn.severity  == MGMT_FAIL)
          printf("\n     severity invalid");
     }
   else if (status_rec->status != MGMT_SUCCESS) {
      printf("\nCall to add trap failed with status %d",
             status_rec->trap_status_rec_u.rc);
        xdr_free(xdr_trap_status_rec, status_rec);
        free(status_rec);
      return(MGMT_FAIL);
   }
   else
      printf("\nCall to add trap was executed");
        xdr_free(xdr_trap_status_rec, status_rec);
        free(status_rec);
    return(0);
}
```

In the preceding example, the `ACMSMGMT_ADD_TRAP_1` procedure is called to add a row to the Trap table. The new row will contain an entity type of ACC, an entity name of * (all), and a trap parameter of EXISTS. The value of the `trap_min` field is -1 (ignored), and the value of the `trap_max` field is 0. The severity of the trap will be error. The effect of this addition is to cause an error-level trap to be generated whenever the ACC is started on the target node. If the call succeeds, the trap is added to the Trap table. Otherwise, an error message is displayed.

8.6. ACMSMGMT_DELETE_COLLECTION_1

ACMSMGMT_DELETE_COLLECTION_1

ACMSMGMT_DELETE_COLLECTION_1 — This procedure deletes entries from the Remote Manager Collection table. Collection table entries can also be added and updated.

Format

```
int *acmsmgmt_delete_collection_1(coll_del_rec *set_struct,CLIENT *cl)
```

Parameters

set_struct

Type	Coll_del_rec

Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Collection table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	entity_type	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	The type of ACMS entity the process is.
	entity_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Pointer to a character string containing a full or partial entity name. May contain wildcard characters (*, !).
	collection_class	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	The type of collection class to delete.

cl

Type	CLIENT *

Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to a status code containing a success or failure status code. MGMT_SUCCESS indicates success. Other values indicate failure.

Description

This procedure deletes a row from the Collection table.

Calls to this procedure must specify `entity_type`, `entity_name`, and `collection_class`. The combination of these fields must exactly match an existing row in the table for the row to be deleted. *Table 8.1, "Collection Classes"*and *Table 8.4, "Entity Types"* contain symbolic values used to populate the `collection_class` and `entity_type` fields; `entity_name` is specified as a null-terminated string.

ID and CONFIG class rows cannot be deleted.

The Collection table contains a fixed number of rows, which is determined by the Remote Manager Parameter table field `total_entity_slots`. This is a nondynamic parameter and requires a restart of the ACMS system in order to be changed. The default is 20 rows. When a row is deleted, it becomes immediately available for reuse.

Deletions from the collection table are processed immediately and may affect more than one ACMS process.

Example

```
int delete_collection_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[] = "*";
   static coll_del_rec set_struct;
   int *status;

   set_struct.client_id         = client_id;
   set_struct.entity_type       = MGMT_ALL;
   set_struct.entity_name       = c_name_all;
   set_struct.collection_class  = MGMT_CLASS_RT;

   status = acmsmgmt_delete_collection_1(&set_struct,cl);

   if (!status) {
      printf("\n Call to delete collection failed");
      return(MGMT_FAIL);
```

```
   }

   if (*status != MGMT_SUCCESS) {
      printf("\n Call to delete collection failed with status %d",*status);
        free (status);
      return(MGMT_FAIL);
   }
   else
      printf("\nCall to delete collection was executed");
        free (status);
      return(0);
}
```

In the preceding example, the ACMSMGMT_DELETE_COLLECTION_1 procedure is called to delete a row from the Collection table. The row deleted is for all entity types (*), all entity names (*), and the RUNTIME collection class. If the call succeeds, the collection table row is deleted, and the RUNTIME collection state for some processes may be changed depending on the collection state of the row before it was deleted. Otherwise, an error message is displayed.

8.7. ACMSMGMT_DELETE_ERR_FILTER_2

ACMSMGMT_DELETE_ERR_FILTER_2

ACMSMGMT_DELETE_ERR_FILTER_2 — This procedure deletes entries from the Remote Manager Error Filter table. Error Filter table entries can also be added.

Format

```
int *acmsmgmt_delete_err_filter_2(err_del_rec *set_struct,CLIENT *cl2)
```

Parameters

set_struct

Type	Err_del_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Error Filter table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the

		ACMS$MGMT_GET_CREDS procedure.
	error_code	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	The type of ACMS entity the process is.

cl2

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to a status code containing a success or failure status code. MGMT_SUCCESS indicates success. Other values indicate failure.

Description

This procedure deletes rows from the Error Filter table.

The Error Filter table is dynamic and does not have a fixed upper boundary. The size of the table fluctuates as entries are added and deleted. When a row becomes empty or unoccupied, it is removed and the remaining rows are renumbered.

Changes to the Error Filter table (see *Section 9.6, "Error Filter Table"*) are processed immediately and may affect more than one ACMS process.

Example

```
int delete_err_filter(int client_id,CLIENT *cl2)
 {

   int *status;
   err_filter_config_rec_r_2 set_struct;

   set_struct.client_id       = client_id;
   set_struct.err_code        = 16638720;

   status = acmsmgmt_delete_err_filter_2(&set_struct,cl2);
```

```
   if (!status) {
        printf("\n RPC Call to delete filter failed");
        return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\n Call to delete error filter failed with status
 %d",*status);
      free(status);
      return(MGMT_FAIL);
   }
   else {
      printf("\n Call to delete error filter was executed");
   }
   free(status);
   return(0);
}
```

In the preceding example, the `acmsmgmt_delete_err_filter_2` procedure is called to delete a row from the Error Filter table.

8.8. ACMSMGMT_DELETE_TRAP_1

ACMSMGMT_DELETE_TRAP_1

ACMSMGMT_DELETE_TRAP_1 — This procedure deletes entries from the Remote Manager Trap table. Trap table entries can also be added and updated.

Format

```
int *acmsmgmt_delete_trap_1(trap_del_rec *set_struct,CLIENT *cl)
```

Parameters

set_struct

Type	Trap_del_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Trap table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value

		for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	entity_type	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	The type of ACMS entity the process is.
	entity_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Pointer to a character string containing a full or partial entity name. May contain wildcard characters (*, !).
	param_to_trap	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	The type of parameter to be monitored for trap conditions.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to a status code containing a success or failure status code. MGMT_SUCCESS indicates success. Other values indicate failure.

Description

This procedure deletes rows from the Trap table.

Calls to this procedure must specify `entity_type`, `entity_name`, and `param_to_trap`. These fields must exactly match an existing record in the Trap table for the delete to be performed. *Table 8.1, "Collection Classes"* and *Table 8.4, "Entity Types"* contain symbolic values used to populate the `collection_class` and `entity_type` fields; symbolic values to the `param_to_trap` field are described in *Table 8.8, "Trap Parameters"*.

Deletions from the Trap table are processed immediately and may affect more than one ACMS process.

Example

```
int delete_trap_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[2] = "*";
   static trap_del_rec set_struct;
   static int *status;

   set_struct.client_id       = client_id
   set_struct.entity_type     = MGMT_ACC;
   set_struct.entity_name     = c_name_all;
   set_struct.param_to_trap   = MGMT_EXISTS;

   status = acmsmgmt_delete_trap_1(&set_struct,cl);

   if (!status) {
      printf("\n Call to delete trap failed");
      return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\nCall to delete trap failed with status %d",*status);
        free(status);
      return(MGMT_FAIL);
   }
   else
      printf("\nCall to delete trap was executed");
        free(status);
      return(0);
}
```

In the preceding example, the ACMSMGMT_DELETE_TRAP_1 procedure is called to delete a row from the Trap table. The row to be deleted contains an entity type of ACC, an entity name of * (all), and a trap parameter of EXISTS. If the call succeeds, the trap is deleted from the Trap table. Otherwise, an error message is displayed.

8.9. ACMSMGMT_GET_ACC_2

ACMSMGMT_GET_ACC_2

ACMSMGMT_GET_ACC_2 — ACMS Remote Manager clients call this procedure to obtain class information about an ACMS Central Controller (ACC) on a local or remote node.

Format

```
acc_rec_out_2 *acmsmgmt_get_acc_2 (sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Acc_rec_out_2
Access	Write
Mechanism	By reference
Usage	Pointer to record returned. If NULL, the RPC has failed. If not null, the record contains either an error code in the status field (the RPC succeeded, but the call failed for another reason) or the data requested.

Description

This procedure obtains class information about an ACC. The return pointer points to a record of type `acc_rec_out_2`, which contains a union consisting of either a failure return code or a pointer to an ACC record.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Example

```
int get_acc_data(int client_id,CLIENT *cl)
 {

   acc_rec_r_2      *accs;
   acc_rec_out_2    *acc_rec;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;

   acc_rec = acmsmgmt_get_acc_2(&sub_rec,cl);

   if (!acc_rec) {
      printf("\n RPC Call to get ACC data failed");
      return(MGMT_FAIL);
   }

   if (acc_rec->status != MGMT_SUCCESS) {
      printf("\n Call to get ACC data failed, returning status code %d",
             acc_rec->status);
        xdr_free(xdr_acc_rec_out_2, acc_rec);
        free(acc_rec);
      return(status);
   }

   accs = &acc_rec->acc_rec_out_2_u.acc_rec;

      printf("\n ACC version is %s",accs->acms_version);
        xdr_free(xdr_acc_rec_out_2, acc_rec);
        free(acc_rec);
      return(0);

}
```

In the preceding example, the ACMSMGMT_GET_ACC_2 procedure is called to fetch ACC management information. If the call succeeds, the ACC version is printed from the retreived record. Otherwise, an error message is displayed.

8.10. ACMS$MGMT_GET_CREDS

ACMS$MGMT_GET_CREDS

ACMS$MGMT_GET_CREDS — Clients that support explicit authentication call this procedure to obtain a client ID. A client ID is issued for the client process when the client process logs in to the

ACMS Remote Manager using the ACMSMGR **LOGIN** command. Once obtained by this procedure, the client ID is used on subsequent RPC calls.

Format

```
int ACMS$MGMT_GET_CREDS(char *server_node,char *user_name, int *client)
```

Parameters

server_node

Type	String
Access	Read
Mechanism	By reference
Usage	Name of the node the server that issued the client ID was running on; the node that will be accessed. `Client_id` is valid only for the server that issued it.

user_name

Type	String
Access	Read
Mechanism	By reference
Usage	Name of the user the client ID was issued to, and on whose behalf the client ID is used. The name may the same as or different than the account name of the client process.

client

Type	Integer
Access	Write
Mechanism	By reference
Usage	The client ID to be used for the target user on the target server node. The client ID is valid only for the client process that created it.

Return Value

Type	Integer	
Access	Write	
Mechanism	By value	
Usage	The completion status of the call. The following are possible return values:	
	Value	**Description**

	MGMT_SUCCESS	Client ID was fetched; credentials verified.
	MGMT_NO_NODELOGICAL	Can't translate UCX$INET_HOST logical name to get local node name.
	MGMT_NO_CREDS_FILE	Credentials file was not found.
	MGMT_CREDS_DATA_ERR	Credentials file is corrupt.
	MGMT_WRONG_PID	PID in credentials file doesn't match client process's PID.
	MGMT_WRONG_NODE	Node name in credentials file doesn't match `server_node` argument.

Description

Clients call this procedure to fetch a previously created client ID from an encrypted credentials file. Credentials files can be created only by the ACMSMGR **`LOGIN`** command. They are stored in the directory pointed to by the logical name ACMS$MGMT_CREDS_DIR (or SYS$LOGIN if ACMS$MGMT_CREDS_DIR is not defined). Credentials files are named using the following format:

```
user-name_pid_target-node_current-node.dat
```

In this format:

- *user-name* must match the `user_name` argument string.
- *pid* must match the PID of the client process.
- *target-node* must match the `server_node` argument string.
- *current-node* must be the local node name (as determined by the logical name UCX$INET_HOST).

Note

For credentials information to be created, the client process must first execute the **`LOGIN`** command of the ACMSMGR utility. The only way to create credentials files is by using the ACMSMGR utility.

If the credentials file cannot be located, opened, and read, an error is returned. Once opened and read, the credentials in the file are verified. If the credentials are acceptable, the `client_id` field is populated and the procedure returns a status that indicates success.

This procedure is statically linked and locally executed.

Example

```
#include <rpc/rpc.h>
#include string
#include "acmsmgmt_rpc.h"

CLIENT *cl;
char sname[] = "sparks";
char *username_p, username[13] = "";
```

```
int client_id;
int status;

int ACMS$MGMT_GET_CREDS();

int main ()
{

    /* if the logical is defined, credential information will be used */
    username_p = getenv("ACMS$MGMT_USER");
    if (username_p)
          strcpy(username,username_p);

    /* establish an rpc connection to the server */
    cl = clnt_create(sname, ACMSMGMT_RPC, ACMSMGMT_VERSION, "tcp");

    /* if the connection was established */
    if (cl != NULL) {

        /* create a security context */
        cl->cl_auth = authunix_create_default();
        client_id = 0;

        /* optionally, get credentials for this user & server */
        if (strlen(username))
           status = ACMS$MGMT_GET_CREDS(sname,username,&client_id);

      }

return(1);
}
```

The preceding example is a program that performs initialization for an ACMS Remote Manager client. The program calls the ACMS$MGMT_GET_CREDS procedure to obtain the client ID for the user whose name is defined by the logical name ACMS$MGMT_USER on the node SPARKS.

8.11. ACMSMGMT_GET_ERR_FILTER_2

ACMSMGMT_GET_ERR_FILTER_2

ACMSMGMT_GET_ERR_FILTER_2 — ACMS Remote Manager clients call this procedure to obtain a listing of system messages currently being filtered from the Remote Manager, and subsequently, the error log.

Format

```
int *acmsmgmt_get_err_filter_2 (sub_id_struct *sub_id_rec,CLIENT *cl2)
```

Parameters

sub_id_rec

Type	Sub_id_struct
Access	Read

Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl2

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Err_filter_data_list_2
Access	Write
Mechanism	By reference
Usage	Pointer to record returned. If NULL, the RPC has failed. If not null, the record contains either an error code in the status field (the RPC succeeded, but the call failed for another reason) or the data requested.

Description

This procedure obtains class information about an Error Filter. The return pointer points to a record of type `err_filter_data_list_2`, which is a union containing either an error code or a pointer to an Error Filter record.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Example

```
int get_err_filter(int client_id,CLIENT *cl2)
```

```
 {

   int status;
   err_filter_data_list_2      *err_filter;
   err_filter_link_2           *nl;
   struct sub_id_struct sub_rec;

   sub_rec.client_id = client_id;

   err_filter = acmsmgmt_get_err_filter_2(&sub_rec,cl2);

   if (!err_filter) {
        printf("\n RPC Call to get Error Filter failed");
        return(MGMT_FAIL);
   }

   if (err_filter->status != MGMT_SUCCESS) {
        printf("\n Call to get Error Filter failed, returning status code
 %d",
               err_filter->status);
        status = err_filter->status;
        xdr_free(xdr_err_filter_data_list_2, err_filter);
        free(err_filter);
        return(status);
   }

   for (n1 = err_filter->err_filter_data_list_2_u.list; nl != NULL;
        nl = nl->pNext) {
        printf("Filter name = %s, and code =%X\n",
              nl->err_filter_data.err_msg_name,
              nl->err_filter_data.err_code);
   }

   xdr_free(xdr_err_filter_data_list_2, err_filter);
   free(err_filter);
   return(0);

}
```

In the preceding example, the `acmsmgmt_get_err_filter_2` procedure is called to fetch error filter information. If the call succeeds, the message code and symbolic name are fetched. Otherwise, an error message is displayed.

8.12. ACMSMGMT_GET_MGR_STATUS_1

ACMSMGMT_GET_MGR_STATUS_1

ACMSMGMT_GET_MGR_STATUS_1 — ACMS Remote Manager clients call this procedure to obtain run-time status information about a Remote Manager on a particular node.

Format

```
mgr_status_rec_out *acmsmgmt_get_mgr_status_1(sub_id_struct *sub_rec,
 CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Mgr_status_rec_out	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting either of a failure code or a pointer to a structure of type `mgr_status_rec`, which contains the status data. The following are the contents of this union:	
	rc	
	Type	Integer
	Access	Write

	Mechanism	By value
	Usage	Failure return code.
	data	
	Type	Mgr_status_rec
	Access	Write
	Mechanism	By reference
	Usage	Remote Manager status data record. Contains the fields from the Manager Status table.

Description

This procedure gets run-time information about a Remote Manager on a particular node. The return pointer points to a record of type `mgr_status_rec_out`, which contains a union consisting of a failure returns code or a pointer to a manager status record.

This procedure does not require the ACMS run-time system in order to execute.

Example

```
int get_mgr_data(int client_id,CLIENT *cl)
 {

   mgr_status_rec         *mgrs;
   mgr_status_rec_out     *mgr_data;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;

   mgr_data = acmsmgmt_get_mgr_status_1(&sub_rec,cl);

   if (!mgr_data) {
      printf("\n RPC Call to get RM data failed");
      return(MGMT_FAIL);
   }

   if (mgr_data->status != MGMT_SUCCESS) {
      printf("\n Call to get RM data failed, returning status code %d",
             mgr_data->status);
      status = mgr_data->status;
        xdr_free(xdr_mgr_status_rec_out, mgr_data);
        free(mgr_data);
      return(status);
   }

   mgrs = &mgr_data->mgr_status_rec_out_u.data;

      printf("\n RPC UDP state is %d",mgrs->rpc_udp_state);
        xdr_free(xdr_mgr_status_rec_out, mgr_data);
        free(mgr_data);
      return(0);
}
```

In the preceding example, the ACMSMGMT_GET_MGR_STATUS_1 procedure is called to fetch the contents of the Manager Status table. If the call succeeds, the current state of the TCP/UDP protocol in the RPC interface is printed from the retrieved record. Otherwise, an error message is displayed.

8.13. ACMSMGMT_GET_PARAM_2

ACMSMGMT_GET_PARAM_2

ACMSMGMT_GET_PARAM_2 — ACMS Remote Manager clients call this procedure to obtain configuration information about a Remote Manager on a particular node.

Format

```
param_rec_out2 *acmsmgmt_get_param_2(sub_id_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Param_rec_out2
Access	Write
Mechanism	By reference
Usage	Pointer to record returned. If NULL, the RPC has failed. If not null, the record contains either an error code in the status field (the RPC succeeded, but the call failed for another reason), or the data requested.

Description

This procedure gets configuration information about a Remote Manager on a particular node. The return pointer points to a record of type `param_rec_out2`, which contains a union consisting of either a failure return code or a pointer to a Parameter record.

This procedure does not require the ACMS run-time system in order to execute.

Example

```
int get_param_data(int client_id,CLIENT *cl)
 {
   int x = 0;
   int y = 0;

   param_rec2       *params;
   param_rec_out2   *param_rec;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;

   param_rec = acmsmgmt_get_param_2(&sub_rec,cl);

   if (!param_rec) {
      printf("\n RPC Call to get Parameter data failed");
      return(MGMT_FAIL);
   }

   if (param_rec->status != MGMT_SUCCESS) {
      printf("\n Call to get Parameter data failed, returning status code
 %d",
             param_rec->status);
      status = param_rec->status;
        xdr_free(xdr_param_rec_out2, param_rec);
        free(param_rec);
      return(status);
   }

   params = &param_rec->param_rec_out2_u.data;

      printf("\n Maximum logins allowed is %d",params->max_logins);
        xdr_free(xdr_param_rec_out2, param_rec);
```

```
        free(param_rec);
      return(0);
}
```

In the preceding example, the ACMSMGMT_GET_PARAM_2 procedure is called to fetch the contents of the Parameter table. If the call succeeds, the maximum number of logins is printed from the retrieved record. Otherwise, an error message is displayed.

8.14. ACMSMGMT_GET_QTI_2

ACMSMGMT_GET_QTI_2

ACMSMGMT_GET_QTI_2 — ACMS Remote Manager clients call this procedure to obtain class information about a Queued Task Initiator (QTI) on a local or remote node.

Format

```
qti_rec_out_2 *acmsmgmt_get_qti_2(sub_id_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value

Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Qti_rec_out2
Access	Write
Mechanism	By reference
Usage	Pointer to record returned. If NULL, the RPC has failed. If not null, the record contains either an error code in the status field (the RPC succeeded, but the call failed for another reason) or the data requested.

Description

This procedure obtains class information about a QTI on a local or remote node. The return pointer points to a record of type `qti_rec_out_2`, which contains a union consisting of either a failure return code or a pointer to a QTI record.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Example

```
int get_qti_data(int client_id,CLIENT *cl)
 {

   qti_rec_r_2      *qtis;
   qti_rec_out_2    *qti_rec;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;

   qti_rec = acmsmgmt_get_qti_2(&sub_rec,cl);

   if (!qti_rec) {
      printf("\n RPC Call to get QTI data failed");
      return(MGMT_FAIL);
   }

   if (qti_rec->status != MGMT_SUCCESS) {
      printf("\n Call to get QTI data failed, returning status code %d",
             qti_rec->status);
      status = qti_rec->status;
        xdr_free(xdr_qti_rec_out_2, qti_rec);
        free(qti_rec);
      return(status);
   }

   qtis = &qti_rec->qti_rec_out_2_u.qti_rec;

      printf("\n QTI process name is %s",qtis->process_name);
```

```
        xdr_free(xdr_qti_rec_out_2, qti_rec);
        free(qti_rec);
      return(0);
}
```

In the preceding example, the ACMSMGMT_GET_QTI_2 procedure is called to fetch QTI management information. If the call succeeds, the QTI process name is printed from the retrieved record. Otherwise, an error message is displayed.

8.15. ACMSMGMT_GET_TSC_2

ACMSMGMT_GET_TSC_2

ACMSMGMT_GET_TSC_2 — ACMS Remote Manager clients call this procedure to obtain class information about a Terminal Subsystem Controller (TSC) on a local or remote node.

Format

```
tsc_rec_out_2 *acmsmgmt_get_tsc_2(sub_id_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value

Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Tsc_rec_out_2
Access	Write
Mechanism	By reference
Usage	Pointer to record returned. If NULL, the RPC has failed. If not null, the record contains either an error code in the status field (the RPC succeeded, but the call failed for another reason) or the data requested.

Description

The return pointer points to a record of type `tsc_rec_out_2`, which contains a union consisting of either a failure return code or a pointer to a TSC record.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Example

```
int get_tsc_data(int client_id,CLIENT *cl)
 {

   tsc_rec_r_2      *tscs;
   tsc_rec_out_2    *tsc_rec;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;

   tsc_rec = acmsmgmt_get_tsc_2(&sub_rec,cl);

   if (!tsc_rec) {
      printf("\n RPC Call to get TSC data failed");
      return(MGMT_FAIL);
   }

   if (tsc_rec->status != MGMT_SUCCESS) {
      printf("\n Call to get TSC data failed, returning status code %d",
             tsc_rec->status);
      status = tsc_rec->status;
        xdr_free(xdr_tsc_rec_out_2, tsc_rec);
        free(tsc_rec);
      return(status);
   }

   tscs = &tsc_rec->tsc_rec_out_2_u.tsc_rec;

      printf("\n TSC process name is %s",tscs->process_name);
        xdr_free(xdr_tsc_rec_out_2, tsc_rec);
```

```
        free(tsc_rec);
      return(0);
}
```

In the preceding example, the ACMSMGMT_GET_TSC_2 procedure is called to fetch TSC management information. If the call succeeds, the TSC's process name is printed from the retrieved record. Otherwise, an error message is displayed.

8.16. ACMSMGMT_GET_VERSION_2

ACMSMGMT_GET_VERSION_2

ACMSMGMT_GET_VERSION_2 — ACMS Remote Manager clients call this procedure to obtain version information for ACMS.

Format

```
version_data_list_2 *acmsmgmt_get_version_2(sub_id_struct *sub_rec, CLIENT
 *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value

Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	version_data_list_2
Access	Write
Mechanism	By reference
Usage	Pointer to record returned. If NULL, the RPC has failed. If not null, the record contains either an error code in the status field (the RPC succeeded, but the call failed for another reason) or the data requested.

Description

The return pointer points to a record of type `version_data_list_2`, which constains a union consisting of either a failure return code or a pointer to a version record.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Example

```
int get_version_data(int client_id,CLIENT *cl2)
 {

   struct sub_id_struct sub_rec;
   version_data_list_2  *version;
   int status;

   sub_rec.client_id = client_id;

   version = acmsmgmt_get_version_2(&sub_rec,cl2);

   if (!version) {
      printf("\n RPC Call to get Version data failed");
      return(MGMT_FAIL);
   }

   if (version->status != MGMT_SUCCESS) {
      printf("\n Call to get Version data failed, returning status code
 %d",
             version->status);
      status = version->status;
        xdr_free(xdr_version_data_list_2, version);
        free(version);
      return(status);
   }

   printf("\n ACMS version is %s",version->
          version_data_list_2_u.data.acms_version);
     xdr_free(xdr_version_data_list_2, version);
     free(version);
```

```
   return(0);
}
```

In the preceding example, the ACMSMGMT_GET_VERSION_2 procedure is called to fetch ACMS version information. If the call succeeds, the version of the installed ACMS software is printed from the retrieved record. Otherwise, an error message is displayed.

8.17. ACMSMGMT_LIST_AGENT_2

ACMSMGMT_LIST_AGENT_2

ACMSMGMT_LIST_AGENT_2 — ACMS Remote Manager clients call this procedure to obtain a list of Agent Process table entries.

Format

```
agent_data_list *acmsmgmt_list_agent_2(agent_sel_struct *sub_rec, CLIENT
 *cl)
```

Parameters

sub_rec

Type	agent_sel_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information:	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	proc_name	
	Type	String
	Access	Read
	Mechanism	By value
	Usage	String that lists the OpenVMS process name for each Agent.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	agent_data_list		
Access	Write		
Mechanism	By reference		
Usage	Pointer to a union. The union contains either a failure code or a pointer to a structure of type `agent_data_list`, which contains the start of a linked list of records. The following are the contents of this union:		
	rc		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	list		
	Type	agent_list	
	Access	Write	
	Mechanism	By reference	
	Usage	Start of linked list. Pointer to a structure of Agent table record, and a forward pointer to the next node in the linked list. The following are the contents of this structure:	
		pNext	
		Type	agent_write
		Access	Write
		Mechanism	By value
		Usage	Start of linked list. Pointer to a structure of type `coll_list`.
		agent_data	
		Type	agent_rec_r
		Access	Write
		Mechanism	By reference
		Usage	Agent table row. Agent table fields.

Description

The ACMSMGMT_LIST_AGENT_2 procedure returns a linked list of Agent table rows. All Agent table rows are returned in each call. Records are returned sequentially from the table, beginning at the start of the table. Entire table rows are returned.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Rows in the Agent table are subject to reuse. Rows are assigned round-robin, and are not cleared until they have been reassigned. Some rows may contain data for inactive Agents. It is the callers responsibility to examine the `record_state` field to determine whether this row belongs to an active (`record_state` field is MGMT_VALID) or inactive (`record_state` field is MGMT_INACTIVE) Agent, and to process the row accordingly.

Example

```
int list_agent_data(int client_id,CLIENT *cl)
{
    static char c_all_agents[2] = "*";
    agent_data_list *agent_data;
    agent_link *nl;
    static struct agent_sel_struct sub_rec;
    int status;

sub_rec.client_id = client_id;
sub_rec.proc_name = c_all_agents;

agent_data = acmsmgmt_list_agent_2(&sub_rec,cl);

if (!agent_data) {
    printf("\n RPC Call to get agent data failed");
    return(MGMT_FAIL);
}
if (agent_data->status == MGMT_FAIL) {
    if (agent_data->agent_data_list_u.rc == MGMT_NOMORE_DATA) {
        printf("\n No agent data found");
        xdr_free(xdr_agent_data_list, agent_data);
        free(agent data);
      return(MGMT_FAIL);
    }
    printf("\n Call to get agent data failed, returning status code %d",
        agent_data->agent_data_list_u.rc);
    status = agent_data->agent_data_list_u.rc;
        xdr_free(xdr_agent_data_list, agent_data);
        free(agent_data);
    return(status);
}
if (agent_data->status == MGMT_WARN)
    printf("\n ** Warning, some data may be from inactive processes **");

for (nl = agent_data->agent_data_list_u.list; nl != NULL; nl = nl->pNext) {
    if (nl->agent_data.record_state == MGMT_INACTIVE)
        printf("\n INACTIVE ");
    else
        printf("\n ");
    printf(" PID: %8X Process Name: %-s",
```

```
            nl->agent_data.pid,
            nl->agent_data.process_name);
}
    printf("\n End of data");
        xdr_free(xdr_agent_data_list, agent_data);
        free(agent_data);
    return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_AGENT_2 procedure is called to fetch a linked list of agents running on the system. If the call succeeds, the `pid` and name of each agent in the list are printed. Otherwise, an error message is displayed.

8.18. ACMSMGMT_LIST_COLLECTIONS_2

ACMSMGMT_LIST_COLLECTIONS_2

ACMSMGMT_LIST_COLLECTIONS_2 — ACMS Remote Manager clients call this procedure to obtain a list of Collection table entries.

Format

```
coll_data_list_2 *acmsmgmt_list_collections_2(coll_sel_struct *coll_rec,
 CLIENT *cl)
```

Parameters

coll_rec

Type	Coll_sel_struct	
Access	Read	
Mechanism	By reference	
Usage	Defines starting point for list of records to be returned. Also identifies the user. The `coll_rec` structure contains the following fields:	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used.
	starting_rec	
	Type	Integer
	Access	Read

	Mechanism	By value
	Usage	Sequential record number (starting at 0) of record to begin list from. Records are returned sequentially from the table. Up to `max_rpc_return_recs` (Parameter table configuration value) are returned in each call.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Coll_data_list_2		
Access	Write		
Mechanism	By reference		
Usage	Pointer to a union. The union contains either a failure code or a pointer to a structure of type `coll_list`, which contains the start of a linked list of records. The following are the contents of this union:		
	rc		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	list		
	Type	Coll_list_2	
	Access	Write	
	Mechanism	By reference	
	Usage	Start of linked list. Pointer to a structure of collection table record, and a forward pointer to the next node in the linked list. The following are the contents of this structure:	
		pNext	
		Type	Coll_list_2
		Access	Write
		Mechanism	By value

		Usage	Start of linked list. Pointer to a structure of type `coll_list`.
		coll_data	
		Type	Coll_rec_2
		Access	Write
		Mechanism	By reference
		Usage	Collection table row. Collection table fields are described in *Section 9.4, "Collection Table"*.

Description

The ACMSMGMT_LIST_COLLECTIONS_2 procedure returns a linked list of collection table rows. The number of rows returned in a single call is bounded by the value of the Parameter table field `max_rpc_return_recs`. More than one call may be required to fetch all the rows. The selection record field `starting_rec` determines the table row to begin with. Records are returned sequentially from the table, beginning with the `starting_rec` row. Row numbering begins at 0.

Entire table rows are returned. See *Section 9.4, "Collection Table"* for a description of the fields in the `coll_rec` structure.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

If the end of the table is reached during execution of this procedure, MGMT_NOMORE_DATA is returned in the status field.

Example

```
int list_collection_data(int client_id,CLIENT *cl)
{

   int rec_count = 0;
   coll_data_list_2  *coll;
   coll_link_2       *nl;
   static struct coll_sel_struct coll_rec;
   int status;
   char c_states[2][9] = {"enabled","disabled"};
   char c_entities[10][9] = {"unknown","*","acc","tsc","qti","cp","exc",
                             "server","group","mgr"};
   char c_classes[6][8] = {"*","id","config","runtime","pool","error"};

   coll_rec.client_id = client_id;
top:
   coll_rec.starting_rec = rec_count;

   coll = acmsmgmt_list_collections_2(&coll_rec,cl);

   if (!coll) {
      printf("\n RPC Call to get Collection data failed");
      return(MGMT_FAIL);
   }
```

```
   if ((coll->status != MGMT_SUCCESS) && (coll->status !=
 MGMT_NOMORE_DATA)) {
      printf("\n Call to get Collection data failed, returning status code
             %d",coll->status);
        xdr_free(xdr_coll_data_list2, coll);
        free(coll);
      return(status);
   }

   for (nl = coll->coll_data_list_2_u.list; nl != NULL; nl = nl->pNext) {
       rec_count++;
       if (nl->coll_data.entity_name_s > 0)
          printf("\n Entity: %-9s  Name: %-32s  Class: %-9s
                 Collection State: %-9s",
                 c_entities[nl->coll_data.entity_type],
                 nl->coll_data.entity_name,
                 c_classes[nl->coll_data.collection_class],
                 c_states[nl->coll_data.collection_state]);
   }

   if (coll->status != MGMT_NOMORE_DATA)
      goto top;

   printf("\n End of data");
     xdr_free(xdr_coll_data_list_2, coll);
     free(coll);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_COLLECTIONS_2 procedure is called to fetch the contents of the Collection table. If the call succeeds, the entity type, name, class, and collection state are printed for each row in the table. Otherwise, an error message is displayed.

8.19. ACMSMGMT_LIST_CP_2

ACMSMGMT_LIST_CP_2

ACMSMGMT_LIST_CP_2 — ACMS Remote Manager clients call this procedure to obtain a list of Command Process (CP) table entries.

Format

```
cp_data_list_2 *acmsmgmt_list_cp_2(cp_sel_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Cp_sel_struct
Access	Read
Mechanism	By reference
Usage	Structure that contains the following client authorization information.

	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the `ACMS$MGMT_GET_CREDS` procedure.
	proc_name	
	Type	String
	Access	Read
	Mechanism	By value
	Usage	String that lists the OpenVMS process name for each CP.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cp_data_list_2	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting of either a failure code or a pointer to a structure of type `cp_data_list2`, which contains the start of a linked list of records. The following are the contents of this union:	
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	list	

	Type	Cp_list_2	
	Access	Write	
	Mechanism	By reference	
	Usage	Start of linked list. Pointer to a structure of CP table record, and a forward pointer to the next node in the linked list. The following are the contents of this structure:	
		pNext	
		Type	Cp_list_2
		Access	Write
		Mechanism	By value
		Usage	Start of linked list. Pointer to a structure of type `coll_list`.
		cp_data	
		Type	Cp_rec_r_2
		Access	Write
		Mechanism	By reference
		Usage	CP table row. CP table fields are described in *Section 9.5, "CP Table"*.

Description

The ACMSMGMT_LIST_CP_2 procedure returns a linked list of CP table rows. All CP table rows are returned in each call. Records are returned sequentially from the table, beginning at the start of the table.

Entire table rows are returned. See *Section 9.5, "CP Table"* for a description of the fields in the `cp_rec_r` structure.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Rows in the CP table are subject to reuse. Rows are assigned round-robin, and are not cleared until they have been reassigned. So some rows may contain data for inactive CPs. It is the caller's responsibility to examine the `record_state` field to determine whether this row belongs to an active (`record_state` field is MGMT_VALID) or inactive (`record_state` field is MGMT_INACTIVE) CP, and to process the row accordingly.

Example

```
int list_cp_data(int client_id,CLIENT *cl)
{
   static char c_all_cps[2] = "*";
   cp_data_list_2  *cp_data;
   cp_link_2       *nl;
   static struct cp_sel_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;
   sub_rec.proc_name = c_all_cps;
```

```
   cp_data = acmsmgmt_list_cp_2(&sub_rec,cl);

   if (!cp_data) {
      printf("\n RPC Call to get CP data failed");
      return(MGMT_FAIL);
   }

   if (cp_data->status == MGMT_FAIL)  {
      if (cp_data->cp_data_list_2_u.rc == MGMT_NOMORE_DATA) {
         printf("\n No CP data found");
           xdr_free(xdr_cp_data_list_2, cp_data);
           free(cp data);
         return(MGMT_FAIL);
      }
      printf("\n Call to get CP data failed, returning status code %d",
             cp_data->cp_data_list_2_u.rc);
      status = cp_data->cp_data_list_2_u.rc;
        xdr_free(xdr_cp_data_list_2, cp_data);
        free(cp_data);
      return(status);
   }

   if (cp_data->status == MGMT_WARN)
      printf("\n ** Warning, some data may be from inactive processes **");

   for (nl = cp_data->cp_data_list_2_u.list; nl != NULL; nl = nl->pNext) {
       if (nl->cp_data.record_state == MGMT_INACTIVE)
          printf("\n INACTIVE ");
       else
          printf("\n          ");
       printf(" PID: %8X  Process Name: %-s",
               nl->cp_data.pid,
               nl->cp_data.process_name);
   }

   printf("\n End of data");
     xdr_free(xdr_cp_data_list_2, cp_data);
     free(cp_data);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_CP_2 procedure is called to fetch the contents of the CP table. If the call succeeds, the state of the CP (if INACTIVE), its PID, and process name are displayed for each table row returned. Otherwise, an error message is displayed.

8.20. ACMSMGMT_LIST_ERR_2

ACMSMGMT_LIST_ERR_2

ACMSMGMT_LIST_ERR_2 — ACMS Remote Manager clients call this procedure to obtain a list of the error log entries.

Format

```
err_data_list *acmsmgmt_list_err_2(err_sel_struct *err_sel, CLIENT *cl)
```

Parameters

log_rec

Type	Log_sel_struct	
Access	Read	
Mechanism	By reference	
Usage	Defines which log records to return. The `log_sel_struct` contains the following fields:	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	before_time	
	Type	Null-terminated character string
	Access	Read, optional
	Mechanism	By reference
	Usage	Pointer to a null-terminated character string containing a valid OpenVMS ASCII time string. This field determines the chronological starting point for the list of records to be returned. If omitted, records are returned beginning at the start of the file. Format is OpenVMS ASCII time (*DD-MMM-YY HH:MM:SS.hh*).
	since_time	
	Type	Null-terminated character string
	Access	Read, optional
	Mechanism	By reference
	Usage	Pointer to a null-terminated character string containing a valid OpenVMS ASCII time string. This field determines

	the chronological ending point for the list of records to be returned. If omitted, records are returned until the end of the file is reached. Format is OpenVMS ASCII time (*DD-MMM-YY HH:MM:SS.hh*).
file_name	
Type	Null-terminated character string
Access	Read, optional
Mechanism	By reference
Usage	Pointer to a null-terminated character string containing either a valid OpenVMS file specification or a logical name pointing to a valid OpenVMS file specification. This field determines the log file to be processed. An empty string requests the default (currently open) log file.
dup_count	
Type	Integer
Access	Read
Mechanism	By value
Usage	A sequential counter of records with the same time. This allows records to be unique even if they were generated at the same time. Set this value to -1 for the initial call.
facility	
Type	Integer
Access	Read, optional
Mechanism	By value
Usage	Value of a valid Remote Manager facility. If specified, only records with matching facility codes are returned.
severity	
Type	Integer
Access	Read, optional
Mechanism	By value

	Usage	Value of a valid Remote Manager severity. If specified, only records with matching severity are returned.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Log_data_list	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a pointer to the start of a linked list of records. See the Description section for a discussion of the structure of the union. The following are the contents of this record:	
	log_data_list	
	Type	Logging_rec
	Access	Write
	Mechanism	By reference
	Usage	Pointer to a structure of type `logging_rec`.
	log_msg	
	Type	Null-terminated character string
	Access	Write
	Mechanism	By reference
	Usage	Pointer to a null-terminated character string containing the error information.
	pNext	
	Type	Log_list
	Access	Write
	Mechanism	By value
	Usage	Pointer to the next record in the linked list.

Description

The ACMSMGMT_LIST_ERR_2 procedure returns a linked list of error log entries, ordered by time. The records to be returned are determined by the fields specified in the `log_sel_struct`

input argument. Records can be selected by date and time, facility, and severity. Note that only `max_rpc_return_rec` data (Parameter table field) is returned in each call. The end of data is signaled by the status field (see the following example). If the end of data is not signaled, repeated calls are needed to fetch all matching records.

The return record is a union containing either a failure code or the first record in the list:

```
struct log_data_list {
        int status;
        union {
                int rc;
                log_list list;
        } log_data_list_u;
};
```

To determine the status of the call and the contents of the return record, first check the status field. The following are possible values in the status field:

- MGMT_FAIL

  The call has failed and the `rc` field contains a specific error code describing the failure.

- MGMT_NOMORE_DATA

  There are no more records in the file. The linked list may or may not contain the final records, depending on whether any more records matched the selection criteria.

- MGMT_SUCCESS

  The call completed successfully. More records exist than were returned in the call.

The ACMSMGMT_LIST_ERR_2 procedure returns *n* records per call, where *n* is determined by the Remote Manager parameter field `max_rpc_return_recs`. Therefore, repeated calls may be necessary to retrieve all records that match the selection criteria. Context is not maintained by the server between calls; the selection criteria are evaluated on each call by the Remote Manager. Following the initial call, callers should place the correct time value in the `since_time` field of the `log_sel_struct` input argument, as well as the correct `dup_count` value in order to have the chronologically next *n* records returned.

This procedure does not require the ACMS run-time system to execute.

Example

```
int list_err_data(int client_id,CLIENT *cl)
{
    int skip_rec = 0;
    char null_time_str[24] = "";
    char first_of_jan[24] = "01-JAN-1998 00:00:00.00";
    char file_spec[] = "acms$mgmt_err_log";
    char time_cache[MGMT_S_TIME_A+1];
    struct log_sel_struct log_rec;
    log_data_list *log;
    log_link *nl;
    int status;

    /* Initialize err selection data */

    log_rec.client_id = client_id;
```

```
    log_rec.before_time = null_time_str;
    log_rec.since_time = first_of_jan;
    log_rec.file_name = file_spec;
    log_rec.dup_count = -1;
    log_rec.facility = =1;
    log_rec.severity = -1;

  top:
    log = acmsmgmt_list_log_1(&log_rec,cl);

    if (!log) {
        return(MGMT_FAIL);
    }

    if (log->status == MGMT_FAIL) {
        status = log->log_data_list_u.rc;
            xdr_free(xdr_log_data_list, log);
            free(log);
        return(status);
    }

    for (nl = log->log_data_list_u.list; nl != NULL; n1 = n1->pNext) {
        if (skip_rec)
            skip_rec = 0;
        else
            printf("\n %-12s\t%-s",sname,nl->log_data.log_msg);
        memcpy(&time_cache[0],nl->log_data.log_msg,23);
        log_rec.dup_count = nl->log_data.dup_count;
        log_rec.since_time = time_cache;
    }

    if (log->status == MGMT_NOMORE_DATA)
        printf("\n *** End of data**");
    else {
        skip_rec = 1;
        goto top;
    }
        xdr_free(xdr_log_data_list, log);
        free(log);
    return(0);
   }
```

In the preceding example, the ACMSMGMT_LIST_ERR_2 procedure is called to fetch the contents of the error log. If the call succeeds, the node, symbolic name, and code equivalent are displayed for each ACMS run-time error relayed to the Remote Manager. Otherwise, an error message is displayed.

8.21. ACMSMGMT_LIST_EXC_2

ACMSMGMT_LIST_EXC_2

ACMSMGMT_LIST_EXC_2 — ACMS Remote Manager clients call this procedure to obtain a list of Application Execution Controller (EXC) (ACMS application) table entries.

Format

```
exc_data_list_2 *acmsmgmt_list_exc_2(exc_sel_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Exc_sel_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains client information and application selection criteria. The structure contains the following fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	appl_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	A pointer to an application name. The name may contain wildcard characters (*, !). Specify in all uppercase characters.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Exc_data_list_2
Access	Write
Mechanism	By reference

Usage	Pointer to a record that contains a union consisting of either a failure code or a pointer to a structure of type `exc_data_list_2`, which contains the start of a linked list of records. The following are the contents of this union:		
	rc		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	list		
	Type	Exc_list_2	
	Access	Write	
	Mechanism	By reference	
	Usage	Start of linked list. Pointer to a structure containing an EXC table record, and a forward pointer to the next node in the linked list. The following are the contents of this structure:	
		pNext	
		Type	Exc_list_2
		Access	Write
		Mechanism	By value
		Usage	Start of linked list. Pointer to a structure of type `coll_list`.
		exc_data	
		Type	Exc_rec_r_2
		Access	Write
		Mechanism	By reference
		Usage	EXC table row. EXC table fields are described in *Section 9.7, "EXC Table"*.

Description

The ACMSMGMT_LIST_EXC_2 procedure returns a linked list of EXC table rows. All EXC table rows whose application_name field matches the `appl_name` field in the selection record are returned in each call.

Entire table rows are returned. See *Section 9.7, "EXC Table"* for a description of the fields in the `exc_rec_r` structure.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Rows in the EXC table are subject to reuse. Rows are assigned round-robin, and are not cleared until they have been reassigned. Therefore, some rows may contain data for inactive EXCs. It is the caller's responsibility to examine the `record_state` field to determine whether this row belongs

to an active (`record_state field` is MGMT_VALID) or inactive (`record_state` field is MGMT_INACTIVE) EXC, and to process the row accordingly.

Example

```
int list_exc_data(int client_id,CLIENT *cl)
{

   static char c_all_appls[2] = "*";
   exc_data_list_2   *exc_data;
   exc_link_2      *nl;
   static struct exc_sel_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;
   sub_rec.appl_name = c_all_appls;

   exc_data = acmsmgmt_list_exc_2(&sub_rec,cl);

   if (!exc_data) {
      printf("\n RPC Call to get EXC data failed");
      return(MGMT_FAIL);
   }

   if (exc_data->status == MGMT_FAIL)  {
      if (exc_data->exc_data_list_2_u.rc == MGMT_NOMORE_DATA) {
         printf("\n No EXC data found");
           xdr_free(xdr_exc_data_list_2, exc_data);
           free(exc_data);
         return(MGMT_FAIL);
      }
      printf("\n Call to get EXC data failed, returning status code %d",
             exc_data->exc_data_list_2_u.rc);
      status = exc_data->exc_data_list_2_u.rc;
        xdr_free(xdr_exc_data_list_2, exc_data);
        free(exc_data);
      return(status);
   }

   if (exc_data->status == MGMT_WARN)
      printf("\n ** Warning, some data may be from inactive processes **");

   for (nl = exc_data->exc_data_list_2_u.list; nl != NULL; nl = nl->pNext)
 {
       if (nl->exc_data.record_state == MGMT_INACTIVE)
          printf("\n INACTIVE ");
       else
          printf("\n          ");
       printf(" PID: %8X  Application : %-s",
               nl->exc_data.pid,
               nl->exc_data.appl_name);
   }

   printf("\n End of data");
     xdr_free(xdr_exc_data_list_2, exc_data);
     free(exc_data);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_EXC_2 procedure is called to fetch the contents of the EXC table. If the call succeeds, the state of the EXC (if inactive), its PID, and its application name are displayed for each table row returned. Otherwise, an error message is displayed.

8.22. ACMSMGMT_LIST_INTERFACES_1

ACMSMGMT_LIST_INTERFACES_1

ACMSMGMT_LIST_INTERFACES_1 — ACMS Remote Manager clients call this procedure to obtain information about all configured interfaces for a Remote Manager server on a local or remote node.

Format

```
interfaces_rec_out *acmsmgmt_list_interfaces_1 (sub_id_struct *sub_rec,
 CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Interfaces_rec_out
Access	Write
Mechanism	By reference
Usage	Pointer to a record that contains a union consisting of either a failure code or a pointer to the start of a linked list of records. See the Description section for a discussion of the structure of the union. The records contain all the fields of the Interfaces table (see *Section 9.8, "Interfaces Table"*).

Description

The ACMSMGMT_LIST_INTERFACES_1 procedure returns an array of Remote Manager Interfaces table rows. All records in the table are returned. Each record represents a separate interface, as determined by the `interface_type` field.

The return record is a union containing either a failure code or the first record in the list, as follows:

```
struct interfaces_rec_out {
        int status;
        union {
                interfaces_rec_out_r interfaces;
                int rc;
        } interfaces_rec_out_u;
};
```

To determine the status of the call and the contents of the return record, first check the status field. The following are possible values in the status field:

- MGMT_FAIL

  The call has failed and the `rc` field contains a specific error code describing the failure.

- MGMT_SUCCESS

  The call completed successfully. All rows in the table were returned.

The array is contained in a structure of type *interfaces_rec_out_r* with an integer field (*num_elements*) containing the size of the array, as follows:

```
struct interfaces_rec_out_r {
        int         num_elements;
        interfaces_rec   values[MGMT_K_MAX_IF];
        };
```

Example

```
int list_interfaces_data(int client_id,CLIENT *cl)
 {

   interfaces_rec_out   *if_ptr;
   interfaces_rec_out_r *inter;
   static struct sub_id_struct sub_rec;
```

```
   int status;

   sub_rec.client_id = client_id;

   if_ptr = acmsmgmt_list_interfaces_1(&sub_rec,cl);

   if (!if_ptr) {
      printf("\n RPC Call to get Interfaces data failed");
      return(MGMT_FAIL);
   }

   inter = &if_ptr->interfaces_rec_out_u.interfaces;
   if (if_ptr->status == MGMT_FAIL) {
      printf("\n Call to get Interfaces data failed, returning status code
             %d",if_ptr->interfaces_rec_out_u.rc);
      status = if_ptr->interfaces_rec_out_u.rc;
        xdr_free(xdr_interfaces_rec_out, if_ptr);
        free(if_ptr);
      return(status);
   }

   printf("\n RPC  interface has processed %d read requests",
          inter->values[0].get_request_count);
   printf("\n SNMP interface has processed %d read requests",
          inter->values[1].get_request_count);
     xdr_free(xdr_interfaces_rec_out, if_ptr);
     free(if_ptr);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_INTERFACES_1 procedure is called to fetch the contents of the Interfaces table. If the call succeeds, the number of read requests by each interface is printed from the retrieved record. Otherwise, an error message is displayed. The example in *Section 6.5.1, "Get Example"* shows how to declare and initialize the input arguments to this procedure.

8.23. ACMSMGMT_LIST_LOG_1

ACMSMGMT_LIST_LOG_1

ACMSMGMT_LIST_LOG_1 — ACMS Remote Manager clients call this procedure to obtain information from a Remote Manager log on a local or remote node.

Format

```
log_data_list *acmsmgmt_list_log_1 (log_sel_struct *log_rec, CLIENT *cl)
```

Parameters

log_rec

Type	Log_sel_struct
Access	Read
Mechanism	By reference

Usage	Defines which log records to return. The `log_sel_struct` contains the following fields:	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used.
	before_time	
	Type	Null-terminated character string
	Access	Read, optional
	Mechanism	By reference
	Usage	Pointer to a null-terminated character string containing a valid OpenVMS ASCII time string. This field determines the chronological starting point for the list of records to be returned. If omitted, records are returned beginning at the start of the file. The format is OpenVMS ASCII time (*DD-MMM-YY HH:MM:SS.hh*).
	since_time	
	Type	Null-terminated character string
	Access	Read, optional
	Mechanism	By reference
	Usage	Pointer to a null-terminated character string containing a valid OpenVMS ASCII time string. This field determines the chronological ending point for the list of records to be returned. If omitted, records are returned until end of file is reached. The format is OpenVMS ASCII time (*DD-MMM-YY HH:MM:SS.hh*).
	file_name	
	Type	Null-terminated character string
	Access	Read, optional

	Mechanism	By reference
	Usage	Pointer to a null-terminated character string containing either a valid OpenVMS file specification or a logical name pointing to a valid OpenVMS file specification. This field determines the log file to be processed. An empty string requests the default (currently open) log file.
	dup_count	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	A sequential counter of records with the same time. This allows records to be unique even if they were generated at the same time. Set this value to -1 for the initial call.
	facility	
	Type	Integer
	Access	Read, optional
	Mechanism	By value
	Usage	Value of a valid Remote Manager facility. If specified, only audit records with matching facility codes are returned.
	severity	
	Type	Integer
	Access	Read, optional
	Mechanism	By value
	Usage	Value of a valid Remote Manager severity. If specified, only audit records with matching severity are returned.

cl

Type	CLIENT *
Access	Read
Mechanism	By value

Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Log_data_list	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a pointer to the start of a linked list of records. See the Description section for a discussion of the structure of the union. The following are the contents of this record:	
	log_data_list	
	Type	Logging_rec
	Access	Write
	Mechanism	By reference
	Usage	Pointer to a structure of type logging_rec.
	dup_count	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Integer value with uniquely identifies records generated at the same time.
	log_msg	
	Type	Null-terminated character string
	Access	Write
	Mechanism	By reference
	Usage	Pointer to a null-terminated character string containing the audit information.
	pNext	
	Type	Log_list
	Access	Write
	Mechanism	By value
	Usage	Pointer to the next record in the linked list.

Description

Note

The ACMSMGMT_LIST_LOG_1 procedure is also described in detail in *Section 6.7.1, "Set Example"*.

The ACMSMGMT_LIST_LOG_1 procedure returns a linked list of Remote Manager log entries, ordered by time. The records to be returned are determined by the fields specified in the `log_sel_struct` input argument. Records can be selected by date and time, facility, and severity. Note that only `max_rpc_return_rec` data (Parameter table field) is returned in each call. The end of data is signaled by the status field (see the following example). If the end of data is not signaled, repeated calls are needed to fetch all matching records.

The return record is a union containing either a failure code or the first record in the list:

```
struct log_data_list {
        int status;
        union {
                int rc;
                log_list list;
        } log_data_list_u;
};
```

To determine the status of the call and the contents of the return record, first check the status field. The following are possible values in the status field:

- MGMT_FAIL

  The call has failed and the `rc` field contains a specific error code describing the failure.

- MGMT_NOMORE_DATA

  There are no more records in the file. The linked list may or may not contain the final records, depending on whether any more records matched the selection criteria.

- MGMT_SUCCESS

  The call completed successfully. More records exist than were returned in the call.

The ACMSMGMT_LIST_LOG_1 procedure returns *n* records per call, where *n* is determined by the Remote Manager parameter field `max_rpc_return_recs`. Therefore, repeated calls may be necessary to retrieve all records that match the selection criteria. Context is not maintained by the server between calls; the selection criteria are evaluated on each call by the Remote Manager. Following the initial call, callers should place the correct time value in the `since_time` field of the `log_sel_struct` input argument, as well as the correct `dup_count` value in order to have the chronologically next *n* records returned.

This procedure does not require the ACMS run-time system to execute.

Example

```
int list_log_data(int client_id,CLIENT *cl)
{

   int  skip_rec          = 0;
   char null_time_str[24] = "";
```

```
   char first_of_jan[24]  = "01-JAN-1998 00:00:00.00";
   char file_spec[]       = ""; /* use default, i.e. active log file */
   char time_cache[MGMT_S_TIME_A+1];
   static struct  log_sel_struct log_rec;
   log_data_list  *log;
   log_link  *nl;
   int status;

   /* Initialize log selection data */

   log_rec.client_id   = client_id;
   log_rec.before_time = null_time_str;
   log_rec.since_time  = first_of_jan;
   log_rec.file_name   = file_spec;
   log_rec.dup_count   = -1;
   log_rec.facility    = -1; /* don't match on facility */
   log_rec.severity    = -1; /* don't match on severity */

top:

   log = acmsmgmt_list_log_1(&log_rec,cl);

   if (!log)
      return(MGMT_FAIL);

   if (log->status == MGMT_FAIL) {
      status = log->log_data_list_u.rc;
        xdr_free(xdr_log_data_list, log);
        free(log);
      return(status);
   }

   for (nl = log->log_data_list_u.list; nl != NULL; nl = nl->pNext) {
      if (skip_rec)
         skip_rec = 0;
      else
         printf("\n %-12s\t%-s",sname,nl->log_data.log_msg);

         memcpy(&time_cache[0],nl->log_data.log_msg,23);
         log_rec.dup_count  = nl->log_data.dup_count;
         log_rec.since_time = time_cache;
   }

   if (log->status == MGMT_NOMORE_DATA)
         printf("\n *** End of data **");

   else {
         skip_rec = 1;
         goto top;
   }

     xdr_free(xdr_log_data_list, log);
     free(log);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_LOG_1 procedure is called to fetch the contents of the RM log file. All entries since January 1, 1998 are requested. If the call succeeds, each entry is

printed out. Otherwise, an error message is displayed. This example is very similar to the one described in detail in *Chapter 6, "Management Programming Using ONC RPC"*

8.24. ACMSMGMT_LIST_PROC_1

ACMSMGMT_LIST_PROC_1

ACMSMGMT_LIST_PROC_1 — ACMS Remote Manager clients call this procedure to obtain a list of ACMS processes running on a particular node, along with some collection state information for each process.

Format

```
proc_data_list *acmsmgmt_list_proc_1 (sub_id_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Proc_data_list			
Access	Write			
Mechanism	By reference			
Usage	Pointer to a record that contains a union consisting of either a failure code or a pointer to a structure of type `proc_link`, which contains the start of a linked list of records. The following are the contents of this union:			
	rc			
	Type	Integer		
	Access	Write		
	Mechanism	By value		
	Usage	Failure return code.		
	list			
	Type	Proc_list		
	Access	Write		
	Mechanism	By reference		
	Usage	Start of linked list. Pointer to a structure of process data, and a forward pointer to the next node in the linked list. The following are the contents of this structure:		
		pNext		
		Type	Proc_list	
		Access	Write	
		Mechanism	By value	
		Usage	Start of linked list. Pointer to a structure of type `user_list`.	
		proc_data		
		Type	Proc_rec	
		Access	Write	
		Mechanism	By reference	
		Usage	The data describing the process. This record contains the following fields:	
			record_state	
			Type	Integer
			Access	Write
			Mechanism	By value
			Usage	The current state of the record. Will be either

	MGMT_VALID or MGMT_INACTIVE.
entity_type	
Type	Integer
Access	Write
Mechanism	By value
Usage	The type of ACMS entity the process is.
pid	
Type	Integer
Access	Write
Mechanism	By value
Usage	OpenVMS Process ID.
process_name	
Type	Null-terminated string
Access	Write
Mechanism	By reference
Usage	The OpenVMS process name.
class_states	
Type	Array of integers
Size:	5
Access	Write
Mechanism	By value
Usage	An array of integers. Each array element represents the collection state for a class. Positions are: • 0: ID • 1: CONFIG • 2: RUNTIME • 3: POOL • 4: ERROR

Description

The ACMSMGMT_LIST_PROC_1 procedure returns a linked list of processes that a particular Remote Manager is aware of. The Remote Manager builds this list from the various ACMS Entity tables. For each process, the Remote Manager populates a `proc_data` record.

Note that some entity tables may contain rows with inactive data, that is, data for processes that are no longer active. The data in these rows may or may not be interesting to the caller. To distinguish active and inactive processes, the Remote Manager sets the `record_state` field to MGMT_VALID for active processes and to MGMT_INACTIVE for inactive processes. The caller is responsible for checking this field and taking appropriate action.

The `collection_states` field is a simple array of five integers. Each array element contains either a 1 (if the collection class is enabled) or a 0 (if the collection class is disabled). Array elements are positional, as described in the Return Value section.

Like other procedures that return linked lists, the return parameter is a union containing either a failure status code or a linked list of records.

To determine the status of the call and the contents of the return record, first check the status field. The following are possible values in the status field:

- MGMT_FAIL

  The call has failed and the `rc` field contains a specific error code describing the failure.

- MGMT_SUCCESS

  The call completed successfully. All user records have been returned.

If the status field value is MGMT_SUCCESS, a linked list has been returned. The linked list contains a structure containing the process data, and a forward pointer. By following the forward pointer, all the records in the list can be retrieved.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Example

```
int list_process_data(int client_id,CLIENT *cl)
{

   proc_data_list  *proc;
   proc_link     *nl;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;

   proc = acmsmgmt_list_proc_1(&sub_rec,cl);

   if (!proc) {
      printf("\n RPC Call to get Process data failed");
      return(MGMT_FAIL);
   }
```

```
   if ((proc->status != MGMT_SUCCESS) && (proc->status !=
 MGMT_NOMORE_DATA)) {
      printf("\n Call to get Process data failed, returning status code
 %d",
             proc->proc_data_list_u.rc);
      status = proc->proc_data_list_u.rc;
        xdr_free(xdr_proc_data_list, proc);
        free(proc);
      return(status);
   }

   for (nl = proc->proc_data_list_u.list; nl != NULL; nl = nl->pNext) {
       if (nl->proc_data.record_state == MGMT_INACTIVE)
          printf("\n INACTIVE ");
       else
          printf("\n          ");
       printf(" PID: %8X  Process Name: %s",nl->proc_data.pid,
              nl->proc_data.process_name);
   }

   printf("\n End of data");
     xdr_free(xdr_proc_data_list, proc);
     free(proc);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_PROC_1 procedure is called to fetch information about collection states from all processes accessible to the Remote Manager. If the call succeeds, the name of the process, along with its state is displayed (inactive processes have that string printed before the process name). Otherwise, an error message is displayed.

8.25. ACMSMGMT_LIST_SERVER_1

ACMSMGMT_LIST_SERVER_1

ACMSMGMT_LIST_SERVER_1 — ACMS Remote Manager clients call this procedure to obtain a list of procedure server table (Server table) entries.

Format

```
ser_data_list *acmsmgmt_list_server_1(ser_sel_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Ser_sel_struct
Access	Read
Mechanism	By reference
Usage	Structure that contains client information and procedure server selection criteria. The structure contains the following fields.
	client_id

	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	appl_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	A pointer to an application name. The name may contain wildcard characters (*, !). Specify in all uppercase characters.
	server_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	A pointer to a procedure server name. The name may contain wildcard characters (*, !). Specify in all uppercase characters.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Ser_data_list
Access	Write
Mechanism	By reference

Usage	Pointer to a record that contains a union consisting of either a failure code or a pointer to a structure of type `ser_link`, which contains the start of a linked list of records. The following are the contents of this union:		
	rc		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	list		
	Type	Ser_list	
	Access	Write	
	Mechanism	By reference	
	Usage	Start of linked list. Pointer to a structure containing an EXC table record, and a forward pointer to the next node in the linked list. The following are the contents of this structure:	
		pNext	
		Type	Ser_list
		Access	Write
		Mechanism	By value
		Usage	Start of linked list. Pointer to a structure of type `coll_list`.
		ser_data	
		Type	Ser_rec_r
		Access	Write
		Mechanism	By reference
		Usage	Server table row. Server table fields are described in *Section 9.12, "Server Table"*.

Description

The ACMSMGMT_LIST_SER_1 procedure returns a linked list of Server table rows. All matching Server table rows are returned in each call. Matching is performed first on application name, and then on server name. Therefore, all matching servers for all matching applications are returned.

Entire table rows are returned. See *Section 9.12, "Server Table"* for a description of the fields in the `ser_rec_r` structure.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Rows in the EXC table are subject to reuse. Rows are assigned round-robin, and are not cleared until they have been reassigned. Therefore, some rows may contain data for inactive EXCs. The

Remote Manager attempts to retrieve server information for inactive EXCs. It is the caller's responsibility to examine the `record_state` field to determine whether this row belongs to an active (`record_state` field is MGMT_VALID) or inactive (`record_state` field is MGMT_INACTIVE) EXC, and to process the row accordingly.

Example

```
int list_ser_data(int client_id,CLIENT *cl)
{

   static char c_all_appls[2] = "*";
   ser_data_list     *ser_data;
   ser_link      *nl;
   static struct ser_sel_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;
   sub_rec.appl_name = c_all_appls;
   sub_rec.server_name = c_all_appls;

   ser_data = acmsmgmt_list_server_1(&sub_rec,cl);

   if (!ser_data) {
      printf("\n RPC Call to get Server data failed");
      return(MGMT_FAIL);
   }

   if (ser_data->status == MGMT_FAIL)  {
      if (ser_data->ser_data_list_u.rc == MGMT_NOMORE_DATA) {
         printf("\n No SERVER data found");
           xdr_free(xdr_ser_data_list, ser_data);
           free(ser_data);
         return(MGMT_FAIL);
       }
      printf("\n Call to get Server data failed, returning status code %d",
             ser_data->ser_data_list_u.rc);
      status = ser_data->ser_data_list_u.rc;
        xdr_free(xdr_ser_data_list, ser_data);
        free(ser_data);
      return(status);
   }

   if (ser_data->status == MGMT_WARN)
      printf("\n ** Warning, some data may be from inactive processes **");

   for (nl = ser_data->ser_data_list_u.list; nl != NULL; nl = nl->pNext) {
       if (nl->ser_data.record_state == MGMT_INACTIVE)
          printf("\n INACTIVE ");
       else
          printf("\n          ");
       printf(" Application : %-32s Server: %-s",
              nl->ser_data.appl_name,
              nl->ser_data.server_name);
   }

   printf("\n End of data");
     xdr_free(xdr_ser_data_list, ser_data);
```

```
      free(ser_data);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_SERVER_1 procedure is called to fetch the contents of the Server tables for all applications on the target node. If the call succeeds, the state of the server (if inactive), the name of the application it belongs to, and the name of the server are displayed for each table row returned. Otherwise, an error message is displayed.

8.26. ACMSMGMT_LIST_TG_2

ACMSMGMT_LIST_TG_2

ACMSMGMT_LIST_TG_2 — ACMS Remote Manager clients call this procedure to obtain a list of Task Group table entries.

Format

```
tg_data_list_2 *acmsmgmt_list_tg_2(tg_sel_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Tg_sel_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains client information and task group selection critera. The structure contains the following fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	appl_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference

	Usage	A pointer to an application name. The name may contain wildcard characters (*, !). Specify in all uppercase characters.
	tg_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	A pointer to a task group name. The name may contain wildcard characters (*, !). Specify in all uppercase characters.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Tg_data_list_2	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting of either a failure code or a pointer to a structure of type `tg_link_2`, which contains the start of a linked list of records. The following are the contents of this union:	
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	list	
	Type	Tg_list_2
	Access	Write
	Mechanism	By reference
	Usage	Start of linked list. Pointer to a structure containing a Task Group table record, and a forward pointer to the next node in the linked list. The following are the contents of this structure:

		pNext	
		Type	Tg_list_2
		Access	Write
		Mechanism	By value
		Usage	Start of linked list. Pointer to a structure of type `coll_list`.
		tg_data	
		Type	Tg_rec_r_2
		Access	Write
		Mechanism	By reference
		Usage	Task Group table row. Task Group table fields are described in *Section 9.13, "Task Group Table"*.

Description

The ACMSMGMT_LIST_TG_2 procedure returns a linked list of Task Group table rows. All matching Task Group table rows are returned in each call. Matching is performed first on the application name, and then on the task group name. Therefore, all matching task groups for all matching applications are returned.

Entire table rows are returned. See *Section 9.13, "Task Group Table"* for a description of the fields in the `tg_rec_r` structure.

If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Rows in the EXC table are subject to reuse. Rows are assigned round-robin, and are not cleared until they have been reassigned. Therefore, some rows may contain data for inactive EXCs. The Remote Manager will attempt to retrieve task group information for inactive EXCs. It is the caller's responsibility to examine the `record_state` field to determine whether this row belongs to an active (`record_state` field is MGMT_VALID) or inactive (`record_state` field is MGMT_INACTIVE) EXC, and to process the row accordingly.

Example

```
int list_group_data(int client_id,CLIENT *cl)
{

   static char c_all_appls[2] = "*";
   tg_data_list_2    *tg_data;
   tg_link_2      *nl;
   static struct tg_sel_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;
   sub_rec.appl_name = c_all_appls;
   sub_rec.tg_name = c_all_appls;

   tg_data = acmsmgmt_list_tg_2(&sub_rec,cl);
```

```
   if (!tg_data) {
      printf("\n RPC Call to get Task Group data failed");
      return(MGMT_FAIL);
   }

   if (tg_data->status == MGMT_FAIL)  {
      if (tg_data->tg_data_list_2_u.rc == MGMT_NOMORE_DATA) {
         printf("\n No GROUP data found");
           xdr_free(xdr_tg_data_list_2, tg_data);
           free(tg_data);
         return(MGMT_FAIL);
      }
      printf("\n Call to get Task Group data failed, returning status code
             %d",tg_data->tg_data_list_2_u.rc);
      status = tg_data->tg_data_list_2_u.rc;
        xdr_free(xdr_tg_data_list_2, tg_data);
        free(tg_data);
      return(status);
   }

   if (tg_data->status == MGMT_WARN)
      printf("\n ** Warning, some data may be from inactive processes **");

   for (nl = tg_data->tg_data_list_2_u.list; nl != NULL; nl = nl->pNext) {
       if (nl->tg_data.record_state == MGMT_INACTIVE)
          printf("\n INACTIVE ");
       else
          printf("\n          ");
       printf(" Application: %-32s  Task Group: %-s",
               nl->tg_data.appl_name,
               nl->tg_data.tg_name);
   }

   printf("\n End of data");
     xdr_free(xdr_tg_data_list_2, tg_data);
     free(tg_data);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_TG_1 procedure is called to fetch the contents of the Task Group tables for all applications on the target node. If the call succeeds, the state of the task group (if inactive), the name of the application it belongs to, and the name of the task group are displayed for each table row returned. Otherwise, an error message is displayed.

8.27. ACMSMGMT_LIST_TRAP_1

ACMSMGMT_LIST_TRAP_1

ACMSMGMT_LIST_TRAP_1 — ACMS Remote Manager clients call this procedure to obtain a list of Trap table entries.

Format

```
trap_data_list *acmsmgmt_list_trap_1(sub_id_struct *sub_id_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct *	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Trap_data_list	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a pointer to a structure of type `trap_list`, which contains the start of a linked list of records. The following are the contents of this union:	
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.

	list		
	Type	Trap_list	
	Access	Write	
	Mechanism	By reference	
	Usage	Start of linked list. Pointer to a structure of trap table rows, and a forward pointer to the next node in the linked list. The following are the contents of this structure:	
		pNext	
		Type	Proc_list
		Access	Write
		Mechanism	By value
		Usage	Start of linked list. Pointer to a structure of type `trap_list`.
		trap_data	
		Type	Trap_rec
		Access	Write
		Mechanism	By reference
		Usage	Trap table row. Trap table fields are described in *Section 9.14, "Trap Table"*.

Description

The `acmsmgmt_list_trap_1` procedure returns a linked list of Trap table rows. All Trap table rows are returned in each call. Records are returned sequentially from the table, beginning at the start of the table.

Entire table rows are returned. See *Section 9.14, "Trap Table"* for a description of the fields in the `trap_rec` structure.

This procedure does not require the ACMS run-time system in order to execute.

Example

```
int list_trap_data(int client_id,CLIENT *cl)
{

   char c_states[2][9] = {"enabled","disabled"};
   char c_entities[10][9] = {"unknown","*","acc","tsc","qti","cp","exc",
                             "server","group","mgr"};
   char c_classes[6][8] = {"*","id","config","runtime","pool","error"};
   char c_trap_params[2][15] = {"exists","event severity"};

   trap_data_list  *trap;
   trap_link       *nl;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;
```

```
   trap = acmsmgmt_list_trap_1(&sub_rec,cl);

   if (!trap) {
      printf("\n RPC Call to get Trap data failed");
      return(MGMT_FAIL);
   }

   if (trap->status != MGMT_SUCCESS)  {
      printf("\n Call to get Trap data failed, returning status code %d",
             trap->trap_data_list_u.rc);
      status = trap->trap_data_list_u.rc;
        xdr_free(xdr_trap_data_list, trap);
        free(trap);
      return(status);
   }

   for (nl = trap->trap_data_list_u.list; nl != NULL; nl = nl->pNext) {
       printf("\n Entity: %-9s  Name: %-32s  Param: %-15s  Trap Min: %d
              Trap Max: %d",
              c_entities[nl->trap_data.entity_type],
              nl->trap_data.entity_name,
              c_trap_params[nl->trap_data.param_to_trap],
              nl->trap_data.min,
              nl->trap_data.max);
   }

   printf("\n End of data");
     xdr_free(xdr_trap_data_list, trap);
     free(trap);
   return(0);
}
```

In the preceding example, the `acmsmgmt_list_trap_1` procedure is called to fetch the contents of the Trap table. If the call succeeds, the `entity_type`, `entity_name`, `parameter`, `trap_min`, and `trap_max` fields are displayed for each row in the table. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.28. ACMSMGMT_LIST_USERS_1

ACMSMGMT_LIST_USERS_1

ACMSMGMT_LIST_USERS_1 — ACMS Remote Manager clients call this procedure to obtain information about users attached to a Remote Manager server on a local or remote node.

Format

```
user_data_list *acmsmgmt_list_users_1 (sub_id_struct *sub_rec, CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct

Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	User_data_list	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting of either a failure code or a pointer to a structure of type `user_link`, which contains the start of a linked list of records. The following are the contents of this union:	
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	list	
	Type	User_list
	Access	Write

Mechanism	By reference			
Usage	Start of linked list. Pointer to a structure of user data, and a forward pointer to the next node in the linked list. The following are the contents of this structure:			
	pNext			
	Type	User_list		
	Access	Write		
	Mechanism	By value		
	Usage	Start of linked list. Pointer to a structure of type `user_list`.		
	user_data			
	Type	User_rec		
	Access	Write		
	Mechanism	By reference		
	Usage	The data describing the user. This record contains the following fields:		
		client_id		
		Type	Integer	
		Access	Write	
		Mechanism	By value	
		Usage	Integer value containing the client ID for the user.	
		reserved		
		Type	Integer	
		Access	Write	
		Mechanism	By value	
		Usage	Reserved for VSI use.	
		gid		
		Type	Word	
		Access	Write	
		Mechanism	By value	
		Usage	UIC group identifier.	
		uid		
		Type	Word	

Access	Write
Mechanism	By value
Usage	UIC user identifier.
proxy_gid	
Type	Word
Access	Write
Mechanism	By value
Usage	UIC group identifier of the proxy user, if proxy is being used.
proxy_uid	
Type	Word
Access	Write
Mechanism	By value
Usage	UIC user identifier of the proxy user, if proxy is being used.
node-name	
Type	Null-terminated string
Access	Write
Mechanism	By reference
Usage	Pointer to a null-terminated string containing the name of the node from which the user logged in.
expires	
Type	Null-terminated string
Access	Write
Mechanism	By reference
Usage	Time the user's credentials expire. Time is expressed in OpenVMS ASCII time format

				(*DD-MMM-YYYY HH:MM:SS.hh*).
			user-name	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	Pointer to a null-terminated string containing the user name.
			rights	
			Type	Array of integers
			Access	Write
			Mechanism	By value
			Usage	ACMS management rights identifiers held by the user.
			proxy_flag	
			Type	Integer
			Access	Write
			Mechanism	By value
			Usage	Indicates whether the record is for a proxy user (`proxy_flag` = 1) or is not for a proxy user (`proxy_flag` = 0).

Description

The ACMSMGMT_LIST_USERS_1 procedure returns a linked list of users who are logged in to a particular Remote Manager. All user records are returned on each call to this procedure.

Like other procedures that return linked lists, the return parameter is a union containing either a failure status code or a linked list of records.

To determine the status of the call and the contents of the return record, first check the status field. The following are possible values for the status field:

- MGMT_FAIL

  The call has failed, and the `rc` field contains a specific error code describing the failure.

- MGMT_SUCCESS

  The call completed successfully. All user records have been returned.

If the status field is equal to MGMT_SUCCESS, a linked list has been returned. The linked list contains a structure containing the user data and a forward pointer. By following the forward pointer, all the records in the list can be retrieved.

This procedure does not require the ACMS run-time system to execute.

Example

```
int list_users_data(int client_id,CLIENT *cl)
{

   user_data_list  *user;
   user_link    *nl;
   static struct sub_id_struct sub_rec;
   int status;

   sub_rec.client_id = client_id;

   user = acmsmgmt_list_users_1(&sub_rec,cl);

   if (!user) {
      printf("\n RPC Call to get User data failed");
      return(MGMT_FAIL);
   }

   if ((user->status != MGMT_SUCCESS) && (user->status !=
 MGMT_NOMORE_DATA)) {
      printf("\n Call to get User data failed, returning status code %d",
             user->user_data_list_u.rc);
      status = user->user_data_list_u.rc;
        xdr_free(xdr_user_data_list, user);
        free(user);
      return(status);
   }

   for (nl = user->user_data_list_u.list; nl != NULL; nl = nl->pNext)
       printf("\n User %s is logged in from node %s",nl->user_data.uname,
              nl->user_data.nodename);

   printf("\n End of data");
     xdr_free(xdr_user_data_list, user);
     free(user);
   return(0);
}
```

In the preceding example, the ACMSMGMT_LIST_USERS_1 procedure is called to fetch information about the users who have logged in to the Remote Manager. If the call succeeds, the name of the user and the node they logged in from are displayed. Otherwise, an error message is displayed. Note that the name displayed is the name by which the user is known to the server, and may be a proxy account. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.29. ACMSMGMT_REPLACE_SERVER_1

ACMSMGMT_REPLACE_SERVER_1

ACMSMGMT_REPLACE_SERVER_1 — This procedure requests the Remote Manager to replace an ACMS procedure server in an ACMS application on the same node on which the Remote Manager is running.

Format

```
cmd_output_rec *acmsmgmt_replace_server_1(ser_sel_struct *sub_rec,CLIENT
 *cl)
```

Parameters

sub_rec

Type	Ser_sel_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains client information and procedure server selection criteria. The structure contains the following fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	appl_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	A pointer to an application name. The name may contain wildcard characters (*, !). Specify in all uppercase characters.
	server_name	

	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	A pointer to a procedure server name. The name may contain wildcard characters (*, !). Specify in all uppercase characters.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec`, which points to a linked list containing status messages. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data, data_warn	
	Type	Cmd_rec
	Access	Write
	Mechanism	By value

	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:		
		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests to have an ACMS procedure server replaced (stopped and started) in an application that is running on the same node on which the Remote Manager is running. The combination of `appl_name` and `server_name` in the input record determines which server will be replaced.

This call executes synchronously. It does not return to the caller until the attempt to replace the server is complete. Any messages associated with an unsuccessful replacing of the server are returned in the `cmd_output` linked list.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of both structures will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of both structures will be MGMT_SUCCESS. There are no status messages associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. There are no status messages returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int replace_server(int client_id,CLIENT *cl)
 {

   dcl_link    *nl;
   static char c_name_all[2] = "*";
   static char vr_read_server[] = "VR_READ_SERVER";
   static struct ser_sel_struct sub_rec;
   static cmd_output_rec *ret_struct;

   sub_rec.client_id = client_id;
   sub_rec.appl_name = c_name_all;
   sub_rec.server_name = vr_read_server;

   ret_struct = acmsmgmt_replace_server_1(&sub_rec,cl);

   if (!ret_struct) {
      printf("\n Call to replace server failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

   if (ret_struct->status != MGMT_WARN) {
      printf("\nCall to replace procedure server %s failed",
             sub_rec.server_name);
      return(MGMT_FAIL);
   }

   printf("\n Call to replace procedure server %s completed with warnings
 or
          errors",sub_rec.server_name);

   for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
        nl= nl->pNext)
      printf("\n %s",nl->dcl_msg);
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
      return(MGMT_FAIL);
   }

   else {
      printf("\nCall to replace procedure server %s was executed",
             sub_rec.server_name);
      for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
           nl = nl->pNext)
      printf("\n %s",nl->dcl_msg);
   }
      xdr_free(xdr_cmd_output_rec, ret_struct);
      free(ret_struct);
   return(0);
}
```

In the preceding example, the `acmsmgmt_replace_server_1` procedure is called to replace servers named VR_READ_SERVER in any application on the target node. If the call succeeds, all VR_READ_SERVER servers are replaced (stopped and started). Otherwise, any error messages

associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.30. ACMSMGMT_RESET_LOG_1

ACMSMGMT_RESET_LOG_1

ACMSMGMT_RESET_LOG_1 — This procedure requests the Remote Manager to close the current version of its log file and open a new one.

Format

```
int *acmsmgmt_reset_log_1(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to a status code containing a success or failure status code. MGMT_SUCCESS indicates success. Other values indicate failure.

Description

This procedure requests the Remote Manager to close the currently open version of its log and to open a new one. All subsequent log entries are posted to the new version, and the old version can be safely removed.

Example

```
int reset_log_data(int client_id,CLIENT *cl)
 {

   static struct sub_id_struct sub_rec;
   int *status;

   sub_rec.client_id = client_id;

   status = acmsmgmt_reset_log_1(&sub_rec,cl);

   if (!status) {
      printf("\n Call to reset log failed");
      return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\n Call to reset log failed with status %d",*status);
        free(status);
      return(MGMT_FAIL);
   }
   else
      printf("\n Call to reset log completed");
        free(status);
   return(0);
}
```

In the preceding example, the `acmsmgmt_reset_log_1` procedure is called to close the current Remote Manager log and to open a new one. If the call succeeds, a success message is displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.31. ACMSMGMT_RESET_ERR_2

ACMSMGMT_RESET_ERR_2

ACMSMGMT_RESET_ERR_2 — This procedure requests the Remote Manager to close the current version of the error log file and open a new one.

Format

```
int *acmsmgmt_reset_err_2(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write

Mechanism	By reference
Usage	Pointer to a status code containing a success or failure status code. MGMT_SUCCESS indicates success. Other values indicate failure.

Description

This procedure requests the Remote Manager to close the currently open version of the error log and to open a new one. All subsequent erro log entries are posted to the new version, and the old version can be safely removed.

Example

```
int reset_err_data(int client_id,CLIENT *cl)
 {

   static struct sub_id_struct sub_rec;
   int *status;

   sub_rec.client_id = client_id;

   status = acmsmgmt_reset_err_2(&sub_rec,cl);

   if (!status) {
      printf("\n Call to reset log failed");
      return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\n Call to reset log failed with status %d",*status);
        free(status);
      return(MGMT_FAIL);
   }
   else
      printf("\n Call to reset log completed");
        free(status);
   return(0);
}
```

In the preceding example, the ACMSMGMT_RESET_ERR_2 procedure is called to close the current error log and to open a new one. If the call succeeds, a success message is displayed. Otherwise, an error message is displayed.

8.32. ACMSMGMT_SAVE_ERR_FILTER_2

ACMSMGMT_SAVE_ERR_FILTER_2

ACMSMGMT_SAVE_ERR_FILTER_2 — This procedure saves the current error filter records to an error filter file.

Format

```
int *acmsmgmt_save_err_filter_2(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

set_struct

Type	Err_filter_config_rec_r_2	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization and error filter record information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	err_filter_file_name	
	Type	file_spec
	Access	Read
	Mechanism	By value
	Usage	Specifies the OpenVMS file specification for the error filter file.
	err_msg_name	
	Type	String
	Access	Read
	Mechanism	By value
	Usage	Symbolic name of the error message.
	err_code	
	Type	String
	Access	Read
	Mechanism	By value
	Usage	Decimal or hexadecimal code for the error message.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to a status code containing a success or failure status code. MGMT_SUCCESS indicates success. Other values indicate failure.

Description

This procedure saves all records in the Error Filter table to the specified ASCII text file.

Example

```
int save_err_filter( int client_id, CLIENT *cl)
 {
   int *status;
   static char c_null_str[2] = "";
   static char file_spec = "sys$login:err_filter.dat";
   err_filter_config_rec_r_2 set_struct;

   set_struct.client_id = client_id;
   set_struct.err_filter_file_name  = file_spec;
   set_struct.err_msg_name = c_null_str;
   set_struct.err_code = -2;

   status = acmsmgmt_save_err_filter_file_2(&set_struct, cl);

   if (!status) {
        printf("\n Call to save error filter failed");
        return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
        printf("\n Call to save error filter failed with status %d",
               *status);
          free(status);
        return(MGMT_FAIL);
   }
   else {
        printf("\n Call to save error filter completed");
   }

  free(status);
```

```
  return(0);
}
```

In the preceding example, the ACMSMGMT_SAVE_ERR_FILTER_2 procedure is called to save all the records in the Error Filter table to the file SYS$LOGIN:ERR_FILTER.DAT. If the call succeeds, a success message is displayed. Otherwise, an error message is displayed.

8.33. ACMSMGMT_SET_ACC_2

ACMSMGMT_SET_ACC_2

ACMSMGMT_SET_ACC_2 — This procedure modifies ACMS Central Controller (ACC) Config class fields.

Format

```
acc_status_rec_2 *acmsmgmt_set_acc_2(acc_config_rec_2 *set_struct,CLIENT
 *cl)
```

Parameters

set_struct

Type	Acc_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and ACC table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	active_sw	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Indicates whether active variables should be updated

	(`active_sw` = 1). Active variables are currently in use by the ACMS system; updates to active variables take effect immediately but are not durable (that is, they do not survive a restart of the ACMS system). Not all variables are dynamic, however. Refer to *Section 9.2, "ACC Table"* and to the field descriptions in this section to determine whether a particular variable can be updated dynamically.
current_sw	
Type	Integer
Access	Read
Mechanism	By value
Usage	Indicates whether current variables should be updated (`current_sw` = 1). Current variables are those stored in the ACMSGEN file currently in use by the ACMS system and are durable (that is, they can survive a restart of the ACMS system). Updates to current variables take effect when the ACMS system is restarted.
acc_priority, audit_state, max_appl, mss_maxobj, mss_maxbuf, mss_poolsize, wsc_poolsize, tws_poolsize, twsc_poolsize	
Type	Integer
Access	Read
Mechanism	By value
Usage	Values to be updated. These fields correspond to fields of the same names in the ACC table, depending on the value of `active_sw` and `current_sw` in this record (for example, `acc_priority` will update the `acc_priority_active` field if `active_sw` is equal to 1). See *Section 9.2, "ACC Table"* for a discussion of these fields.

		Note that not all fields can be updated dynamically.
	acc_username, username_default, node_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Values to be updated. These fields correspond to fields of the same names in the ACC table, depending on the value of `active_sw` and `current_sw` in this record (for example, `username_default` will update the `username_default_active` field if `active_sw` is equal to 1). See *Section 9.2, "ACC Table"* for a discussion of these fields. Note that not all fields can be updated dynamically. In order to have any of these fields set to null (that is, ""), set the field to the string "NULL".

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Struct acc_status_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `acc_config_rec_out`, which contains status codes for each field, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer

Access	Write		
Mechanism	By value		
Usage	Failure return code.		
data, data_warn			
Type	Acc_config_rec_out		
Access	Write		
Mechanism	By value		
Usage	Structure containing fields corresponding to the fields in the `acc_config_rec` structure, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this structure:		
	acc_priority, audit_state, max_appl, mss_maxobj, mss_maxbuf, mss_poolsize, wsc_poolsize, tws_poolsize, twsc_poolsize		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Status fields corresponding to the fields in the input argument.	
	acc_username, username_default, node_name		
	Type	Null-terminated string	
	Access	Write	
	Mechanism	By reference	
	Usage	These fields contain the values that were supplied in the input argument, and can be ignored.	
	cmd_output		
	Type	Dcl_list	
	Access	Write	
	Mechanism	By reference	
	Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
		dcl_msg	
		Type	Null-terminated string
		Access	Write

			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests updates to ACMS ACC Config class fields contained in the ACC table (see *Section 9.2, "ACC Table"*). Note that the ACC table contains both active and stored values. The `active_sw` field and `current_sw` field control which fields are to be updated.

Attempting to update an active field that is nondynamic is essentially useless, since the value of the active field value will not change. For instance, calling this procedure with the `active_sw` field set to 1 and the `acc_username` field populated produces no change to the system.

Setting the `current_sw` field to 1 causes updates to be written to the current ACMSGEN file. These updates are durable, that is, they can survive a restart of the ACMS sytem, but they do not affect the active system until the system is restarted.

Both `current_sw` and `active_sw` can be set on a given call. If they are, both the active and stored values for any nonnegative or nonnull fields will be updated.

For any nonnegative integer fields, the completion status of the update is returned in the corresponding field in the return structure. For string fields, the string field value is returned regardless of the status of the call.

In order to have one of the string fields set to a null string (that is, ""), populate the field with the value NULL. To have one of the string fields ignored, pass in a null string.

Example

```
int set_acc_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[2] = "*";
   static char c_null_str[2] = "";
   static acc_config_rec_2 set_struct;
   acc_status_rec_2        *ret_struct;
   dcl_link                *nl;

   memset(&set_struct,-1,sizeof(set_struct));
   set_struct.client_id      = client_id;
   set_struct.active_sw      = 1;
   set_struct.current_sw     = 0;
   set_struct.audit_state    = MGMT_STATE_DISABLED;

   /* Have to provide a pointer for string conversions by XDR
```

```
      or it will access vio. RM will ignore any fields with
      strlen of 0 */
   set_struct.acc_username = c_null_str;
   set_struct.username_default = c_null_str;
   set_struct.node_name  = c_null_str;

   ret_struct = acmsmgmt_set_acc_2(&set_struct,cl);

   if (!ret_struct) {
      printf("\n Call to modify ACC failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS)
      printf("\n Call to modify ACC returned the following warnings or
             errors\n");
   else
      printf("\n Call to modify ACC completed\n");

   for (nl = ret_struct->acc_status_rec_2_u.data.cmd_output; nl != NULL;
        nl = nl->pNext)
      printf("\n %s",nl->dcl_msg);
        xdr_free(xdr_acc_status_rec_2, ret_struct);
        free(ret_struct);
      return(0);
}
```

In the preceding example, the `acmsmgmt_set_acc_1` procedure is called to disable system auditing on the target node. If the call succeeds, system auditing is disabled on the target node, and a success message is displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.34. ACMSMGMT_SET_AGENT_2

ACMSMGMT_SET_AGENT_2

ACMSMGMT_SET_AGENT_2 — This procedure modifies the ACMS Agent Process class attributes.

Format

```
agent_status_rec *acmsmgmt_set_agent_2(agent_config_rec
 *agent_cfg_rec,CLIENT *cl)
```

Parameters

set_struct

Type	agent_config_rec
Access	Read
Mechanism	By reference
Usage	Structure that contains the following client identification and Agent table fields.
	client_id

Type	Integer
Access	Read
Mechanism	By value
Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
current_sw	
Type	Integer
Access	Read
Mechanism	By value
Usage	Update current values flag, required to be set to 1.
pid	
Type	Integer
Access	Read
Mechanism	By reference
Usage	PID of the Agent process to be updated, required.
user1_time, user2_time, user3_time	
Type	Null-terminated String
Access	Read
Mechanism	By reference
Usage	Time fields provided for use by programmers. Pointers to character strings representing VMS time (for example, "18-NOV-2003 00:00:00.00"). If these fields are not to be set, the fields should be initialized to a null string (""). See the note at the end of this section for a discussion of these fields.
user1_data, user2_data, user3_data, user4_data, user5_data, user6_data	
Type	Integer
Access	Read

	Mechanism	By value
	Usage	Integer fields provided for use by programmers.
	astlm, biolm, bytlm, diolm, enqlm, fillm, pgflquota, tqelm, wsdefault, wsextent, wsquota	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Quota values to be updated. These fields correspond to the stored fields of the same names in the Agent table (for example, `astlm` will update `astlm_stored`). See the note at the end of this section for a discussion of these fields.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	agent_status_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting of either a failure code or a structure of type `agent_config_rec_out`, which contains status codes for each quota field, as well as a linked list of status messages associated with the update. The following are the contents of this union:	
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data, data_warn	
	Type	agent_config_rec_out

	Access	Write		
	Mechanism	By value		
	Usage	Structure containing fields corresponding to the quota fields in the `agent_config_rec` structure, as well as a linked list of status messages associated with the update. The following are the contents of this structure:		
		astlm, biolm, bytlm, diolm, enqlm, fillm, pgflquota, tqelm, wsdefault, wsextent, wsquota		
		Type	Integer	
		Access	Write	
		Mechanism	By value	
		Usage	Status fields corresponding to the quota fields in the input argument.	
		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Note

This procedure requests updates to ACMS Agent Config class fields contained in the Agent table at the end of this section. Note that the Agent table contains both active and stored quota values; however, only the stored fields can be changed.

Description

The ACMSMGMT_SET_AGENT_2 procedure sets the three user time fields and six user data fields, these fields are provided for agent developers to use as they see fit for individual agents. If the ACMS run-time system is not running when this call is issued, the Remote Manager returns the MGMT_NOT_MAPPED error code.

Example

```
int set_agent_data(int client_id,CLIENT *cl)
 {
    static char null_time[] = "";
    static agent_config_rec set_struct;
    agent_status_rec *ret_struct;
    dcl_link *nl;

    static char time1[MGMT$S_TIME_A+1] = "18-NOV-1858 00:00:00.00";
    static char time2[MGMT$S_TIME_A+1] = "19-NOV-1858 00:00:00.00";
    static char time3[MGMT$S_TIME_A+1] = "20-NOV-1858 00:00:00.00";

    // Initialize the agent config record.
    memset(&set_struct,-1,sizeof(set_struct));
    set_struct.user1_time = null_time;
    set_struct.user2_time = null_time;
    set_struct.user3_time = null_time;

    set_struct.pid     = 0x45400931;
    set_struct.client_id     = client_id;
    set_struct.current_sw     = 1;
    set_struct.astlm     = 500;

    set_struct.user1_time = time1;
    set_struct.user2_time = time2;
    set_struct.user3_time = time3;
    set_struct.user1_data = 1;
    set_struct.user2_data = 2;
    set_struct.user3_data = 3;
    set_struct.user4_data = 4;
    set_struct.user5_data = 5;
    set_struct.user6_data = 6;

    ret_struct = acmsmgmt_set_agent_2(&set_struct,cl);
    if (!ret_struct) {
        printf("\n Call to modify Agent failed");
        return(MGMT_FAIL);
    }
    if (ret_struct->status != MGMT_SUCCESS)
        printf("\n Call to modify Agent returned the following warnings or
 errors\n");
    else
        printf("\n Call to modify Agent completed\n");

    for (nl = ret_struct->agent_status_rec_u.data.cmd_output;
        nl != NULL;
        nl = nl->pNext)
      printf("\n %s",nl->dcl_msg);
```

```
      xdr_free(xdr_agent_status_rec, ret_struct);
      free(ret_struct);
    return(0);
}
```

In the preceding example, the ACMSMGMT_SET_AGENT_2 procedure is call to set the user time and data fields. If the call succeeds, a success message is displayed. Otherwise, an error message is displayed.

8.35. ACMSMGMT_SET_COLLECTION_2

ACMSMGMT_SET_COLLECTION_2

ACMSMGMT_SET_COLLECTION_2 — This procedure modifies entries in the Remote Manager Collection table. Collection table entries can also be added and deleted.

Format

```
coll_status_rec_2 *acmsmgmt_set_collection_2(coll_config_rec_2
 *set_struct,CLIENT *cl)
```

Parameters

set_struct

Type	Coll_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and collection table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	coll	
	Type	Struct coll_update_rec_r
	Access	Read
	Mechanism	By value

	Usage	Structure containing a Collection table record. Collection table fields are described in *Section 9.4, "Collection Table"*. See the Description section for information on how to initialize this record.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Struct coll_status_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `coll_update_rec_r`, which contains status codes for each field. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data_warn	
	Type	Coll_update_rec_r
	Access	Write
	Mechanism	By value
	Usage	Structure containing a Collection table record. The entries in this field contain status codes that correspond to the fields in the `coll` structure. See the Description section for a discussion of how to determine the update status for any field.

Description

This procedure requests updates to fields in the Collection table (see *Section 9.4, "Collection Table"*).

Updates to this table are not durable; that is, they do not survive a restart of the Remote Manager. To make nondynamic, permanent updates to the collection table, use the ACMSCFG utility.

Calls to this procedure must specify `entity_type`, `entity_name`, and `collection_class`. These fields must exactly match an existing record in the Collection table for the update to be applied. *Table 8.1, "Collection Classes"* and *Table 8.4, "Entity Types"* contain symbolic values used to populate the `collection_class` and `entity_type` fields; `entity_name` is specified as a null-terminated string.

For any nonnegative fields, the completion status of the update is returned in the corresponding field in the return structure. This includes the key fields of `entity_type`, `entity_name`, and `collection_class`. If no matching record is found in the table, `entity_type` and `collection_class` contain values of MGMT_FAIL.

Updates to the collection table are processed immediately and may affect more than one ACMS process. See *Section 5.1, "Managing Data Collection"* for discussion of how the collection table affects ACMS data collection.

Example

```
int set_collection_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[2] = "*";
   static coll_config_rec_2 set_struct;
   struct coll_status_rec_2 *status_rec;

   set_struct.client_id              = client_id;
   set_struct.coll.entity_type       = MGMT_ALL;
   set_struct.coll.entity_name       = c_name_all;
   set_struct.coll.collection_class  = MGMT_CLASS_RT;
   set_struct.coll.collection_state  = MGMT_STATE_ENABLED;

   status_rec = acmsmgmt_set_collection_2(&set_struct,cl);
   if (!status_rec) {
      printf("\n Call to modify collection failed");
      return(MGMT_FAIL);
   }

   if (status_rec->status == MGMT_WARN) {
      printf("\nThe following updates failed: ");
      if (status_rec->coll_status_rec_2_u.data_warn.entity_type  ==
MGMT_FAIL)
         printf("\n     Record not found");
      if (status_rec->coll_status_rec_2_u.data_warn.collection_state
          == MGMT_FAIL)
         printf("\n     coll_state invalid");
      if (status_rec->coll_status_rec_2_u.data_warn.storage_state ==
MGMT_FAIL)
         printf("\n     storage_state invalid");
      if (status_rec->coll_status_rec_2_u.data_warn.storage_interval
          == MGMT_FAIL)
         printf("\n     storage_interval invalid");
```

```
     }

   else if (status_rec->status != MGMT_SUCCESS) {
      printf("\n Call to modify collection failed with status
             %d",status_rec->coll_status_rec_2_u.rc);
        xdr_free(xdr_coll_status_rec_2, status_rec);
        free(status_rec);
      return(MGMT_FAIL);
   }
   else
      printf("\nCall to modify collection was executed");
        xdr_free(xdr_coll_status_rec_2, status_rec);
        free(status_rec);
      return(0);
}
```

In the preceding example, the `acmsmgmt_set_collection_1` procedure is called to set the collection state to ENABLED for the Collection table record with entity of * (all), name of * (all), and class of RUNTIME. If the call set the collection state to ENABLED for the Collection table record with an entity of * (all), a name of * (all), and class of RUNTIME. If the call succeeds, the new value will be stored in the Collection table, all ACMS processes on the target node will begin collecting run-time data, and a success message will be displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.36. ACMSMGMT_SET_CP_2

ACMSMGMT_SET_CP_2

ACMSMGMT_SET_CP_2 — This procedure modifies the ACMS Central Process (CP) class attributes.

Format

```
cp_status_rec_2 *acmsmgmt_set_cp_2(cp_config_rec_2 *cp_cfg_rec,CLIENT *cl)
```

Parameters

cp_cfg_rec_2

Type	Cp_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and collection table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID

		must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	astlm, biolm, bytlm, current_sw, diolm, enqlm, fillm, pgflquota, tqelm, wsdefault, wsextent, wsquota	
	Type	Cp_rec_r
	Access	Read
	Mechanism	By value
	Usage	Structure containing a CP table record. CP table fields are described in *Section 9.5, "CP Table"*. See the Description section for information on how to initialize this record.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cp_status_rec_2	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a record that contains a union consisting of either a failure code or a structure of type `config_rec_out_2`, which contains status codes for each field. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data_warn	

	Type	Config_rec_out_2
	Access	Write
	Mechanism	By value
	Usage	Structure containing a CP table record. The entries in this field contain status codes that correspond to the fields in the cp structure. See the Description section for a discussion of how to determine the update status for any field.

Description

This procedure requests updates to fields in the CP table.

Updates to this table are not durable; that is, they do not survive a restart of the Remote Manager. To make nondynamic, permanent updates to the collection table, use the ACMSCFG utility.

Updates to the CP table are processed immediately and may affect more than one ACMS process.

Example

```
int set_cp_data(int client_id,CLIENT *cl)
 {
   cp_config_rec_2  set_struct;
   cp_status_rec_2 *ret_struct;
   dcl_link    *nl;

   memset(&set_struct,-1,sizeof(set_struct));

   set_struct.client_id      = client_id;
   set_struct.current_sw     = 1;
   set_struct.astlm      = 500;

   ret_struct = acmsmgmt_set_cp_2(&set_struct,cl);

   if (!ret_struct) {
       printf("\n Call to modify CP failed");
        return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS)
      printf("\n Call to modify CP returned the following warnings or
        errors\n");
   else
      printf("\n Call to modify CP completed\n");

   for (nl = ret_struct->cp_status_rec_2_u.data.cmd_output; nl != NULL;
        nl = nl->pNext)
     printf("\n %s",nl->dcl_msg);
       xdr_free(xdr_cp_status_rec_2, ret_struct);
       free(ret_struct);
   return(0);
}
```

In the preceding example, the ACMSMGMT_SET_CP_2 procedure is called. Otherwise, an error message is displayed.

8.37. ACMSMGMT_SET_EXC_2

ACMSMGMT_SET_EXC_2

ACMSMGMT_SET_EXC_2 — This procedure modifies the ACMS Application Execution Controller (EXC) Config class attributes.

Format

```
exc_status_rec_2 *acmsmgmt_set_exc_2(exc_config_rec_2 *set_struct,CLIENT
 *cl)
```

Parameters

set_struct

Type	Exc_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and EXC table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value of `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	appl_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Name of the application to update.
	audit_state, max_tasks, sp_mon_interval, max_servers, server_proc_dmpflag, transaction_timeout	

	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Values to be updated. These fields correspond to the active fields of the same names in the EXC table (for example, `max_tasks` will update `max_tasks_active`). See *Section 9.7, "EXC Table"* for a discussion of these fields. All fields in this record can be updated dynamically. Stored values cannot be changed for EXCs (application must be rebuilt).

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Struct exc_status_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `exc_config_rec_out`, which contains status codes for each field, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data, data_warn	
	Type	Exc_config_rec_out
	Access	Write

	Mechanism	By value		
	Usage	Structure containing fields corresponding to the fields in the `exc_config_rec` structure, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this structure:		
		audit_state, max_tasks, sp_mon_interval, max_servers, server_proc_dmpflag, transaction_timeout		
		Type	Integer	
		Access	Write	
		Mechanism	By value	
		Usage	Status fields corresponding to the fields in the input argument.	
		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests updates to ACMS EXC Config class fields contained in the EXC table (see *Section 9.7, "EXC Table"*). Note that the EXC table contains both active and stored values; however, only the active fields can be changed. In order to change the stored values, the application must be rebuilt.

For any nonnegative integer fields, the completion status of the update is returned in the corresponding field in the return structure.

Example

```
int set_exc_data(int client_id,CLIENT *cl)
 {

   static char vr_appl[] = "VR_APPL";
   static exc_config_rec_2 set_struct;
   exc_status_rec_2        *ret_struct;
   dcl_link                *nl;

   memset(&set_struct,-1,sizeof(set_struct));
   set_struct.client_id      = client_id;
   set_struct.audit_state    = MGMT_STATE_DISABLED;
   set_struct.appl_name      = vr_appl;

   ret_struct = acmsmgmt_set_exc_2(&set_struct,cl);

   if (!ret_struct) {
      printf("\n Call to modify EXC failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS)
      printf("\n Call to modify EXC returned the following warnings or
            errors\n");
   else
      printf("\n Call to modify EXC completed\n");

   for (nl = ret_struct->exc_status_rec_2_u.data.cmd_output; nl != NULL; nl
 =
        nl->pNext)
       printf("\n %s",nl->dcl_msg);
         xdr_free(xdr_exc_status_rec_2, ret_struct);
         free(ret_struct);
       return(0);
}
```

In the preceding example, the `acmsmgmt_set_exc_1` procedure is called to disable application auditing for the application VR_APPL on the target node. If the call succeeds, the VR_APPL no longer writes application auditing messages, and a success message is displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.38. ACMSMGMT_SET_INTERFACE_1

ACMSMGMT_SET_INTERFACE_1

ACMSMGMT_SET_INTERFACE_1 — This procedure modifies the status of a Remote Manager interface. Either the SNMP or RPC interface can be modified. Note: ACMS Remote Manager will not allow the RPC interface to be DISABLED through this call. The only way to disable the RPC interface dynamically is to use the SNMP interface.

Format

```
int *acmsmgmt_set_interface_1(interface_config_rec *if_cfg_rec,CLIENT *cl)
```

Parameters

if_cfg_rec

Type	Interface_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and interface configuration fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	interface_type	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Indicates the interface to be modified. *Table 8.2, "Interface Types"* shows the valid symbolic values for interface types.
	state	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Indicates desired state of the interface. *Table 8.3, "Enable States"* shows the valid symbolic values for the allowable states.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to status value returned. If NULL or MGMT_SUCCESS, the RPC has succeeded. If neither NULL nor MGMT_SUCCESS, the procedure call failed and the value pointed to is the reason for failure.

Description

This procedure modifies the status of an interface. Interfaces can be enabled (that is, requested to start) or disabled (that is, requested to stop) by setting the state field in `if_cfg_rec` to the appropriate value.

Note that it is not possible to use the RPC interface to enable the RPC interface. In order to use the RPC interface, it must already be enabled. In order to start the RPC interface, either use the SNMP interface, or use the ACMSCFG utility to configure the RPC interface to be enabled when the Remote Manager starts up.

It is also not possible to use this call to disable the RPC interface. The ACMS Remote Manager does not allow an interface to disable itself. The only way to disable the RPC interface dynamically is to use the SNMP interface.

Example

```
int set_interface_data(int client_id,CLIENT *cl)
 {

   static interface_config_rec set_struct;
   int *status;

   memset(&set_struct,-1,sizeof(set_struct));

   set_struct.client_id = client_id;
   set_struct.interface_type = MGMT_IF_SNMP;
   set_struct.state = MGMT_STATE_ENABLED;

   status = acmsmgmt_set_interface_1(&set_struct,cl);

   if (!status) {
      printf("\n Call to update SNMP interface failed");
```

```
      return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\n Call to update SNMP interface failed with status
 %d",*status);
        free(status);
      return(MGMT_FAIL);
   }
   else
      printf("\n Call to set SNMP interface completed");
        free(status);
   return(0);
}
```

In the preceding example, the `acmsmgmt_set_interface_1` procedure is called to enable the SNMP interface. If the call succeeds, the SNMP interface is running on the target node, and a success message is displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.39. ACMSMGMT_SET_PARAM_2

ACMSMGMT_SET_PARAM_2

ACMSMGMT_SET_PARAM_2 — This procedure requests updates to fields in the Remote Manager Parameter table.

Format

```
param_status_rec2 *acmsmgmt_set_param_2(param_config_rec2
 *set_struct,CLIENT *cl)
```

Parameters

set_struct

Type	Param_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and parameter configuration fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id`

		is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	params	
	Type	Struct param_rec
	Access	Read
	Mechanism	By value
	Usage	Structure containing a Parameter table record. Parameter table fields are described in *Section 9.10, "Parameter Table"*. See the Description section for information on how to initialize this record.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Struct param_status_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union either contains a failure code or a structure of type `param_rec`, which contains status codes for each field. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data, data_warn	
	Type	Param_rec
	Access	Write
	Mechanism	By value

	Usage	Structure containing a Parameter table record. The entries in this field contain status codes correspond to the fields in the `params` structure. See the Description section for a discussion of how to determine the update status for any field.

Description

This procedure requests updates to fields in the Parameter table (see *Section 9.10, "Parameter Table"*). Some field updates are dynamic; others are not. Updates to this table are not durable; that is, they do not survive a restart of the Remote Manager.

When this procedure is called, any fields with negative values are ignored. Callers should initialize any fields to a negative value, for example, -1, for which updates are not to be applied. All nonnegative fields are validated prior to being updated.

For any nonnegative fields, the completion status of the update is returned in the corresponding field in the return structure. For instance, if the `mss_coll_interval` and `max_logins` fields in the `params` structure of the `param_config_rec` are nonnegative when this procedure is called, the `mss_coll_interval` and `max_logins` field of the data or `data_warn` structures of the `param_status_rec` will contain the completion status for those updates.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of either structure is MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of either structure is MGMT_SUCCESS. There are no status messages associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. There are no status messages returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int set_param_data(int client_id,CLIENT *cl)
 {

   static param_config_rec2 set_struct;
   param_status_rec2 *ret_struct;
   int status;

   memset(&set_struct,-1,sizeof(set_struct));

   set_struct.client_id = client_id;
   set_struct.params.max_logins = 25;

   ret_struct = acmsmgmt_set_param_2(&set_struct,cl);

   if (!ret_struct) {
      printf("\n Call to modify parameters failed");
      return(MGMT_FAIL);
   }
```

```
   if (ret_struct->status != MGMT_SUCCESS) {

      if (ret_struct->status != MGMT_WARN) {
         printf("\nCall to modify parameters failed, returning %d",
                ret_struct->status);
         status = ret_struct->status;
           xdr_free(xdr_param_status_rec2, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
     }

   if (ret_struct->param_status_rec2_u.data.max_logins  != MGMT_SUCCESS)
      printf("\n max_logins specified was invalid ");
        xdr_free(xdr_param_status_rec2, ret_struct);
        free(ret_struct);
      return(MGMT_FAIL);
    }

   else
      printf("\n Call to set params completed");
        xdr_free(xdr_param_status_rec2, ret_struct);
        free(ret_struct);
   return(0);
}
```

In the preceding example, the `acmsmgmt_set_param_1` procedure is called to set the maximum number of logins to the Remote Manager to 25. If the call succeeds, the new value will be stored in the Parameter table and a success message will be displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.40. ACMSMGMT_SET_QTI_2

ACMSMGMT_SET_QTI_2

ACMSMGMT_SET_QTI_2 — This procedure modifies Queued Task Initator (QTI) Config class attributes.

Format

```
qti_status_rec_2 *acmsmgmt_set_qti_2(qti_config_rec_2 *set_struct,CLIENT
 *cl)
```

Parameters

set_struct

Type	Qti_config_rec
Access	Read
Mechanism	By reference
Usage	Structure that contains the following client identification and QTI table fields.

client_id	
Type	Integer
Access	Read
Mechanism	By value
Usage	If explicit authentication is being used, a valid client ID must be provided. If the value of `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
active_sw	
Type	Integer
Access	Read
Mechanism	By value
Usage	Indicates whether active variables should be updated (`active_sw` = 1). Active variables are currently in use by the ACMS system; updates to active variables take effect immediately but are not durable (that is, they do not survive a restart of the ACMS system). Not all variables are dynamic, however. Refer to *Section 9.11, "QTI Table"* and to the field descriptions in this section to determine whether a particular variable can be updated dynamically.
current_sw	
Type	Integer
Access	Read
Mechanism	By value
Usage	Indicates whether current variables should be updated (`current_sw` = 1). Current variables are those stored in the ACMSGEN file currently in use by the ACMS system and are durable (that is, they survive a restart of the ACMS system). Updates to current variables take

		effect when the ACMS system is restarted.
	qti_priority, max_threads, sub_timeout, retry_timer, polling_timer	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Values to be updated. These fields correspond to fields of the same names in the QTI table, depending on the value of `active_sw` and `current_sw` in this record (for example, `qti_priority` will update the `qti_priority_stored` field if `current_sw` is equal to 1). See *Section 9.11, "QTI Table"* for a discussion of these fields. Note that not all fields can be updated dynamically.
	qti_username	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Values to be updated. This field corresponds to the `qti_username` field in the QTI table; the exact field depends on the value of `active_sw` and `current_sw` in this record (for example, `qti_username` will update the `qti_username_stored` field if `current_sw` is equal to 1). See *Section 9.11, "QTI Table"* for a discussion of these fields. Note that not all fields can be updated dynamically. In order to have this field set to null (that is, ""), set the field to the string "NULL".

cl

Type	CLIENT *

Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Struct qti_status_rec		
Access	Write		
Mechanism	By reference		
Usage	Pointer to a union. The union contains either a failure code or a structure of type `qti_config_rec_out`, which contains status codes for each field, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:		
	status		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	data, data_warn		
	Type	Qti_config_rec_out	
	Access	Write	
	Mechanism	By value	
	Usage	Structure containing fields corresponding to the fields in the `qti_config_rec` structure, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this structure:	
		qti_priority, max_threads, sub_timeout, retry_timer, polling_timer	
		Type	Integer
		Access	Write
		Mechanism	By value
		Usage	Status fields corresponding to the fields in the input argument.
		qti_username	
		Type	Null-terminated string
		Access	Write

		Mechanism	By reference	
		Usage	This field contains the value that was supplied in the input argument and can be ignored.	
		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests updates to ACMS QTI Config class fields contained in the QTI table (see *Section 9.11, "QTI Table"*). Note that the QTI table contains both active and stored values. The `active_sw` field and `current_sw` field control which fields should be updated.

Attempting to update an active field that is nondynamic is essentially useless, since the active field value does not change. For instance, calling this procedure with the `active_sw` field set to 1 and the `qti_username` field populated produces no change to the system.

Setting the `current_sw` field to 1 causes updates to be written to the current ACMSGEN file. These updates are durable, that is, they survive a restart of the ACMS sytem, but do not affect the active system until the system is restarted.

Both `current_sw` and `active_sw` can be set on a given call. In this case, both the active and stored values for any nonnegative or nonnull fields are updated.

For any nonnegative integer fields, the completion status of the update is returned in the corresponding field in the return structure. For string fields, the string field value is returned, regardless of the status of the call.

In order to have one of the string fields set to a null string (that is, ""), populate the field with value "NULL". To have one of the string fields ignored, pass in a null string.

Example

```
int set_qti_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[2] = "*";
   static char c_null_str[2] = "";
   static qti_config_rec_2 set_struct;
   qti_status_rec_2        *ret_struct;
   dcl_link                *nl;

   memset(&set_struct,-1,sizeof(set_struct));
   set_struct.client_id      = client_id;
   set_struct.active_sw      = 1;
   set_struct.current_sw     = 0;
   set_struct.polling_timer  = 4999;

   /* Have to provide a pointer for string conversions by XDR
      or it will qtiess vio. RM will ignore any fields with
      strlen of 0 */
   set_struct.qti_username = c_null_str;

   ret_struct = acmsmgmt_set_qti_2(&set_struct,cl);

   if (!ret_struct) {
      printf("\n Call to modify qti failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS)
      printf("\n Call to modify QTI returned the following warnings or
             errors\n");
   else
      printf("\n Call to modify QTI completed\n");

   for (nl = ret_struct->qti_status_rec_2_u.data.cmd_output; nl != NULL;
        nl = nl->pNext)
      printf("\n %s",nl->dcl_msg);
        xdr_free(xdr_qti_status_rec_2, ret_struct);
        free(ret_struct);
      return(0);
}
```

In the preceding example, the `acmsmgmt_set_qti_1` procedure is called to set the ACMSGEN parameter `qti_polling_timer` to 4999 milliseconds. If the call succeeds, only the active value is modified, the stored value is unchanged, and a success message is displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.41. ACMSMGMT_SET_SERVER_1

ACMSMGMT_SET_SERVER_1

ACMSMGMT_SET_SERVER_1 — This procedure modifies server (ACMS procedure server) Config class attributes.

Format

```
ser_status_rec *acmsmgmt_set_server_1(ser_config_rec *set_struct,CLIENT
 *cl)
```

Parameters

set_struct

Type	Ser_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Server table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value of `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	appl_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Name of the application to which the server to be updated belongs.
	server_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference

	Usage	Name of the server to update.
	creation_delay, creation_interval, deletion_delay, deletion_interval, server_proc_dmpflag, minimum_instances, maximum_instances	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Values to be updated. These fields correspond to the active fields of the same names in the Server table (for example, `creation_delay` updates the `creation_delay_active field`). See *Section 9.12, "Server Table"* for a discussion of these fields. All fields in this record can be updated dynamically.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Struct ser_status_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `ser_config_rec_out`, which contains status codes for each field, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.

	data, data_warn			
	Type	Ser_config_rec_out		
	Access	Write		
	Mechanism	By value		
	Usage	Structure containing fields corresponding to the fields in the `ser_config_rec` structure, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this structure:		
		creation_delay, creation_interval, deletion_delay, deletion_interval, server_proc_dmpflag, minimum_instances, maximum_instances		
		Type	Integer	
		Access	Write	
		Mechanism	By value	
		Usage	Status fields corresponding to the fields in the input argument.	
		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests updates to ACMS server Config class fields contained in the Server table (see *Section 9.12, "Server Table"*). Note that the Server table contains only active values.

For any nonnegative integer fields, the completion status of the update is returned in the corresponding field in the return structure.

Example

```
int set_ser_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[2] = "*";
   static char vr_appl[] = "VR_APPL";
   static ser_config_rec set_struct;
   ser_status_rec        *ret_struct;
   dcl_link              *nl;

   memset(&set_struct,-1,sizeof(set_struct));
   set_struct.client_id      = client_id;
   set_struct.appl_name      = vr_appl;
   set_struct.server_name    = c_name_all;
   set_struct.creation_delay = 20;

   ret_struct = acmsmgmt_set_server_1(&set_struct,cl);

   if (!ret_struct) {
      printf("\n Call to modify Server failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS)
      printf("\n Call to modify Server returned the following warnings or
             errors\n");
   else
      printf("\n Call to modify Server completed\n");

   for (nl = ret_struct->ser_status_rec_u.data.cmd_output; nl != NULL;
        nl = nl->pNext)
      printf("\n %s",nl->dcl_msg);
        xdr_free(xdr_ser_status_rec, ret_struct);
        free(ret_struct);
      return(0);
}
```

In the preceding example, the `acmsmgmt_set_server_1` procedure is called to set the `creation_delay` parameter field for all servers in application VR_APPL to 20 seconds. If the call succeeds, this parameter field is modified for all servers in the VR_APPL, and a success message is displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.42. ACMSMGMT_SET_TRAP_1

ACMSMGMT_SET_TRAP_1

ACMSMGMT_SET_TRAP_1 — This procedure modifies entries in the Remote Manager Trap table. Trap table entries can also be added and deleted.

Format

```
trap_status_rec *acmsmgmt_set_trap_1(trap_config_rec *set_struct,CLIENT
 *cl)
```

Parameters

set_struct

Type	Trap_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Trap table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value of `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	trap_entry	
	Type	Structtrap_update_rec_r
	Access	Read
	Mechanism	By value
	Usage	Structure containing a Trap table record. Trap table fields are described in *Section 9.14, "Trap Table"*. See the Description section for information on how to initialize this record.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Struct trap_status_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `trap_update_rec_r`, which contains status codes for each field. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data_warn	
	Type	Trap_update_rec_r
	Access	Write
	Mechanism	By value
	Usage	Structure containing a Trap table record. The entries in this field contain status codes corresponding to the fields in the `trap_entry` structure. See the Description section for a discussion of how to determine the update status for any field.

Description

This procedure requests updates to fields in the Trap table (see *Section 9.14, "Trap Table"*).

Updates to this table are not durable; that is, they do not survive a restart of the Remote Manager. To make nondynamic, permanent updates to the Trap table, use the ACMSCFG utility.

Calls to this procedure must specify `entity_type`, `entity_name`, and `param_to_trap`. These fields must exactly match an existing record in the Trap table for the update to be applied. *Table*

8.1, "Collection Classes" and *Table 8.4, "Entity Types"* contain symbolic values used to populate the `collection_class` and `entity_type` fields; symbolic values to the `param_to_trap` field are described in *Table 8.8, "Trap Parameters"*.

Setting fields `trap_min`, `trap_max`, or severity to -2 excludes them from being updating. Otherwise, the corresponding field in the matching trap record is modified. -1 is a special value that causes the field to be ignored when evaluating the trap conditions; see *Section 7.8, "SNMP Traps"*.

Updates to the Trap table are processed immediately and may affect more than one ACMS process. See *Section 7.8, "SNMP Traps"* for a discussion of how to set SNMP traps.

Example

```
int set_trap_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[2] = "*";
   static trap_config_rec set_struct;
   struct trap_status_rec *status_rec;

   set_struct.client_id                  = client_id
   set_struct.trap_entry.entity_type     = MGMT_ACC;
   set_struct.trap_entry.entity_name     = c_name_all;
   set_struct.trap_entry.param_to_trap   = MGMT_EXISTS;
   set_struct.trap_entry.min             = 1;
   set_struct.trap_entry.max             = -1;
   set_struct.trap_entry.severity        = MGMT_SEV_FATAL;

   status_rec = acmsmgmt_set_trap_1(&set_struct,cl);

   if (!status_rec) {
      printf("\n Call to modify trap failed");
      return(MGMT_FAIL);
   }

   if (status_rec->status == MGMT_WARN) {
      printf("\nThe following updates failed: ");
      if (status_rec->trap_status_rec_u.data_warn.entity_type  ==
MGMT_FAIL)
          printf("\n     entity_type not found or invalid");
      if (status_rec->trap_status_rec_u.data_warn.param_to_trap ==
MGMT_FAIL)
          printf("\n     param not found or invalid");
      if (status_rec->trap_status_rec_u.data_warn.min   == MGMT_FAIL)
          printf("\n     min invalid");
      if (status_rec->trap_status_rec_u.data_warn.max   == MGMT_FAIL)
          printf("\n     max invalid");
      if (status_rec->trap_status_rec_u.data_warn.severity  == MGMT_FAIL)
          printf("\n     severity invalid");
     }

   else if (status_rec->status != MGMT_SUCCESS) {
      printf("\nCall to modify trap failed with status %d",
             status_rec->trap_status_rec_u.rc);
        xdr_free(xdr_trap_status_rec, status_rec);
        free(status_rec);
      return(MGMT_FAIL);
```

```
   }
   else
      printf("\nCall to modify trap was executed");
        xdr_free(xdr_trap_status_rec, status_rec);
        free(status_rec);
    return(0);
}
```

In the preceding example, the `acmsmgmt_set_trap_1` procedure is called to set the `trap_min` field to 1, the `trap_max` field to -1, and the trap severity to FATAL for a trap based on an `entity_type` of ACC, an `entity_name` of * (all), and a trap parameter of EXISTS. The effect of this change is to cause a fatal-level trap to be generated if the ACC on the target node is stopped. If the call succeeds, the trap is reconfigured in the Trap table in memory. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.43. ACMSMGMT_SET_TSC_2

ACMSMGMT_SET_TSC_2

ACMSMGMT_SET_TSC_2 — This procedure modifies Terminal Subsystem Controller (TSC) Config class attributes.

Format

```
tsc_status_rec_2 *acmsmgmt_set_tsc_2(tsc_config_rec_2 *set_struct,CLIENT
 *cl)
```

Parameters

set_struct

Type	Tsc_config_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and TSC table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value of `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

active_sw	
Type	Integer
Access	Read
Mechanism	By value
Usage	Indicates whether active variables should be updated (`active_sw` = 1). Active variables are currently in use by the ACMS system; updates to active variables take effect immediately but are not durable (that is, they do not survive a restart of the ACMS system). Not all variables are dynamic, however. Refer to *Section 9.15, "TSC Table"* and to the field descriptions in this section to determine whether a particular variable can be updated dynamically.
current_sw	
Type	Integer
Access	Read
Mechanism	By value
Usage	Indicates whether current variables should be updated (`current_sw` = 1). Current variables are those stored in the ACMSGEN file currently in use by the ACMS system and are durable (that is, they survive a restart of the ACMS system). Updates to current variables take effect when the ACMS system is restarted.
tsc_priority, cp_priority, cp_slots, max_logins, max_tts_cp, perm_cps, min_cpis	
Type	Integer
Access	Read
Mechanism	By value
Usage	Values to be updated. These fields correspond to fields of the same names in the TSC table, depending on the value of `active_sw`

		and `current_sw` in this record (for example, `tsc_priority` will update the `tsc_priority_stored` field if `current_sw` is equal to 1). See *Section 9.15, "TSC Table"* for a discussion of these fields. Note that not all fields can be updated dynamically.
	tsc_username, cp_username	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Values to be updated. These fields correspond to fields of the same names in the TSC table, depending on the value of `active_sw` and `current_sw` in this record (for example, `tsc_username` will update the `tsc_username_stored` field if `current_sw` is equal to 1). See *Section 9.15, "TSC Table"* for a discussion of these fields. Note that not all fields can be updated dynamically. In order to have any of these fields set to null (that is, ""), set the field to the string "NULL".

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Struct tsc_status_rec
Access	Write
Mechanism	By reference
Usage	Pointer to a union. The union contains either a failure code or a structure of type `tsc_config_rec_out`, which contains status codes for each field, as well as a linked list of status messages associated with the update. See the Description

	section for a discussion of how to determine the update status for any field. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data, data_warn	
	Type	Tsc_config_rec_out
	Access	Write
	Mechanism	By value
	Usage	Structure containing fields corresponding to the fields in the `acc_config_rec` structure, as well as a linked list of status messages associated with the update. See the Description section for a discussion of how to determine the update status for any field. The following are the contents of this structure:
		tsc_priority, cp_priority, cp_slots, max_logins, max_tts_cp, perm_cps, min_cpis
		Type: Integer
		Access: Write
		Mechanism: By value
		Usage: Status fields corresponding to the fields in the input argument.
		tsc_username, cp_username
		Type: Null-terminated string
		Access: Write
		Mechanism: By reference
		Usage: These fields contain the values that were supplied in the input argument, and can be ignored.
		cmd_output
		Type: Dcl_list
		Access: Write
		Mechanism: By reference
		Usage: Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:

			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests updates to ACMS TSC Config class fields contained in the TSC table (see *Section 9.15, "TSC Table"*). Note that the TSC table contains both active and stored values. The `active_sw` field and `current_sw` field control which fields are attempted to be updated.

Attempting to update an active field that is nondynamic is essentially useless, since the active field value will not change. For instance, calling this procedure with the `active_sw` field set to 1 and the `tsc_username` field populated does not result in any change to the system.

Setting the `current_sw` field to 1 causes updates to be written to the current ACMSGEN file. These updates are durable, that is, they survive a restart of the ACMS sytem, but do not affect the active system until the system is restarted.

Both `current_sw` and `active_sw` may be set on a given call. In this case, both the active and stored values for any nonnegative or nonnull fields will be updated.

For any nonnegative integer fields, the completion status of the update is returned in the corresponding field in the return structure. For string fields, the string field value is returned, regardless of the status of the call.

In order to have one of the string fields set to a null string (that is, ""), populate the field with value "NULL". To have one of the string fields ignored, pass in a null string.

Example

```
int set_tsc_data(int client_id,CLIENT *cl)
 {

   static char c_name_all[2] = "*";
   static char c_null_str[2] = "";
   static tsc_config_rec set_struct;
   tsc_status_rec        *ret_struct;
   dcl_link              *nl;

   memset(&set_struct,-1,sizeof(set_struct));
```

```
   set_struct.client_id      = client_id;
   set_struct.active_sw      = 1;
   set_struct.current_sw     = 0;
   set_struct.max_logins     = 61;

   /* Have to provide a pointer for string conversions by XDR
      or it will tscess vio. RM will ignore any fields with
      strlen of 0 */
   set_struct.tsc_username = c_null_str;
   set_struct.cp_username  = c_null_str;

   ret_struct = acmsmgmt_set_tsc_2(&set_struct,cl);

   if (!ret_struct) {
      printf("\n Call to modify TSC failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS)
      printf("\n Call to modify TSC returned the following warnings or
             errors\n");
   else
      printf("\n Call to modify TSC completed\n");
   for (nl = ret_struct->tsc_status_rec_2_u.data.cmd_output; nl != NULL;
        nl = nl->pNext)
     printf("\n %s",nl->dcl_msg);
       xdr_free(xdr_tsc_status_rec_2, ret_struct);
       free(ret_struct);
     return(0);
}
```

In the preceding example, the `acmsmgmt_set_tsc_1` procedure is called to set the ACMSGEN parameter `max_logins` to 61. If the call succeeds, only the active value is modified; the stored value is unchanged, and a success message is displayed. Otherwise, an error message is displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure

8.44. ACMSMGMT_START_ACC_1

ACMSMGMT_START_ACC_1

ACMSMGMT_START_ACC_1 — This procedure requests that the Remote Manager start the ACMS system.

Format

```
cmd_output_rec *acmsmgmt_start_acc_1(acc_startup_rec *start_struct,CLIENT
 *cl)
```

Parameters

start_struct

Type	Acc_startup_rec

Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Trap table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	audit_sw	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Indicates whether system auditing should be enabled (`audit_sw` = 1), or disabled (`audit_sw` = 0).
	qti_sw	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Indicates whether the Queued Task Initiator (QTI) should be started (`qti_sw` = 1), or not (`qti_sw` = 0).
	terminals_sw	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Indicates whether the Terminal Subsystem Controller (TSC) should be started (`terminals_sw` = 1), or not (`terminals_sw` = 0).

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec		
Access	Write		
Mechanism	By reference		
Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec_r`, which points to a linked list containing status messages. The following are the contents of this union:		
	status		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	rc		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	data, data_warn		
	Type	Cmd_rec	
	Access	Write	
	Mechanism	By value	
	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:	
		cmd_output	
		Type	Dcl_list
		Access	Write
		Mechanism	By reference
		Usage	Pointer to a linked list of records containing status messages related to the

			failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests startup of the ACMS run-time system on the same node that the Remote Manager is running on. Fields in the input argument determine how the ACMS system will be started (that is, with or without auditing, terminals or QTI).

This call executes synchronously. It does not return to the caller until the attempt to start the system is complete. Any messages associated with an unsuccessful start of the system are returned in the `cmd_output` linked list.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of both structures will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of both structures will be MGMT_SUCCESS. There are no status messages associated with a successful call. If the status field contains MGMT_FAIL, the call failed. No status messages are returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int start_acc(int client_id,CLIENT *cl)
{
   dcl_link    *nl;
   static acc_startup_rec start_struct;
   cmd_output_rec *ret_struct;

   start_struct.client_id = client_id;
   start_struct.audit_sw  = 1;
   start_struct.qti_sw  = 1;
   start_struct.terminals_sw  = 1;

   ret_struct = acmsmgmt_start_acc_1(&start_struct,cl);
```

```
   if (!ret_struct) {
      printf("\n Call to start system failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

      if (ret_struct->status != MGMT_WARN) {
         printf("\nCall to start ACMS system failed with status %d",
                ret_struct->status);
           xdr_free(xdr_cmd_output_rec, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
       }

        printf("\n Call to start ACMS system completed with warnings or
               errors");

       for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
            nl = nl->pNext)
         printf("\n %s",nl->dcl_msg);
           xdr_free(xdr_cmd_output_rec, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
   }

   else {
        printf("\nCall to start ACMS system was executed");
        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
            printf("\n %s",nl->dcl_msg);
   }
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
 return(0);
}
```

In the preceding example, the `acmsmgmt_start_acc_1` procedure is called to start the ACMS run-time system on the target node. The system is started with system auditing enabled, the QTI started, and terminals started. If the call succeeds, the ACMS run-time system is started on the target node. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.45. ACMSMGMT_START_EXC_1

ACMSMGMT_START_EXC_1

ACMSMGMT_START_EXC_1 — This procedure requests that the Remote Manager start an ACMS application on the same node on which the Remote Manager is running.

Format

```
cmd_output_rec *acmsmgmt_start_exc_1(exc_startup_rec *start_struct,CLIENT
 *cl)
```

Parameters

start_struct

Type	Exc_startup_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Trap table fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	appl_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Pointer to the application name of the application to be started.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec
Access	Write
Mechanism	By reference

Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec_r`, which points to a linked list containing status messages. The following are the contents of this union:			
	status			
	Type	Integer		
	Access	Write		
	Mechanism	By value		
	Usage	Failure return code.		
	rc			
	Type	Integer		
	Access	Write		
	Mechanism	By value		
	Usage	Failure return code.		
	data, data_warn			
	Type	Cmd_rec		
	Access	Write		
	Mechanism	By value		
	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:		
		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write

			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure starts an ACMS application on the same node on which the Remote Manager is running. The `appl_name` field in the input record determines which application will be started.

This call executes synchronously. It does not return to the caller until the attempt to start the application is complete. Any messages associated with an unsuccessful start of the application are returned in the `cmd_output` linked list.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of either structure will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of either structure will be MGMT_SUCCESS. No status messages are associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. No status messages are returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int start_exc(int client_id,CLIENT *cl)
{
   dcl_link    *nl;
   static char c_appl_name[] = "VR_APPL";
   static exc_startup_rec start_struct;
   cmd_output_rec *ret_struct;

   start_struct.client_id = client_id;
   start_struct.appl_name = c_appl_name;

   ret_struct = acmsmgmt_start_exc_1(&start_struct,cl);

   if (!ret_struct) {
      printf("\n Call to start EXC failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

      if (ret_struct->status != MGMT_WARN) {
         printf("\nCall to start ACMS EXC failed with status %d",
                ret_struct->status);
           xdr_free(xdr_cmd_output_rec, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
       }

       printf("\n Call to start ACMS EXC completed with warnings or
 errors");
```

```
       for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
            nl = nl->pNext)
            printf("\n %s",nl->dcl_msg);
              xdr_free(xdr_cmd_output_rec, ret_struct);
              free(ret_struct);
            return(MGMT_FAIL);
   }

   else {
        printf("\nCall to start ACMS EXC was executed");
        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
            printf("\n %s",nl->dcl_msg);
   }
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
 return(0);
}
```

In the preceding example, the `acmsmgmt_start_exc_1` procedure is called to start an application named VR_APPL on the target node. If the call succeeds, the VR_APPL application is started on the target node. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.46. ACMSMGMT_START_QTI_1

ACMSMGMT_START_QTI_1

ACMSMGMT_START_QTI_1 — This procedure requests that the Remote Manager start a Queued Task Initiator (QTI) on the same node on which the Remote Manager is running.

Format

```
cmd_output_rec *acmsmgmt_start_qti_1(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct *	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value

	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec_r`, which points to a linked list containing status messages. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data, data_warn	
	Type	Cmd_rec
	Access	Write
	Mechanism	By value
	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:

		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure starts an ACMS QTI on the same node on which the Remote Manager is running.

This call executes synchronously. It does not return to the caller until the attempt to start the QTI is complete. Any messages associated with an unsuccessful start of the QTI are returned in the `cmd_output` linked list.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of both structures will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of both structures will be MGMT_SUCCESS. No status messages are associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. No status messages are returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int start_qti(int client_id,CLIENT *cl)
{
   dcl_link    *nl;
   static struct sub_id_struct sub_rec;
   cmd_output_rec *ret_struct;
```

```
   sub_rec.client_id = client_id;

   ret_struct = acmsmgmt_start_qti_1(&sub_rec,cl);

   if (!ret_struct) {
      printf("\n Call to start QTI failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

      if (ret_struct->status != MGMT_WARN) {
         printf("\nCall to start ACMS QTI failed with status %d",
                ret_struct->status);
           xdr_free(xdr_cmd_output_rec, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
       }

       printf("\n Call to start ACMS QTI completed with warnings or
 errors");

       for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
            nl = nl->pNext)
          printf("\n %s",nl->dcl_msg);
           xdr_free(xdr_cmd_output_rec, ret_struct);
           free(ret_struct);
          return(MGMT_FAIL);
   }

   else {
        printf("\nCall to start ACMS QTI was executed");
        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
 \
             nl = nl->pNext)
            printf("\n %s",nl->dcl_msg);
   }
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
   return(0);
}
```

In the preceding example, the `acmsmgmt_start_qti_1` procedure is called to start the Queued Task Initiator (QTI) on the target node. If the call succeeds, the QTI is started on the target node. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.47. ACMSMGMT_START_TRACE_MONITOR_1

ACMSMGMT_START_TRACE_MONITOR_1

ACMSMGMT_START_TRACE_MONITOR_1 — This procedure requests that the Remote Manager start the ACMS$TRACE_MONITOR process. The ACMS$TRACE_MONITOR process is an intermediate process used by the Remote Manager to communicate with ACMS run-time processes to enable and disable collections.

Format

```
int *acmsmgmt_start_trace_monitor_1(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to status value returned. If the value is NULL or MGMT_SUCCESS, the RPC has succeeded. If the value is neither NULL nor MGMT_SUCCESS, the call failed and the value pointed to is the reason for failure.

Description

This procedure requests that the Remote Manager start the ACMS$TRACE_MONITOR process on the target node. The ACMS$TRACE_MONITOR process is an intermediate process used by the Remote Manager to communicate with ACMS run-time processes to enable and disable collections.

In general, external entities do not require a startup and shutdown request of the trace monitor process. The Remote Manager starts the trace monitor during process initialization and stops it during process shutdown. Additionally, the Remote Manager starts the trace monitor anytime it is needed if it is not already started. Once started, the trace monitor continues to run until the Remote Manager shuts down.

After issuing the **START** command to the trace monitor, the Remote Manager waits for a period of up to `trace_start_wait_time`, a Parameter table parameter that is dynamic and expressed in seconds. If the trace monitor fails to start during that period, the Remote Manager returns an error to the caller.

Example

```
int start_trace(int client_id,CLIENT *cl)
{
   int *status;
   static struct sub_id_struct sub_rec;

   sub_rec.client_id = client_id;

   status = acmsmgmt_start_trace_monitor_1(&sub_rec,cl);

   if (!status) {
      printf("\nStartup of Trace Monitor has failed");
      return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\nStartup of Trace Monitor has failed with return code %d",
             *status);
      return(*status);
   }

   printf("\nTrace Monitor has been started ");
     free(status);
   return(MGMT_SUCCESS);
}
```

In the preceding example, the `acmsmgmt_start_trace_monitor_1` procedure is called to start the ACMS$TRACE_MON process on the target node. If the call succeeds, the process is started. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.48. ACMSMGMT_START_TSC_1

ACMSMGMT_START_TSC_1

ACMSMGMT_START_TSC_1 — This procedure requests that the Remote Manager start a Terminal Subsystem Controller (TSC) on the same node on which it is running.

Format

```
cmd_output_rec *acmsmgmt_start_tsc_1(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct *	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec_r`, which points to a linked list containing status messages. The following are the contents of this union:	
	status	
	Type	Integer

	Access	Write		
	Mechanism	By value		
	Usage	Failure return code.		
	rc			
	Type	Integer		
	Access	Write		
	Mechanism	By value		
	Usage	Failure return code.		
	data, data_warn			
	Type	Cmd_rec		
	Access	Write		
	Mechanism	By value		
	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:		
		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests that an ACMS TSC be started on the same node on which the Remote Manager is running.

This call executes synchronously. It does not return to the caller until the attempt to start the TSC is complete. Any messages associated with an unsuccessful start of the TSC are returned in the `cmd_output` linked list.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of both structures will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of both structures will be MGMT_SUCCESS. No status messages are associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. No status messages are returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int start_tsc(int client_id,CLIENT *cl)
{
   dcl_link    *nl;
   static struct sub_id_struct sub_rec;
   cmd_output_rec *ret_struct;

   sub_rec.client_id = client_id;

   ret_struct = acmsmgmt_start_tsc_1(&sub_rec,cl);

   if (!ret_struct) {
      printf("\n Call to start TSC failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

      if (ret_struct->status != MGMT_WARN) {
         printf("\nCall to start ACMS TSC failed with status %d",
                ret_struct->status);
           xdr_free(xdr_cmd_output_rec, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
       }

        printf("\n Call to start ACMS TSC completed with warnings or
 errors");

       for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
            nl = nl->pNext)
         printf("\n %s",nl->dcl_msg);
           xdr_free(xdr_cmd_output_rec, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
   }
```

```
   else {
        printf("\nCall to start ACMS TSC was executed");
        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
            printf("\n %s",nl->dcl_msg);
   }
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
 return(0);
}
```

In the preceding example, the `acmsmgmt_start_tsc_1` procedure is called to start the terminal subsystem on the target node. If the call succeeds, the terminal subsystem is started on the target node. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.49. ACMSMGMT_STOP_1

ACMSMGMT_STOP_1

ACMSMGMT_STOP_1 — This procedure initiates shutdown of the Remote Manager server on a particular node.

Format

```
int *acmsmgmt_stop_1(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct *	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to status value returned. If the value is NULL or MGMT_SUCCESS, the RPC has succeeded. If the value is neither null nor MGMT_SUCCESS, the call failed and the value pointed to is the reason for failure.

Description

This procedure shuts down the Remote Manager server on the target node. As part of shutdown, the RPC interface is stopped, which may result in a NULL pointer being returned to the caller. A NULL pointer in this case signals success of the shutdown request.

Note that the success of this procedure does not guarantee that the Remote Manager server has actually shut down. It guarantees only that the shutdown has been requested.

Example

```
int stop_manager(int client_id,CLIENT *cl)
{
   static int *status;
   static struct sub_id_struct sub_rec;
   sub_rec.client_id = client_id;

   status = acmsmgmt_stop_1(&sub_rec,cl);

   if (!status) {
      printf("\nServer shutdown has been requested");
      return(0);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\n Call to stop server failed with status %d",*status);
      return(MGMT_FAIL);
   }

   printf("\n Server shutdown has been requested");
```

```
 return(0);
}
```

In the preceding example, the `acmsmgmt_stop_1` procedure is called to request shutdown of the ACMS Remote Manager. A message is displayed indicating the success or failure of the operation. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.50. ACMSMGMT_STOP_ACC_1

ACMSMGMT_STOP_ACC_1

ACMSMGMT_STOP_ACC_1 — This procedure requests that the Remote Manager stop the ACMS system.

Format

```
cmd_output_rec *acmsmgmt_stop_acc_1(acc_shutdown_rec *stop_struct,CLIENT
 *cl)
```

Parameters

stop_struct

Type	Acc_shutdown_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and ACC control fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	cancel_sw	
	Type	Integer
	Access	Read
	Mechanism	By value

	Usage	Indicates whether the system should be stopped immediately (`cancel_sw` = 1), or whether currently executing tasks should be allowed to complete first (`cancel_sw` = 0).

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec_r`, which points to a linked list containing status messages. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	rc	
	Type	Integer
	Access	Write
	Mechanism	By value
	Usage	Failure return code.
	data, data_warn	
	Type	Cmd_rec
	Access	Write
	Mechanism	By value
	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:
		cmd_output

		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure shuts down the ACMS run-time system on the same node on which the Remote Manager is running. Fields in the input argument determine how the ACMS system will be stopped. If the value for `cancel_sw` is 1, currently executing tasks are cancelled, and the system is stopped. If the value for `cancel_sw` is 0, currently executing tasks are allowed to complete before the system is shut down.

This call executes synchronously. It does not return to the caller until the attempt to stop the system is complete. Any messages associated with an unsuccessful stop of the system are returned in the `cmd_output` linked list.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of both structures will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of both structures will be MGMT_SUCCESS. No status messages are associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. No status messages are returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int stop_acc(int client_id,CLIENT *cl)
{
```

```
   dcl_link    *nl;
   static acc_shutdown_rec stop_struct;
   cmd_output_rec *ret_struct;

   stop_struct.client_id = client_id;
   stop_struct.cancel_sw  = 1;

   ret_struct = acmsmgmt_stop_acc_1(&stop_struct,cl);

   if (!ret_struct) {
      printf("\n Call to stop ACC failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {
      if (ret_struct->status != MGMT_WARN) {
         printf("\nCall to stop ACMS ACC failed with status %d",
                ret_struct->status);
           xdr_free(xdr_cmd_output_rec, ret_struct);
           free(ret_struct);
         return(MGMT_FAIL);
   }

        printf("\n Call to stop ACMS ACC completed with warnings or
 errors");

 for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
            printf("\n %s",nl->dcl_msg);
              xdr_free(xdr_cmd_output_rec, ret_struct);
              free(ret_struct);
            return(MGMT_FAIL);
   }
   else {
        printf("\nCall to stop ACMS ACC was executed");
        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
     printf("\n %s",nl->dcl_msg);
   }
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
 return(0);
}
```

In the preceding example, the `acmsmgmt_stop_acc_1` procedure is called to stop the ACMS run-time system on the target node. The system is stopped abruptly (**`/CANCEL`**), terminating any in-process tasks. If the call succeeds, the ACMS system is stopped on the target node. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.51. ACMSMGMT_STOP_EXC_1

ACMSMGMT_STOP_EXC_1

ACMSMGMT_STOP_EXC_1 — This procedure requests that the Remote Manager stop the ACMS system.

Format

```
cmd_output_rec *acmsmgmt_stop_exc_1(exc_shutdown_rec *stop_struct,CLIENT
 *cl)
```

Parameters

stop_struct

Type	Exc_shutdown_rec	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client identification and Application Execution Controller (EXC) control fields.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.
	cancel_sw	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	Indicates whether the application should be stopped immediately (`cancel_sw` = 1), or whether currently executing tasks should be allowed to complete first (`cancel_sw` = 0).
	appl_name	
	Type	Null-terminated string
	Access	Read
	Mechanism	By reference
	Usage	Name of the application to be stopped.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec		
Access	Write		
Mechanism	By reference		
Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec_r`, which points to a linked list containing status messages. The following are the contents of this union:		
	status		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	rc		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	data, data_warn		
	Type	Cmd_rec	
	Access	Write	
	Mechanism	By value	
	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:	
		cmd_output	
		Type	Dcl_list
		Access	Write
		Mechanism	By reference
		Usage	Pointer to a linked list of records containing status messages related to the

			failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure shuts down an ACMS application on the same node on which the Remote Manager is running. Fields in the input argument determine which application to stop (`appl_name`) and how the application will be stopped. If the value for `cancel_sw` is 1, currently executing tasks are cancelled, and the application is stopped. If the value for `cancel_sw` is 0, currently executing tasks are allowed to complete before the application is shut down.

This call executes synchronously. It does not return to the caller until the attempt to stop the application is complete. Any messages associated with an unsuccessful stop of the system are returned in the `cmd_output` linked list.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of both structure will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of both structures will be MGMT_SUCCESS. No status messages are associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. No status messages are returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int stop_exc(int client_id,CLIENT *cl)
{
   dcl_link    *nl;
   static char c_appl_name[] = "VR_APPL";
   static exc_shutdown_rec stop_struct;
   cmd_output_rec *ret_struct;

   stop_struct.client_id = client_id;
   stop_struct.cancel_sw  = 1;
```

```
   stop_struct.appl_name  = c_appl_name;

   ret_struct = acmsmgmt_stop_exc_1(&stop_struct,cl);

   if (!ret_struct) {
      printf("\n Call to stop EXC failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

   if (ret_struct->status != MGMT_WARN) {
      printf("\nCall to stop ACMS EXC failed with status %d",
             ret_struct->status);
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
      return(MGMT_FAIL);
   }

   printf("\n Call to stop ACMS EXC completed with warnings or errors");

        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
     printf("\n %s",nl->dcl_msg);
              xdr_free(xdr_cmd_output_rec, ret_struct);
              free(ret_struct);
            return(MGMT_FAIL);
   }

   else {
 printf("\nCall to stop ACMS EXC was executed");
        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
     printf("\n %s",nl->dcl_msg);
   }
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
 return(0);
}
```

In the preceding example, the `acmsmgmt_stop_exc_1` procedure is called to stop an application named VR_APPL on the target node. If the call succeeds, the VR_APPL application is stopped on the target node. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.52. ACMSMGMT_STOP_QTI_1

ACMSMGMT_STOP_QTI_1

ACMSMGMT_STOP_QTI_1 — This procedure requests that the Remote Manager stop a Queued Task Initiator (QTI) on the same node on which the Remote Manager is running.

Format

```
cmd_output_rec *acmsmgmt_stop_qti_1(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct *	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *
Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec	
Access	Write	
Mechanism	By reference	
Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec_r`, which points to a linked list containing status messages. The following are the contents of this union:	
	status	
	Type	Integer
	Access	Write
	Mechanism	By value

	Usage	Failure return code.		
	rc			
	Type	Integer		
	Access	Write		
	Mechanism	By value		
	Usage	Failure return code.		
	data, data_warn			
	Type	Cmd_rec		
	Access	Write		
	Mechanism	By value		
	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:		
		cmd_output		
		Type	Dcl_list	
		Access	Write	
		Mechanism	By reference	
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:	
			dcl_msg	
			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests to stop an ACMS QTI on the same node on which the Remote Manager is running.

This call executes synchronously. It does not return to the caller until the attempt to stop the QTI is complete. Any messages associated with an unsuccessful stop of the QTI are returned in the `cmd_output` linked list.

The `data` and `data_warn` structures contain identical data. If the operation fails, the status field of both structures will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of both structures will be MGMT_SUCCESS. No status messages are associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. No status messages are returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int stop_qti(int client_id,CLIENT *cl)
{
   dcl_link    *nl;
   static struct sub_id_struct sub_rec;
   cmd_output_rec *ret_struct;

   sub_rec.client_id = client_id;

   ret_struct = acmsmgmt_stop_qti_1(&sub_rec,cl);

   if (!ret_struct) {
      printf("\n Call to stop qti failed");
      return(MGMT_FAIL);
   }

   if (ret_struct->status != MGMT_SUCCESS) {

 if (ret_struct->status != MGMT_WARN) {
            printf("\nCall to stop ACMS QTI failed with status %d",
                   ret_struct->status);
              xdr_free(xdr_cmd_output_rec, ret_struct);
              free(ret_struct);
            return(MGMT_FAIL);
   }

        printf("\n Call to stop ACMS QTI completed with warnings or
 errors");

        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
     printf("\n %s",nl->dcl_msg);
              xdr_free(xdr_cmd_output_rec, ret_struct);
              free(ret_struct);
      return(MGMT_FAIL);
   }

   else {
        printf("\nCall to stop ACMS QTI was executed");
        for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
             nl = nl->pNext)
     printf("\n %s",nl->dcl_msg);
   }
```

```
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
   return(0);
}
```

In the preceding example, the `acmsmgmt_stop_qti_1` procedure is called to stop the Queued Task Initiator (QTI) on the target node. If the call succeeds, the QTI is stopped on the target node. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.53. ACMSMGMT_STOP_TRACE_MONITOR_1

ACMSMGMT_STOP_TRACE_MONITOR_1

ACMSMGMT_STOP_TRACE_MONITOR_1 — This procedure requests that the Remote Manager stop the ACMS$TRACE_MONITOR process. The ACMS$TRACE_MONITOR process is an intermediate process used by the Remote Manager to communicate with ACMS run-time processes to enable and disable collections.

Format

```
int *acmsmgmt_stop_trace_monitor_1(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *

Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Integer
Access	Write
Mechanism	By reference
Usage	Pointer to status value returned. If the value is NULL or MGMT_SUCCESS, the RPC has succeeded. If the value is neither NULL nor MGMT_SUCCESS, the call failed and the value pointed to is the reason for failure.

Description

This procedure requests that the Remote Manager stop the ACMS$TRACE_MONITOR process on the target node. The ACMS$TRACE_MONITOR process is an intermediate process used by the Remote Manager to communicate with ACMS run-time processes to enable and disable collections.

In general, external entities do not require a startup and shutdown request of the trace monitor process. The Remote Manager starts the trace monitor during process initialization and stops it during process shutdown. Additionally, the Remote Manager starts the trace monitor anytime it is needed if it is not already started. Once started, the trace monitor continues to run until the Remote Manager shuts down.

After issuing the **`STOP`** command to the trace monitor, the Remote Manager waits for a period of up to `trace_start_wait_time`, a Parameter table parameter that is dynamic and expressed in seconds. If the trace monitor fails to stop during that period, the Remote Manager returns an error to the caller.

Example

```
int stop_trace(int client_id,CLIENT *cl)
{
   int *status;
   static struct sub_id_struct sub_rec;

   sub_rec.client_id = client_id;

   status = acmsmgmt_stop_trace_monitor_1(&sub_rec,cl);

   if (!status) {
      printf("\nShutdown of Trace Monitor has failed");
      return(MGMT_FAIL);
   }

   if (*status != MGMT_SUCCESS) {
      printf("\nShutdown of Trace Monitor has failed with return code %d",
             *status);
        free(status);
      return(MGMT_FAIL);
```

```
   }

   printf("\nTrace Monitor has been stopped ");
     free(status);
   return(MGMT_SUCCESS);
}
```

In the preceding example, the `acmsmgmt_stop_trace_monitor_1` procedure is called to stop the ACMS$TRACE_MON process on the target node. If the call succeeds, the process is stopped. Otherwise, any error messages associated with the failure are displayed. The example in *Section 6.4.1, "Initialization Example"* shows how to declare and initialize the input arguments to this procedure.

8.54. ACMSMGMT_STOP_TSC_1

ACMSMGMT_STOP_TSC_1

ACMSMGMT_STOP_TSC_1 — This procedure requests that the Remote Manager stop a Terminal Subsystem Controller (TSC) on the same node on which the Remote Manager is running.

Format

```
cmd_output_rec *acmsmgmt_stop_tsc_1(sub_id_struct *sub_rec,CLIENT *cl)
```

Parameters

sub_rec

Type	Sub_id_struct	
Access	Read	
Mechanism	By reference	
Usage	Structure that contains the following client authorization information.	
	client_id	
	Type	Integer
	Access	Read
	Mechanism	By value
	Usage	If explicit authentication is being used, a valid client ID must be provided. If the value for `client_id` is 0, proxy access is used. `client_id` is obtained by calling the ACMS$MGMT_GET_CREDS procedure.

cl

Type	CLIENT *

Access	Read
Mechanism	By value
Usage	Pointer to an RPC client handle previously obtained by calling the RPC routine CLNT_CREATE.

Return Value

Type	Cmd_output_rec		
Access	Write		
Mechanism	By reference		
Usage	Pointer to a union. The union contains either a failure code or a structure of type `cmd_rec`, which points to a linked list containing status messages. The following are the contents of this union:		
	status		
	Type		Integer
	Access		Write
	Mechanism		By value
	Usage		Failure return code.
	rc		
	Type	Integer	
	Access	Write	
	Mechanism	By value	
	Usage	Failure return code.	
	data, data_warn		
	Type	Cmd_rec	
	Access	Write	
	Mechanism	By value	
	Usage	Structure containing the first node in a linked list of status messages (type `dcl_list`). The following are the contents of this structure:	
		cmd_output	
		Type	Dcl_list
		Access	Write
		Mechanism	By reference
		Usage	Pointer to a linked list of records containing status messages related to the failure of any updates. This structure contains the following fields:
			dcl_msg

			Type	Null-terminated string
			Access	Write
			Mechanism	By reference
			Usage	The status message.
			pNext	
			Type	Dcl_list
			Access	Write
			Mechanism	By reference
			Usage	Pointer to the next node in the linked list.

Description

This procedure requests to stop an ACMS TSC on the same node on which the Remote Manager is running.

This call executes synchronously. It does not return to the caller until the attempt to stop the TSC is complete. Any messages associated with an unsuccessful start of the TSC are returned in the `cmd_output` linked list.

The data and data_warn structures contain identical data. If the operation fails, the status field of both structure will be MGMT_WARN; in this case, use the `data_warn` structure to fetch the status messages from the `cmd_output` linked list.

If the operation is successful, the status field of both structures will be MGMT_ SUCCESS. No status messages are associated with a successful call.

If the status field contains MGMT_FAIL, the call failed. No status messages are returned; instead, the reason for the failure is contained in the `rc` field.

Example

```
int stop_tsc(int client_id,CLIENT *cl)
{
    dcl_link *nl;
    static struct sub_id_struct sub_rec;
    cmd_output_rec *ret_struct;

    sub_rec.client_id = client_id;

    ret_struct = acmsmgmt_stop_tsc_1(&sub_rec,cl);

    if (!ret_struct) {
        printf("\n Call to stop TSC failed");
        return(MGMT_FAIL);
    }
    if (ret_struct->status != MGMT_SUCCESS) {
    if (ret_struct->status != MGMT_WARN) {
```

```
    printf("\nCall to stop ACMS TSC failed with status %d",
            ret_struct->status);
        xdr_free(xdr_cmd_output_rec, ret_struct);
        free(ret_struct);
    return(MGMT_FAIL);
}
printf("\n Call to stop ACMS TSC completed with warnings or errors");

for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
    nl = nl->pNext)
  printf("\n %s",nl->dcl_msg);
    xdr_free(xdr_cmd_output_rec, ret_struct);
    free(ret_struct);
  return(MGMT_FAIL);
}
else {
    printf("\nCall to stop ACMS TSC was executed");
    for (nl = ret_struct->cmd_output_rec_u.data.cmd_output; nl != NULL;
        nl = nl->pNext)
  printf("\n %s",nl->dcl_msg);
}
    xdr_free(xdr_cmd_output_rec, ret_struct);
    free(ret_struct);
  return(0);
}
```

In the preceding example, the ACMSMGMT_STOP_TSC_1 procedure is called to stop the terminal subsystem on the target node. If the call succeeds, the terminal subsystem is stopped on the target node. Otherwise, any error messages associated with the failure are displayed.

Chapter 9. Remote Manager Reference Tables

This chapter contains information about data types that the Remote Manager implements and the reference tables for the Remote Manager. The Remote Manager reference tables include the following:

- ACC table
- Agent table
- Collection table
- CP table
- EXC table
- Interfaces table
- Manager status table
- Parameter table
- QTI table
- Server table
- Task Group table
- Trap table
- TSC table
- Users table

Note

The following sections describe the records and fields in each Remote Manager reference table. Many of these tables contain a subset of fields intended solely for use on or by systems running ACMS Version 4.4 or higher.

9.1. Data Types

The ACMS Remote Manager implements the following data types:

- Gauge and Min Gauge

  Gauge fields are structures containing the following fields:

  - `current_value`

    The value of the object when last observed. Represents the most current known value.

- `max_value` or `min_value`

  The largest or smallest observed value for the object.

- `time_max_seen` or `time_min_seen`

  The date and time the `max_value` or `min_value` was set.

- Integer

  Integer fields are 32-bit signed integers.

- State 1

  State 1 fields are integers with two possible values:

  - MGMT$K_STATE_DISABLED
  - MGMT$K_STATE_ENABLED

- State 2

  State 2 fields are integers with the following possible values:

  - MGMT$K_STATE_INITED
  - MGMT$K_STATE_INITING
  - MGMT$K_STATE_LOAD_DONE
  - MGMT$K_STATE_LOADING
  - MGMT$K_STATE_STARTED
  - MGMT$K_STATE_STARTING
  - MGMT$K_STATE_STOPPED
  - MGMT$K_STATE_STOPPING

- String

  String fields are null-terminated ASCII strings.

- Time

  Time fields are stored internally in OpenVMS internal time format and are generally displayed as *DD-MMM-YYY HH:MM:SS.hh*. When present in a record supplied by the Remote Manager (that is, from either an RPC or SNMP call, or in a file), time is always an ASCII value in the default OpenVMS format (*DD-MMM-YYYY HH:MM:SS.hh*) and is stored as a null-terminated string.

9.2. ACC Table

The ACC table contains a single entry for ACC management information.

Table 9.1. ACC Table

Class	Field	Data Type	SNMP Access	RPC Access
ID	record_state	integer	R	R
ID	id_coll_state	integer	R	R
ID	process_name	string	R	R
ID	pid	integer	R	R
ID	username_active	string	R	R
ID	username_stored	string	R	R
ID	start_time	time	R	R
ID	end_time	time	R	R
ID	acms_version	string	R	R
CONFIG	config_coll_state	integer	R	R
CONFIG	acms_state §	integer	RW	R
CONFIG	acc_priority_active	integer	R	R
CONFIG	acc_priority_stored	integer	RW	RW
CONFIG	max_appl_active	integer	R	R
CONFIG	max_appl_stored	integer	RW	RW
CONFIG	mss_maxobj_active	integer	R	R
CONFIG	mss_maxobj_stored	integer	RW	RW
CONFIG	mss_maxbuf_active	integer	R	R
CONFIG	mss_maxbuf_stored	integer	RW	RW
CONFIG	mss_poolsize_active	integer	R	R
CONFIG	mss_poolsize_stored	integer	RW	RW
CONFIG	mss_process_pool_active	integer	R	R
CONFIG	mss_process_pool_stored	integer	RW	RW
CONFIG	mss_net_retry_timer_active §	integer	RW	RW
CONFIG	mss_net_retry_timer_stored	integer	RW	RW
CONFIG	audit_state §	integer	RW	RW
CONFIG	username_default_active §	integer	RW	RW
CONFIG	username_default_stored	integer	RW	RW

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Class	Field	Data Type	SNMP Access	RPC Access
CONFIG	node_name_active	integer	R	R
CONFIG	node_name_stored	integer	RW	RW
CONFIG	ws_poolsize_active	integer	R	R
CONFIG	ws_poolsize_stored	integer	RW	RW
CONFIG	wsc_poolsize_active	integer	R	R
CONFIG	wsc_poolsize_stored	integer	RW	RW
CONFIG	tws_poolsize_active	integer	R	R
CONFIG	tws_poolsize_stored	integer	RW	RW
CONFIG	twsc_poolsize_active	integer	R	R
CONFIG	twsc_poolsize_stored	integer	RW	RW
CONFIG	astlm_active »	integer	R	R
CONFIG	astlm_stored »	integer	RW	RW
CONFIG	biolm_active »	integer	R	R
CONFIG	biolm_stored »	integer	RW	RW
CONFIG	bytlm_active »	integer	R	R
CONFIG	bytlm_stored »	integer	RW	RW
CONFIG	diolm_active »	integer	R	R
CONFIG	diolm_stored »	integer	RW	RW
CONFIG	enqlm_active »	integer	R	R
CONFIG	enqlm_stored »	integer	RW	RW
CONFIG	fillm_active »	integer	R	R
CONFIG	fillm_stored »	integer	RW	RW
CONFIG	pgflquota_active »	integer	R	R
CONFIG	pgflquota_stored »	integer	RW	RW
CONFIG	tqelm_active »	integer	R	R
CONFIG	tqelm_stored »	integer	RW	RW
CONFIG	wsdefault_active »	integer	R	R
CONFIG	wsdefault_stored »	integer	RW	RW
CONFIG	wsextent_active »	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Class	Field	Data Type	SNMP Access	RPC Access
CONFIG	wsextent_stored »	integer	RW	RW
CONFIG	wsquota_active »	integer	R	R
CONFIG	wsquota_stored »	integer	RW	RW
CONFIG	channelcnt_active »	integer	R	R
CONFIG	channelcnt_stored »	integer	RW	RW
CONFIG	gblsections_active »	integer	R	R
CONFIG	gblsections_stored »	integer	RW	RW
CONFIG	gblpages_active »	integer	R	R
CONFIG	gblpages_stored »	integer	RW	RW
CONFIG	gblpagfil_active »	integer	R	R
CONFIG	gblpagfil_stored »	integer	RW	RW
RUNTIME	runtime_coll_state	integer	R	R
RUNTIME	current_appls	gauge	R	R
RUNTIME	current_users	gauge	R	R
RUNTIME	current_local_users	gauge	R	R
RUNTIME	current_remote_users	gauge	R	R
RUNTIME	appl_starts	integer	R	R
RUNTIME	decnet_object	integer	R	R
RUNTIME	astlm_current »	gauge	R	R
RUNTIME	biolm_current »	gauge	R	R
RUNTIME	bytlm_current »	gauge	R	R
RUNTIME	diolm_current »	gauge	R	R
RUNTIME	enqlm_current »	gauge	R	R
RUNTIME	fillm_current »	gauge	R	R
RUNTIME	pgflquota_current »	gauge	R	R
RUNTIME	tqelm_current »	gauge	R	R
RUNTIME	wssize_current »	gauge	R	R
RUNTIME	channelcnt_current »	gauge		
POOL	pool_coll_state	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Class	Field	Data Type	SNMP Access	RPC Access
POOL	mss_shared_total	integer	R	R
POOL	mss_shared_free	min gauge	R	R
POOL	mss_shared_largest	min gauge	R	R
POOL	mss_shared_failures	integer	R	R
POOL	mss_shared_garbage	integer	R	R
POOL	mss_process_total	integer	R	R
POOL	mss_process_free	min gauge	R	R
POOL	mss_process_largest	min gauge	R	R
POOL	mss_process_failures	integer	R	R
POOL	mss_process_garbage	integer	R	R
POOL	mss_objects	gauge	R	R
POOL	mss_msgs_1-1024 »	integer	R	R
POOL	mss_msgs_1025-2048 »	integer	R	R
POOL	mss_msgs_2049-4096 »	integer	R	R
POOL	mss_msgs_4097-8192 »	integer	R	R
POOL	mss_msgs_8193-16384 »	integer	R	R
POOL	mss_msgs_16385-32768 »	integer	R	R
POOL	mss_msgs_32769-65636 »	integer	R	R
POOL	mss_msg_count_resets »	gauge	R	R
POOL	tws_poolsize_total »	gauge	R	R
POOL	tws_pool_large_used »	gauge	R	R
POOL	tws_pool_min_free »	gauge	R	R
POOL	twsc_poolsize_total »	gauge	R	R
POOL	twsc_pool_large_used »	gauge	R	R
POOL	twsc_pool_min_free »	gauge	R	R
POOL	ws_pool_large_used »	gauge	R	R
POOL	ws_pool_min_free »	gauge	R	R
POOL	wsc_pool_large_used »	gauge	R	R
POOL	wsc_pool_min_free »	gauge	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Class	Field	Data Type	SNMP Access	RPC Access
ERROR	err_coll_state »	integer	R	R
ERROR	err_count »	integer	R	R
ERROR	last_err_msg »	string	R	R
ERROR	time_of_last_error »	time	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.2.1. Field Descriptions

Following are descriptions of the fields in *Table 9.1, "ACC Table"*.

- `record_state`

  The current state of this table entry. Valid states are VALID (the process is currently running and maintaining this table entry) or INACTIVE (the process is no longer running). Inactive rows are subject to reuse.

- `id_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `process_name`

  The OpenVMS process name for the process.

- `pid`

  The OpenVMS process identifier for the process.

- `username_active`

  The OpenVMS user name under which the process is currently running. This is the value that was in the ACMSGEN file when the process was started.

- `username_stored`

  The OpenVMS process name currently stored in the ACMSGEN file for this process.

- `start_time`

  Date and time the process was started.

- `end_time`

  Date and time the process ended. If the process has not yet ended, this field will be null.

- `acms_version`

  Current version of the ACC.

- `config_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `acms_state`

  Current ACMS state of the process. This field can be set (to DISABLED or to 0) by the SNMP interface only. RPC users use the ACMSMGMT_STOP_ACC_1 procedure described in *Section 8.50, "ACMSMGMT_STOP_ACC_1"*. ACMSMGR users use the **`STOP SYSTEM`** command.

- Current ACMS state of the process. This field can be set (to DISABLED or to 0) by the SNMP interface only. RPC users use the ACMSMGMT_STOP_ACC_1 procedure described in *ACMSMGMT_STOP_ACC_1*. ACMSMGR users use the **`STOP SYSTEM`** command.

- `acc_priority_active`

  The base priority for this process. This is the value of the ACMSGEN field when the process was started.

- `acc_priority_stored`

  The base priority currently stored in the ACMSGEN file for this process.

- `max_appl_active`

  The maximum number of ACMS applications that can be started simultaneously on this node. This is the value of the ACMSGEN field when the ACC process was started.

- `max_appl_stored`

  The value of the `max_appl` field currently stored in the ACMSGEN file.

- `mss_maxobj_active`

  The maximum number of ACMS message switch objects that can be started simultaneously on this node. This is the value of the ACMSGEN field when the ACC process was started. See the MSS class field `mss_objects` for a count of the current and maximum number of MSS objects instantiated on the system.

- `mss_maxobj_stored`

  The value of the `mss_maxobj` field currently stored in the ACMSGEN file.

- `mss_maxbuf_active`

  The maximum size of a message segment of an ACMS message switch message. This is the value of the ACMSGEN field when the ACC process was started.

- `mss_maxbuf_stored`

  The value of the `mss_maxbuf` field currently stored in the ACMSGEN file.

- `mss_poolsize_active`

  The size of the MSS shared pool. This is the value of the ACMSGEN field when the ACC process was started.

- `mss_poolsize_stored`

  The value of the `mss_poolsize` field currently stored in the ACMSGEN file.

- `mss_process_pool_active`

  The default size of the MSS pool allocated for each ACMS process. This is the value of the ACMSGEN field when the ACC process was started.

- `mss_process_pool_stored`

  The value of the `mss_process_pool` field currently stored in the ACMSGEN file.

- `mss_net_retry_active`

  The time ACMS processes will wait before retrying an MSS network operation. This field can be modified dynamically.

- `mss_net_retry_timer_stored`

  The value of the `mss_net_retry_timer` field currently stored in the ACMSGEN file.

- `audit_state`

  The current system auditing state.

- `username_default_active`

  The default user name for remote users. This is the value of the ACMSGEN field when the ACC process was started.

- `username_default_stored`

  The value of the `username_default` field currently stored in the ACMSGEN file.

- `node_name_active`

  The node name for the current node. This is the value of the ACMSGEN field when the ACC process was started.

- `node_name_stored`

  The value of the `node_name` field currently stored in the ACMSGEN file.

- `ws_poolsize_active`

  The default size for WS pools. This is the value of the ACMSGEN field when the ACC process was started.

- `ws_poolsize_stored`

  The value of the `ws_poolsize` field currently stored in the ACMSGEN file.

- `wsc_poolsize_active`

  The default size for WSC pools. This is the value of the ACMSGEN field when the ACC process was started.

- `wsc_poolsize_stored`

  The value of the `wsc_poolsize` field currently stored in the ACMSGEN file.

- `tws_poolsize_active`

  The default size for TWS pools. This is the value of the ACMSGEN field when the ACC process was started.

- `tws_poolsize_stored`

  The value of the `twsc_poolsize` field currently stored in the ACMSGEN file.

- `twsc_poolsize_active`

  The default size for TWSC pools. This is the value of the ACMSGEN field when the ACC process was started.

- `twsc_poolsize_stored`

  The value of the `twsc_poolsize` field currently stored in the ACMSGEN file.

- `runtime_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `current_appls`

  The number of applications currently running on the node.

- `current_users`

  The number of users currently logged in to the node.

- `current_local_users`

  The number of current users logged in to ACMS locally.

- `current_remote_users`

  The number of current users who are logged in to ACMS remotely.

- `appl_starts`

  The number of applications that have been started on the node since the system was started.

- `decnet_object`

If the process has a current DECnet object, the value of this field is STARTED. Otherwise, the value is STOPPED. If the DECnet object is stopped (and the `runtime_coll_state` is enabled for this process), either distributed processing has not been enabled (that is, the `node_name` parameter in the ACMSGEN file is NULL) or there is currently a problem with DECnet. Also, check the ACC CONFIG parameters `node_name_active` and `node_name_stored` to determine the current status of the ACMSGEN `node_name` field.

- `astlm_current`, `biolm_current`, `bytlm_current`, `diolm_current`, `enqlm_current`, `fillm_current`, `pgflquota_current`, `tqelm_current`, `wssize_current`, `channelcnt_current`

  The actual amount of the related OpenVMS process or system resource that is being consumed by the ACC process. The frequency with which these fields are updated is based on the value of the `vms_coll_interval` field in the *Table 9.9, "Parameter Table"*.

- `pool_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `mss_shared_total`

  The total size (in bytes) of the MSS shared pool on this node. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_shared_free`

  The amount (in bytes) of unused MSS shared pool. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_shared_largest`

  The largest unused block (in bytes) available in the MSS shared pool. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_shared_failures`

  The number of failed attempts to allocate space from the MSS shared pool. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_shared_garbage`

  The number of garbage collections that have been run to reclaim space in the MSS shared pool. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_total`

The total size (in bytes) of the MSS process pool allocated for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_free`

  The amount of unused MSS process pool (in bytes) for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_largest`

  The largest unused block (in bytes) available in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_failures`

  The number of failed attempts to allocate space in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_garbage`

  The number of garbage collections for this process that have been run to reclaim space in the MSS process pool. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_objects`

  The number of MSS objects currently instantiated on the node. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_msgs_1-1024`, `mss_msgs_1025-2048`, `mss_msgs_2049-4096`, `mss_msgs_4097-8192`, `mss_msgs_8193-16384`, `mss_msgs_16385-32768`, `mss_msgs_32769-65636`

  The number of MSS messages whose sizes fall within the specified range (in bytes).

- `mss_msg_count_resets`

  The number of MSS message size counter resets that have occurred. Each of the preceding counters (`mss_msgs_`*`n`*`-`*`n`*) are reset if one of the counters overflows.

- `tws_poolsize_total`, `twsc_poolsize_total`

  The total amount (in pagelets) of TWS or TWSC process pool being used by all task groups running on the current node. The frequency with which these fields are updated is based on the value of the `wksp_coll_interval` field in the*Table 9.9, "Parameter Table"*.

- `tws_pool_large_used`, `twsc_pool_large_used`

  The largest amount (in bytes) of TWS or TWSC process pool being used by a task group running on the current node. The frequency with which these fields are updated is based on the value of the `wksp_coll_interval` field in the *Table 9.9, "Parameter Table"*.

- `tws_pool_min_free`, `twsc_pool_min_free`

  The minimum amount (in bytes) of unused TWS or TWSC process pool available on the current node. The frequency with which these fields are updated is based on the value of the `wksp_coll_interval` field in the *Table 9.9, "Parameter Table"*.

- `ws_pool_large_used`, `wsc_pool_large_used`

  The largest amount (in bytes) of WS or WSC process pool being used by an application running on the current node. The frequency with which these fields are updated is based on the value of the `wksp_coll_interval` field in the *Table 9.9, "Parameter Table"*.

- `ws_pool_min_free`, `wsc_pool_min_free`

  The minimum amount (in bytes) of unused WS or WSC process pool available on the current node. The frequency with which these fields are updated is based on the value of the `wksp_coll_interval` field in the *Table 9.9, "Parameter Table"*.

- `err_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the *Table 9.3, "Collection Table"*. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  Errors for this process are only sent to the Remote Manager if this field is set to ENABLED. If this field is set to DISABLED, the process will not collect data for the fields in this class; existing field values reflect activity during a prior period when collection was enabled.

- `err_count`

  The total number of errors related to this process that were sent to the Remote Manager.

- `last_err_msg`

  The text of the most recent error message related to this process that was sent to the Remote Manager

- `time_of_last_error`

  Date and time of the most recent error message related to this process that was sent to the Remote Manager.

9.3. Agent Table

Table 9.2. Agent Table

Class	Field	Data Type	SNMP Access	RPC Access
ID	record_state	integer	R	R
Table Key § – Field is dynamic » – Only valid for use with systems running ACMS Version 4.4 or higher R – Read Access RW – Read/Write Access Blank – Not available to the interface				

Class	Field	Data Type	SNMP Access	RPC Access
ID	id_coll_state	integer	R	R
ID	process_name	string	R	R
ID	pid	integer	R	R
ID	start_time	time	R	R
ID	end_time	time	R	R
ID	user_name	string	R	R
ID	acms_state	integer	R	R
CONFIG	cfg_coll_state	integer	R	R
CONFIG	astlm_active »	integer	R	R
CONFIG	astlm_stored »	integer	RW	RW
CONFIG	biolm_active »	integer	R	R
CONFIG	biolm_stored »	integer	RW	RW
CONFIG	bytlm_active »	integer	R	R
CONFIG	bytlm_stored »	integer	RW	RW
CONFIG	diolm_active »	integer	R	R
CONFIG	diolm_stored »	integer	RW	RW
CONFIG	enqlm_active »	integer	R	R
CONFIG	enqlm_stored »	integer	RW	RW
CONFIG	fillm_active »	integer	R	R
CONFIG	fillm_stored »	integer	RW	RW
CONFIG	pgflquota_active »	integer	R	R
CONFIG	pgflquota_stored »	integer	RW	RW
CONFIG	tqelm_active »	integer	R	R
CONFIG	tqelm_stored »	integer	RW	RW
CONFIG	wsdefault_active »	integer	R	R
CONFIG	wsdefault_stored »	integer	RW	RW
CONFIG	wsextent_active »	integer	R	R
CONFIG	wsextent_stored »	integer	RW	RW
CONFIG	wsquota_active »	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Class	Field	Data Type	SNMP Access	RPC Access
CONFIG	wsquota_stored »	integer	RW	RW
RUNTIME	rt_coll_state	integer	R	R
RUNTIME	decnet_object	integer	R	R
RUNTIME	active_task_calls	gauge	R	R
RUNTIME	current_attached_terms	gauge	R	R
RUNTIME	active_tdms_menu_reqs	gauge	R	R
RUNTIME	total_tdms_menu_reqs	integer	R	R
RUNTIME	active_tdms_reqs	gauge	R	R
RUNTIME	active_tdms_msgrd	gauge	R	R
RUNTIME	active_tdms_msgwt	gauge	R	R
RUNTIME	active_tdms_cancel	gauge	R	R
RUNTIME	total_tdms_reqs	integer	R	R
RUNTIME	total_tdms_msgrd	integer	R	R
RUNTIME	total_tdms_msgwt	integer	R	R
RUNTIME	total_tdms_cancel	integer	R	R
RUNTIME	active_vf_menu_reqs	gauge	R	R
RUNTIME	total_vf_menu_reqs	integer	R	R
RUNTIME	active_vf_reqs	gauge	R	R
RUNTIME	active_vf_enable	gauge	R	R
RUNTIME	active_vf_disable	gauge	R	R
RUNTIME	active_vf_cancel	gauge	R	R
RUNTIME	active_vf_send	gauge	R	R
RUNTIME	active_vf_receive	gauge	R	R
RUNTIME	active_vf_xceive	gauge	R	R
RUNTIME	total_vf_reqs	integer	R	R
RUNTIME	total_vf_enable	integer	R	R
RUNTIME	total_vf_disable	integer	R	R
RUNTIME	total_vf_cancel	integer	R	R
RUNTIME	total_vf_send	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Class	Field	Data Type	SNMP Access	RPC Access
RUNTIME	total_vf_receive	integer	R	R
RUNTIME	total_vf_xceive	integer	R	R
RUNTIME	total_tasks_executed	integer	R	R
RUNTIME	user1_time	time	RW	RW
RUNTIME	user2_time	time	RW	RW
RUNTIME	user3_time	time	RW	RW
RUNTIME	user1_data	integer	RW	RW
RUNTIME	user2_data	integer	RW	RW
RUNTIME	user3_data	integer	RW	RW
RUNTIME	user4_data	integer	RW	RW
RUNTIME	user5_data	integer	RW	RW
RUNTIME	user6_data	integer	RW	RW
RUNTIME	astlm_current »	gauge	R	R
RUNTIME	biolm_current »	gauge	R	R
RUNTIME	bytlm_current »	gauge	R	R
RUNTIME	diolm_current »	gauge	R	R
RUNTIME	enqlm_current »	gauge	R	R
RUNTIME	fillm_current »	gauge	R	R
RUNTIME	pgflquota_current »	gauge	R	R
RUNTIME	tqelm_current »	gauge	R	R
RUNTIME	wssize_current »	gauge	R	R
RUNTIME	channelcnt_current »	gauge	R	R
POOL	pool_coll_state	integer	R	R
POOL	mss_process_total	integer	R	R
POOL	mss_process_free	min gauge	R	R
POOL	mss_process_largest	min gauge	R	R
POOL	mss_process_failures	integer	R	R
POOL	mss_process_garbage	integer	R	R
ERROR	err_coll_state »	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Class	Field	Data Type	SNMP Access	RPC Access
ERROR	err_count »	integer	R	R
ERROR	last_err_msg »	string	R	R
ERROR	time_of_last_error »	time	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.3.1. Field Descriptions

Following are descriptions of the fields in the table above.

- `record_state`

  The current state of this table entry. Valid states are VALID (the process is currently running and maintaining this table entry) or INACTIVE (the process is no longer running). Inactive rows are subject to reuse.

- `id_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `process_name`

  The OpenVMS process name for the process.

- `pid`

  The OpenVMS process identifier for the process.

- `start_time`

  Date and time the process was started.

- `end_time`

  Date and time the process ended. If the process has not yet ended, this field is null.

- `user_name`

  The OpenVMS account under which the process is running.

- `acms_state`

  The ACMS state of the process.

- `cfg_coll_state`

  Collection states can be modified by modifying entries in the Collection table. If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `astlm_active`, `biolm_active`, `bytlm_active`, `diolm_active`, `enqlm_active`, `fillm_ active`, `pgflquota_active`, `tqelm_active`, `wsdefault_active`, `wsextent_active`, `wsquota_active`

  The default value of the related OpenVMS process quota. This is the value of the quota when the Agent process was started.

- `astlm_stored`, `biolm_stored`, `bytlm_stored`, `diolm_stored`, `enqlm_stored`, `fillm_stored`, `pgflquota_stored`, `tqelm_stored`, `wsdefault_stored`, `wsextent_stored`, `wsquota_stored`

  The value of the related process quota currently stored in the OpenVMS system user authorization file (SYSUAF.DAT).

- `rt_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `decnet_object`

  If the process has a current DECnet object, the value of this field is STARTED. Otherwise, the value is STOPPED. If the DECnet object is stopped (and the `runtime_coll_state` is enabled for this process), either distributed processing has not been enabled (that is, the `node_name` parameter in the ACMSGEN file is NULL) or there is currently a problem with DECnet. Also, check the ACC CONFIG parameters `node_name_active` and `node_name_stored` to determine the current status of the ACMSGEN `node_name` field.

- `active_task_calls`

  The number of task calls currently being executed by all users of the Agent.

- `current_attached_terms`

  The number of terminals currently attached to the Agent.

- `active_tdms_menu_reqs`

  The number of TDMS menu requests currently being executed by all users of the Agent.

- `total_tdms_menu_reqs`

  The total number of TDMS menu requests executed by all users of the Agent since the Agent was started.

- `active_tdms_reqs`

  The number of TDMS requests of all types currently being executed by all users of the Agent.

- `active_tdms_msgrd`

  The number of TDMS read messages currently being executed by all users of the Agent.

- `active_tdms_msgwt`

  The number of TDMS write messages currently being executed by all users of the Agent.

- `active_tdms_cancel`

  The number of TDMS cancels currently being executed by all users of the Agent.

- `total_tdms_reqs`

  The total number of TDMS requests (menu and exchange) executed by all users of the Agent since the Agent was started.

- `total_tdms_msgrd`

  The total number of TDMS read messages executed by all users of the Agent since the Agent was started.

- `total_tdms_msgwt`

  The total number of TDMS write messages executed by all users of the Agent since the Agent was started.

- `total_tdms_cancel`

  The total number of TDMS cancels executed by all users of the Agent since the Agent was started.

- `active_vf_menu_reqs`

  The number of VSI DECforms menu requests currently being executed by all users of the Agent.

- `total_vf_menu_reqs`

  The total number of VSI DECforms menu requests executed by all users of the Agent since the Agent was started.

- `active_vf_reqs`

  The number of VSI DECforms requests of all types currently being executed by all users of the Agent.

- `active_vf_enable`

  The number of VSI DECforms enable requests currently being executed by all users of the Agent.

- `active_vf_disable`

  The number of VSI DECforms disable requests currently being executed by all users of the Agent.

- `active_vf_cancel`

  The number of VSI DECforms cancel requests currently being executed by all users of the Agent.

- `active_vf_send`

  The number of VSI DECforms requests currently being executed by all users of the Agent.

- `active_vf_receive`

  The number of VSI DECforms receive requests currently being executed by all users of the Agent.

- `active_vf_xceive`

  The number of VSI DECforms enable transceives currently being executed by all users of the Agent.

- `total_vf_reqs`

  The total number of VSI DECforms requests of all types executed by all users of the Agent since the Agent was started.

- `total_vf_enable`

  The total number of VSI DECforms enable requests executed by all users of the Agent since the Agent was started.

- `total_vf_disable`

  The total number of VSI DECforms disable requests executed by all users of the Agent since the Agent was started.

- `total_vf_cancel`

  The total number of VSI DECforms cancel requests executed by all users of the Agent since the Agent was started.

- `total_vf_send`

  The total number of VSI DECforms send requests executed by all users of the Agent since the Agent was started.

- `total_vf_receive`

  The total number of VSI DECforms receive requests executed by all users of the Agent since the Agent was started.

- `total_vf_xceive`

  The total number of VSI DECforms transceive requests executed by all users of the Agent since the Agent was started.

- `total_tasks_executed`

  The total number of tasks started in the Agent since the Agent was started.

- `user1_time`, `user2_time`, `user3_time`, `user1_data`, `user2_data`, `user3_data`, `user4_data`, `user5_data`, `user6_data`

  Additional generic runtime fields that are available to programmers and Agent developers.

- `astlm_current`, `biolm_current`, `bytlm_current`, `diolm_current`, `enqlm_current`, `fillm_current`, `pgflquota_current`, `tqelm_current`, `wssize_current`, `channelcnt_current`

  The actual amount of the related OpenVMS process or system resource that is being consumed by the Agent process. The frequency with which these fields are updated is based on the value of the `vms_coll_interval` field in the Parameter table.

- `pool_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `mss_process_total`

  The total size of the MSS process pool allocated for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval`.

- `mss_process_free`

  The amount of unused MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval`.

- `mss_process_largest`

  The largest unused block available in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval`.

- `mss_process_failures`

  The number of failed attempts to allocate space in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval`.

- `mss_process_garbage`

  The number of garbage collections that have been run to reclaim space in this processes MSS process pool. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval`.

- `error_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. Errors for this process are only sent to the Remote Manager if this field is set to ENABLED. If this field is set to DISABLED, the process will not collect data for the fields in this class; existing field values reflect activity during a prior period when collection was enabled.

- `error_count`

  The total number of errors related to this process that were sent to the Remote Manager.

- `last_error_message`

The text of the most recent error message related to this process that was sent to the Remote Manager.

- `time_of_last_error`

  Date and time of the most recent error message related to this process that was sent to the Remote Manager.

9.4. Collection Table

The Collection table is populated from the configuration file maintained by the user on the local system (or in a cluster common area) when the ACMS run-time system is started.

This table can be used at run time to enable or disable data collection by entity and class. The primary key to this table is the combination of entity, class, and name. Duplicate rows are not allowed.

Table 9.3. Collection Table

Field	Data Type	SNMP Access	RPC Access	Configuration Access
entity §	string	R	RW	RW
class §	string	R	RW	RW
name §	string	R	RW	RW
coll_state §	state1	RW	RW	RW
storage_location § »	string	R	RW	RW
storage_state § »	state1	R	RW	RW
storage_interval § »	integer	R	RW	RW
storage_begin_time § »	time	RW	RW	RW
storage_end_time § »	time	RW	RW	RW

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.4.1. Field Descriptions

Following are descriptions of the fields in *Table 9.3, "Collection Table"*.

- `entity`

  Name or type of the entity. Valid values are ACC, CP, EXC, GROUP, QTI, SERVER, and TSC.

- `class`

  Class of data to be collected. Valid values are CONFIG, ID, POOL, and RUNTIME.

- `name`

  A name for the entity that helps to uniquely identify an instance of the entity type. Possible entity names are:

  - ACC, CP, QTI, TSC (process name)
  - EXC (application name)
  - GROUP (task group name
  - SERVER (server name)

  Name can include the following wildcard values:

  - asterisk (*) (matches all characters)
  - exclamation point (!) (negation)

- `coll_state`

  Current state as configured, either from the configuration file or by a user at run time. Valid values are ENABLED or DISABLED. A change to this field causes collection to be initiated or terminated.

- `storage_location`

  An OpenVMS file specification (or logical name) that indicates where collection data is to be writer. The default value for this field the logical ACMS$MGMT_SNAPSHOT.

  If a directory is not specified as part of the file specification, the file is placed in the default directory of the account under which the Remote Manager process is running.

  Multiple collection entities can share and write output to the same data snapshot file.

  See *Section 5.1, "Managing Data Collection"* for more information on creating data snapshots by periodically saving data collections.

- `storage_state`

  Current state of data snapshots as configured either from the configuration file or by a user at run time. Valid values are ENABLED or DISABLED. A change to this field causes data snapshots to be either initiated or terminated.

  In order for data snapshots to be fully functional, the `coll_state field` must be set to ENABLED, and the current time must fall between the specified `storage_begin_time` and `storage_end_time` values.

  See *Section 5.1, "Managing Data Collection"* for more information on creating data snapshots by saving periodically data collections.

- storage_interval

  Controls the frequency (in seconds) at which data snapshots are collected. A lower value causes data snapshots to be collected more often; a higher value causes data snapshots to be collected less often.

  The `storage_interval` value should be a multiple of the `timer_interval` value. The `timer_interval` value determines the minimum elapsed time for many Remote Manager

parameters, including the storage interval setting. The relationship of the values in these two fields determine how often data snapshots are performed, for example:

- If the `timer_interval` value is greater, its value is used by default. For instance, if the `timer_interval` is 10 and the `storage_interval` is 5, snapshots will be written at 10 second intervals.
- If the `storage_interval` value is greater and is a multiple of the `timer_ interval`, the `storage_interval` value is used. For example, if the `timer_ interval` is 10 and the `storage_interval` is 30, snapshots will be written at 30 second intervals.
- If the `storage_interval` value is greater and is not a multiple of the `timer_interval`, the next multiple of the `timer_interval` value is used. For example, if the `timer_interval` is 10 and the `storage_interval` is 15, snapshots will be written at 20 second intervals.

See *Section 5.1, "Managing Data Collection"* for more information on creating data snapshots by periodically saving data collections.

- `storage_begin_time`

  The date and time (*DD-MMM-YY:hh:mm:ss.nn*) when the data snapshot interval is scheduled to begin. The default value is NOW (begin immediately).

  See *Section 5.1, "Managing Data Collection"* for more information on creating data snapshots by periodically saving data collections.

- `storage_end_time`

  The date and time (*DD-MMM-YY:hh:mm:ss.nn*) when the data snapshot interval is scheduled to end. The default value is NEVER (run indefinitely).

  See *Section 5.1, "Managing Data Collection"* for more information on creating data snapshots by periodically saving data collections.

9.5. CP Table

The CP table contains a row for each terminal Command Process (CP) running on the node.

Table 9.4. CP Table

Collection Class	Field	Data Type	SNMP Access	RPC Access
ID	record_state	integer	R	R
ID	id_coll_state	integer	R	R
ID	process_name	string	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Access
ID	pid	integer	R	R
ID	start_time	time	R	R
ID	end_time	time	R	R
ID	user_name	string	R	R
CONFIG	astlm_active »	integer	R	R
CONFIG	astlm_stored »	integer	RW	RW
CONFIG	biolm_active »	integer	R	R
CONFIG	biolm_stored »	integer	RW	RW
CONFIG	bytlm_active »	integer	R	R
CONFIG	bytlm_stored »	integer	RW	RW
CONFIG	diolm_active »	integer	R	R
CONFIG	diolm_stored »	integer	RW	RW
CONFIG	enqlm_active »	integer	R	R
CONFIG	enqlm_stored »	integer	RW	RW
CONFIG	fillm_active »	integer	R	R
CONFIG	fillm_stored »	integer	RW	RW
CONFIG	pgflquota_active »	integer	R	R
CONFIG	pgflquota_stored »	integer	RW	RW
CONFIG	tqelm_active »	integer	R	R
CONFIG	tqelm_stored »	integer	RW	RW
CONFIG	wsdefault_active »	integer	R	R
CONFIG	wsdefault_stored »	integer	RW	RW
CONFIG	wsextent_active »	integer	R	R
CONFIG	wsextent_stored »	integer	RW	RW
CONFIG	wsquota_active »	integer	R	R
CONFIG	wsquota_stored »	integer	RW	RW
RUNTIME	runtime_coll_state	integer	R	R
RUNTIME	acms_state	integer	R	R
RUNTIME	decnet_object	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Access
RUNTIME	current_attached_terminals	gauge	R	R
RUNTIME	active_task_calls	gauge	R	R
RUNTIME	active_tdms_menu_reqs	gauge	R	R
RUNTIME	total_tdms_menu_reqs	integer	R	R
RUNTIME	active_tdms_reqs	gauge	R	R
RUNTIME	active_tdms_read_msgs	gauge	R	R
RUNTIME	active_tdms_write_msgs	gauge	R	R
RUNTIME	active_tdms_cancels	gauge	R	R
RUNTIME	total_tdms_reqs	integer	R	R
RUNTIME	total_tdms_read_msgs	integer	R	R
RUNTIME	total_tdms_write_msgs	integer	R	R
RUNTIME	total_tdms_cancels	integer	R	R
RUNTIME	active_df_menu_reqs	gauge	R	R
RUNTIME	total_df_menu_reqs	integer	R	R
RUNTIME	active_df_reqs	gauge	R	R
RUNTIME	active_df_enables	gauge	R	R
RUNTIME	active_df_disables	gauge	R	R
RUNTIME	active_df_cancels	gauge	R	R
RUNTIME	active_df_sends	gauge	R	R
RUNTIME	active_df_receives	gauge	R	R
RUNTIME	active_df_transceives	gauge	R	R
RUNTIME	total_df_reqs	integer	R	R
RUNTIME	total_df_enables	integer	R	R
RUNTIME	total_df_disables	integer	R	R
RUNTIME	total_df_cancels	integer	R	R
RUNTIME	total_df_sends	integer	R	R
RUNTIME	total_df_receives	integer	R	R
RUNTIME	total_df_transceives	integer	R	R
RUNTIME	data_set_hangups	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Access
RUNTIME	astlm_current »	gauge	R	R
RUNTIME	biolm_current »	gauge	R	R
RUNTIME	bytlm_current »	gauge	R	R
RUNTIME	diolm_current »	gauge	R	R
RUNTIME	enqlm_current »	gauge	R	R
RUNTIME	fillm_current »	gauge	R	R
RUNTIME	pgflquota_current »	gauge	R	R
RUNTIME	tqelm_current »	gauge	R	R
RUNTIME	wssize_current »	gauge	R	R
RUNTIME	channelcnt_current »	gauge	R	R
POOL	pool_coll_state	integer	R	R
POOL	mss_process_total	integer	R	R
POOL	mss_process_free	min gauge	R	R
POOL	mss_process_largest	min gauge	R	R
POOL	mss_process_failures	integer	R	R
POOL	mss_process_garbage	integer	R	R
ERROR	err_coll_state »	integer	R	R
ERROR	err_count »	integer	R	R
ERROR	last_err_msg »	string	R	R
ERROR	time_of_last_error »	time	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.5.1. Field Descriptions

Following are descriptions of the fields in *Table 9.4, "CP Table"*.

- `record_state`

  The current state of this table entry. Valid states are VALID (the process is currently running and maintaining this table entry) or INACTIVE (the process is no longer running). Inactive rows are subject to reuse.

- `id_coll_state`

The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `process_name`

  The OpenVMS process name for the process.

- `pid`

  The OpenVMS process identifier for the process.

- `start_time`

  Date and time the process was started.

- `end_time`

  Date and time the process ended. If the process has not yet ended, this field is null.

- `user_name`

  The OpenVMS account under which the process is running.

- `astlm_active`, `biolm_active`, `bytlm_active`, `diolm_active`, `enqlm_active`, `fillm_active`, `pgflquota_active`, `tqelm_active`, `wsdefault_active`, `wsextent_active`, `wsquota_active`

  The default value of the related OpenVMS process quota. This is the value of the quota when the CP process was started.

- `astlm_stored`, `biolm_stored`, `bytlm_stored`, `diolm_stored`, `enqlm_stored`, `fillm_ stored`, `pgflquota_stored`, `tqelm_stored`, `wsdefault_stored`, `wsextent_stored`, `wsquota_stored`

  The value of the related process quota currently stored in the OpenVMS system user authorization file (SYSUAF.DAT).

- `runtime_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `acms_state`

  The ACMS state of the process.

- `decnet_object`

  If the process has a current DECnet object, the value of this field is STARTED. Otherwise, the value is STOPPED. If the DECnet object is stopped (and the `runtime_coll_state` is enabled for this

process), either distributed processing has not been enabled (that is, the node_name parameter in the ACMSGEN file is NULL) or there is currently a problem with DECnet. Also, check the ACC CONFIG parameters node_name_active and node_name_stored to determine the current status of the ACMSGEN node_name field.

- current_attached_terminals

  The number of terminals currently attached to the CP.

- active_task_calls

  The number of task calls currently being executed by all users of the CP.

- active_tdms_menu_reqs

  The number of TDMS menu requests currently being executed by all users of the CP.

- total_tdms_menu_reqs

  The total number of TDMS menu requests executed by all users of the CP since the CP was started.

- active_tdms_reqs

  The number of TDMS requests of all types currently being executed by all users of the CP.

- active_tdms_read_msgs

  The number of TDMS read messages currently being executed by all users of the CP.

- active_tdms_write_msgs

  The number of TDMS write messages currently being executed by all users of the CP.

- active_tdms_cancels

  The number of TDMS cancels currently being executed by all users of the CP.

- total_tdms_reqs

  The total number of TDMS requests (menu and exchange) executed by all users of the CP since the CP was started.

- total_tdms_read_msgs

  The total number of TDMS read messages executed by all users of the CP since the CP was started.

- total_tdms_write_msgs

  The total number of TDMS write messages executed by all users of the CP since the CP was started.

- total_tdms_cancels

  The total number of TDMS cancels executed by all users of the CP since the CP was started.

- active_df_menu_reqs

  The number of DECforms menu requests currently being executed by all users of the CP.

- `total_df_menu_reqs`

  The total number of DECforms menu requests executed by all users of the CP since the CP was started.

- `active_df_reqs`

  The number of DECforms requests of all types currently being executed by all users of the CP.

- `active_df_enables`

  The number of DECforms enable requests currently being executed by all users of the CP.

- `active_df_disables`

  The number of DECforms disable requests currently being executed by all users of the CP.

- `active_df_cancels`

  The number of DECforms cancel requests currently being executed by all users of the CP.

- `active_df_sends`

  The number of DECforms requests currently being executed by all users of the CP.

- `active_df_receives`

  The number of DECforms receive requests currently being executed by all users of the CP.

- `active_df_transceives`

  The number of DECforms enable transceives currently being executed by all users of the CP.

- `total_df_reqs`

  The total number of DECforms requests of all types executed by all users of the CP since the CP was started.

- `total_df_enables`

  The total number of DECforms enable requests executed by all users of the CP since the CP was started.

- `total_df_disables`

  The total number of DECforms disable requests executed by all users of the CP since the CP was started.

- `total_df_cancels`

  The total number of DECforms cancel requests executed by all users of the CP since the CP was started.

- `total_df_sends`

  The total number of DECforms send requests executed by all users of the CP since the CP was started.

- `total_df_receives`

  The total number of DECforms receive requests executed by all users of the CP since the CP was started.

- `total_df_transceives`

  The total number of DECforms transceive requests executed by all users of the CP since the CP was started.

- `data_set_hangups`

  The total number of data set hangups detected by the CP since the CP was started.

- `astlm_current`, `biolm_current`, `bytlm_current`, `diolm_current`, `enqlm_current`, `fillm_current`, `pgflquota_current`, `tqelm_current`, `wssize_current`, `channelcnt_current`

  The actual amount of the related OpenVMS process or system resource that is being consumed by the CP process. The frequency with which these fields are updated is based on the value of the `vms_coll_interval` field in the *Table 9.9, "Parameter Table"*.

- `pool_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 9.4, "Collection Table"* and *Section 5.1, "Managing Data Collection"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `mss_process_total`

  The total size of the MSS process pool allocated for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_free`

  The amount of unused MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_largest`

  The largest unused block available in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_failures`

  The number of failed attempts to allocate space in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_garbage`

The number of garbage collections that have been run to reclaim space in this processes MSS process pool. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `err_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 9.4, "Collection Table"* and *Section 5.1, "Managing Data Collection"* for discussions of data collection.

  Errors for this process are only sent to the Remote Manager if this field is set to ENABLED. If this field is set to DISABLED, the process will not collect data for the fields in this class; existing field values reflect activity during a prior period when collection was enabled.

- `err_count`

  The total number of errors related to this process that were sent to the Remote Manager.

- `last_err_msg`

  The text of the most recent error message related to this process that was sent to the Remote Manager.

- `time_of_last_error`

  Date and time of the most recent error message related to this process that was sent to the Remote Manager.

9.6. Error Filter Table

The Error Filter table enables users to increase the usefulness and manage the size of the Remote Manager error log by preventing an ACMS process from relaying repetitive or non-critical system messages. Each error filter record specifies a particular system message to suppress. These messages continue to be displayed on ACMS systems, but they are not sent to or recorded by the Remote Manager.

The following table shows the fields in an ACMS error filter record. Note that the data in this table is persistent and durable. However, table entries can removed manually using the ACMSMGR **`DELETE FILTER`** command.

Table 9.5. Error Filter Table

Field Name	Data Type	SNMP Access	RPC Access
node_name »	string	RW	RW
message_code »	integer	RW	RW
message_name »	string	RW	RW

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.6.1. Field Descriptions

Following are descriptions of the fields in *Table 9.5, "Error Filter Table"*:

- `node_name`

  Name of the OpenVMS node on which to filter error messages.

- `message_code`

  The hexadecimal value or symbolic error name associated with the message being filtered.

9.7. EXC Table

The EXC table is sized according the MAX_APPLS ACMSGEN parameter.

Table 9.6. EXC Table

Collection Class	Field	Data Type	SNMP Access	RPC Acess
ID	record_state	integer	R	R
ID	id_coll_state	integer	R	R
ID	process_name	string	R	R
ID	user_name	string	R	R
ID	pid	integer	R	R
ID	start_time	time	R	R
ID	end_time	time	R	R
ID	appl_name	string	R	R
ID	build_time	time	R	R
ID	exc_appl_tbl_state	integer	R	
ID	exc_server_types	integer	R	
ID	exc_task_groups	integer	R	
CONFIG	config_coll_state	integer	R	R
CONFIG	acms_state §	integer	RW	R
CONFIG	audit_state_active §	integer	RW	RW
CONFIG	audit_state_stored	state1	R	R
CONFIG	max_tasks_active §	integer	RW	RW
CONFIG	max_tasks_stored	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Acess
CONFIG	sp_monitoring_interval_active §	integer	RW	RW
CONFIG	sp_monitoring_interval_stored	state1	R	R
CONFIG	max_servers_active §	integer	RW	RW
CONFIG	max_servers_stored	integer	R	R
CONFIG	transaction_timeout_active §	integer	RW	RW
CONFIG	transaction_timeout_stored	integer	R	R
CONFIG	astlm_active »	integer	R	R
CONFIG	astlm_stored »	integer	RW	RW
CONFIG	biolm_active »	integer	R	R
CONFIG	biolm_stored »	integer	RW	RW
CONFIG	bytlm_active »	integer	R	R
CONFIG	bytlm_stored »	integer	RW	RW
CONFIG	diolm_active »	integer	R	R
CONFIG	diolm_stored »	integer	RW	RW
CONFIG	enqlm_active »	integer	R	R
CONFIG	enqlm_stored »	integer	RW	RW
CONFIG	fillm_active »	integer	R	R
CONFIG	fillm_stored »	integer	RW	RW
CONFIG	pgflquota_active »	integer	R	R
CONFIG	pgflquota_stored »	integer	RW	RW
CONFIG	tqelm_active »	integer	R	R
CONFIG	tqelm_stored »	integer	RW	RW
CONFIG	wsdefault_active »	integer	R	R
CONFIG	wsdefault_stored »	integer	RW	RW
CONFIG	wsextent_active »	integer	R	R
CONFIG	wsextent_stored »	integer	RW	RW
CONFIG	wsquota_active »	integer	R	R
CONFIG	wsquota_stored »	integer	RW	RW
CONFIG	ws_poolsize »	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Acess
CONFIG	wsc_poolsize »	integer	R	R
CONFIG	tws_poolsize »	integer	R	R
CONFIG	twsc_poolsize »	integer	R	R
RUNTIME	runtime_coll_state	integer	R	R
RUNTIME	decnet_object	integer	R	R
RUNTIME	current_servers	gauge	R	R
RUNTIME	current_submitters	gauge	R	R
RUNTIME	current_tasks	gauge	R	R
RUNTIME	total_tasks_executed	integer	R	R
RUNTIME	total_submitters	integer	R	R
RUNTIME	current active_servers	gauge	R	R
RUNTIME	current_free_servers	gauge	R	R
RUNTIME	current_waiting_tasks	gauge	R	R
RUNTIME	server_start_count	integer	R	R
RUNTIME	server_failure_count	integer	R	R
RUNTIME	task_failures	integer	R	R
RUNTIME	task_start_failures	integer	R	R
RUNTIME	task_security_failures	integer	R	R
RUNTIME	task_cancels	integer	R	R
RUNTIME	active_tdms_requests	gauge	R	R
RUNTIME	active_tdms_read_messages	gauge	R	R
RUNTIME	active_tdms_write_messages	gauge	R	R
RUNTIME	active_tdms_cancels	gauge	R	R
RUNTIME	total_tdms_requests	integer	R	R
RUNTIME	total_tdms_read_messages	integer	R	R
RUNTIME	total_tdms_write_messages	integer	R	R
RUNTIME	total_tdms_cancels	integer	R	R
RUNTIME	total_dataset_hangups	integer	R	R
RUNTIME	astlm_current »	gauge	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Acess
RUNTIME	biolm_current »	gauge	R	R
RUNTIME	bytlm_current »	gauge	R	R
RUNTIME	diolm_current »	gauge	R	R
RUNTIME	enqlm_current »	gauge	R	R
RUNTIME	fillm_current »	gauge	R	R
RUNTIME	pgflquota_current »	gauge	R	R
RUNTIME	tqelm_current »	gauge	R	R
RUNTIME	wssize_current »	gauge	R	R
RUNTIME	channelcnt_current »	gauge	R	R
POOL	pool_coll_state	integer	R	R
POOL	mss_process_total	integer	R	R
POOL	mss_process_free	min gauge	R	R
POOL	mss_process_largest	min gauge	R	R
POOL	mss_process_failures	integer	R	R
POOL	mss_process_garbage	integer	R	R
POOL	ws_pool_total	integer	R	R
POOL	ws_pool_used »	gauge	R	R
POOL	ws_pool_free	min gauge	R	R
POOL	ws_pool_largest	min gauge	R	R
POOL	ws_pool_failures	integer	R	R
POOL	ws_pool_garbage	integer	R	R
POOL	wsc_pool_total	integer	R	R
POOL	wsc_pool_used »	gauge	R	R
POOL	wsc_pool_free	min gauge	R	R
POOL	wsc_pool_largest	min gauge	R	R
POOL	wsc_pool_failures	integer	R	R
POOL	wsc_pool_garbage	integer	R	R
POOL	tws_pool_total »	integer	R	R
POOL	twsc_poolsize_total »	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Acess
POOL	tws_pool_large_used »	gauge	R	R
POOL	twsc_pool_total »	integer	R	R
POOL	twsc_poolsize_total »	integer	R	R
POOL	twsc_pool_large_used »	gauge	R	R
ERROR	err_coll_state »	integer	R	R
ERROR	err_count »	integer	R	R
ERROR	last_err_msg »	string	R	R
ERROR	time_of_last_error »	time	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.7.1. Field Descriptions

Following are descriptions of the fields in *Table 9.6, "EXC Table"*.

- `record_state`

  The current state of this table entry. Valid states are VALID (the process is currently running and maintaining this table entry) or INACTIVE (the process is no longer running). Inactive rows are subject to reuse.

- `id_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `process_name`

  The OpenVMS process name for the process.

- `user_name`

  The OpenVMS account under which the process is running.

- `pid`

  The OpenVMS process identifier of the process.

- `start_time`

  Date and time the process was started.

- `end_time`

Date and time the process ended. If the process has not yet ended, this field will be null.

- `appl_name`

  Name of the application.

- `build_time`

  Date and time the application database (ADB) was built.

- `exc_appl_tbl_state`

  This field is available to the SNMP interface only. It contains the state of the application global section for this EXC. When EXCs have completed their startup, they construct global sections containing server and task group tables. If this field is not MGMT$K_VALID (2), the Server and Task Group tables are not available.

- `exc_server_types`

  This field is available to the SNMP interface only. It contains the number of server types contained in the application, which is also the number of rows in the Server table for this EXC.

- `exc_task_groups`

  This field is available to the SNMP interface only. It contains the number of task groups contained in the application, which is also the number of rows in the Task Group table for this EXC.

- `config_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `audit_state_active`

  Current auditing state of the application.

- `audit_state_stored`

  Value of the auditing state of the application as specified in the ACMS application database (.ADB file).

- `max_tasks_active`

  The current maximum number of executing tasks allowed.

- `max_tasks_stored`

  The maximum number of executing tasks allowed as specified in the ACMS application database (.ADB file).

- `sp_monitoring_interval_active`

  The current server process monitoring interval for the application.

- `sp_monitoring_interval_stored`

The server process monitoring interval for the application as specified in the ACMS application database (.ADB file).

- `max_servers_active`

  The current maximum number of started server instances for the application.

- `max_servers_stored`

  The maximum number of started server instances for the application as specified in the ACMS application database (.ADB file).

- `transaction_timeout_active`

  The current default task timeout for the application.

- `transaction_timeout_stored`

  The default task timeout for the application as specified in the ACMS application database (.ADB file).

- `astlm_active`, `biolm_active`, `bytlm_active`, `diolm_active`, `enqlm_active`, `fillm_active`, `pgflquota_active`, `tqelm_active`, `wsdefault_active`, `wsextent_active`, `wsquota_active`

  The default value of the related OpenVMS process quota. This is the value of the quota when the EXC process was started.

- `astlm_stored`, `biolm_stored`, `bytlm_stored`, `diolm_stored`, `enqlm_stored`, `fillm_ stored`, `pgflquota_stored`, `tqelm_stored`, `wsdefault_stored`, `wsextent_stored`, `wsquota_stored`

  The value of the related process quota currently stored in the OpenVMS system user authorization file (SYSUAF.DAT).

- `ws_poolsize`, `wsc_poolsize`

  The default size (in pagelets) of WS or WSC process pool for the application.

- `tws_poolsize`, `twsc_poolsize`

  The default size (in pagelets) of TWS or TWSC process pool for the application.

- `acms_state`

  The current ACMS state of this process.

- `runtime_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `decnet_object`

  If the process has a current DECnet object, the value of this field is STARTED. Otherwise, the value is STOPPED. If the DECnet object is stopped (and the `runtime_coll_state` is enabled for this process), either distributed processing has not been enabled (that is, the `node_name` parameter in the ACMSGEN file is NULL) or there is currently a problem with DECnet. Also, check the ACC CONFIG parameters `node_name_active` and `node_name_stored` to determine the current status of the ACMSGEN `node_name` field.

- `current_servers`

  The number of server instances currently started in this application.

- `current_submitters`

  The number of submitters currently logged in to this application.

- `current_tasks`

  The number of tasks currently started in the application.

- `total_tasks_executed`

  The total number of tasks started in the application since the application was started.

- `total_submitters`

  The total number of submitters who have submitted tasks to this application since the application was started.

- `current_active_servers`

  The currrent number of active servers (that is, those servers peforming processing steps).

- `current_free_servers`

  The number of started servers which are not currently active (that is, not currently executing processing steps).

- `current_waiting_tasks`

  The number of tasks that are not executing, waiting for a procedure server to become available.

- `server_start_count`

  The number of times servers have been started in this application.

- `server_failure_count`

  The number of times servers have been stopped in this application.

- `server_process_total`

  The number of concurrent server processes (active or free) running in this application.

- `task_failures`

  The number of tasks in this application that have failed to complete successfully.

- `task_start_failures`

  The number of tasks in this application that have failed to start.

- `task_security_failures`

  The number of tasks in this application that have failed to start because of security violations.

- `task_cancels`

  The number of tasks in this application that have been cancelled.

- `active_tdms_requests`

  The number of TDMS requests (both exchange and menu) that are currently executing for this process.

- `active_tdms_read_messages`

  The number of TDMS read messages currently outstanding for this process.

- `active_tdms_write_messages`

  The number of TDMS write messages currently outstanding for this process.

- `active_tdms_cancels`

  The number of TDMS cancels currently outstanding for this process.

- `total_tdms_requests`

  The total number of TDMS requests (both exchange and menu) processed by this process while the `runtime_coll_state` has been ENABLED.

- `total_tdms_read_messages`

  The total number of TDMS read messages processed by this process while the `runtime_coll_state` has been ENABLED.

- `total_tdms_write_messages`

  The total number of TDMS write messages processed by this process while the `runtime_coll_state` has been ENABLED.

- `total_tdms_cancels`

  The total number of TDMS cancels processed by this process while the `runtime_coll_state` has been ENABLED.

- `total_dataset_hangups`

  The total number of TDMS dataset hangups (unexpected session interruptions) processed by this process while the `runtime_coll_state` has been ENABLED.

- `astlm_current`, `biolm_current`, `bytlm_current`, `diolm_current`, `enqlm_current`, `fillm_current`, `pgflquota_current`, `tqelm_current`, `wssize_current`, `channelcnt_current`

The actual amount of the related OpenVMS process or system resource that is being consumed by the EXC process.

- `pool_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `mss_process_total`

  The total size of the MSS process pool allocated for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_free`

  The amount of unused MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_largest`

  The largest unused block available in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_failures`

  The number of failed attempts to allocate space in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_garbage`

  The number of garbage collections for this process that have been run to reclaim space in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `ws_pool_total`, `wsc_pool_total`

  The total size of the WS or WSC pool (in bytes) allocated for this application. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `ws_pool_used`, `wsc_pool_used`

  The amount of WS or WSC pool (in bytes) used by this application. The frequency with which these fields are updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `ws_pool_largest`, `wsc_pool_largest`

The largest unused block available in the WS or WSC pool for this application. The frequency with which these fields are updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `ws_pool_free`, `wsc_pool_free`

  The amount of unused WS or WSC pool (in bytes) for this application. The frequency with which these fields are updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `ws_pool_failures`, `wsc_pool_failures`

  The number of failed attempts to allocate space in the WS or WSC pool for this application. The frequency with which these fields are updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `ws_pool_garbage`, `wsc_pool_garbage`

  The number of garbage collections that have been run to reclaim space in this application's WS pool. The frequency with which these fields are updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `tws_pool_total`,`twsc_pool_total`

  The default size for TWS or TWSC pools (in bytes) allocated for this application. The frequency with which these fields are updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `tws_poolsize_total`, `twsc_poolsize_total`

  The total size of all TWS or TWSC pools (in bytes) for all task groups in this application. The frequency with which these fields are updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `tws_pool_large_used`, `twsc_pool_large_used`

  The largest amount of TWS or TWSC pool used (in bytes) by a task group in this application. The frequency with which these fields are updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `err_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 9.4, "Collection Table"* and *Section 5.1, "Managing Data Collection"* for discussions of data collection.

  Errors for this process are only sent to the Remote Manager if this field is set to ENABLED. If this field is set to DISABLED, the process will not collect data for the fields in this class; existing field values reflect activity during a prior period when collection was enabled.

- `err_count`

  The total number of errors generated by this process. related to this process that were sent to the Remote Manager.

- `last_err_msg`

The text of the most recent error message related to this process that was sent to the Remote Manager.

- `time_of_last_error`

  Date and time of the most recent error message related to this process that was sent to the Remote Manager.

9.8. Interfaces Table

The Interfaces table is populated from the configuration file by the ACMS Remote Manager process during process startup. This table specifies which interfaces are active and contains parameters associated with each interface. By default, the RPC interface is started; SNMP is not started.

Table 9.7. Interfaces Table

Field	Data Type	SNMP Access	RPC Access	Configuration Access
interface	string	R	R	R
state §	state1	RW	RW	RW
running_state	state2	R	R	R
get_request_count	integer	R	R	
set_request_count	integer	R	R	
alarms_sent	integer	R	R	
time_alarm_last_sent	integer	R	R	

Table Key

§ – Field is dynamic
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.8.1. Field Descriptions

Following are descriptions of the fields in *Table 9.7, "Interfaces Table"*.

- `interface`

  Name or type of the interface. Valid values are RPC or SNMP.

- `state`

  Current state as configured, either from the configuration file or by a user at run time. Valid values are ENABLED or DISABLED. Note that `state` and `acms_state` are not always the same because of potential run-time failures in a thread. For instance, if a thread fails to start, `state` may be ENABLED, but `acms_state` may be STOPPED.

  A thread can be enabled only if the `acms_state` value is STOPPED. A thread can be disabled only if the `acms_state` value is not STOPPED.

- `acms_state`

  Actual execution state. Interfaces go through the following states:

  - INITING

    The Remote Manager is in the process of creating the interface thread.

  - STARTING

    The interface thread has been created and is initializing.

  - STARTED

    The interface thread has completed initializing and is now running.

  - STOPPING

    The thread is starting shutdown, as the result of either a stop request or a fatal error.

  - STOPPED

    The thread is no longer executing.

- `get_request_count`

  The number of read requests submitted to the interface. This includes requests that are rejected because of authorization failures.

- `set_request_count`

  The number of write requests submitted to the interface. This includes requests that are rejected because of authorization failures.

- `alarms_sent`

  The number of alarms that have been sent by this interface. For SNMP, these are SNMP traps. For RPC, this field is undefined.

- `time_alarm_last_sent`

  The time the most recent alarm was sent by this interface. For SNMP, this is the time the last SNMP trap was sent. For RPC, this field is undefined.

9.9. Manager Status Table

The Manager Status table contains run-time values that reflect Remote Manager activity. This table is maintained internally by the Remote Manager and is read only to all external entities. Values in the table can be accessed through one of the supported interfaces. No changes can be made to the table by external users.

In general, the values in this table are informational only.

Table 9.8. Manager Status Table

Field Name	Data Type	SNMP Access	RPC Access
collection_count	integer	R	R
interfaces_count	integer	R	R
timer_count	integer	R	R
trap_count	integer	R	R
rpc_udp_state	state1	R	R
rpc_tcp_state	state1	R	R

Table Key

§ – Field is dynamic
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.9.1. Field Descriptions

Following are descriptions of the fields in *Table 9.8, "Manager Status Table"*.

- `collection_count`

  Current number of Collection table entries.

- `interfaces_count`

  Current number of entries in the Interfaces table.

- `timer_count`

  Current number of entries in the Timer table.

- `trap_count`

  Current number of entries in the Trap table.

- `rpc_udp_state`

  Current state of the RPC interface using the UDP protocol. A value of 1 means that the UDP protocol is active. A value of 0 means that the UDP protocol is inactive.

- `rpc_tcp_state`

  Current state of the RPC interface using the TCP protocol. A value of 1 means that the UDP protocol is active. A value of 0 means that the UDP protocol is inactive.

9.10. Parameter Table

The Parameter table contains values that control the operation of the ACMS Remote Manager and that are not directly related to any ACMS entity. This table is populated initially from the ACMSCFG file.

The Remote Manager maintains the table internally at run time; users can access data in the table only through one of the supported interfaces. Changes made to the table at run time are lost when the Remote Manager is stopped.

In general, the values in this table should be modified for fine tuning only, and only if a demonstrated need exists.

Note

All the fields in *Table 9.9, "Parameter Table"* are of type integer, and all fields have read and write access.

Table 9.9. Parameter Table

Field	Default Value	Minimum Value	Maximum Value	Interface
dcl_audit_level §	E	0	F	S,R,F
dcl_mgr_priority	5	1	10	S,R,F
dcl_stacksize	300	1	2147483647	S,R,F
error_interval § »	10	1	863999999	S,R,F
event_log_priority	5	1	10	S,R,F
log_stacksize	300	1	2147483647	S,R,F
login_creds_lifetime §	60	1	14399999	S,R,F
max_logins §	20	1	2147483647	S,R,F
max_rpc_return_recs	20	1	2147483647	S,R,F
mgr_audit_level §	E	0	F	S,R,F
msg_proc_audit_level §	E	0	F	S,R,F
msg_proc_priority	5	1	10	S,R,F
msg_proc_stacksize	300	1	2147483647	S,R,F
mss_coll_interval §	10	1	863999999	S,R,F
proc_mon_audit_level	E	0	F	S,R,F
proc_mon_interval §	30	1	14399999	S,R,F
proc_mon_priority	5	1	10	S,R,F
proc_mon_stacksize	300	1	2147483647	S,R,F
proxy_creds_lifetime §	60	1	14399999	S,R,F
rpc_audit_level §	E	0	F	S,R,F

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Field	Default Value	Minimum Value	Maximum Value	Interface
rpc_priority	5	1	10	S,R,F
rpc_stacksize	300	1	2147483647	S,R,F
security_audit_level §	E	0	F	S,R,F
snap_audit_level »	E	0	F	S,R,F
snap_priority »	5	1	10	S,R,F
snap_stacksize »	400	300	2147483647	S,R,F
snmp_agent_time_out §	10	1	863999999	S,R,F
snmp_audit_level §	E	0	F	S,R,F
snmp_are_you_there	300	1	863999999	S,R,F
snmp_priority	5	1	10	S,R,F
snmp_sel_time_out	5	0	863999999	S,R,F
snmp_stacksize	300	1	2147483647	S,R,F
timer_audit_level §	E	0	F	S,R,F
timer_interval §	30	1	863999999	S,R,F
timer_priority	5	1	10	S,R,F
timer_stacksize	300	1	2147483647	S,R,F
total_entity_slots	20	1	2147483647	S,R,F
trace_msg_wait_time §	5	1	14399999	S,R,F
trace_start_wait_time §	5	1	14399999	S,R,F
trap_audit_level §	E	0	F	S,R,F
trap_priority	5	1	10	S,R,F
trap_stacksize	300	1	2147483647	S,R,F
udp_enabled »	1	0	1	S,R,F
vms_coll_interval §	10	1	863999999	S,R,F
wksp_coll_interval § »	10	1	863999999	S,R,F
max_agents	2	1	2147483647	S,R,F

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.10.1. Field Descriptions

Following are descriptions of the fields in *Table 9.9, "Parameter Table"*.

- `dcl_audit_level`, `mgr_audit_level`, `msg_audit_level`, `proc_mon_audit_level`, `rpc_audit_level`, `security_audit_level`, `snmp_audit_level`, `timer_audit_level`, `trap_audit_level`

  Audit levels determine the amount of auditing information written for a given facility. Audit levels are specified using a hexidecimal value from 0 (none) to F (all). The integer values are a logical ORing of the following:

INFO	1
WARN	2
ERROR	4
FATAL	8

  For example, to specify auditing of both error and fatal information, specify a value of `C`. For more information about auditing and audit levels see *Section 4.7, "Managing Log Files"*.

- `dcl_mgr_priority`

  Relative priority of the DCL manager thread. The DCL manager is used to send ACMS run-time changes to the ACMS system. Priority is specified as a whole number between 1 and 10, where 1 is the lowest priority and 10 is the highest. This value should be left at the default.

- `dcl_stacksize`, `log_stacksize`, `msg_proc_stacksize`, `proc_mon_stacksize`, `rpc_stacksize`, `snmp_stacksize`, `timer_stacksize`, `trap_stacksize`

  These values determine the internal stack sizes for each thread. Stack sizes are set during thread creation and are not adjusted after the thread has been started. Restartable threads (such as RPC and SNMP) can be adjusted while the Remote Manager is running by disabling the interface, modifying the parameter, and then reenabling the interface.

- `event_log_priority`

  Relative priority of the event log thread. The event log thread writes audit messages to the audit log. Priority is specified as a whole number between 1 and 10, where 1 is the lowest priority and 10 is the highest. This value should be left at the default.

- `login_creds_lifetime`

  The amount of time (in minutes) that explict logins are valid. When a user logs in to a Remote Manager process using a valid OpenVMS account and password, a login is created for the user, and the expiration of that login is calculated and stored based on this parameter. When the current time is greater than the expiration time, the user is logged out and must log in again using the ACMSMGR **`LOGIN`** command. A change to this parameter takes effect for any login that takes place after the change is made. A change to this parameter does not take effect for any login that took place before the change was made.

- `max_logins`

  Maximum number of external processes allowed to concurrently connect to the Remote Manager. Starting the SNMP interface counts as one login. Each RPC client counts as one login. RPCs are serviced serially.

- `max_rpc_return_recs`

The maximum number of records to be returned to any given request for data. This parameter allows network bandwidth to be conserved by sending data in user-managed chunks.

- `msg_proc_priority`

  Relative priority of the message processor thread. The message processor is responsible for removing messages sent by ACMS processes to the Remote Manager from the error input queue and for processing messages according to configuration values specified in the Collection and Trap tables. This value should be left at the default. Priority is specified as a whole number between 1 and 10, where 1 is the lowest priority and 10 is the highest.

- `mss_coll_interval`

  Controls the frequency (in seconds) at which MSS values are collected. A lower value causes MSS values to be collected more often; a higher value causes MSS values to be collected less often. MSS values are collected by all ACMS run-time processes except SWL, ATR, and procedure servers.

- `proc_mon_interval`

  The frequency (in seconds) at which the process monitor thread should run. The process monitor thread checks for the existence of the ACC and other ACMS run-time processes in order to map the MGMT global section and to send alarms.

- `proc_mon_priority`

  Relative priority of the process monitor thread. The process monitor thread periodically checks for the existence of the ACC process in order to map the MGMT global section and to send alarms. Priority is specified as a whole number between 1 and 10, where 1 is the lowest priority and 10 is the highest. This value should be left at the default.

- `proxy_creds_lifetime`

  The amount of time (in minutes) that proxy logins are valid. When a user first accesses a Remote Manager process using an ACMS proxy, a login is created for the user, and the expiration of that login is calculated and stored based on this parameter. When the current time is greater than the expiration time, the user's proxy information is refreshed. A change to this parameter takes effect for any login that takes place after the change is made. A change to this parameter does not take effect for any login that took place before the change was made.

- `rpc_priority`

  Relative priority of the RPC management thread. The RPC management thread responds to RPC requests to get or set data values. Priority is specified as a whole number between 1 and 10, where 1 is the lowest priority and 10 is the highest.

- `snmp_agent_time_out`

  Number of seconds that the SNMP Master agent waits for a response from the Remote Manager. The maximum is 10 seconds for VSI TCP/IP Services Version 4.2. For VSI TCP/IP Services Version 5.0 and higher, the maximum is 60 seconds.

- `snmp_are_you_there`

  Controls how often are you there messages are sent by the Remote Manager to the SNMP Master agent. This value should be entered as a multiple of the `snmp_sel_time_out` value. Each time

a timeout occurs, a timeout counter is incremented. The product of the timeout counter and the snmp_sel_timeout are then compared to the snmp_are_you_there value. If the product is greater than the snmp_are_you_there value, an are_you_there message is sent.

- snmp_priority

  Relative priority of the SNMP management thread. The SNMP management thread responds to SNMP requests to get or set data values. Priority is specified as a whole number between 1 and 10, where 1 is the lowest and 10 is the highest.

- snmp_sel_time_out

  Controls how long the Remote Manager waits for a response from the SNMP master agent. If the timeout valueis reached and no messages are expected, the snmp_are_you_there interval is checked (see snmp_are_you_there). If a message is expected and is not received before the select times out, the connection to the master agent is assumed to have been lost and an attempt is made to reregister. There is a hard-coded 2 second wait prior to reregistration.

  This value also controls how long it takes to begin disabling this interface. Requests to disable the interface do not interrupt the socket select – they wait for it to either timeout or end naturally (that is, when a message is received). At worse case, a request to disable the interface has to wait snmp_sel_time_out seconds before the shutdown of the interface begins. Once it begins, it usually shuts down quickly – within a second or two.

- tcp_enabled

  Controls how the Remote Manager client and desktop server work together. If this field is set to 0, then the current TCP/IP connection is disabled. If both the tcp_enabled and udp_enabled fields are set to 0, the entire RPC thread is shut down.

- timer_interval

  The Remote Manager runs one internal timer that controls the operation of all other timers. The interval of this timer effectively sets the smallest timer interval for the process. The interval is set in seconds. If the value is too small, the timer will run frequently with no work to do. This value should be set to smallest desired timer interval.

- timer_priority

  Relative priority of the timer thread. The timer thread manages all internal timers. Priority is specified as a whole number between 1 and 10, where 1 is the lowest priority and 10 is the highest. This value should be left at the default.

- total_entity_slots

  The total number of Collection table entries to allow. When this number is reached, additional ACMSMGR **ADD COLLECTION** requests are rejected. Slots are allocated when the ACMS run-time system is started.

- trace_msg_wait_time

  The number of seconds the Remote Manager should wait for updates to the mss_coll_interval and wksp_coll_interval parameters to become effective (processed by the ACC). Updates to the ACC are sent by means of the trace monitor. The Remote Manager will poll the value being changed for up to trace_msg_wait_time seconds to see

whether the value was in fact changed. If it is not changed within this timeframe, the Remote Manager logs an error and returns an error to the caller.

- `trace_start_wait_time`

  The number of seconds the Remote Manager should wait for the trace monitor to be started. The Remote Manager communicates to ACMS process through the trace monitor. The Remote Manager attempts to start the trace monitor if the Remote Manager needs to send a message and the trace monitor is not already running. This value controls how long the Remote Manager will wait for the trace monitor to start before aborting the message send. Messages that are not sent are discarded (lost).

- `trap_priority`

  Relative priority of the trap sender thread. The trap sender thread dispatches trap messages to SNMP and RPC receivers. Priority is specified as a whole number between 1 and 10, where 1 is the lowest priority and 10 is the highest. This value should be left at the default.

- `udp_enabled`

  Controls how the Remote Manager client and desktop server work together. If this field is set to 0, a UDP listener is not registered for the Remote Manager. If both the `tcp_enabled` and `udp_enabled` fields are set to 0, the entire RPC thread is shut down.

- `vms_coll_interval`

  Controls the frequency (in seconds) at which OpenVMS process quota and system parameter values are collected. A lower value causes these values to be collected more often; a higher value causes these values to be collected less often. OpenVMS process quota values are collected by all ACMS processes; system parameter values are collected only by ACC.

- `wksp_coll_interval`

  Controls the frequency (in seconds) at which workspace (WS, WSC, TWS, TWSC) pool values are collected. A lower value causes workspace values to be collected more often; a higher value causes workspace pool values to be collected less often. Workspace pool values are collected only by ACC and EXC.

- `max_agents`

  Controls the value for the number of agents to be monitored. The default value is 2, and the minimum value is 1. This value sets the size of the table used to monitor agents. If the value is set too low, newly created agents will not be monitored.

9.11. QTI Table

The QTI table contains a single entry for QTI management information.

Table 9.10. QTI Table

Collection Class	Field	Data Type	SNMP Access	RPC Access
ID	record_state	integer	R	R
ID	id_coll_state	integer	R	R
ID	process_name	string	R	R
ID	pid	integer	R	R
ID	start_time	time	R	R
ID	end_time	time	R	R
CONFIG	config_coll_state	integer	R	R
CONFIG	acms_state §	integer	RW	R
CONFIG	qti_username_active	string	R	R
CONFIG	qti_username_stored	string	RW	RW
CONFIG	qti_priority_active	integer	R	R
CONFIG	qti_priority_stored	integer	RW	RW
CONFIG	max_threads	integer	RW	RW
CONFIG	sub_timeout_active §	integer	RW	RW
CONFIG	sub_timeout_stored	integer	RW	RW
CONFIG	retry_timer_active §	integer	RW	RW
CONFIG	retry_timer_stored	integer	RW	RW
CONFIG	polling_timer_active §	integer	RW	RW
CONFIG	polling_timer_stored	integer	RW	RW
CONFIG	astlm_active »	integer	R	R
CONFIG	astlm_stored »	integer	RW	RW
CONFIG	biolm_active »	integer	R	R
CONFIG	biolm_stored »	integer	RW	RW
CONFIG	bytlm_active »	integer	R	R
CONFIG	bytlm_stored »	integer	RW	RW
CONFIG	diolm_active »	integer	R	R
CONFIG	diolm_stored »	integer	RW	RW
CONFIG	enqlm_active »	integer	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Access
CONFIG	enqlm_stored »	integer	RW	RW
CONFIG	fillm_active »	integer	R	R
CONFIG	fillm_stored »	integer	RW	RW
CONFIG	pgflquota_active »	integer	R	R
CONFIG	pgflquota_stored »	integer	RW	RW
CONFIG	tqelm_active »	integer	R	R
CONFIG	tqelm_stored »	integer	RW	RW
CONFIG	wsdefault_active »	integer	R	R
CONFIG	wsdefault_stored »	integer	RW	RW
CONFIG	wsextent_active »	integer	R	R
CONFIG	wsextent_stored »	integer	RW	RW
CONFIG	wsquota_active »	integer	R	R
CONFIG	wsquota_stored »	integer	RW	RW
RUNTIME	runtime_coll_state	integer	R	R
RUNTIME	started_ queues	gauge	R	R
RUNTIME	current_tasks	gauge	R	R
RUNTIME	current_submitters	gauge	R	R
RUNTIME	task_successes	integer	R	R
RUNTIME	task_failures	integer	R	R
RUNTIME	task_retries	integer	R	R
RUNTIME	errors_queued	integer	R	R
RUNTIME	astlm_current »	gauge	R	R
RUNTIME	biolm_current »	gauge	R	R
RUNTIME	bytlm_current »	gauge	R	R
RUNTIME	diolm_current »	gauge	R	R
RUNTIME	enqlm_current »	gauge	R	R
RUNTIME	fillm_current »	gauge	R	R
RUNTIME	pgflquota_current »	gauge	R	R
RUNTIME	tqelm_current »	gauge	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Access
RUNTIME	wssize_current »	gauge	R	R
RUNTIME	channelcnt_current »	gauge	R	R
POOL	pool_coll_state	integer	R	R
POOL	mss_process_total	integer	R	R
POOL	mss_process_free	min gauge	R	R
POOL	mss_process_largest	min gauge	R	R
POOL	mss_process_failures	integer	R	R
POOL	mss_process_garbage	integer	R	R
ERROR	err_coll_state »	integer	R	R
ERROR	err_count »	integer	R	R
ERROR	last_err_msg »	string	R	R
ERROR	time_of_last_error »	time	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.11.1. Field Descriptions

Following are descriptions of the fields in *Table 9.10, "QTI Table"*.

- `record_state`

  The current state of this table entry. Valid states are VALID (the process is currently running and maintaining this table entry) or INACTIVE (the process is no longer running). Inactive rows are subject to reuse.

- `id_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `process_name`

  The OpenVMS process name for the process.

- `pid`

  The OpenVMS process identifier for the process.

- `start_time`

  Date and time the process was started.

- `end_time`

  Date and time the process ended. If the process has not yet ended, this field will be null.

- `config_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `qti_username_active`

  The OpenVMS account under which the QTI will run. This is the value of the ACMSGEN field when the process was started.

- `qti_username_stored`

  The value of the qti_username field currently stored in the ACMSGEN file.

- `qti_priority_active`

  The base priority for this process. This is the value of the ACMSGEN field when the process was started.

- `qti_priority_stored`

  The base priority currently stored in the ACMSGEN file for this process.

- `sub_timeout_active`

  The current value of the QTI submitter timeout.

- `sub_timeout_stored`

  The value of the `qti_sub_timeout` field in the current ACMSGEN file.

- `retry_timer_active`

  The current value of the QTI retry timer.

- `retry_timer_stored`

  The value of the `qti_retry_timer` field in the current ACMSGEN file.

- `polling_timer_active`

  The current value of the QTI polling timer.

- `polling_timer_stored`

  The value of the `qti_polling_timer` field in the current ACMSGEN file.

- `astlm_active`, `biolm_active`, `bytlm_active`, `diolm_active`, `enqlm_active`, `fillm_active`, `pgflquota_active`, `tqelm_active`, `wsdefault_active`, `wsextent_active`, `wsquota_active`

  The default value of the related OpenVMS process quota. This is the value of the quota when the QTI process was started.

- astlm_stored, biolm_stored, bytlm_stored, diolm_stored, enqlm_stored, fillm_stored, pgflquota_stored, tqelm_stored, wsdefault_stored, wsextent_stored, wsquota_stored

  The value of the related process quota currently stored in the OpenVMS system user authorization file (SYSUAF.DAT).

- acms_state

  The current ACMS state of this process.

- runtime_coll_state

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- max_threads

  The maximum number of threads allowed.

- started_queues

  The number of queues currently started on the node.

- current_tasks

  The number of tasks currently executed that were submitted by the QTI.

- current_submitters

  The number of submitters currently logged in by the QTI.

- task_successes

  The number of tasks successfully submitted and executed by the QTI.

- task_failures

  The number of tasks that failed to complete successfully after being submitted by the QTI.

- task_retries

  The number of times the QTI has attempted to re-run a task that is currently failed.

- errors_queued

  The number of tasks queued to error queues by the QTI.

- astlm_current, biolm_current, bytlm_current, diolm_current, enqlm_current, fillm_current, pgflquota_current, tqelm_current, wssize_current, channelcnt_current

The actual amount of the related OpenVMS process or system resource that is being consumed by the QTI process. The frequency with which these fields are updated is based on the value of the `vms_coll_interval` field in the Parameter table (see *Table 9.9, "Parameter Table"*).

- `pool_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `mss_process_total`

  The total size of the MSS process pool allocated for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_free`

  The amount of MSS process pool for this process that is currently unused. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_largest`

  The largest unused block available in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_failures`

  The number of failed attempts to allocate space in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_garbage`

  The number of garbage collections for this process that have been run to reclaim space in the MSS process pool. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `err_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  Errors for this process are only sent to the Remote Manager if this field is set to ENABLED. If this field is set to DISABLED, the process will not collect data for the fields in this class; existing field values reflect activity during a prior period when collection was enabled.

- `err_count`

The total number of errors related to this process that were sent to the Remote Manager.

- `last_err_msg`

  The text of the most recent error message related to this process that was sent to the Remote Manager.

- `time_of_last_error`

  Date and time of the most recent error message related to this process that was sent to the Remote Manager.

9.12. Server Table

The Server table contains a separate row for each server type (not server instance) in the application. Totals are for all instances of the server type.

Table 9.11. Server Table

Collection Class	Field	Data Type	SNMP Access	RPC Access
ID	record_state	integer	R	R
ID	id_coll_state	integer	R	R
ID	appl_name	string	R	R
ID	server_name	string	R	R
CONFIG	config_coll_state	integer	R	R
CONFIG	creation_delay_active §	integer	RW	RW
CONFIG	creation_interval_active §	integer	RW	RW
CONFIG	deletion_delay_active §	integer	RW	RW
CONFIG	deletion_interval_active §	integer	RW	RW
CONFIG	server_process_dump_flag_active §	integer	RW	RW
CONFIG	server_replace_flag §	integer	RW	RW
CONFIG	minimum_instances_active §	integer	RW	RW
CONFIG	maximum_instances_active §	integer	RW	RW
RUNTIME	runtime_coll_state	integer	R	R
RUNTIME	current_servers	gauge	R	R
RUNTIME	current_waiting tasks	gauge	R	R
RUNTIME	server_start_count	integer	R	R

Table Key

§ – Field is dynamic
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Access
RUNTIME	server_failures	integer	R	R

Table Key

§ – Field is dynamic
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.12.1. Field Descriptions

Following are descriptions of the fields in *Table 9.11, "Server Table"*.

- `record_state`

  The current state of this table entry. Valid states are VALID (the process is currently running and maintaining this table entry) or INACTIVE (the process is no longer running). Inactive rows are subject to reuse.

- `id_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `appl_name`

  Name of the application to which this server type belongs.

- `server_name`

  Name of this server type.

- `config_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `creation_delay_active`

  The current creation delay for this server type.

- `creation_interval_active`

  The current creation interval for this server type.

- `deletion_delay_active`

  The current deletion delay for this server type.

- `deletion_interval_active`

  The current deletion interval for this server type.

- `server_process_dump_flag_active`

  The current server process dump flag for this server type.

- `server_replace_flag`

  This field provides the ability for SNMP users to replace a server type by setting this value to 1. This field is available only to the SNMP interface.

- `minimum_instances_active`

  The current minimum number of started instances for this server type.

- `maximum_instances_active`

  The current maximum number of started instances for this server type.

- `runtime_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `current_servers`

  The current number of started servers of this type in the application.

- `current_waiting_tasks`

  The current number of tasks waiting to execute processing steps that call servers of this type in this application.

- `server_start_count`

  The number of times a server instance has been created for servers of this type in this application.

9.13. Task Group Table

The Task Group table contains a row for each task group in the application.

Table 9.12. Task Group Table

Collection Class	Field	Data Type	SNMP Access	RPC Access
ID	record_state	integer	R	R
Table Key § – Field is dynamic R – Read Access RW – Read/Write Access Blank – Not available to the interface				

Collection Class	Field	Data Type	SNMP Access	RPC Access
ID	id_coll_state	integer	R	R
ID	appl_name	string	R	R
ID	task_group_name	string	R	R
ID	build_time	time	R	R
POOL	pool_coll_state	integer	R	R
POOL	tws_pool_total	integer	R	R
POOL	tws_pool_free	min gauge	R	R
POOL	tws_pool_largest	min gauge	R	R
POOL	tws_pool_failures	integer	R	R
POOL	tws_pool_garbage	integer	R	R
POOL	twsc_pool_total	integer	R	R
POOL	twsc_pool_free	min gauge	R	R
POOL	twsc_pool_largest	min gauge	R	R
POOL	twsc_pool_failures	integer	R	R
POOL	twsc_pool_garbage	integer	R	R

Table Key

§ – Field is dynamic
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.13.1. Field Descriptions

Following are descriptions of the fields in *Table 9.12, "Task Group Table"*.

- `record_state`

  The current state of this table entry. Valid states are VALID (the process is currently running and maintaining this table entry) or INACTIVE (the process is no longer running). Inactive rows are subject to reuse.

- `id_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `appl_name`

  Name of the application to which this server type belongs.

- `task_group_name`

  Name of this task group.

- `build_time`

  The date and time the task group database (TDB) was built.

- `pool_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `tws_pool_total`

  The total size of the TWS pool allocated for this task group. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `tws_pool_free`

  The amount of unused TWS pool this task group. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `tws_pool_largest`

  The largest unused block available in this task group's TWS pool. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `tws_pool_failures`

  The number of failed attempts to allocate space in the TWS pool for this task group, The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `tws_pool_garbage`

  The number of garbage collections that have been run to reclaim space in this task group's TWS pool. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `twsc_pool_total`

  The total size of the TWSC pool allocated for this task group. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `twsc_pool_free`

  The amount of unused TWSC Pool for this task group. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `twsc_pool_largest`

The largest unused block available in this task group's TWSC pool. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `twsc_pool_failures`

  The number of failed attempts to allocate space in this task group's TWSC pool. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `twsc_pool_garbage`

  The number of garbage collections that have been run to reclaim space in this task group's TWSC pool. The frequency with which this field is updated is based on the value of the Parameter table field `wksp_coll_interval` (see *Table 9.9, "Parameter Table"*).

9.14. Trap Table

The Trap table is used to control which events trigger the Remote Manager to generate an SNMP trap. The ACMS Remote Manager populates this table from the configuration file at system startup. Thereafter, users make modifications to this table through either the SNMP interface or the ACMSMGR interface.

The primary key to this table is the combination of entity, name, and parameter. Duplicate rows are not allowed.

Table 9.13. Trap Table

Field	Data Type	SNMP Access	RPC Access	Configuration Access
entity §	integer	R	RW	RW
name §	string	R	RW	RW
parameter §	integer	R	RW	RW
min_value §	integer	RW	RW	RW
max_value §	integer	RW	RW	RW
severity §	integer	RW	RW	RW
alarms_sent	integer	R	R	
alarm_last_sent	integer	R	R	
trap_delete §	integer	RW		
Table Key § – Field is dynamic R – Read Access RW – Read/Write Access Blank – Not available to the interface				

9.14.1. Field Descriptions

Following are descriptions of the fields in *Table 9.13, "Trap Table"*.

- `entity`

  Name or type of the entity. Valid values are ACC, CP, EXC, MGR, QTI, and TSC. Symbolic values for the RPC interface are defined in *Section 8.1.4, "Entity Types"*. This value for entity cannot be changed from the SNMP interface.

- `name`

  A name for the entity that helps to uniquely identify an instance of the entity type. This value cannot be changed from the SNMP interface. Possible entity names are:

  - ACC, CP, QTI, TSC (process name)
  - EXC (application name)
  - GROUP (task group name)
  - MGR: Must be *
  - SERVER (server name): One of the following wildcard values:
    - asterisk (*) (matches all characters)
    - exclamation point (!) (negation)

- `parameter`

  Parameter specifies the value or condition to be monitored for potential alarms. This value cannot be changed from the SNMP interface.

  Not all parameters are valid for all entity types (see *Table 9.14, "Trap Minimums and Maximums"*). Valid values are:

  - EVENT_SEVERITY

    This parameter causes a test to be performed each time an auditable event is raised in the Remote Manager. Remote Manager events are filtered using the fields in the Parameter table (see *Section 9.10, "Parameter Table"*) and are stored in the Remote Manager log (see *Section 4.7, "Managing Log Files"*). Events are monitored for traps even if the event is not currently being logged.

  - EXISTS

    This parameter causes a test to be performed each time the Remote Manager detects that a process has started or stopped.

- `min_value`

  The minimum allowable value for the parameter. Valid minimums are parameter dependent (see *Table 9.14, "Trap Minimums and Maximums"*). If the field or condition being monitored is less than the value specified, an alarm is generated. A value of -1 is used when this field is not to be evaluated.

- `max_value`

  The maximum allowable value for the parameter. Valid maximums are parameter dependent (see *Table 9.14, "Trap Minimums and Maximums"*). If the field or condition being monitored is greater than the value specified, an alarm is generated. A value of -1 is used when this field is not to be evaluated.

- `severity`

  A severity to be associated with the trap. Severity codes are embedded in the trap message (see *Section 9.14.3, "SNMP Trap Format"*) and must be parsed by the trap receiver. Valid values are:

  - INFO
  - WARN
  - ERROR
  - FATAL

- `alarms_sent`

  A count of the number of alarms that have been sent.

- `alarm_last_sent`

  The date and time the last alarm was sent.

- `trap_delete`

  This field is available only through the SNMP interface. Set this field to 1 to delete the table row. RPC users call the procedure shown in *ACMSMGMT_DELETE_TRAP_1*. ACMSMGR and ACMSCFG each provide a **`DELETE TRAP`** command for this purpose.

9.14.2. Valid Trap Minimums and Maximums

Table 9.14, "Trap Minimums and Maximums" lists the values that can be specified as the minimum or maximum for each parameter type.

Table 9.14. Trap Minimums and Maximums

Parameter	Value	Meaning	Valid for These Entities	Comments
EVENT_SEVERITY	1	Informational	MGR	When configuring alarms for event severities, remember how the values are evaluated. For example, specifying the value 8 (FATAL) as a minimum results in an alarm being generated by all lesser severities. Similarly, specifying the value 1 (INFO) as a maximum results in an alarm being generated by all greater severities.
	2	Warning	MGR	
	4	Error	MGR	
	8	Fatal	MGR	
	–1	Ignore this field	MGR	The value of -1 causes the field to be ignored. When configuring traps, it is not always desirable to specify both minimum and maximum values. The value -1 can be used as a null placeholder when either value is to be ignored.
EXISTS	0	Stopped	ACC, CP,	When specified as a maximum, this value causes an alarm to be generated

			EXC, QTI, TSC	whenever the associated entity type and name is started. This value can be used, for example, to signal when the QTI has been started on a node on which it should not run.
	1	Started	ACC, CP, EXC, QTI, TSC	When specified as a minimum, this value causes an alarm to be generated whenever the associated entity type and name is stopped. This value can be used, for example, to signal when a particular application has been stopped.
	–1	Ignore this field	ACC, CP, EXC, QTI, TSC	The value of -1 causes the field to be ignored. When configuring traps, it is not always desirable to specify both minimum and maximum values. The value -1 can be used as a null placeholder when either value is to be ignored.

9.14.3. SNMP Trap Format

The following is the format of an SNMP trap message. Note that the message is generated as an ASCII string. Fields within the string are separated by a colon.

```
time: severity: entity_type: entity_name: parameter: value
```

In this format:

- *time* is a 23-character ASCII time in the format *DD-MMM-YYYY HH:MM:SS.hh*.
- *severity* is a single ASCII character that specifies the severity as determined from the severity field in the table that raised the alarm. Severities are:
  - I (informational)
  - W (warning)
  - E (error)
  - F (fatal)
- *entity_type* is one of the valid entity types (ACC, TSC, QTI, EXC, CP, MGR) and represents the entity that caused the alarm to be raised.
- *entity_name* is the process name of the entity that raised the alarm.
- *parameter* is the parameter that caused the alarm to be raised.
- *value* is the value that caused the alarm to be raised.

9.15. TSC Table

The TSC table contains a single entry for TSC management information.

Table 9.15. TSC Table

Collection Class	Field	Data Type	SNMP Access	RPC Access
ID	record_state	integer	R	R
ID	id_coll_state	integer	R	R
ID	process_name	string	R	R
ID	pid	integer	R	R
ID	start_time	time	R	R
ID	end_time	time	R	R
CONFIG	config_coll_state	integer	R	R
CONFIG	tsc_priority_active	integer	R	R
CONFIG	acms_state §	integer	RW	R
CONFIG	tsc_priority_stored	integer	RW	RW
CONFIG	tsc_username_active	string	R	R
CONFIG	tsc_username_stored	integer	RW	RW
CONFIG	cp_priority_active	integer	R	R
CONFIG	cp_priority_stored	integer	RW	RW
CONFIG	cp_slots_active	integer	R	R
CONFIG	cp_slots_stored	integer	RW	RW
CONFIG	max_logins_active §	integer	RW	RW
CONFIG	max_logins_stored	integer	RW	RW
CONFIG	max_tts_cp_active §	integer	RW	RW
CONFIG	max_tts_cp_stored	integer	RW	RW
CONFIG	perm_cps_active §	integer	RW	RW
CONFIG	perm_cps_stored	integer	RW	RW
CONFIG	min_cpis_active §	integer	RW	RW
CONFIG	min_cpis_stored	integer	RW	RW
CONFIG	cp_username_active	integer	R	R
CONFIG	cp_username_stored	integer	RW	RW
CONFIG	astlm_active »	integer	R	R
CONFIG	astlm_stored »	integer	RW	RW

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Access
CONFIG	biolm_active »	integer	R	R
CONFIG	biolm_stored »	integer	RW	RW
CONFIG	bytlm_active »	integer	R	R
CONFIG	bytlm_stored »	integer	RW	RW
CONFIG	diolm_active »	integer	R	R
CONFIG	diolm_stored »	integer	RW	RW
CONFIG	enqlm_active »	integer	R	R
CONFIG	enqlm_stored »	integer	RW	RW
CONFIG	fillm_active »	integer	R	R
CONFIG	fillm_stored »	integer	RW	RW
CONFIG	pgflquota_active »	integer	R	R
CONFIG	pgflquota_stored »	integer	RW	RW
CONFIG	tqelm_active »	integer	R	R
CONFIG	tqelm_stored »	integer	RW	RW
CONFIG	wsdefault_active »	integer	R	R
CONFIG	wsdefault_stored »	integer	RW	RW
CONFIG	wsextent_active »	integer	R	R
CONFIG	wsextent_stored »	integer	RW	RW
CONFIG	wsquota_active »	integer	R	R
CONFIG	wsquota_stored »	integer		
RUNTIME	runtime_coll_state	integer	R	R
RUNTIME	current_users	gauge	R	R
RUNTIME	cp_slots_used »	gauge	R	R
RUNTIME	terminals_per_cp_avg »	gauge	R	R
RUNTIME	astlm_current »	gauge	R	R
RUNTIME	biolm_current »	gauge	R	R
RUNTIME	bytlm_current »	gauge	R	R
RUNTIME	diolm_current »	gauge	R	R
RUNTIME	enqlm_current »	gauge	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

Collection Class	Field	Data Type	SNMP Access	RPC Access
RUNTIME	fillm_current »	gauge	R	R
RUNTIME	pgflquota_current »	gauge	R	R
RUNTIME	tqelm_current »	gauge	R	R
RUNTIME	wssize_current »	gauge	R	R
RUNTIME	channelcnt_current »	gauge	R	R
POOL	pool_coll_state	integer	R	R
POOL	mss_process_total	integer	R	R
POOL	mss_process_free	min gauge	R	R
POOL	mss_process_largest	min gauge	R	R
POOL	mss_process_failures	integer	R	R
POOL	mss_process_garbage	integer	R	R
ERROR	err_coll_state »	integer	R	R
ERROR	err_count »	integer	R	R
ERROR	last_err_msg »	string	R	R
ERROR	time_of_last_error »	time	R	R

Table Key

§ – Field is dynamic
» – Only valid for use with systems running ACMS Version 4.4 or higher
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.15.1. Field Descriptions

Following are descriptions of the fields in *Table 9.15, "TSC Table"*.

- `record_state`

  The current state of this table entry. Valid states are VALID (the process is currently running and maintaining this table entry) or INACTIVE (the process is no longer running). Inactive rows are subject to reuse.

- `id_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `process_name`

  The OpenVMS process name for the process.

- `pid`

The OpenVMS process identifier for the process.

- `start_time`

  Date and time the process was started.

- `end_time`

  Date and time the process ended. If the process has not yet ended, this field will be null.

- `config_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

- `tsc_priority_active`

  The base priority for this process. This is the value of the ACMSGEN field when the process was started.

- `tsc_priority_stored`

  The base priority currently stored in the ACMSGEN file for this process.

- `tsc_username_active`

  The OpenVMS account under which the TSC will run. The tsc_username_active is the value of the ACMSGEN field when the process was started.

- `tsc_username_stored`

  The value of the tsc_username field currently stored in the ACMSGEN file.

- `cp_priority_active`

  The base priority for CP processes. This is the value of the ACMSGEN field when the TSC process was started.

- `cp_priority_stored`

  The base priority currently stored in the ACMSGEN file for CP processes.

- `cp_slots_active`

  The current number of CP slots. This is the value of the ACMSGEN field when the TSC process was started. This field also represents the maximum number of entries in the CP table.

- `cp_slots_stored`

  The value of the cp_slots field in the current ACMSGEN file.

- `max_logins_active`

  The current maximum number of logins allowed.

- `max_logins_stored`

The value of the max_logins field in the current ACMSGEN file.

- `max_tts_cp_active`

  The current maximum number of terminals that a CP will support.

- `max_tts_cp_stored`

  The value of the max_tts_cp field in the current ACMSGEN file.

- `perm_cps_active`

  The number of permanent CPs that will be maintained on the system.

- `perm_cps_stored`

  The value of the perm_cps field in the current ACMSGEN file.

- `min_cpis_active`

  The number of CP slots that will be left open on a given CP.

- `min_cpis_stored`

  The value of the min_cpis field in the current ACMSGEN file.

- `cp_username_active`

  The current user name under which CP processes will run.

- `cp_username_stored`

  The value of the cp_username field in the current ACMSGEN file.

- `acms_state`

  Current ACMS state of the process.

- `runtime_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

  If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `current_users`

  The current number of terminal users in all CPs started by this TSC.

- `pool_coll_state`

  The current collection state for this class and this process. Collection states can be modified by modifying entries in the Collection table. See *Section 5.1, "Managing Data Collection"* and *Section 9.4, "Collection Table"* for discussions of data collection.

If this field is set to DISABLED, the process is not currently collecting data for the fields in this class. Any field values reflect activity during a prior period when collection was enabled.

- `mss_process_total`

  The total size of the MSS process pool allocated for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_free`

  The amount of unused MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_largest`

  The largest unused block available in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_failures`

  The number of failed attempts to allocate space in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

- `mss_process_garbage`

  The number of garbage collections that have been run to reclaim space in the MSS process pool for this process. The frequency with which this field is updated is based on the value of the Parameter table field `mss_coll_interval` (see *Table 9.9, "Parameter Table"*).

9.16. Users Table

The Users table contains information about users who have logged in to Remote Manager, either explicitly (using a user name and password) or implicitly (using a proxy account). This table is maintained internally by the Remote Manager and is read only to all external entities. Values in the table can be accessed through one of the supported interfaces. External users cannot make changes to this table.

In general, the values in this table are informational only.

Table 9.16. Users Table

Field Name	Data Type	SNMP Access	RPC Access
client_id	integer	R	R
Table Key § – Field is dynamic R – Read Access RW – Read/Write Access Blank – Not available to the interface			

Field Name	Data Type	SNMP Access	RPC Access
gid	short integer	R	R
uid	short integer	R	R
proxy_gid	short integer	R	R
proxy_uid	short integer	R	R
nodename	string	R	R
expires	string	R	R
uname	string	R	R
rights	integer array	R	R
proxy_flag	integer	R	R

Table Key

§ – Field is dynamic
R – Read Access
RW – Read/Write Access
Blank – Not available to the interface

9.16.1. Field Descriptions

Following are descriptions of the fields in *Table 9.16, "Users Table"*.

- `client_id`

  Unique number that identifies this client.

- `gid`

  Group portion of the user's UIC on the client node.

- `uid`

  User portion of the user's UIC on the client node,

- `proxy_gid`

  Group portion of the UIC from the proxy account on the server node.

- `proxy_uid`

  User portion of the UIC from the proxy account on the server node.

- `nodename`

  Name of client node. This is the only node from which the client ID is valid.

- `expires`

  Date and time the user log in expires. Full OpenVMS ASCII date (*DD-MMM-YYYY HH:MM:SS.hh*).

- `uname`

Account name of the account being used for authorization. This can be the account name for the proxy.

- `rights`

  Rights held by the user. This is an array of three elements. The first element contains the ACMS$MGMT_READ rights identifier; the second contains the ACMS$MGMT_WRITE rights identifier; the third contains the ACMS$MGMT_OPER identifier. A value of 0 indicates the user does not hold the right.

- `proxy_flag`

  A flag indicating whether the user has explicitly logged in (value of 0) or implicitly logged in by means of proxy (value of 1).

Chapter 10. ACMSCFG Commands

This chapter provides reference information about the commands of the ACMSCFG utility.

10.1. ACMSCFG Overview

The ACMSCFG utility is provided for performing operations on the ACMS Remote Manager configuration file. Similar to the ACMSMGR utility (described in *Chapter 11, "ACMSMGR Commands"*), the ACMSCFG utility performs only a subset of the operations that the ACMSMGR utility performs.

The ACMSCFG utility performs operations on the Remote Manager configuration file only, and only on configuration files that are directly accessible to the process running the utility.

Section 4.2, "Configuring Remote Manager Startup" discusses the purpose and use of the configuration file, as well as file defaults.

10.1.1. Command Format

The format for ACMSCFG commands is as follows:

```
ACMSCFG verb object qualifiers
```

The following verbs are supported:

- **ADD**
- **DELETE**
- **HELP**
- **SET**
- **SHOW**

Each verb has associated objects. The following sections list the objects and any qualifiers for each ACMSCFG command.

10.1.2. Command Objects and Qualifiers

The objects and qualifiers for the ACMSCFG commands are summarized in *Table 10.1, "ACMSCFG Command Objects and Qualifiers"*.

Table 10.1. ACMSCFG Command Objects and Qualifiers

Objects	Qualifiers
ADD Command	
COLLECTION	/CLASS, /COLL_STATE, /ENTITY, /NAME, /STORAGE_END_TIME, /STORAGE_INTERVAL,

Objects	Qualifiers
	/STORAGE_LOCATION, /STORAGE_ BEGIN_TIME, /STORAGE_STATE
TRAP	/ENTITY, /NAME, /PARAMETER, /SEVERITY, /TRAP_MIN, /TRAP_ MAX
DELETE Command	
COLLECTION	/CLASS, /ENTITY, /NAME
TRAP	/ENTITY, /NAME, /PARAMETER
HELP Command	
None.	None.
SET Command	
COLLECTION	/CLASS, /COLL_STATE, /ENTITY, /NAME, /STORAGE_END_TIME, /STORAGE_INTERVAL, /STORAGE_LOCATION, /STORAGE_ BEGIN_TIME, /STORAGE_STATE
INTERFACE	/INTERFACE, /STATE
PARAMETER	/DCL_AUDIT_LEVEL, /DCL_MGR_PRIORITY, /DCL_STACKSIZE, /ERROR_INTERVAL, /EVENT_LOG_PRIORITY, /LOG_STACKSIZE, /LOGIN_CREDS_LIFETIME, /MAX_LOGINS, /MAX_RPC_RETURN_RECS, /MGR_AUDIT_LEVEL, /MSG_PROC_AUDIT_LEVEL, /MSG_PROC_PRIORITY, /MSG_PROC_STACKSIZE, /MSS_ COLL_INTERVAL, /PROC_MON_AUDIT_LEVEL, /PROC_MON_INTERVAL, /PROC_MON_PRIORITY, /PROC_MON_STACKSIZE, /PROXY_CREDS_LIFETIME, /RPC_AUDIT_LEVEL, /RPC_PRIORITY, /RPC_STACKSIZE, /SECURITY_AUDIT_LEVEL, /SNAP_AUDIT_LEVEL, /SNAP_PRIORITY, /SNAP_STACKSIZE, /SNMP_AGENT_TIME_OUT, /SNMP_ARE_YOU_THERE, /SNMP_AUDIT_LEVEL, /SNMP_PRIORITY, /SNMP_SEL_TIME_OUT, /SNMP_STACKSIZE, /TCP_ENABLED, /TIMER_AUDIT_LEVEL, /TIMER_INTERVAL, /TIMER_PRIORITY, /TIMER_STACKSIZE, /TOTAL_ ENTITY_SLOTS, /TRACE_MSG_WAIT_TIME,

Objects	Qualifiers
	/TRACE_START_WAIT_TIME, /TRAP_AUDIT_LEVEL, /TRAP_PRIORITY, /TRAP_STACKSIZE, /UDP_ENABLED, /VMS_COLL_INTERVAL, /WKSP_ COLL_INTERVAL, /MAX_AGENTS
TRAP	/ENTITY, /NAME, /PARAMETER, /SEVERITY, /TRAP_MIN, /TRAP_ MAX
SHOW Command	
COLLECTION	/BRIEF, /FULL
CONTROL	None
INTERFACE	None
PARAMETER	None
TRAP	None

10.2. ACMSCFG ADD COLLECTION

ACMSCFG ADD COLLECTION

ACMSCFG ADD COLLECTION — Adds records to the collection table in the configuration file.

Format

```
ACMSCFG ADD COLLECTION [/qualifiers]
```

Command Qualifier	Default
/CLASS=*keyword*	* (all)
/COLL_STATE=*keyword*	DISABLED
/ENTITY=*keyword*	None
/NAME=[*,*proc_name*]	* (all)
/STORAGE_END_TIME=[NEVER, *time*]	NEVER; run until DISABLED Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_INTERVAL=*value*	300 Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_LOCATION=*file-name*	Translation of logical ACMS $MGMT_SNAPSHOT. Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_BEGIN_TIME=[NOW, *time*]	NOW; start as soon as ENABLED.

Command Qualifier	Default
	Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_STATE=*keyword*	DISABLED. Only for use on systems running ACMS Version 4.4 or higher.

Privileges Required

None.

Parameters

None.

Qualifiers

/CLASS=[*, ERROR, POOL, RUNTIME]

This qualifier determines the class that will be enabled or disabled. The default is all (*). See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a description of each class type.

/COLL_STATE=[ENABLED, DISABLED]

This qualifier specifies the state of the collection. The default is DISABLED. When a **SHOW ENTITY** command is issued, data for those classes that have their collection state set to ENABLED is displayed. Note that while the collection state is DISABLED, the data displayed for an entity may not be accurate. Data cannot be written to the data snapshot file when this qualifier is DISABLED, even when the storage state is ENABLED.

/ENTITY=[*, ACC, CP, EXC, GROUP, QTI, SERVER, TSC]

This required qualifier specifies the entity for which collection should be enabled or disabled.

/NAME=[*, *entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

For ACC, CP, QTI, and TSC entity types, the entity name is the process name. For the EXC entity type, the entity name is the name of the application (for example, VR_APPL).

Server and task group names can be specified as compound names made up of an application name and a server or task group name, separated by a period (for example, VR_APPL.VR_READ_SERVER). Either part of server or task group names can be a wildcard (for example, *.VR_READ_SERVER or VR_APPL.*). If only one part of a server or task group name is specified, it is assumed to be the application name, and the server or task group name is wildcarded. For example, VR_APPL is equivalent to VR_APPL.*.

The default is all (*), which is equivalent to *.* for a compound name.

/STORAGE_END_TIME=[NEVER, *time*]

This qualifier specifies a time after which the collection data should no longer be written to the snapshot file. The format of *time* is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for

example, 10-OCT or 09:00) are supported. If this qualifier is not specified, the default keyword of NEVER is applied, which equates to the OpenVMS zero date of 17-NOV-1858 00:00:00.00. With a value of NEVER, collection data continues to be written to the snapshot file until the storage state is set to DISABLED.

/STORAGE_INTERVAL=*value*

This qualifier controls the frequency (in seconds) at which data snapshots are performed. The default value is 300 seconds.

The storage interval value should be a multiple of the timer interval parameter (SET PARAMETER/TIMER_INTERVAL). The timer interval value determines the minimum elapsed time for many Remote Manager parameters, including the storage interval setting. The relationship of these values determine how often data snapshots are performed, for example:

- If the timer interval value is greater, its value is used by default. For instance, if the timer interval is 10 and the storage interval is 5, snapshots will be written at 10 second intervals.
- If the storage interval value is greater and is a multiple of the timer interval, the storage interval value is used. For example, if the timer interval is 10 and the storage interval is 30, snapshots will be written at 30 second intervals.
- If the storage interval value is greater and is not a multiple of the timer interval, the next multiple of the timer interval value is used. For example, if the timer interval is 10 and the storage interval is 15, snapshots will be written at 20 second intervals.

/STORAGE_LOCATION=*file-name*

This qualifier specifies an OpenVMS file specification to which collection data is to be written. The format of *file-name* is a valid OpenVMS pathname or logical (such as DISK$1:[SYSTEM.SNAPSHOTS] or SYS$SYSTEM:SNAPSHOTS.DAT).

If the **/STORAGE_LOCATION** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SNAPSHOT. If the logical is defined, the value of the logical is used by default. If a directory is not provided as part of the specification, the file is written to the default directory of the account under which the Remote Manager process is running.

Multiple collections can share a single snapshot file or be stored in separate files. For continuity, VSI recommends that EXC, Server, and Task Group collection information be written to the same snapshot file.

/STORAGE_START_TIME=[NOW, *time*]

This qualifier specifies a time after which the collection data should be written to the snapshot file. The format of *time* is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. If this qualifier is not specified, the default keyword of NOW is applied, which equates to the OpenVMS zero date of 17-NOV-1858 00:00:00.00. With a value of NOW, collection data is written to the snapshot file immediately, or as soon as the storage state is set to ENABLED.

/STORAGE_STATE=[ENABLED, DISABLED]

This qualifier specifies the state of the data snapshots. If this qualifier is not specified, data snapshots are disabled by default. To fully enable data snapshots, both the storage state and the collection state (**/COLL_STATE**) must be set to ENABLED.

Notes

When adding new collection records, the combination of class, entity, and name must be unique.

It is not possible to add records for the ID and CONFIG class. By default, all ACMS processes collect ID and CONFIG class data.

ACMS processes read the Collection table during process startup to determine which classes to begin collecting. Once the Remote Manager has been started, the ACMSMGR **SHOW PROCESS** command can be used to determine the class states for the currently running ACMS processes.

In order for collection data to be written to a snapshot file, the following conditions must be met:

- A qualifying entity must be running (one with an entity type and name matching fields in the Collection table).
- The collection state and storage state for that entity must be enabled.
- The current time must fall between the storage start time and storage end time.

If all these conditions are met, the Remote Manager opens the snapshot file for shared write operations. The file remains open until the storage state is set to DISABLED or until the snapshot period expires.

When multiple collection records apply to a given process, the records are assigned weights according to a precedence of name, then entity, and then class. Within a column, wildcard entries are weighted less than nonwildcard entries. The row with the highest weight that applies to a process is used. The command ACMSMGR **SHOW COLLECTIONS** displays weights for each row in the table. See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a discussion of the Collection table and how weights are assigned.

In contrast to typical collections, weighting for data snapshot threads does not apply. Therefore, it is possible for redundant collection data to be written to one or more snapshot files. If multiple collection records compile overlapping data, and each has their storage state set to ENABLED, each record writes data to the designated snapshot file.

See *Section 11.2, "ACMSMGR ADD COLLECTION"* for a discussion about adding collection records at run time.

Examples

```
$ ACMSCFG ADD COLLECTION/ENTITY=EXC/CLASS=RUNTIME/NAME=VR_APPL
```

This command creates an entry in the Collection table in the configuration file.

10.3. ACMSCFG ADD TRAP

ACMSCFG ADD TRAP

ACMSCFG ADD TRAP — Adds records to the trap table in the configuration file.

Format

```
ACMSCFG ADD TRAP [/qualifiers]
```

Command Qualifier	Default
/ENTITY=*keyword*	None
/NAME=[*,*entity-name*]	* (all)
/PARAMETER=*keyword*	EXISTS
/SEVERITY=[I,W,E,F]	E
/TRAP_MIN=*value*	-1
/TRAP_MAX=*value*	-1

Privileges Required

None.

Parameters

None.

Qualifiers

/ENTITY=[*, ACC, CP, EXC, MGR, QTI, TSC]

This required qualifier determines the entities for which a trap should be set.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. In general, the entity name is the process name. The exceptions are the EXC entity and the MGR entity.

For the EXC entity, use the assigned application name.

For the MGR entity, you must specify all (*).

Wildcards (*) are allowed in names. The default qualifier is the asterisk (*) wildcard.

/PARAMETER=[EVENT_SEVERITY, EXISTS]

The field that should be monitored. Valid values are:

- EVENT_SEVERITY

  This parameter is used for monitoring internal Remote Manager events. The Remote Manager logs internal events in the Remote Manager log. See *Section 4.7, "Managing Log Files"* and *Section 11.35, "ACMSMGR SHOW LOG"* for discussions of the Remote Manager log. Traps can be generated based on the severity levels of these events.

- EXISTS

  This parameter is used for monitoring process existence. Traps are generated if the associated entity type and name either start or stop.

/SEVERITY=[I, W, E, F]

A severity to be associated with the trap. Severity codes are embedded in the trap message and must be parsed by the trap receiver. Severities can be informational (I), warning (W), error (E), and fatal (F).

/TRAP_MIN=*value*

This qualifier specifies the minimum allowable value for the parameter being monitored. A trap is generated if the parameter value is less than the minimum value. See *Table 9.13, "Trap Table"* for a list of valid **/TRAP_MIN** values.

/TRAP_MAX=*value*

This qualifier specifies the maximum allowable value for the parameter being monitored. A trap is generated if the parameter value is greater than the maximum value. See *Table 9.13, "Trap Table"* for a list of valid **/TRAP_MAX** values.

Notes

When adding new trap records, the combination of entity, name, and parameter must be unique.

See *Table 9.14, "Trap Minimums and Maximums"* for a discussion about setting appropriate trap minimums and maximums. See *Section 9.14.3, "SNMP Trap Format"* for a description of the trap message generated.

Examples

```
$  ACMSCFG ADD TRAP /ENTITY=ACC/PARAMETER=EXISTS/TRAP_MIN=1
```

This command causes an SNMP trap to be generated whenever the ACC process stops if the SNMP interface is running.

10.4. ACMSCFG DELETE COLLECTION

ACMSCFG DELETE COLLECTION

ACMSCFG DELETE COLLECTION — Deletes records from the collection table in the configuration file.

Format

```
ACMSCFG DELETE COLLECTION [/qualifiers]
```

Command Qualifier	Default
/CLASS=*keyword*	* (all)
/ENTITY=*keyword*	None
/NAME=[*,*entity-name*]	* (all)

Privileges Required

None.

Parameters

None.

Qualifiers

/CLASS=[*, CONFIG, ERROR, ID, POOL, RUNTIME]

This qualifier determines the class that will be enabled or disabled. See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a description of each class type.

/ENTITY=[*, ACC, AGENT, CP, EXC, GROUP, QTI, SERVER, TSC]

This required qualifier determines the entities for which collection should be enabled or disabled.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

For ACC, AGENT, CP, QTI, and TSC entity types, the entity name is the process name. For EXCs, the entity name is the name of the application (for example, VR_APPL).

Server and task group names can be specified as compound names made up of application name and server or task group name, separated by a period (for example, VR_APPL.VR_READ_SERVER). Either part of server or task group names can be a wildcard (for example, *.VR_READ_SERVER or VR_APPL.*). If only one part of a server or task group name is specified, it is assumed to be the application name, and a wildcard is used as the server or task group name. For example, VR_APPL is equivalent to VR_APPL.*.

The default is all (*), which is equivalent to *.* for a compound name.

Notes

When deleting collection records, the combination of class, entity, and name must exactly match the row to be deleted.

It is not possible to delete records for the ID and CONFIG class. By default, all ACMS processes collect ID and CONFIG class data.

When multiple collection records apply to a given process, the records are assigned weights according to a precedence of name, then entity, then class. Within a column, wildcard entries are weighted less than nonwildcard entries. The row with the highest weight that applies to a process is used. The ACMSMGR **SHOW COLLECTIONS** command displays weights for each row in the table. See also *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a discussion of the Collection table and how weights are assigned.

See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a discussion about deleting collection records at run time.

Examples

```
$ ACMSCFG DELETE COLLECTION/ENTITY=EXC/CLASS=RUNTIME/NAME=VR_APPL
```

This command deletes the entry in the Collection table for run-time collection by the VR_APPL application.

10.5. ACMSCFG DELETE TRAP

ACMSCFG DELETE TRAP

ACMSCFG DELETE TRAP — Deletes a record from the trap table in the configuration file.

Format

`ACMSCFG DELETE TRAP [/qualifiers]`

Command Qualifier	Default
/ENTITY=*keyword*	None
/NAME=[*,*entity-name*]	* (all)
/PARAMETER=*keyword*	EXISTS

Privileges Required

None.

Parameters

None.

Qualifiers

/ENTITY=[*, ACC, CP, EXC, MGR, QTI, TSC]

This required qualifier determines the entity or entities for which a trap should be set.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. In general, the entity name is the process name. The exceptions are the EXC entity and the MGR entity.

For the EXC entity, use the assigned application names.

For the MGR entity, you must specify all (*).

Wildcards (*) are allowed in names. The default qualifier is the asterisk (*) wildcard.

/PARAMETER=[EVENT_SEVERITY,EXISTS]

The field that should be monitored. Valid values are:

- EVENT_SEVERITY

  This parameter is used for monitoring internal Remote Manager events. The Remote Manager logs internal events in the Remote Manager log. See *Section 4.7, "Managing Log Files"* and *Section 11.35, "ACMSMGR SHOW LOG"* for discussions of the Remote Manager log. Traps can be generated based on the severity levels of these events.

- EXISTS

  This parameter is used for monitoring process existence. Traps are generated if the associated entity type and name either start or stop.

Notes

When deleting trap records, the combination of entity, name, and parameter must exactly match a row in the Trap table.

Examples

```
$ ACMSCFG DELETE TRAP/ENTITY=ACC/PARAMETER=EXISTS
```

This command deletes a trap from the Trap table in the configuration file.

ACMSCFG HELP

ACMSCFG HELP — Displays help information about the ACMS Configuration utility (ACMSCFG) and its commands.

Format

```
ACMSCFG HELP
```

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

Online help is available for each ACMSCFG command. Each help topic summarizes the valid syntax, abbreviations, parameters, and qualifiers for a particular command and also indicates all default and required values.

For a comprehensive list of ACMS utilities that offer online help or for further instructions on how to invoke help, see ACMS Help.

Examples

```
$ ACMSCFG HELP
```

This command invokes online help for the ACMSCFG utility and displays a list of available topics.

10.6. ACMSCFG SET COLLECTION

ACMSCFG SET COLLECTION

ACMSCFG SET COLLECTION — Adds records to the collection table in the configuration file.

Format

```
ACMSCFG SET COLLECTION [/qualifiers]
```

Command Qualifier	Default
/CLASS=*keyword*	* (all)
/COLL_STATE=*keyword*	None
/ENTITY=*keyword*	None
/NAME=[*,*entity-name*]	* (all)
/STORAGE_END_TIME=[NEVER, *time*]	None Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_INTERVAL=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_LOCATION=*file-name*	Translation of logical ACMS $MGMT_SNAPSHOT Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_BEGIN_TIME=[NOW, *time*]	None Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_STATE=*keyword*	Only for use on systems running ACMS Version 4.4 or higher.

Privileges Required

None.

Parameters

None.

Qualifiers

/CLASS=[*, CONFIG, ERROR, ID, POOL, RUNTIME]

This qualifier determines the class that will be enabled or disabled. See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a description of each class type.

/COLL_STATE=[ENABLED, DISABLED]

This qualifier specifies the state of the collection. When an ACMSMGR **SHOW ENTITY** command is issued, data for those classes that have their collection state set to ENABLED is displayed. Note that while the collection state is DISABLED, data is not collected. As a result, data cannot be written to the data snapshot file when this qualifier is DISABLED, even when the storage state is ENABLED.

/ENTITY=[*, ACC, AGENT, CP, EXC, GROUP, QTI, SERVER, TSC]

This required qualifier determines the entities for which collection should be enabled or disabled.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

For ACC, AGENT, CP, QTI, and TSC entity types, the entity name is the process name. For the EXC entity, the entity name is the name of the application (for example, VR_APPL).

Server and task group names can be specified as compound names made up of application name and server or task group name, separated by a period (for example, VR_APPL.VR_READ_SERVER). Either part of server or task group names can be a wildcard (for example, *.VR_READ_SERVER or VR_APPL.*). If only one part of a server or task group name is specified, it is assumed to be the application name, and a wildcard is used as the server or task group name. For example, VR_APPL is equivalent to VR_APPL.*.

The default is all (*), which is equivalent to *.* for a compound name.

/STORAGE_END_TIME=[NEVER,*time*]

This qualifier specifies a time after which the collection data should no longer be written to the snapshot file. The format of *time* is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. The keyword NEVER is also supported, which equates to the OpenVMS zero date of 17-NOV-1858 00:00:00.00. With a value of NEVER, collection data continues to be written to the snapshot file until the storage state is set to DISABLED.

If this qualifier is not specified, the existing value remains unchanged. This value can be modified dynamically.

/STORAGE_INTERVAL=*value*

This qualifier controls the frequency (in seconds) at which data snapshots are performed.

The storage interval value should be a multiple of the timer interval parameter (**`SET PARAMETER/TIMER_INTERVAL`**). The timer interval value determines the minimum elapsed time for many Remote Manager parameters, including the storage interval setting. The relationship of these values determine how often data snapshots are performed, for example:

- If the timer interval value is greater, its value is used by default. For instance, if the timer interval is 10 and the storage interval is 5, snapshots will be written at 10 second intervals.
- If the storage interval value is greater and is a multiple of the timer interval, the storage interval value is used. For example, if the timer interval is 10 and the storage interval is 30, snapshots will be written at 30 second intervals.
- If the storage interval value is greater and is not a multiple of the timer interval, the next multiple of the timer interval value is used. For example, if the timer interval is 10 and the storage interval is 15, snapshots will be written at 20 second intervals.

/STORAGE_LOCATION=*file-name*

This qualifier specifies an OpenVMS file specification to which collection data is to be written. The format of *file-name* is a valid OpenVMS pathname or logical (such as DISK$1:[SYSTEM.SNAPSHOTS] or SYS$SYSTEM:SNAPSHOTS.DAT).

If the **/STORAGE_LOCATION** qualifier is not specified, the current value remains unchanged. If a directory is not provided as part of the specification, the file is written to the default directory of the account under which the Remote Manager process is running.

Multiple collections can share a single snapshot file or be stored in separate files. For continuity, VSI recommends that EXC, Server, and Task Group collection information be written to the same snapshot file.

This value can be modified dynamically.

/STORAGE_BEGIN_TIME=*time*

This qualifier specifies a time after which the collection data should be written to the snapshot file. The format of *time* is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. The keyword NOW is also supported, which equates to the OpenVMS zero date of 17-NOV-1858 00:00:00.00. With a value of NOW, collection data is written to the snapshot file immediately, or as soon as the storage state is set to ENABLED.

If this qualifier is not specified, the current value remains unchanged. This value can be modified dynamically.

/STORAGE_STATE=[ENABLED, DISABLED]

This qualifier specifies the state of the data snapshots. To fully enable data snapshots, both the storage state and the collection state (**/COLL_STATE**) must be set to ENABLED. If this qualifier is not specified, the current value remains unchanged. This value can be modified dynamically.

Notes

When updating collection records, the combination of class, entity, and name must exactly match a record in the collection table.

By default, processes collect only ID and CONFIG class data during process initialization. If these classes were disabled during process startup, that information would not be available until the class was enabled and the process was restarted.

ACMS processes read the Collection table during process startup to determine which classes to begin collecting. Once the Remote Manager has been started, the ACMSMGR **SHOW PROCESS** command can be used to determine the class states for the currently running ACMS processes.

In order for collection data to be written to a snapshot file, the following conditions must be met:

- A qualifying entity must be running (one with an entity type and name matching fields in the Collection table).
- The collection state and storage state for that entity must be enabled.
- The current time must fall between the storage start time and storage end time.

If all these conditions are met, the Remote Manager opens the snapshot file for shared write operations. The file remains open until the storage state is set to DISABLED or until the snapshot period expires.

Changes to snapshot values are processed dynamically. Updated storage interval and storage state values are applied immediately; updated storage location and storage end time values are applied during the next snapshot interval.

When multiple collection records apply to a given process, the records are assigned weights according to a precedence of name, then entity, and then class. Within a column, wildcard entries are weighted less than nonwildcard entries. The row with the highest weight that applies to a process is used. The command ACMSMGR **SHOW COLLECTION** displays weights for each row in the table. See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a discussion of the Collection table and how weights are assigned.

In contrast to typical collections, weighting for data snapshot threads does not apply. Therefore, it is possible for redundant collection data to be written to one or more snapshot files. If multiple collection records compile overlapping data, and each has their storage state set to ENABLED, each record writes data to the designated snapshot file.

Examples

```
$ ACMSCFG SET COLLECTION/ENTITY=EXC/CLASS=RUNTIME/NAME=VR_APPL/
COLL_STATE=DISABLED
```

This command disables run-time data collection for the VR_APPL application.

ACMSCFG SET INTERFACE

ACMSCFG SET INTERFACE — Allows Remote Manager interfaces to be enabled or disabled in the configuration file.

Format

ACMSCFG SET INTERFACE [*/qualifiers*]

Command Qualifier	Default
/INTERFACE=*interface*	None
/STATE=*keyword*	DISABLED

Privileges Required

None.

Parameters

None.

Qualifiers

/INTERFACE=*interface*

This required qualifier determines which interface to modify. Valid values are:

- RPC
- SNMP

/STATE=[DISABLED, ENABLED]

This qualifier determines the operation to perform. If the value supplied is ENABLED, the interface will be started (if it is not already running). If the value supplied is DISABLED, the interface will be stopped.

Notes

The ACMSMGR uses the RPC interface. Stopping an interface disables communication to the Remote Manager through that interface. Stopping the RPC interface on a given node prevents ACMSMGR from communicating with the Remote Manager on that node.

Examples

```
$  ACMSCFG SET INTERFACE/INTERFACE=SNMP/STATE=DISABLED
ACMS Remote Management Option -- Command line utility
Call to modify interface on server sparks was executed
%ACMSMGMT-S-SUCCESS, Operation completed
```

ACMSCFG SET PARAMETER

ACMSCFG SET PARAMETER — Allows Remote Manager parameters to be updated in the configuration file.

Format

```
ACMSMGR SET PARAMETER [/qualifiers]
```

Command Qualifier	**Default**
/DCL_AUDIT_LEVEL=*value*	None
/DCL_MGR_PRIORITY=*value*	None
/DCL_STACKSIZE=*value*	None
/ERROR_INTERVAL=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/EVENT_LOG_PRIORITY=*value*	None
/LOG_STACKSIZE=*value*	None
/LOGIN_CREDS_LIFETIME=*value*	None
/MAX_LOGINS=*value*	None
/MAX_RPC_RETURN_RECS=*value*	None
/MGR_AUDIT_LEVEL=*value*	None
/MSG_PROC_AUDIT_LEVEL=*value*	None
/MSG_PROC_PRIORITY=*value*	None
/MSG_PROC_STACKSIZE=*value*	None
/MSS_COLL_INTERVAL=*value*	None
/PROC_MON_AUDIT_LEVEL=*value*	None
/PROC_MON_INTERVAL=*value*	None
/PROC_MON_PRIORITY=*value*	None
/PROC_MON_STACKSIZE=*value*	None

Command Qualifier	Default
/PROXY_CREDS_LIFETIME=*value*	None
/RPC_AUDIT_LEVEL=*value*	None
/RPC_PRIORITY=*value*	None
/RPC_STACKSIZE=*value*	None
/SECURITY_AUDIT_LEVEL=*value*	None
/SNAP_AUDIT_LEVEL=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/SNAP_PRIORITY=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/SNAP_STACKSIZE=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/SNMP_AGENT_TIME_OUT	None
/SNMP_ARE_YOU_THERE=*value*	None
/SNMP_AUDIT_LEVEL=*value*	None
/SNMP_PRIORITY=*value*	None
/SNMP_SEL_TIME_OUT=*value*	None
/SNMP_STACKSIZE=*value*	None
/TCP_ENABLED=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/TIMER_AUDIT_LEVEL=*value*	None
/TIMER_INTERVAL=*value*	None
/TIMER_PRIORITY=*value*	None
/TIMER_STACKSIZE=*value*	None
/TOTAL_ENTITY_SLOTS=*value*	None
/TRACE_MSG_WAIT_TIME=*value*	None
/TRACE_START_WAIT_TIME=*value*	None
/TRAP_AUDIT_LEVEL=*value*	None
/TRAP_PRIORITY=*value*	None
/TRAP_STACKSIZE=*value*	None
/UDP_ENABLED=*value*	None

Command Qualifier	Default
	Only for use on systems running ACMS Version 4.4 or higher.
/VMS_COLL_INTERVAL=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/WKSP_COLL_INTERVAL=*value*	None
/MAX_AGENTS=*value*	None

Privileges Required

None.

Parameters

None.

Qualifiers

/[parameter]=value

All qualifiers correspond directly to fields in the Parameter table. See *Section 9.10.1, "Field Descriptions"* for descriptions of each field.

Notes

See *Section 9.10, "Parameter Table"* for a description of each parameter.

Example

```
$ ACMSCFG SET PARAMETER /MGR_AUDIT_LEVEL=E
```

This command modifies the dynamic parameter field `mgr_audit_level`.

ACMSCFG SET TRAP

ACMSCFG SET TRAP — Updates records in the Trap table in the configuration file.

Format

```
ACMSCFG SET TRAP [/qualifiers]
```

Command Qualifier	Default
/ENTITY=*keyword*	None
/NAME=[*,*entity-name*]	* (all)
/PARAMETER=*keyword*	EXISTS
/SEVERITY=[I,W,E,F]	None
/TRAP_MIN=*value*	None

Command Qualifier	Default
/TRAP_MAX=*value*	None

Privileges Required

None.

Parameters

None.

Qualifiers

/ENTITY=[*, ACC, CP, EXC, MGR, QTI, TSC]

This required qualifier determines the entity or entities for which a trap should be set.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Specify the value of the name field for the record you wish to modify.

/PARAMETER=[EVENT_SEVERITY, EXISTS]

The field that should be monitored. Valid values are:

- EVENT_SEVERITY

  This parameter is used for monitoring internal Remote Manager events. The Remote Manager logs internal events in the Remote Manager log. See *Section 4.7, "Managing Log Files"* and *Section 11.35, "ACMSMGR SHOW LOG"* for discussions of the Remote Manager log. Traps can be generated based on the severity levels of these events.

- EXISTS

  This parameter is used for monitoring process existence. Traps are generated if the associated entity type and name either starts or stops.

/SEVERITY=[I,W,E,F]

A severity to be associated with the trap. Severity codes are embedded in the trap message and must be parsed by the trap receiver. Severities are informational (I), warning (W), error (E), and fatal (F).

/TRAP_MIN=*value*

This qualifier specifies the minimum allowable value for the parameter being monitored. A trap is generated if the parameter value is less than the minimum value. See *Table 9.13, "Trap Table"* for a list of valid values.

/TRAP_MAX=*value*

This qualifier specifies the maximum allowable value for the parameter being monitored. A trap is generated if the parameter value is greater than the maximum value. See *Table 9.13, "Trap Table"* for a list of valid values.

Notes

When updating trap records, the combination of entity, name, and parameter must exactly match a record in the trap table.

See *Table 9.14, "Trap Minimums and Maximums"* for a discussion about setting appropriate trap minimums and maximums. See *Section 9.14.3, "SNMP Trap Format"* for a description of the trap message generated.

Examples

```
$ ACMSCFG SET TRAP /ENTITY=QTI/PARAMETER=EXISTS/TRAP_MAX=0
```

This command causes an SNMP trap to be generated whenever the QTI process is started if the SNMP interface is running.

ACMSCFG SHOW COLLECTION

ACMSCFG SHOW COLLECTION — Displays Collection table data from the configuration file.

Format

```
ACMSCFG SHOW COLLECTION [/qualifiers]
```

Privileges Required

None.

Parameters

None.

Qualifier

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. The default is **/BRIEF**.

Note that storage start and end times for data snapshots are only visible when **/FULL** is provided. When not specified, the resulting summary display may contain truncated values for some of the longer fields (such as entity name and storage location).

Notes

See *Section 9.4, "Collection Table"* for a discussion of each field displayed. See *Section 5.1, "Managing Data Collection"* for a discussion of collections.

Examples

```
$  ACMSCFG SHOW COLLECTION
Entity                 Collect  Collect                      Storage
 Storage
```

```
Type    Entity Name    Class    State    Storage location     State
 Interval
------- -------------- -------- -------- -------------------  ---------
 --------
*       *              id       enabled  acms$mgmt_snapshot   enabled
 3600
*       *              config   enabled  acms$mgmt_snapshot   disabled
 3600
*       *              runtime  enabled  acms$mgmt_snapshot   disabled  10
*       *              pool     enabled  acms$mgmt_snapshot   disabled  10
*       *              error    enabled  acms$mgmt_snapshot   disabled  10
```

This command shows the current contents of the Collection table as stored in the configuration file.

ACMSCFG SHOW CONTROL

ACMSCFG SHOW CONTROL — Displays the control record from the configuration file.

Format

```
ACMSCFG SHOW CONTROL
```

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

The control record is used by the Remote Manager and cannot be modified or deleted. The ACMSCFG **SHOW CONTROL** command displays the following fields:

- Interface count – Number of interface records in the file.
- Collection count – Number of collection records in the file.
- Timer count – Number of timer records in the file.
- Trap count – Number of trap records in the file.
- Parameter count – Number of parameter records in the file.
- Version – Internal file version identifier.

Examples

```
$  ACMSCFG SHOW CONTROL
Record Counts
 Record type  Count
```

```
------------  -------

 Interface    2
 Collection   2
 Timer        1
 Trap         1
 Parameter    1
 Version      6
```

This command shows the current contents of the control record in the configuration file.

ACMSCFG SHOW INTERFACE

ACMSCFG SHOW INTERFACE — Displays the Remote Manager interface from the configuration file.

Format

```
ACMSCFG SHOW INTERFACE
```

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

The Remote Manager supports two interfaces: RPC and SNMP. This command displays the enabled states of each interface.

See *Section 9.8, "Interfaces Table"* for a discussion of each field displayed.

Examples

```
$  ACMSCFG SHOW INTERFACE
Interface        Enable
Type             State
-------------------------------------------------------
 rpc            enabled
 snmp           enabled
```

This command shows the current contents of the Interfaces table in the configuration file. As shown, both interfaces are started when the Remote Manager is started.

ACMSCFG SHOW PARAMETER

ACMSCFG SHOW PARAMETER — Displays Remote Manager parameter information from the configuration file.

Format

```
ACMSCFG SHOW PARAMETER
```

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

See *Section 9.10, "Parameter Table"* for a description of each parameter.

Examples

```
$  ACMSCFG SHOW PARAMETER
Management Parameters
Parameter                Value   Default Min     Max         (D)ynamic
------------------------------------------------------------------------
 dcl_audit_level         E       E       0           F     (D)
 dcl_mgr_priority        5       5       1          10
 dcl_stacksize           300     300     1    2147483647    k (Vax), 8k
 (Alpha)
 error_interval          10      10      1     863999999    seconds (D)
 event_log_priority      5       5       1          10
 log_stacksize           300     300     1    2147483647    K (Vax), 8k
 (Alpha)
 login_creds_lifetime    60      60      1      14399999    minutes (D)
 max_logins              20      20      1    2147483647    (D)
 max_rpc_return_recs     20      20      1    2147483647
 mgr_audit_level         E       E       0           F     (D)
 msg_proc_audit_level    E       E       0           F     (D)
 msg_proc_priority       5       5       1          10
 msg_proc_stacksize      300     300     1    2147483647    k (Vax), 8k
 (Alpha)
 mss_coll_interval       10      10      1     863999999    seconds (D)
 proc_mon_audit_level    E       E       0           F     (D)
 proc_mon_interval       30      30      1      14399999    seconds (D)
 proc_mon_priority       5       5       1          10
 proc_mon_stacksize      300     300     1    2147483647    K (Vax), 8k
 (Alpha)
 proxy_creds_lifetime    60      60      1      14399999    minutes (D)
 rpc_audit_level         E       E       0           F     (D)
 rpc_priority            5       5       1          10
 rpc_stacksize           300     300     1    2147483647    k (Vax), 8k
 (Alpha)
 security_audit_level    E       E       0           F     (D)
 snap_audit_level        E       E       0           F     (D)
 snap_priority           5       5       1          10
```

```
snap_stacksize          30      30      1    2147483647    k (Vax), 8k
(Alpha)
snmp_agent_time_out     10      10      1     863999999    seconds
snmp_are_you_there      300     300     2     863999999    seconds
snmp_audit_level        E       E       0             F    (D)
snmp_priority           5       5       1            10
snmp_sel_time_out       5       5       1     863999999    seconds
snmp_stacksize          300     300     1    2147483647    k (Vax), 8k
(Alpha)
tcp_enabled             1       1       0             1    [0,1]
1=enabled
timer_audit_level       E       E       0             F    (D)
timer_interval          30      30      1     863999999    seconds (D)
timer_priority          5       5       1            10
timer_stacksize         300     300     1    2147483647    k (Vax), 8k
(Alpha)
total_entity_slots      20      20      1    2147483647
trace_msg_wait_time     5       5       1      14399999    seconds (D)
trace_start_wait_time   5       5       1      14399999    seconds (D)
trap_audit_level        E       E       0             F    (D)
trap_priority           5       5       1            10
trap_stacksize          300     300     1    2147483647    k (Vax), 8k
(Alpha)
udp_enabled             1       1       0             1    [0,1]
1=enabled
vms_coll_interval       10      10      0     863999999    seconds (D)
wksp_coll_interval      10      10      1     863999999    seconds (D)
```

This command shows the current contents of the Parameter table in the configuration file.

ACMSCFG SHOW TRAP

ACMSCFG SHOW TRAP — Displays SNMP trap configurations from the configuration file.

Format

```
ACMSCFG SHOW TRAP
```

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

SNMP traps are generated only if the SNMP interface is started.

See *Section 9.14, "Trap Table"* for a description of each field displayed.

Example

```
$  ACMSCFG SHOW TRAP
Entity  Entity
Type    Name                    Parameter        Min     Max      Severity
------- ----------------------- ---------------- ------- -------- --------
 *       *                      exists           1       -1       i
```

This command shows the current contents of the Trap table in the configuration file. As shown, a single trap has been configured to send an informational trap when any ACMS process is stopped. This is the default configuration.

Chapter 11. ACMSMGR Commands

This chapter provides reference information about the commands for the ACMSMGR utility.

11.1. ACMSMGR Overview

The ACMSMGR utility is used to perform operations on running ACMS systems.

You can use the ACMSMGR utility to perform the following functions:

- User authentication (login, logout)
- Display and update ACMS system management data
- Manage the ACMS Remote Manager

The ACMSMGR utility uses the ACMS management interface, which is based on ONC RPC. Commands can be executed remotely from any node in the network, and many commands can be executed against more than one node.

See *Chapter 4, "Managing the Remote Manager"* for a description of how to use ACMSMGR to manage the Remote Manager.

11.1.1. Command Format

The format for ACMSMGR commands is as follows:

```
ACMSMGR verb object qualifiers
```

The following verbs are supported:

- **ADD**
- **DELETE**
- **HELP**
- **LOGIN**
- **LOGOUT**
- **REPLACE**
- **RESET**
- **SET**
- **SHOW**
- **START**
- **STOP**

Each verb has an associated object and set of qualifiers.

11.1.2. Command Objects and Qualifiers

The objects and qualifiers for the ACMSMGR commands are summarized in the table below.

Table 11.1. ACMSMGR Command Objects and Qualifiers

Objects	Qualifiers
ADD Command	
COLLECTION	/CLASS, /COLL_STATE, /ENTITY, /NAME, /NODE, /STORAGE_END_TIME, /STORAGE_INTERVAL, /STORAGE_LOCATION, /STORAGE_BEGIN_TIME, /STORAGE_STATE, /USER
FILTER	/CODE, /FILE, /NAME, /NODE, /USER
TRAP	/ENTITY, /NAME, /NODE, /PARAMETER, /SEVERITY, /TRAP_MIN, /TRAP_MAX, /USER
DELETE Command	
COLLECTION	/CLASS, /ENTITY, /NAME, /NODE, /USER
FILTER	/CODE, /NAME, /NODE, /USER
TRAP	/ENTITY, /NAME, /NODE, /PARAMETER, /USER
HELP Command	
None	None
LOGIN Command	
None	/NODE, /PASSWORD, /USER
LOGOUT Command	
None	/NODE, /USER
REPLACE Command	
SERVER	/APPLICATION, /NODE, /SERVER, /USER
RESET Command	
ERROR	/NODE, /USER
LOG	/NODE, /USER
SAVE Command	
FILTER	/FILE, /NODE, /USER
SET Command	
ACC	/ACC_PRIORITY, /ACC_USERNAME, /ACTIVE, /ASTLM, /AUDIT_STATE, /BIOLM, /BYTLM, /CHANNELCNT, /DIOLM, /ENQLM, /FILLM, /GBLPAGES,

	/GBLPAGFIL, /GBLSECTIONS, /LOG, /MAX_APPL, /MSS_MAXBUF, /MSS_MAXOBJ, /MSS_NET_RETRY_TIMER, /MSS_POOLSIZE, /MSS_PROCESS_POOL, /NODE, /NODE_NAME, /PGFLQUOTA, /STORED, /TQELM, /TWS_POOLSIZE, /TWSC_POOLSIZE, /USER, /USERNAME_DEFAULT, /WS_POOLSIZE, /WSC_POOLSIZE, /WSDEFAULT, /WSEXTENT, /WSQUOTA
COLLECTION	/CLASS, /COLL_STATE, /ENTITY, /NAME, /NODE, /STORAGE_END_TIME, /STORAGE_INTERVAL, /STORAGE_LOCATION, /STORAGE_START_TIME, /STORAGE_STATE, /USER
CP	/ASTLM, /BIOLM, /BYTLM, /DIOLM, /ENQLM, /FILLM, /LOG, /PGFLQUOTA, /TQELM, /WSEXTENT, /WSDEFAULT, /WSQUOTA
EXC	/ACTIVE, /APPLICATION, /ASTLM, /AUDIT_STATE, /BIOLM, /BYTLM, /DIOLM, /ENQLM, /FILLM, /LOG, /MAX_SERVERS, /MAX_TASKS, /NODE, /PGFLQUOTA, /SP_MON_INTERVAL, /STORED, /TQELM, /TRANSACTION_TIMEOUT, /USERNAME, /WSEXTENT, /WSDEFAULT, /WSQUOTA
INTERFACE	/INTERFACE, /NODE, /STATE, /USER
PARAMETER	/DCL_AUDIT_LEVEL, /DCL_MGR_PRIORITY, /DCL_STACKSIZE, /ERROR_INTERVAL, /EVENT_LOG_PRIORITY, /LOG_STACKSIZE, /LOGIN_CREDS_LIFETIME, /MAX_LOGINS, /MAX_RPC_RETURN_RECS, /MGR_AUDIT_LEVEL, /MSG_PROC_AUDIT_LEVEL, /MSG_PROC_PRIORITY, /MSG_PROC_STACKSIZE, /MSS_COLL_INTERVAL, /NODE, /PROC_MON_AUDIT_LEVEL, /PROC_MON_INTERVAL, /PROC_MON_PRIORITY, /PROC_MON_STACKSIZE, /PROXY_CREDS_LIFETIME, /RPC_AUDIT_LEVEL, /RPC_PRIORITY, /RPC_STACKSIZE, /SECURITY_AUDIT_LEVEL, /SNAP_AUDIT_LEVEL, /SNAP_PRIORITY, /SNAP_STACKSIZE, /SNMP_AGENT_TIME_OUT, /SNMP_ARE_YOU_THERE,

	/SNMP_AUDIT_LEVEL, /SNMP_PRIORITY, /SNMP_SEL_TIME_OUT, /SNMP_STACKSIZE, /TCP_ENABLED, /TIMER_AUDIT_LEVEL, /TIMER_INTERVAL, /TIMER_PRIORITY, /TIMER_STACKSIZE, /TOTAL_ENTITY_SLOTS, /TRACE_MSG_WAIT_TIME, /TRACE_START_WAIT_TIME, /TRAP_AUDIT_LEVEL, /TRAP_PRIORITY, /TRAP_STACKSIZE, /UDP_ENABLED, /USER, /VMS_COLL_INTERVAL, /WKSP_COLL_INTERVAL
QTI	/ACTIVE, /ASTLM, /BIOLM, /BYTLM, /DIOLM, /ENQLM, /FILLM, /LOG, /NODE, /PGFLQUOTA, /POLLING_TIMER, /QTI_PRIORITY, /QTI_USERNAME, /RETRY_TIMER, /STORED, /SUB_TIMEOUT, /TQELM, /WSEXTENT, /WSDEFAULT, /WSQUOTA, /USER
SERVER	/APPLICATION, /CREATION_DELAY, /CREATION_INTERVAL, /DELETION_DELAY, /DELETION_INTERVAL, /LOG, /MAX_INSTANCE, /MIN_INSTANCE, /NODE, /SERVER, /SP_DUMP_FLAG, /USER
TRAP	/ENTITY, /NAME, /NODE, /PARAMETER, /SEVERITY, /TRAP_MIN, /TRAP_MAX, /USER
TSC	/ACTIVE, /ASTLM, /BIOLM, /BYTLM, /CP_PRIORITY, /CP_SLOTS, /CP_USERNAME, /DIOLM, /ENQLM, /FILLM, /LOG, /MAX_LOGINS, /MAX_TTS_CP, /MIN_CPIS, /NODE, /PERM_CPS, /PGFLQUOTA, /STORED, /TQELM, /TSC_PRIORITY, /TSC_USERNAME, /USER, /WSEXTENT, /WSDEFAULT, /WSQUOTA
SHOW Command	
ACC	/ACTIVE, /BRIEF, /CONFIG, /ERROR, /FULL, /ID, /INTERVAL, /NODE, /OUT, /POOL, /RUNTIME, /STORED, /USER
COLLECTION	/BRIEF, /FULL, /INTERVAL, /NODE, /OUT, /USER
CP	/ACTIVE, /ALL, /BRIEF, /CONFIG, /ERROR, /FULL, /ID, /INTERVAL, /NODE, /OUT, /POOL, /PROCESS_NAME, /RUNTIME, /STORED, /USER
ERROR	/BEFORE, /FILENAME, /INTERVAL, /LOCAL, /NODE, /OUT, /SEVERITY, /SINCE, /USER

EXC	/ACTIVE, /ALL, /APPLICATION, /BRIEF, /CONFIG, /ERROR, /FULL, /ID, /INTERVAL, /NODE, /OUT, /POOL, /RUNTIME, /STORED, /USER
FILTER	/NODE, /USER
GROUP	/APPLICATION, /BRIEF, /FULL, /GROUP, /ID, /INTERVAL, /NODE, /OUT, /POOL, /USER
INTERFACE	/INTERVAL, /NODE, /OUT, /USER
LOG	/BEFORE, /FACILITY, /FILENAME, /INTERVAL, /LOCAL, /NODE, /OUT, /SEVERITY, /SINCE, /USER
MANAGER	/INTERVAL, /NODE, /OUT, /USER
PARAMETER	/INTERVAL, /NODE, /OUT, /USER
PROCESS	/ALL, /BRIEF, /FULL, /INTERVAL, /NODE, /OUT, /USER
QTI	/ACTIVE, /ALL, /BRIEF, /CONFIG, /ERROR, /FULL, /ID, /INTERVAL, /NODE, /OUT, /POOL, /RUNTIME, /STORED, /USER
SERVER	/APPLICATION, /BRIEF, /CONFIG, /FULL, /ID, /INTERVAL, /NODE, /OUT, /RUNTIME, /SERVER, /USER
TRAP	/INTERVAL, /NODE, /OUT, /USER
TSC	/ACTIVE, /ALL, /BRIEF, /CONFIG, /ERROR, /FULL, /ID, /INTERVAL, /NODE, /OUT, /POOL, /RUNTIME, /STORED, /USER
USER	/BRIEF, /FULL, /INTERVAL, /NODE, /OUT, /USER
VERSION	None.
START Command	
EXC	/APPLICATION, /NODE, /USER
QTI	/NODE, /USER
SYS	/NOAUDIT, /NODE, /QTI, /NOTERMINALS, /USER
TERMINALS	/NODE, /USER
TRACE_MONITOR	/NODE, /USER
STOP Command	
EXC	/APPLICATION, /CANCEL, /NODE, /USER
MANAGER	/NODE, /USER
QTI	/NODE, /USER
SYS	/CANCEL, /NODE, /USER

TERMINALS	/NODE, /USER
TRACE_MONITOR	/NODE, /USER

11.2. ACMSMGR ADD COLLECTION

ACMSMGR ADD COLLECTION

ACMSMGR ADD COLLECTION — Adds records to the Collection table.

Format

ACMSMGR ADD COLLECTION [/*qualifiers*]

Command Qualifier	Default
/CLASS=*class-name*	* (all)
/COLL_STATE=*keyword*	DISABLED
/ENTITY=*keyword*	Qualifier is required
/NAME=[*,*entity-name*]	* (all)
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/STORAGE_END_TIME=[NEVER, *time*]	NEVER; run until DISABLED
/STORAGE_INTERVAL=*value*	300
/STORAGE_LOCATION=*file-name*	Translation of logical ACMS$MGMT_SNAPSHOT
/STORAGE_START_TIME=[NOW, *time*]	NOW; start as soon as ENABLED
/STORAGE_STATE=*keyword*	DISABLED
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/CLASS=[*, ERROR, POOL, RUNTIME]

This qualifier specifies the class to be enabled or disabled. The default is * (all). See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a description of each class type.

/COLL_STATE=[ENABLED, DISABLED]

This qualifier specifies the state of the collection. The default is DISABLED. When a **SHOW ENTITY** command is issued, data for those classes that have their collection state set to

ENABLED is displayed. Note that while the collection state is DISABLED, the data displayed for an entity may not be accurate. Data cannot be written to the data snapshot file when this qualifier is DISABLED, even when the storage state is ENABLED.

/ENTITY=[*, ACC, AGENT, CP, EXC, GROUP, QTI, SERVER, TSC]

This required qualifier specifies the entity for which collection should be enabled or disabled.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

For ACC, AGENT, CP, QTI, and TSC entity types, the entity name is the process name. For the EXC entity type, the entity name is the name of the application (for example, VR_APPL).

Server and task group names can be specified as compound names made up of an application name and a server or task group name, separated by a period (for example, VR_APPL.VR_READ_SERVER). Either part of server or task group names can be a wildcard (for example, *.VR_READ_SERVER or VR_APPL.*). If only one part of a server or task group name is specified, it is assumed to be the application name, and the server or task group name is wildcarded. For example, VR_APPL is equivalent to VR_APPL.*.

The default is all (*), which is equivalent to *.* for a compound name.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

To execute the command on more than one node, you can specify the node names in a comma-separated list. The ACMSMGR utility attempts to perform the operation sequentially on each node in the list.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/STORAGE_END_TIME=[NEVER, *time*]

This qualifier specifies a time after which the collection data should no longer be written to the snapshot file. The format of *time* is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. If this qualifier is not specified, the default keyword of NEVER is applied, which equates to the OpenVMS zero date of 17-NOV-1858 00:00:00.00. With a value of NEVER, collection data continues to be written to the snapshot file until the storage state is set to DISABLED.

/STORAGE_INTERVAL=*value*

This qualifier controls the frequency (in seconds) at which data snapshots are performed. The default value is 300 seconds.

The storage interval value should be a multiple of the timer interval parameter (**SET PARAMETER/TIMER_INTERVAL**). The timer interval value determines the minimum

elapsed time for many Remote Manager parameters, including the storage interval setting. The relationship of these values determine how often data snapshots are performed, for example:

- If the timer interval value is greater, its value is used by default. For instance, if the timer interval is 10 and the storage interval is 5, snapshots will be written at 10 second intervals.
- If the storage interval value is greater and is a multiple of the timer interval, the storage interval value is used. For example, if the timer interval is 10 and the storage interval is 30, snapshots will be written at 30 second intervals.
- If the storage interval value is greater and is not a multiple of the timer interval, the next multiple of the timer interval value is used. For example, if the timer interval is 10 and the storage interval is 15, snapshots will be written at 20 second intervals.

/STORAGE_LOCATION=*file-name*

This qualifier specifies an OpenVMS file specification to which collection data is to be written. The format of *file-name* is a valid OpenVMS pathname or logical (such as DISK$1:[SYSTEM.SNAPSHOTS] or SYS$SYSTEM:SNAPSHOTS.DAT).

If the **/STORAGE_LOCATION** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SNAPSHOT. If the logical is defined, the value of the logical is used by default. If a directory is not provided as part of the specification, the file is written to the default directory of the account under which the Remote Manager process is running.

Multiple collections can share a single snapshot file or be stored in separate files. For continuity, VSI recommends that EXC, Server, and Task Group collection information be written to the same snapshot file.

/STORAGE_BEGIN_TIME=[NOW, *time*]

This qualifier specifies a time after which the collection data should be written to the snapshot file. The format of *time* is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. If this qualifier is not specified, the default keyword of NOW is applied, which equates to the OpenVMS zero date of 17-NOV-1858 00:00:00.00. With a value of NOW, collection data is written to the snapshot file immediately, or as soon as the storage state is set to ENABLED.

/STORAGE_STATE=[ENABLED, DISABLED]

This qualifier specifies the state of the data snapshots. If this qualifier is not specified, data snapshots are disabled by default. To fully enable data snapshots, both the storage state and the collection state (**/COLL_STATE**) must be set to ENABLED.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

When adding new collection records, the combination of class, entity, and name must be unique.

It is not possible to add records for the ID and CONFIG class. By default, all ACMS processes collect ID and CONFIG class data.

ACMS processes read the Collection table during process startup to determine which classes to begin collecting. Once the Remote Manager has been started, the ACMSMGR **SHOW PROCESS** command can be used to determine the class states for the currently running ACMS processes.

In order for collection data to be written to a snapshot file, the following conditions must be met:

- A qualifying entity must be running (one with an entity type and name matching fields in the Collection table).
- The collection state and storage state for that entity must be enabled.
- The current time must fall between the storage start time and storage end time.

If all these conditions are met, the Remote Manager opens the snapshot file for shared write operations. The file remains open until the storage state is set to DISABLED or until the snapshot period expires.

When multiple collection records apply to a given process, the records are assigned weights according to a precedence of name, then entity, and then class. Within a column, wildcard entries are weighted less than nonwildcard entries. The row with the highest weight that applies to a process is used. The command ACMSMGR **SHOW COLLECTION** displays weights for each row in the table. See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a discussion of the Collection table and how weights are assigned.

In contrast to typical collections, weighting for data snapshot threads does not apply. Therefore, it is possible for redundant collection data to be written to one or more snapshot files. If multiple collection records compile overlapping data, and each has their storage state set to ENABLED, each record writes data to the designated snapshot file.

Examples

```
$ ACMSMGR ADD COLL/ENT=EXC/CLASS=RUNTIME/NAME=VR_APPL
```

This command creates an entry in the Collection table. As a result of this command, the EXC entity for VR_APPL will begin collecting run-time information; however this data will not be saved and written to the data snapshot file.

11.3. ACMSMGR ADD FILTER

ACMSMGR ADD FILTER

ACMSMGR ADD FILTER — Adds records to the Remote Manager Error Filter table. This command (and its qualifiers) is only for use with systems running ACMS Version 4.4 or higher.

Format

ACMSMGR ADD FILTER [/*qualifiers*]

Command Qualifier	Default

/CODE=*value*	None
/FILE=*file-name*	None
/NAME=*error-name*	None
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/CODE=*value*

This qualifier specifies the decimal or hexadecimal value (such as %x5258028) related to the error message being filtered.

/FILE=*file-name*

This qualifier specifies the name of an input file that contains a list of error filter values. The Remote Manager reads this file and adds each code or symbolic name to the Error Filter table.

/NAME=*error-name*

This qualifier specifies the symbolic name (such as %ACMSACC-W-TCS_ LOADING) related to the error message being filtered.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

To execute the command on more than one node, you can specify the node names in a comma-separated list. The ACMSMGR utility attempts to perform the operation sequentially on each node in the list.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

When adding new error filter records, you must specify either the **/CODE**, **/NAME**, or **/FILE** qualifier.

ACMS processes read the Error Filter table during process startup to determine which errors to refrain from sending to the Remote Manager server. Error filtering begins or ends immediately after filter records are added. The Remote Manager signals the appropriate ACMS process as soon as it has reevaluated the Error Filter table following an addition. Messages are sent to active ACMS processes using the ACMS Trace Monitor.

Certain system messages, such as event flags (ACMSACC-I-EVENT), often spawn further status messages indicating the cause of the event (ACMSACC-WFORCEOUT). Error filtering is explicit; that is, only the specified messages are suppressed. If you want to filter the initial and subsequent system messages, you must add each message to the Error Filter table.

Errors are also filtered using the command **SET PARAMETER/ERROR_INTERVAL=*n***. Any errors that are rebroadcast within the specified interval are not sent to the Remote Manager server. The command ACMSMGR **SHOW PARAMETER** can be used to determine the current error interval for ACMS systems.

Examples

```
$ ACMSMGR ADD FILTER /NAME="ACMSACC-W-FORCEOUT" /NODE=SPARKS
```

This command adds the ACMS ACC force out warning message to the Error Filter table. If this message is generated by an ACMS process on node SPARKS, it is not relayed to the Remote Manager.

```
$ ACMSMGR ADD FILTER /FILE=DISK$1:[USER1]FILTER_ERR.DAT
```

This command adds all the system messages in the file FILTER_ERR.DAT to the Error Filter table. These messages are no longer relayed to the Remote Manager.

11.4. ACMSMGR ADD TRAP

ACMSMGR ADD TRAP

ACMSMGR ADD TRAP — Adds records to the Remote Manager Trap table.

Format

ACMSMGR ADD TRAP [/*qualifiers*]

Command Qualifier	Default
/ENTITY=[*,*entity-name*]	Qualifier is required.
/NAME=[*,*entity-name*]	* (all)

/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/PARAMETER=*keyword*	EXISTS
/SEVERITY=[I,W,E,F]	E
/TRAP_MIN=*value*	-1
/TRAP_MAX=*value*	-1
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/ENTITY=[*, ACC, CP, EXC, MGR, QTI, TSC]

This required qualifier specifies the entity or entities for which a trap should be set.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

This field is ignored for the MGR entity.

For ACC, CP, QTI, and TSC entity types, the entity name is the process name. For the EXC entity type, the entity name is the name of the application (for example, VR_APPL).

The default is all (*).

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/PARAMETER=[EVENT_SEVERITY, EXISTS]

This parameter specifies the field that should be monitored.

- EVENT_SEVERITY

Internal Remote Manager events are to be monitored. The Remote Manager logs internal events in the Remote Manager log. See *Section 4.7, "Managing Log Files"* and *Section 11.35, "ACMSMGR SHOW LOG"* for discussions of the Remote Manager log. Traps can be generated based on the severity levels of these events.

- EXISTS

  Process existence is to be monitored. Traps are generated if the associated entity type and name either starts or stops.

/SEVERITY=[I, W, E, F]

This qualifier specifies the severity to be associated with the trap. Severity codes are embedded in the trap message and must be parsed by the trap receiver. Severities are informational (I), warning (W), error (E), or fatal (F).

/TRAP_MIN=*value*

This qualifier specifies the minimum allowable value for the parameter being monitored. A trap is generated if the parameter value is less than the minimum. See *Table 9.14, "Trap Minimums and Maximums"* for a list of valid values.

/TRAP_MAX=*value*

This qualifier specifies the maximum allowable value for the parameter being monitored. A trap is generated if the parameter value is greater than the maximum. See *Table 9.14, "Trap Minimums and Maximums"* for a list of valid values.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access

Notes

When adding new trap records, the combination of entity, name, and parameter must be unique.

Traps become active as soon as they are added to the Trap table and the SNMP interface is running.

See *Table 9.14, "Trap Minimums and Maximums"* for a discussion about setting appropriate trap minimums and maximums. See also *Section 9.14.3, "SNMP Trap Format"* for a description of the trap message generated.

Examples

```
$ ACMSMGR ADD TRAP /ENT=ACC/PARAMETER=EXISTS/TRAP_MIN=1
```

This command causes an SNMP trap to be generated whenever the ACC process stops if the SNMP interface is running.

11.5. ACMSMGR DELETE COLLECTION

ACMSMGR DELETE COLLECTION

ACMSMGR DELETE COLLECTION — Deletes records from the Collection table.

Format

ACMSMGR DELETE COLLECTION [/*qualifiers*]

Command Qualifier	Default
/CLASS=*class-name*	* (all)
/ENTITY=[*,*entity-name*]	Qualifier is required.
/NAME=[*,*entity-name*]	* (all)
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/CLASS=[*, CONFIG, ERROR, ID, POOL, RUNTIME]

This qualifier specifies the class to be enabled or disabled. The default is * (all). See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a description of each class type.

/ENTITY=[*, ACC, AGENT, CP, EXC, GROUP, QTI, SERVER, TSC]

This required qualifier specifies the entity or entities for which collection should be enabled or disabled.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

For ACC, AGENT, CP, QTI, and TSC entity types, the entity name is the process name. For the EXC entity type, the entity name is the name of the application (for example, VR_APPL).

Server and task group names can be specified as compound names made up of an application name and a server or task group name, separated by a period (for example,

VR_APPL.VR_READ_SERVER). Either part of server or task group names can be a wildcard (for example, *.VR_READ_SERVER or VR_APPL.*). If only one part of a server or task group name is specified, it is assumed to be the application name, and the server or task group name is wildcarded. For example, VR_APPL is equivalent to VR_APPL.*.

The default is all (*), which is equivalent to *.* for a compound name.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

When deleting collection records, the combination of class, entity, and name must exactly match the row to be deleted. Deleting a collection record automatically terminates all related snapshot threads.

It is not possible to delete records for the ID and CONFIG class. By default, all ACMS processes collect ID and CONFIG class data.

Collections begin or end immediately after collection records are deleted. The Remote Manager signals the appropriate ACMS process as soon as it has reevaluated the Collection table following a deletion. Messages are sent to the ACMS process using the ACMS Trace Monitor.

ACMS processes read the Collection table during process startup to determine which classes to begin collecting.

The ACMSMGR **SHOW PROCESS** command can be used to determine the class states for the currently running ACMS processes.

Examples

```
$ ACMSMGR DELETE COLL/ENT=EXC/CLASS=RUNTIME/NAME=VR_APPL
```

This command deletes the entry in the Collection table for run-time collection by the VR_APPL application. After the deletion, if there are no other Collection table entries that apply to the run-time class for VR_APPL, run-time collection is disabled.

11.6. ACMSMGR DELETE FILTER

ACMSMGR DELETE FILTER

ACMSMGR DELETE FILTER — Removes records from the Remote Manager Error Filter table. This command (and its qualifiers) is only for use with systems running ACMS Version 4.4 or higher.

Format

ACMSMGR DELETE FILTER [/*qualifiers*]

Command Qualifier	Default
/CODE=*value*	None
/NAME=*error-name*	None
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/CODE=*value*

This qualifier specifies the hexadecimal value (such as %x5258028) related to the error message being filtered.

/NAME=*error-name*

This required qualifier specifies the symbolic name (such as %ACMSACC-WTCS_ LOADING) related to the error message being filtered.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

Either the **/CODE** or **/NAME** qualifier must be specified. When deleting error filter records, the combination of code (or name) and node must exactly match the row to be deleted.

ACMS processes read the Error Filter table during process startup to determine which errors to refrain from sending to the Remote Manager server. Error filtering ends immediately after filter records are deleted. The Remote Manager signals the appropriate ACMS process as soon as it has reevaluated the Error Filter table following a deletion. Messages are sent to active ACMS processes using the ACMS Trace Monitor.

Errors are also filtered using the command **SET PARAMETER/ERROR_INTERVAL=*n***. Any errors that are rebroadcast within the specified interval are not sent to the Remote Manager server. The command ACMSMGR **SHOW PARAMETER** can be used to determine the current error interval for ACMS systems.

Examples

```
$ ACMSMGR DELETE FILTER /NAME="ACMSACC-W-FORCEOUT" /NODE=SPARKS
```

This command deletes the ACMS ACC force out warning message from the Error Filter table. If this message is generated by an ACMS process on node SPARKS, it is relayed to the Remote Manager and written to the error log.

11.7. ACMSMGR DELETE TRAP

ACMSMGR DELETE TRAP

ACMSMGR DELETE TRAP — Deletes a record from the trap table.

Format

ACMSMGR DELETE TRAP [/*qualifiers*]

Command Qualifier	Default
/ENTITY=[*,*entity-name*]	Qualifier is required.

/NAME=[*,*entity-name*]	* (all)
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/PARAMETER=*keyword*	EXISTS
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/ENTITY=[*, ACC, CP, EXC, MGR, QTI, TSC]

This required qualifier specifies the entity or entities for which a trap should be set.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

For the MGR entity, this field should always be set to asterisk (*).

For ACC, CP, QTI, and TSC entity types, the entity name is the process name. For the EXC entity type, the entity name is the name of the application (for example, VR_APPL).

The default is all (*).

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/PARAMETER=[EVENT_SEVERITY, EXISTS]

This parameter specifies the field that should be monitored.

- EVENT_SEVERITY

  Internal Remote Manager events are to be monitored. The Remote Manager logs internal events in the Remote Manager log. See *Section 11.35, "ACMSMGR SHOW LOG"* and *Section 4.7,*

"Managing Log Files" for discussions of the Remote Manager log. Traps can be generated based on the severity levels of these events.

- EXISTS

  Process existence is to be monitored. Traps are generated if the associated entity type and name either starts or stops.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

When deleting trap records, the combination of entity, name and parameter must exactly match a row in the trap table.

Traps become inactive as soon as they are deleted from the Trap table.

Examples

```
$ ACMSMGR DELETE TRAP/ENT=ACC/PARAM=EXISTS
```

This command deletes a trap from the Trap table.

11.8. ACMSMGR HELP

ACMSMGR HELP

ACMSMGR HELP — Displays help information about the ACMS Remote Manager Client (ACMSMGR) and its commands.

Format

ACMSMGR HELP

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

Online help is available for each ACMSMGR command. Each help topic summarizes the valid syntax, abbreviations, parameters, and qualifiers for a particular command and also indicates all default and required values.

For a comprehensive list of ACMS utilities that offer online help or for further instructions on how to invoke help, see ACMS Help.

Examples

```
$ ACMSMGR HELP
```

This command invokes online help for the ACMSMGR utility and displays a list of available topics.

11.9. ACMSMGR LOGIN

ACMSMGR LOGIN

ACMSMGR LOGIN — Logs in to a server on one or more nodes.

Format

ACMSMGR LOGIN [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/PASSWORD=password	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/PASSWORD=*password*

This qualifier specifies the password of the OpenVMS account on the server node to log in as. It is sent encrypted to the server node for verification.

If the **/PASSWORD** parameter is not specified, the ACMSMGR will prompt the user for a password. Login will not be attempted without a password.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node on which to log in.

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility prompts the user for a user name. Login cannot be attempted without a user name.

Notes

In order to access any remote management functions, a valid login is required if proxy access is not enabled or if proxy accounts have not been set up.

A credentials file is created for each node logged in to. Credentials files are specific for a user, process, and node. In addition, a separate credentials file is created for each combination of user name and node. Subsequent ACMSMGR commands pass the authentication information in the appropriate credentials file to the Remote Manager server, which then performs function authorization. Users determine which credentials are used either by using the **/USER** qualifier or by defining the ACMS$MGMT_USER logical.

For example, suppose user BOB on node CLIENT logs in to node SERV1 as ACMS_ADMIN. Also, suppose user BOB on node CLIENT logs in to node SERV2 as ACMS_USER. BOB will have two active logins (two credentials files). He can specify which one to use by either defining the logical ACMS$MGMT_USER, or specifying a user name using the **/USER** qualifier.

Logins are valid for the duration of the login_credentials_lifetime parameter (specified using the ACMSMGR **SET PARAMETER** command).

See *Section 4.4, "Logging In to the Remote Manager"* for a complete discussion of how logins are processed and how credentials files are handled.

Examples

- `$ ACMSMGR LOGIN /NODE=SPARKS /USER=USERNAME /PASSWORD=12345678`

  This command logs in user USERNAME to node SPARKS.

- `$ ACMSMGR LOGIN /NODE=SPARKS,NELSON /USER=USERNAME /PASSWORD=12345678`

This command logs in user USERNAME to nodes SPARKS and NELSON.

- `$ ACMSMGR LOGIN /USER=USERNAME`

  This command logs in user USERNAME to the node specified by the logical name ACMS$MGMT_SERVER_NODE. The ACMSMGR utility will prompt the user for the password.

- `$ ACMSMGR LOGIN`

  This command logs in the user defined by the logical name ACMS$MGMT_ USER to the node specified by the logical name ACMS$MGMT_SERVER_NODE. If the logical name ACMS$MGMT_USER is not defined, the ACMSMGR utility will prompt for the user name. The ACMSMGR utility also will prompt for the password.

11.10. ACMSMGR LOGOUT

ACMSMGR LOGOUT

ACMSMGR LOGOUT — Logs out a user from a server on one or more nodes.

Format

ACMSMGR LOGOUT [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node from which to log out.

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility prompts the user for a user name. Logout cannot be performed without a user name.

Notes

Once logout is complete, subsequent ACMSMGR commands for a user, process, and node will fail authorization checks.

Examples

- `$ ACMSMGR LOGOUT /NODE=SPARKS /USER=USERNAME`

  This command logs out user USERNAME from node SPARKS.

- `$ ACMSMGR LOGOUT /NODE=SPARKS,NELSON /USER=USERNAME /PASSWORD=12345`

  This command logs out user USERNAME from nodes SPARKS and NELSON.

- `$ ACMSMGR LOGOUT /USER=USERNAME`

  This command logs out user USERNAME from the node specified by the logical name ACMS$MGMT_SERVER_NODE.

- `$ ACMSMGR LOGOUT`

  This command logs out the user defined by the logical name ACMS$MGMT_ USER from the node specified by the logical name ACMS$MGMT_SERVER_ NODE. If the logical name ACMS$MGMT_USER is not defined, the ACMSMGR utility will prompt for the user name.

11.11. ACMSMGR REPLACE SERVER

ACMSMGR REPLACE SERVER

ACMSMGR REPLACE SERVER — Replaces a running server in a running ACMS application.

Format

ACMSMGR REPLACE SERVER [/*qualifiers*]

Command Qualifier	Default
/APPLICATION=[*,*application-name*]	* (all)
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE

/SERVER=[*,*server-name*]	* (all)
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/APPLICATION=[*, *application-name*]

This qualifier specifies the name of the application. If this qualifier is not specified, the command is executed for all applications on the target node.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/SERVER=[*, *server-name*]

This qualifier specifies the name of the server. If this qualifier is not specified, the command is executed for all servers on the target node in the target application.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/REPLACE SERVER**. The command is executed synchronously.

Examples

```
$ ACMSMGR REPLACE SERVER /APPL=VR_APPL/SERV=VR_READ_SERVER/NODE=SPARKS
```

This command replaces the VR_READ_SERVER in the VR_APPL application on node SPARKS.

11.12. ACMSMGR RESET ERROR

ACMSMGR RESET ERROR

ACMSMGR RESET ERROR — Resets (closes) the Remote Manager error log file and creates (opens) a new version. This command (and its qualifiers) is only for use with systems running ACMS Version 4.4 or higher.

Format

ACMSMGR RESET ERROR [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command closes the current version of the ACMSMGR error log file and opens a new version.

Examples

```
$ ACMSMGR RESET ERROR /NODE=SPARKS /USER=USERNAME
```

This command resets the Remote Manager error log on node SPARKS.

11.13. ACMSMGR RESET LOG

ACMSMGR RESET LOG

ACMSMGR RESET LOG — Resets (closes) the Remote Manager log file and creates (opens) a new version.

Format

ACMSMGR RESET LOG [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command closes the current version of the ACMSMGR log file and opens a new version.

Examples

```
$ ACMSMGR RESET LOG /NODE=SPARKS /USER=USERNAME
```

This command resets the Remote Manager log on node SPARKS.

11.14. ACMSMGR SAVE FILTER

ACMSMGR SAVE FILTER

ACMSMGR SAVE FILTER — Saves the current records in the Error Filter table to a file. This command (and its qualifiers) is only for use with systems running ACMS Version 4.4 or higher.

Format

ACMSMGR SAVE FILTER [*/qualifiers*]

Command Qualifier	Default
/FILE=*file-name*	Qualifier is required.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/FILE=*file-name*

This required qualifier specifies a full OpenVMS file specification *(node::device:[directory]file-name.ext)* that indicates where the error filter information is to be written. Partial specifications and logical names are not valid.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command provides the ability to write the current set of error filter records to an external file. The **/FILE** qualifier is required and must reference a valid, complete OpenVMS file specification. Logicals and partial names are not recognized.

Examples

```
$ ACMSMGR SAVE FILTER/FILE=VLCROW::DISK1$:[SYSTEM.FILTER]ERROR_FILTER.TXT/
NODE=SPARKS
```

This command saves the error filter records for node SPARKS to the DISK1$:[SYSTEM.FILTER]ERROR_FILTER.TXT on node VLCROW.

11.15. ACMSMGR SET ACC

ACMSMGR SET ACC

ACMSMGR SET ACC — Makes modifications to the ACMS system.

Format

ACMSMGR SET ACC [/*qualifiers*]

Command Qualifier	Default
/ACC_PRIORITY=*value*	None
/ACC_USERNAME=*user-name*	None
/ACTIVE	/STORED
/ASTLM=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/AUDIT_STATE=*keyword*	None
/BIOLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/BYTLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/CHANNELCNT=*value*	None. See /*system-parameter*. Only for use on systems running ACMS Version 4.4 or higher. This special parameter is used by VSI and is subject to change. Do not change this parameter unless VSI recommends that you do so.
/DIOLM=*value*	None. See /*process-quota*.
/ENQLM=*value*	None. See /*process-quota*.
/FILLM=*value*	None. See /*process-quota*.
/GBLPAGES=*value*	None. See /*system-parameter*.
/GBLPAGFIL=*value*	None. See /*system-parameter*.
/GBLSECTIONS=*value*	None. See /*system-parameter*.
/LOG	None
/MAX_APPL=*value*	None
/MSS_MAXBUF=*value*	None
/MSS_MAXOBJ=*value*	None
/MSS_NET_RETRY_TIMER=*value*	None
/MSS_POOLSIZE=*value*	None
/MSS_PROCESS_POOL=*value*	None

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/NODE_NAME=*node-name*	None
/PGFLQUOTA=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/STORED	/STORED
/TQELM=*value*	None. See /*process-quota*.
/TWS_POOLSIZE=*value*	None
/TWSC_POOLSIZE=*value*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER
/USERNAME_DEFAULT=*user-name*	None
/WS_POOLSIZE=*value*	None
/WSC_POOLSIZE=*value*	None
/WSDEFAULT=*value*	None. See /*process-quota*.
/WSEXTENT=*value*	None. See /*process-quota*.
/WSQUOTA=*value*	None. See /*process-quota*.

Privileges Required

ACMS$MGMT_OPER

ACMS$MGMT_SYSUPD (for system parameters)

Parameters

None.

Qualifiers

/process-quota=value

These qualifiers correspond to and update the related process quota fields in the system user authorization (SYSUAF) record for the user specified by ACC_USERNAME. Updated quota values apply to the next process that is created.

Because these qualifiers control the nondynamic values for the related process quotas, the **/ACTIVE** qualifier cannot be specified. The **/STORED** qualifier is the default and causes the specified values to be stored in the current SYSUAF.DAT file.

For information on using AUTHORIZE to modify process quotas, see the *VSI OpenVMS System Manager's Manual* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/]. For more information about the individual quotas and their values, see *VSI OpenVMS System Management Utilities Reference Manual, Volume I: A–L* [https://docs.vmssoftware.com/vsi-

openvms-system-management-utilities-reference-manual-volume-i-a-l/] or access the online help for AUTHORIZE.

/system-parameter=value

These qualifiers correspond to and update the related OpenVMS System Generation utility (SYSGEN) parameters. Updated parameter values apply to the next process that is created. The ACMS$MGMT_SYSUPD rights identifier is required to set these parameters.

Because these qualifiers control the nondynamic values for the related syste parameters, the **/ACTIVE** qualifier cannot be specified. The **/STORED** qualifier is the default and causes the specified values to be stored in the current SYSGEN work area.

For information on using SYSGEN, see the *VSI OpenVMS System Manager's Manual* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/]. For more information about the individual parameters and their values, see *VSI OpenVMS System Management Utilities Reference Manual, Volume II: M–Z* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-ii-m-z/] or access the online help for SYSGEN.

/ACC_PRIORITY=*value*

This qualifier corresponds to and updates the ACMSGEN field ACC_PRIORITY. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/ACC_USERNAME=*user-name*

This qualifier corresponds to and updates the ACMSGEN field ACC_USERNAME. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/ACTIVE

This qualifier causes dynamic ACMSGEN field values to be updated from the current ACMSGEN file. The **/ACTIVE** qualifier cannot be specified on the same command with the **/STORED** qualifier. If neither is specified, the default is **/STORED**. If **/ACTIVE** is specified, no updates are written to the file.

/AUDIT_STATE=[ENABLED, DISABLED]

This qualifier is equivalent to the ACMSOPER command **ACMS/SET SYSTEM/AUDIT** (or **/NOAUDIT** if the value is DISABLED).

/LOG

This qualifier causes status information for the current SET transaction to be displayed to the terminal (SYS$OUTPUT). This qualifier is useful when setting multiple values; a separate status message is displayed for each value that is set.

/MAX_APPL=*value*

This qualifier corresponds to and updates the ACMSGEN field MAX_APPL. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/MSS_MAXBUF=*value*

This qualifer corresponds to and updates the ACMSGEN field MSS_MAXBUF. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/MSS_MAXOBJ=*value*

This qualifier corresponds to and updates the ACMSGEN field MSS_MAXOBJ. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/MSS_NET_RETRY_TIMER=*value*

This qualifier corresponds to and updates the ACMSGEN field MSS_NET_ RETRY_TIMER. As this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified for the running system. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/MSS_POOLSIZE=*value*

This qualifier corresponds to and updates the ACMSGEN field MSS_POOLSIZE. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/MSS_PROCESS_POOL=*value*

This qualifier corresponds to and updates the ACMSGEN field MSS_PROCESS_ POOL. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/NODE_NAME=*node-name*

This qualifier corresponds to and updates the ACMSGEN field NODE_NAME. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/STORED

This qualifier causes ACMSGEN field updates to be written and saved in the current ACMSGEN file. The **/STORED** qualifier cannot be specified on the same command as the **/ACTIVE** qualifier. If neither is specified, the default is **/STORED**.

/TWS_POOLSIZE=*value*

This qualifier corresponds to and updates the ACMSGEN field TWS_POOLSIZE. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/TWSC_POOLSIZE=*value*

This qualifier corresponds to and updates the ACMSGEN field TWSC_POOLSIZE. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the value specified to be stored in the current ACMSGEN file.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

/USERNAME_DEFAULT=*user-name*

This qualifier corresponds to and updates the ACMSGEN field USERNAME_ DEFAULT. Because this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified for the running system. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/WS_POOLSIZE=*value*

This qualifier corresponds to and updates the ACMSGEN field WS_POOLSIZE. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/WSC_POOLSIZE=*value*

This qualifier corresponds to and updates the ACMSGEN field WSC_POOLSIZE. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

Notes

This command provides the ability to remotely update either the running ACMS system or the current ACMSGEN file.

The **/ACTIVE** and **/STORED** qualifiers control how updates are posted to ACMSGEN. The **/ACTIVE** and **/STORED** qualifiers have no effect on the **/AUDIT_ STATE** qualifier, which is processed independently of any ACMSGEN updates.

Examples

- `$ ACMSMGR SET ACC /NODE=SPARKS/MSS_NET_RETRY_TIMER=250/ACTIVE`

This command modifies the ACMSGEN field `mss_net_retry_timer` on node SPARKS and updates the active system only. The change is not saved in the ACMSGEN file.

- `$ ACMSMGR SET ACC /NODE=SPARKS/MSS_NET_RETRY_TIMER=500/STORED`

  This command modifies the ACMSGEN field `mss_net_retry_timer` on node SPARKS and saves the change in the ACMSGEN file. The active system is not updated.

- `$ ACMSMGR SET ACC /NODE=SPARKS/CHANNELCNT=255/STORED`

  This command modifies the CHANNELCNT system parameter on node SPARKS and saves the change in the SYSGEN work area. The active system is not updated.

11.16. ACMSMGR SET AGENT

ACMSMGR SET AGENT

ACMSMGR SET AGENT — Makes modifications to the ACMS system.

Format

ACMSMGR SET AGENT [/*qualifiers*]

Command Qualifier	Default
/PID=*value*	None (Mandatory)
/ASTLM=*value*	None. See */process-quota*.
/BIOLM=*value*	None. See */process-quota*.
/BYTLM=*value*	None. See */process-quota*.
/DIOLM=*value*	None. See */process-quota*.
/ENQLM=*value*	None. See */process-quota*.
/FILLM=*value*	None. See */process-quota*.
/LOG	None
/PGFLQUOTA=*value*	None. See */process-quota*.
/TQELM=*value*	None. See */process-quota*.
/WSDEFAULT=*value*	None. See */process-quota*.
/WSEXTENT=*value*	None. See */process-quota*.
/WSQUOTA=*value*	None. See */process-quota*.
/NODE=*value*	None
/USER=*value*	None

Privileges Required

ACMS$MGMT_OPER

ACMS$MGMT_SYSUPD (for system parameters)

Parameters

None.

Qualifiers

/PID=*pid*

Specifies the Process ID of the Agent the stored values of which to modify. The **/PID** qualifier is Mandatory.

/process-quota=value

These qualifiers correspond to and update the related process quota fields in the system user authorization (SYSUAF) record for the user of the Agent using the supplied process PID. Updated quota values apply to the next Agent that is created.

Because these qualifiers control the nondynamic values for the related process quota fields, the **/ACTIVE** qualifier cannot be specified. The **/STORED** qualifier causes the specified values to be stored in the current SYSUAF.DAT file.

For information on using AUTHORIZE to modify process quotas, see the *VSI OpenVMS System Manager's Manual* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/]. For more information about the individual quotas and their values, see *VSI OpenVMS System Management Utilities Reference Manual, Volume I: A–L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/] or access the online help for AUTHORIZE.

/LOG=*value*

This qualifier causes status information for the current SET transaction to be displayed to the terminal (SYS$OUTPUT). This qualifier is useful when setting multiple values; a separate status message is displayed for each value that is set.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command provides the ability to remotely update the current ACMSGEN file.

The **/ACTIVE** and **/STORED** qualifiers are not used with the ACMSMGR **SET AGENT** command because only stored values can be modified.

Examples

```
$ ACMSMGR SET AGENT /NODE=SPARKS /ASTLM=250 /PID=274009D6
```

This command modifies the ACMSGEN field astlm on node SPARKS for the Agent running in the process of the specified PID.

11.17. ACMSMGR SET COLLECTION

ACMSMGR SET COLLECTION

ACMSMGR SET COLLECTION — Updates records in the Collection table.

Format

ACMSMGR SET COLLECTION [/*qualifiers*]

Command Qualifier	Default
/CLASS=*class-name*	* (all)
/COLL_STATE=*keyword*	None
/ENTITY=[*,*entity-name*]	Qualifier is required.
/NAME=[*,*entity-name*]	* (all)
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/STORAGE_END_TIME=[NEVER, *time*]	None Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_INTERVAL=*value*	None Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_LOCATION=*file-name*	Translation of logical ACMS $MGMT_SNAPSHOT

Command Qualifier	Default
/STORAGE_BEGIN_TIME=[NOW, *time*]	None Only for use on systems running ACMS Version 4.4 or higher.
/STORAGE_STATE=*keyword*	None Only for use on systems running ACMS Version 4.4 or higher.
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/CLASS=[*, CONFIG, ERROR, ID, POOL, RUNTIME]

This qualifier specifies the class to be enabled or disabled. The default is * (all). See *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a description of each class type.

/COLL_STATE=[ENABLED, DISABLED]

This qualifier specifies the state of the collection. When a **`SHOW ENTITY`** command is issued, data for those classes that have their collection state set to ENABLED is displayed. Note that while the collection state is DISABLED, the data displayed for an entity may not be accurate. Data cannot be written to the data snapshot file when this qualifier is DISABLED, even when the storage state is ENABLED.

/ENTITY=[*, ACC, AGENT, CP, EXC, GROUP, QTI, SERVER, TSC]

This required qualifier specifies the entity or entities for which collection should be enabled or disabled.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

For ACC, AGENT, CP, QTI, and TSC entity types, the entity name is the process name. For the EXC entity type, the entity name is the name of the application (for example, VR_APPL).

Server and task group names can be specified as compound names made up of an application name and a server or task group name, separated by a period (for example, VR_APPL.VR_READ_SERVER). Either part of server or task group names can be a wildcard (for example, *.VR_READ_SERVER or VR_APPL.*). If only one part of a server or task group name is specified, it is assumed to be the application name, and the server or task group name is wildcarded. For example, VR_APPL is equivalent to VR_APPL.*.

The default is all (*), which is equivalent to *.* for a compound name.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/STORAGE_END_TIME=[NEVER,*time*]

This qualifier specifies a time after which the collection data should no longer be written to the snapshot file. The format of *time* is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. The keyword NEVER is also supported, which equates to the OpenVMS zero date of 17-NOV-1858 00:00:00.00. With a value of NEVER, collection data continues to be written to the snapshot file until the storage state is set to DISABLED.

If this qualifier is not specified, the existing value remains unchanged. This value can be modified dynamically.

/STORAGE_INTERVAL=*value*

This qualifier controls the frequency (in seconds) at which data snapshots are performed.

The storage interval value should be a multiple of the timer interval parameter (SET PARAMETER/TIMER_INTERVAL). The timer interval value determines the minimum elapsed time for many Remote Manager parameters, including the storage interval setting. The relationship of these values determine how often data snapshots are performed, for example:

- If the timer interval value is greater, its value is used by default. For instance, if the timer interval is 10 and the storage interval is 5, snapshots will be written at 10 second intervals.
- If the storage interval value is greater and is a multiple of the timer interval, the storage interval value is used. For example, if the timer interval is 10 and the storage interval is 30, snapshots will be written at 30 second intervals.
- If the storage interval value is greater and is not a multiple of the timer interval, the next multiple of the timer interval value is used. For example, if the timer interval is 10 and the storage interval is 15, snapshots will be written at 20 second intervals.

/STORAGE_LOCATION=*file-name*

This qualifier specifies an OpenVMS file specification to which collection data is to be written. The format of *file-name* is a valid OpenVMS pathname or logical (such as DISK$1:[SYSTEM.SNAPSHOTS] or SYS$SYSTEM:SNAPSHOTS.DAT).

If the **/STORAGE_LOCATION** qualifier is not specified, the current value remains unchanged. If a directory is not provided as part of the specification, the file is written to the default directory of the account under which the Remote Manager process is running.

Multiple collections can share a single snapshot file or be stored in separate files. For continuity, VSI recommends that EXC, Server, and Task Group collection information be written to the same snapshot file.

This value can be modified dynamically.

/STORAGE_BEGIN_TIME=*time*

This qualifier specifies a time after which the collection data should be written to the snapshot file. The format of *time* is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. The keyword NOW is also supported, which equates to the OpenVMS zero date of 17-NOV-1858 00:00:00.00. With a value of NOW, collection data is written to the snapshot file immediately, or as soon as the storage state is set to ENABLED.

If this qualifier is not specified, the current value remains unchanged. This value can be modified dynamically.

/STORAGE_STATE=[ENABLED, DISABLED]

This qualifier specifies the state of the data snapshots. To fully enable data snapshots, both the storage state and the collection state (**/COLL_STATE**) must be set to ENABLED. If this qualifier is not specified, the current value remains unchanged. This value can be modified dynamically.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

When updating collection records, the combination of class, entity, and name must exactly match a record in the Collection table.

You cannot modify Collection table entries for the ID and CONFIG classes.

ACMS processes read the Collection table during process startup to determine which classes to begin collecting. Once the Remote Manager has been started, the ACMSMGR **SHOW PROCESS** command can be used to determine the class states for the currently running ACMS processes.

In order for collection data to be written to a snapshot file, the following conditions must be met:

- A qualifying entity must be running (one with an entity type and name matching fields in the Collection table).
- The collection state and storage state for that entity must be enabled.

- The current time must fall between the storage start time and storage end time.

If all these conditions are met, the Remote Manager opens the snapshot file for shared write operations. The file remains open until the storage state is set to DISABLED or until the snapshot period expires.

Changes to the Collection table are processed immediately, except for storage location and storage end time values. These values are applied the next time snapshot data is written. The Remote Manager signals the appropriate ACMS process as soon as it has reevaluated the Collection table following an addition. Messages are sent to the ACMS process using the ACMS Trace Monitor.

When multiple collection records apply to a given process, the records are assigned weights according to a precedence of name, then entity, then class. Within a column, wildcard entries are weighted less than nonwildcard entries. The row with the highest weight that applies to a process is used. The ACMSMGR **SHOW COLLECTION** command displays weights for each row in the table. See also *Section 5.1.1, "Entities, Classes, Names, and Collections"* for a discussion of the Collection table and of how weights are assigned.

In contrast to typical collections, weighting for data snapshot threads does not apply. Therefore, it is possible for redundant collection data to be written to one or more snapshot files. If multiple collection records compile overlapping data, and each has their storage state set to ENABLED, each record writes data to the designated snapshot file.

Examples

```
$ ACMSMGR SET COLLECTION/ENT=EXC/CLASS=RUNTIME/NAME=VR_APPL/COLL_S=DISABLED
```

This command disables run-time data collection for the VR_APPL application.

11.18. ACMSMGR SET CP

ACMSMGR SET CP

ACMSMGR SET CP — Makes modifications to an ACMS system process. This command (and its qualifiers) is only for use with systems running ACMS Version 4.4 or higher.

Format

ACMSMGR SET CP [/*qualifiers*]

Command Qualifier	Default
/ASTLM=*value*	None. See */process-quota*.
/BIOLM=*value*	None. See */process-quota*.
/BYTLM=*value*	None. See */process-quota*.
/DIOLM=*value*	None. See */process-quota*.
/ENQLM=*value*	None. See */process-quota*.
/FILLM=*value*	None. See */process-quota*.
/LOG	None

/PGFLQUOTA=*value*	None. See /*process-quota*.
/TQELM=*value*	None. See /*process-quota*.
/WSDEFAULT=*value*	None. See /*process-quota*.
/WSEXTENT=*value*	None. See /*process-quota*.
/WSQUOTA=*value*	None. See /*process-quota*.

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/process-quota=value

These qualifiers correspond to and update the related process quota fields in the system user authorization (SYSUAF) record for the user specified by CP_ USERNAME. Updated quota values apply to the next process that is created.

Because these qualifiers control the nondynamic values for the related process quota fields, the **/ACTIVE** qualifier cannot be specified. The **/STORED** qualifier causes the specified values to be stored in the current SYSUAF.DAT file.

For information on using AUTHORIZE to modify process quotas, see the *VSI OpenVMS System Manager's Manual* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/]. For more information about the individual quotas and their values, see *VSI OpenVMS System Management Utilities Reference Manual, Volume I: A–L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/] or access the online help for AUTHORIZE.

/LOG

This qualifier causes status information for the current SET transaction to be displayed to the terminal (SYS$OUTPUT). This qualifier is useful when setting multiple values; a separate status message is displayed for each value that is set.

Notes

This command provides the ability to remotely update either the running ACMS system or the current ACMSGEN file quota values for subsequent CP processes.

Examples

```
$ ACMSMGR SET CP /ASTLM=1000 /LOG
```

This command modifies the ASTLM process quota and causes informational messages to be displayed that indicate the status of the update. The new ASTLM value will be applied to subsequent CP processes.

11.19. ACMSMGR SET EXC

ACMSMGR SET EXC

ACMSMGR SET EXC — Makes modifications to running ACMS applications.

Format

ACMSMGR SET EXC [*/qualifiers*]

Command Qualifier	Default
/ACTIVE	/ACTIVE
/APPLICATION=*application-name*	None
/ASTLM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/AUDIT_STATE=*keyword*	None
/BIOLM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/BYTLM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/DIOLM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/ENQLM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/FILLM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/LOG	None
/MAX_SERVERS=*value*	None
/MAX_TASKS=*value*	None
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/PGFLQUOTA=*value*	None. See */process-quota*.

Command Qualifier	Default
	Only for use on systems running ACMS Version 4.4 or higher.
/SP_MON_INTERVAL=*value*	None
/STORED	/ACTIVE
/TQELM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/TRANSACTION_TIMEOUT=*value*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER
/WSDEFAULT=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/WSEXTENT=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/WSQUOTA=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/process-quota=value

These qualifiers correspond to and update the related process quota fields in the system user authorization (SYSUAF) record for the user specified by **`SHOW EXC/ID`**. Updated quota values apply to the next process that is created.

Because these qualifiers control the nondynamic values for the related process quota fields, the **`/ACTIVE`** qualifier cannot be specified. The **`/STORED`** qualifier causes the specified values to be stored in the current SYSUAF.DAT file.

For information on using AUTHORIZE to modify process quotas, see the *VSI OpenVMS System Manager's Manual* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/]. For more information about the individual quotas and their values, see *VSI OpenVMS System Management Utilities Reference Manual, Volume I: A–L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/] or access the online help for AUTHORIZE.

/ACTIVE

This qualifier causes dynamic ACMSGEN field values to be updated from the current ACMSGEN file. The **/ACTIVE** qualifier cannot be specified on the same command with the **/STORED** qualifier. If neither is specified, the default is **/ACTIVE** for all values except for process quotas (which default to **/STORED**). If **/ACTIVE** is specified, no updates are written to the file.

/APPLICATION=*application-name*

The name of the application to be modified. The command ACMSMGR **SET EXC** requires the **/APPL** qualifier for any values that are stored values. If the **/APPL** qualifier is missing, the error NOAPPLQUAL will be returned.

A different error, NOSUCHAPPL, will be returned if the application you are attempting to modify is not active.

/AUDIT_STATE=[ENABLED, DISABLED]

This qualifier is equivalent to the ACMSOPER command **ACMS/MODIFY APPLICATION /APPL=AUDIT** (or **/APPL=NOAUDIT** if the value is DISABLED).

/LOG

This qualifier causes status information for the current SET transaction to be displayed to the terminal (SYS$OUTPUT). This qualifier is useful when setting multiple values; a separate status message is displayed for each value that is set.

/MAX_SERVERS=*value*

This qualifier updates the MAX_SERVERS limit in the running application. Updates are lost when the application is restarted.

/MAX_TASKS=*value*

This qualifier updates the MAX_TASKS limit in the running application. Updates are lost when the application is restarted.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/SP_MON_INTERVAL=*value*

This qualifier updates the SP_MON_INTERVAL field in the running application. Updates are lost when the application is restarted.

/STORED

This qualifier causes ACMSGEN field updates to be written and saved in the current ACMSGEN file. The **/STORED** qualifier cannot be specified on the same command as the **/ACTIVE** qualifier.

If neither is specified, the default is **/ACTIVE** for all values except for process quotas (which default to **/STORED**).

/TRANSACTION_TIME=*value*

This qualifier updates the TRANSACTION_TIMEOUT default value in the running application. Updates are lost when the application is restarted.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/MOD APPLICATION**. Any changes made to the running system are lost when the application is restarted.

Examples

```
$ ACMSMGR SET EXC/APPL=VR_APPL/SP_MON_INTERVAL=10/MAX_TASKS=50
```

This command modifies the running application VR_APPL.

11.20. ACMSMGR SET INTERFACE

ACMSMGR SET INTERFACE

ACMSMGR SET INTERFACE — Allows Remote Manager interfaces to be started or stopped.

Format

ACMSMGR SET INTERFACE [/*qualifiers*]

Command Qualifier	Default
/INTERFACE=[RPC,SNMP]	Qualifier is required.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/STATE=[ENABLED,DISABLED]	DISABLED
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/INTERFACE=[RPC, SNMP]

This required qualifier specifies which interface to modify. Only SNMP is supported.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/STATE=[ENABLED, DISABLED]

This qualifier specifies the operation to perform. If the value is ENABLED, the interface will be started (if it is not already running). If the value is DISABLED, the interface will be stopped (if it is not already stopped).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command allows interfaces to be stopped or started. However, an interface cannot disable itself. Since the ACMSMGR utility uses the RPC interface, it cannot be used to disable the RPC interface. To disable the RPC interface, either use the ACMSCFG utility and restart the Remote Manager, or use the SNMP interface.

The SNMP interface can be both enabled and disabled using this command. It may take several seconds for this command to complete if the SNMP interface is in a non-interruptible state when the command is issued.

Examples

```
$ ACMSMGR SET INTERFACE/INTERFACE=SNMP/STATE=DISABLED

ACMS Remote Management Option -- Command line utility
```

```
Call to modify interface on server sparks was executed
%ACMSMGMT-S-SUCCESS, Operation completed
```

This command stops the SNMP interface on the node specified by the logical name ACMS$MGMT_SERVER_NODE. Authorization is either performed for the user specified by the logical ACMS$MGMT_USER, or is based on an ACMS proxy on the target node if the logical is not defined.

11.21. ACMSMGR SET PARAMETER

ACMSMGR SET PARAMETER

ACMSMGR SET PARAMETER — Allows Remote Manager parameters to be modified.

Format

ACMSMGR SET PARAMETER [/*qualifiers*]

Command Qualifier	Default
/DCL_AUDIT_LEVEL=*value*	See /parameter.
/DCL_MGR_PRIORITY=*value*	See /parameter.
/DCL_STACKSIZE=*value*	See /parameter.
/ERROR_INTERVAL=*value*	See /parameter. Only for use on systems running ACMS Version 4.4 or higher.
/EVENT_LOG_PRIORITY=*value*	See /parameter.
/LOG_STACKSIZE=*value*	See /parameter.
/LOGIN_CREDS_LIFETIME=*value*	See /parameter.
/MAX_LOGINS=*value*	See /parameter.
/MAX_RPC_RETURN_RECS=*value*	See /parameter.
/MGR_AUDIT_LEVEL=*value*	See /parameter.
/MSG_PROC_AUDIT_LEVEL=*value*	See /parameter.
/MSG_PROC_PRIORITY=*value*	See /parameter.
/MSG_PROC_STACKSIZE=*value*	See /parameter.
/MSS_COLL_INTERVAL=*value*	See /parameter.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/PROC_MON_AUDIT_LEVEL=*value*	See /parameter.
/PROC_MON_INTERVAL=*value*	See /parameter.
/PROC_MON_PRIORITY=*value*	See /parameter.
/PROC_MON_STACKSIZE=*value*	See /parameter.
/PROXY_CREDS_LIFETIME=*value*	See /parameter.

Command Qualifier	Default
/RPC_AUDIT_LEVEL=*value*	See /parameter.
/RPC_PRIORITY=*value*	See /parameter.
/RPC_STACKSIZE=*value*	See /parameter.
/SECURITY_AUDIT_LEVEL=*value*	See /parameter.
/SNAP_AUDIT_LEVEL=*value*	See /parameter. Only for use on systems running ACMS Version 4.4 or higher.
/SNAP_PRIORITY=*value*	See /parameter. Only for use on systems running ACMS Version 4.4 or higher.
/SNAP_STACKSIZE=*value*	See /parameter. Only for use on systems running ACMS Version 4.4 or higher.
/SNMP_AGENT_TIME_OUT=*value*	See /parameter.
/SNMP_ARE_YOU_THERE=*value*	See /parameter.
/SNMP_AUDIT_LEVEL=*value*	See /parameter.
/SNMP_PRIORITY=*value*	See /parameter.
/SNMP_SEL_TIME_OUT=*value*	See /parameter.
/SNMP_STACKSIZE=*value*	See /parameter.
/TCP_ENABLED	See /parameter. Only for use on systems running ACMS Version 4.4 or higher.
/TIMER_AUDIT_LEVEL=*value*	See /parameter.
/TIMER_INTERVAL=*value*	See /parameter.
/TIMER_PRIORITY=*value*	See /parameter.
/TIMER_STACKSIZE=*value*	See /parameter.
/TOTAL_ENTITY_SLOTS=*value*	See /parameter.
/TRACE_MSG_WAIT_TIME=*value*	See /parameter.
/TRACE_START_WAIT_TIME=*value*	See /parameter.
/TRAP_AUDIT_LEVEL=*value*	See /parameter.
/TRAP_PRIORITY=*value*	See /parameter.
/TRAP_STACKSIZE=*value*	See /parameter.
/UDP_ENABLED	See /parameter.

Command Qualifier	Default
	Only for use on systems running ACMS Version 4.4 or higher.
/USER=*user-name*	Translation of logical ACMS$MGMT_USER
/VMS_COLL_INTERVAL=*value*	See /parameter. Only for use on systems running ACMS Version 4.4 or higher.
/WKSP_COLL_INTERVAL=*value*	See /parameter.

Privileges Required

ACMS$MGMT_WRITE

Parameters

None.

Qualifiers

/parameter=*value*

All qualifiers except NODE and USER correspond directly to fields in the Parameter table. See *Section 9.10.1, "Field Descriptions"* for a description of each field. For a listing of the current default, minimum, and maximum values, use the **SHOW PARAMETER** command.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

Some parameter changes take effect immediately; others take effect only when the Remote Manager is restarted, and only if they are written to the ACMSCFG file (using the ACMSCFG **SET PARAMETER** command). *Table 9.9, "Parameter Table"* shows which parameters are dynamic and which are not.

For parameters that are not dynamic, you must restart the appropriate facility for the change to take effect. For example, to modify the SNMP_SELECT_TIME_ OUT parameter, you must stop and restart the SNMP interface.

The ACMS Remote Manager allows an authorized user to make changes to the VMS parameter file (via SYSGEN), ACMS parameter file (via ACMSGEN), user quotas (via AUTHORIZE), ACMS Remote Manager parameters (via ACMSMGR SET PARAM) and to a running ACMS system (via ACMSOPR commands).

Some values are checked for minimums, like negative numbers and zero. ACMS Remote Manager parameters are checked for limits but Authorize, ACMSGEN and SYSGEN values are not. Use the same caution with the ACMS Remote Manager as you would with SYSGEN and AUTHORIZE and verify any changes you make.

Examples

```
$ ACMSMGR SET PARAMETER /MGR_AUDIT_LEVEL=E /NODE=SPARKS /USER=USERNAME

ACMS Remote Management Option -- Command line utility
Call to modify parameters on server sparks was executed
%ACMSMGMT-S-SUCCESS, Operation completed
```

This command modifies the dynamic parameter MGR_AUDIT_LEVEL on node SPARKS and specifies that authorization be performed for user USERNAME.

11.22. ACMSMGR SET QTI

ACMSMGR SET QTI

ACMSMGR SET QTI — Makes modifications related to the Queued Task Initiator (QTI).

Format

ACMSMGR SET QTI [/*qualifiers*]

Command Qualifier	Default
/ACTIVE	/STORED
/ASTLM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/BIOLM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/BYTLM=*value*	None. See */process-quota*.

Command Qualifier	Default
	Only for use on systems running ACMS Version 4.4 or higher.
/DIOLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/ENQLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/FILLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/LOG	None
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/PGFLQUOTA=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/POLLING_TIMER=*value*	None
/QTI_PRIORITY=*value*	None
/QTI_USERNAME=*user-name*	None
/RETRY_TIMER	None
/STORED	/STORED
/SUB_TIMEOUT	None
/TQELM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/USER=*user-name*	Translation of logical ACMS$MGMT_USER
/WSDEFAULT=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/WSEXTENT=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/WSQUOTA=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/process-quota=value

These qualifiers correspond to and update the related process quota fields in the system user authorization (SYSUAF) record for the user specified by QTI_USERNAME. Updated quota values apply to the next process that is created.

Because these qualifiers control the nondynamic values for the related process quota fields, the **/ACTIVE** qualifier cannot be specified. The **/STORED** qualifier causes the specified values to be stored in the current SYSUAF.DAT file.

For information on using AUTHORIZE to modify process quotas, see the *VSI OpenVMS System Manager's Manual* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/]. For more information about the individual quotas and their values, see *VSI OpenVMS System Management Utilities Reference Manual, Volume I: A–L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/] or access the online help for AUTHORIZE.

/ACTIVE

This qualifier causes dynamic ACMSGEN field values to be updated from the current ACMSGEN file. The **/ACTIVE** qualifier cannot be specified on the same command with the **/STORED** qualifier. If neither is specified, the default is **/STORED**. If **/ACTIVE** is specified, no updates are written to the file.

/LOG

This qualifier causes status information for the current SET transaction to be displayed to the terminal (SYS$OUTPUT). This qualifier is useful when setting multiple values; a separate status message is displayed for each value that is set.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/POLLING_TIMER=*node-name*

This qualifier corresponds to and updates the ACMSGEN field QTI_POLLING_ TIMER. Because this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified

for the running system. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/QTI_PRIORITY=*value*

This qualifier corresponds to and updates the ACMSGEN field QTI_PRIORITY. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/QTI_USERNAME=*user-name*

This qualifier corresponds to and updates the ACMSGEN field QTI_USERNAME. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/RETRY_TIMER=*value*

This qualifier corresponds to and updates the ACMSGEN field QTI_RETRY_ TIMER. Because this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified for the running system. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/STORED

This qualifier causes ACMSGEN field updates to be written and saved in the current ACMSGEN file. The **/STORED** qualifier cannot be specified on the same command as the **/ACTIVE** qualifier. If neither is specified, the default is **/STORED**.

/SUB_TIMEOUT=*value*

This qualifier corresponds to and updates the ACMSGEN field QTI_SUB_ TIMEOUT. Because this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified for the running system. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command allows you to remotely update either the running ACMS system or the current ACMSGEN file.

The **/ACTIVE** and **/STORED** qualifiers control how updates are posted to ACMSGEN. The **/ACTIVE** and **/STORED** qualifiers have no effect on the **/AUDIT_ STATE** qualifier, which is processed independently of any ACMSGEN updates.

Examples

- `$ ACMSMGR SET QTI /NODE=SPARKS/SUB_TIMEOUT=5000/ACTIVE`

  This command modifies the ACMSGEN field `qti_sub_timeout` on node SPARKS and updates the active system only. The change is not saved in the ACMSGEN file.

- `$ ACMSMGR SET QTI /NODE=SPARKS/SUB_TIMEOUT=5000/STORED`

  This command modifies the ACMSGEN field `qti_sub_timeout` on node SPARKS and saves the change in the ACMSGEN file. The active system is not updated.

11.23. ACMSMGR SET SERVER

ACMSMGR SET SERVER

ACMSMGR SET SERVER — Makes modifications to servers running in ACMS applications.

Format

ACMSMGR SET SERVER [*/qualifiers*]

Command Qualifier	Default
/APPLICATION=[*,application-name]	* (all)
/CREATION_DELAY=*value*	None
/CREATION_INTERVAL=*value*	None
/DELETION_DELAY=*value*	None
/DELETION_INTERVAL=*value*	None
/LOG	None
/MAX_INSTANCE=*value*	None
/MIN_INSTANCE=*value*	None
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/SERVER=[*,server_name]	* (all)
/SP_DUMP_FLAG=[ENABLED,DISABLED]	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/APPLICATION=*application-name*

The name of the application to be modified. If this qualifier is not specified, modifications are applied to all applications.

/CREATION_DELAY=*value*

This qualifier updates the CREATION_DELAY for the specified server in the running application. Updates are lost when the application is restarted.

/CREATION_INTERVAL=*value*

This qualifier updates the CREATION_INTERVAL for the specified server in the running application. Updates are lost when the application is restarted.

/DELETION_DELAY=*value*

This qualifier updates the DELETION_DELAY for the specified server in the running application. Updates are lost when the application is restarted.

/DELETION_INTERVAL=*value*

This qualifier updates the DELETION_INTERVAL for the specified server in the running application. Updates are lost when the application is restarted.

/LOG

This qualifier causes status information for the current SET transaction to be displayed to the terminal (SYS$OUTPUT). This qualifier is useful when setting multiple values; a separate status message is displayed for each value that is set.

/MAX_INSTANCE=*value*

This qualifier updates the MAX_INSTANCE limit for the specified server in the running application. Updates are lost when the application is restarted.

/MIN_INSTANCE=*value*

This qualifier updates the MIN_INSTANCE limit for the specified server in the running application. Updates are lost when the application is restarted.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/SERVER=*server-names*

This qualifier specifies the name of the server to be modified. If this qualifier is not specified, all servers in the application are modified.

/SP_DUMP_FLAG=*value*

This qualifier updates the SP_DUMP_FLAG for the specified server in the running application. Updates are lost when the application is restarted.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/MOD APPLICATION/SERVER**. Any changes made to the running system are lost when the application is restarted.

Examples

```
$ ACMSMGR SET SERVER/APPL=VR_APPL/SERVER=VR_READ_SERVER/
SP_DUMP_FLAG=ENABLED
```

This command modifies the SP_DUMP_FLAG field for the server VR_READ_ SERVER running in the VR_APPL application.

11.24. ACMSMGR SET TRAP

ACMSMGR SET TRAP

ACMSMGR SET TRAP — Updates records in the Trap table.

Format

ACMSMGR SET TRAP [/*qualifiers*]

Command Qualifier	Default
/ENTITY=[*,*entity-name*]	Qualifier is required.
/NAME=[*,*entity-name*]	* (all)

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/PARAMETER=*keyword*	EXISTS
/SEVERITY=[I,W,E,F]	E
/TRAP_MIN=*value*	-1
/TRAP_MAX=*value*	-1
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/ENTITY=[*, ACC, CP, EXC, MGR, QTI, TSC]

This required qualifier specifies the entity or entities for which a trap should be set.

/NAME=[*,*entity-name*]

This qualifier specifies particular instances of an entity. Wildcards (*) are allowed in names.

For the MGR entity, this field should always be set to asterisk (*).

For ACC, CP, QTI, and TSC entity types, the entity name is the process name. For the EXC entity type, the entity name is the name of the application (for example, VR_APPL).

The default is all (*).

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/PARAMETER=[EVENT_SEVERITY, EXISTS]

This parameter specifies the field that should be monitored.

- EVENT_SEVERITY

  Internal Remote Manager events are to be monitored. The Remote Manager logs internal events in the Remote Manager log. See *Section 11.35, "ACMSMGR SHOW LOG"* and *Section 4.7, "Managing Log Files"* for discussions of the Remote Manager log. Traps can be generated based on the severity levels of these events.

- EXISTS

  Process existence is to be monitored. Traps are generated if the associated entity type and name either starts or stops.

/SEVERITY=[I, W, E, F]

This qualifier specifies the severity to be associated with the trap. Severity codes are embedded in the trap message and must be parsed by the trap receiver. Severities are informational (I), warning (W), error (E), or fatal (F).

/TRAP_MIN=*value*

This qualifier specifies the minimum allowable value for the parameter being monitored. A trap is generated if the parameter value is less than the minimum. See *Table 9.14, "Trap Minimums and Maximums"* for a list of valid values.

/TRAP_MAX=*value*

This qualifier specifies the maximum allowable value for the parameter being monitored. A trap is generated if the parameter value is greater than the maximum. See *Table 9.14, "Trap Minimums and Maximums"* for a list of valid values.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

When updating trap records, the combination of entity, name, and parameter must exactly match a record in the Trap table.

Changes become active as soon as they are added to the Trap table.

See *Table 9.14, "Trap Minimums and Maximums"* for a discussion about setting appropriate trap minimums and maximums. See *Section 9.14.3, "SNMP Trap Format"* for a description of the trap message generated.

Examples

```
$ ACMSMGR SET TRAP /ENT=QTI/PARAM=EXISTS/MAX=0
```

This command causes an SNMP trap to be generated whenever the QTI process is started if the SNMP interface is running.

11.25. ACMSMGR SET TSC

ACMSMGR SET TSC

ACMSMGR SET TSC — Makes modifications to the ACMS terminal subsystem.

Format

ACMSMGR SET TSC [/*qualifiers*]

Command Qualifier	Default
/ACTIVE	/STORED
/ASTLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/BIOLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/BYTLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/CP_PRIORITY=*value*	None
/CP_SLOTS=*value*	None
/CP_USERNAME=*user-name*	None
/DIOLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/ENQLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/FILLM=*value*	None. See /*process-quota*. Only for use on systems running ACMS Version 4.4 or higher.

Command Qualifier	Default
/LOG	None
/MAX_LOGINS=*value*	None
/MAX_TTS_CP=*value*	None
/MIN_CPIS=*value*	None
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/PERM_CPS=*value*	None
/PGFLQUOTA=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/STORED	/STORED
/TQELM=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/TSC_PRIORITY=*value*	None
/TSC_USERNAME=*user-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER
/WSDEFAULT=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/WSEXTENT=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.
/WSQUOTA=*value*	None. See */process-quota*. Only for use on systems running ACMS Version 4.4 or higher.

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/process-quota=value

These qualifiers correspond to and update the related process quota fields in the system user authorization (SYSUAF) record for the user specified by **`/TSC_USERNAME`**. Updated quota values apply to the next process that is created.

Because these qualifiers control the nondynamic values for the related process quota fields, the **/ACTIVE** qualifier cannot be specified. The **/STORED** qualifier is the default and causes the specified values to be stored in the current SYSUAF.DAT file.

For information on using AUTHORIZE to modify process quotas, see the *VSI OpenVMS System Manager's Manual* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/]. For more information about the individual quotas and their values, see *VSI OpenVMS System Management Utilities Reference Manual, Volume I: A–L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/] or access the online help for AUTHORIZE.

/ACTIVE

This qualifier causes dynamic ACMSGEN field values to be updated from the current ACMSGEN file. The **/ACTIVE** qualifier cannot be specified on the same command with the **/STORED** qualifier. If neither is specified, the default is **/STORED**. If **/ACTIVE** is specified, no updates are written to the file.

/CP_PRIORITY=*value*

This qualifier corresponds to and updates the ACMSGEN field CP_PRIORITY. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/CP_SLOTS=*value*

This qualifier corresponds to and updates the ACMSGEN field CP_SLOTS. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/CP_USERNAME=*user-name*

This qualifier corresponds to and updates the ACMSGEN field CP_USERNAME. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/LOG

This qualifier causes status information for the current SET transaction to be displayed to the terminal (SYS$OUTPUT). This qualifier is useful when setting multiple values; a separate status message is displayed for each value that is set.

/MAX_LOGINS=*value*

This qualifier corresponds to and updates the ACMSGEN field MSS_MAX_ LOGINS. As this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified for the running system. The **/STORED** qualifier causes the value specified to be stored in the current ACMSGEN file.

/MAX_TTS_CP=*value*

This qualifier corresponds to and updates the ACMSGEN field MAX_TTS_CP. Because this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified for

the running system. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/MIN_CPIS=*value*

This qualifier corresponds to and updates the ACMSGEN field MIN_CPIS. Because this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified for the running system. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/PERM_CPS=*value*

This qualifier corresponds to and updates the ACMSGEN field PERM_CPS. Because this is a dynamic ACMSGEN field, the **/ACTIVE** qualifier causes the current value to be modified for the running system. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/STORED

This qualifier causes ACMSGEN field updates to be written and saved in the current ACMSGEN file. The **/STORED** qualifier cannot be specified on the same command as the **/ACTIVE** qualifier. If neither is specified, the default is **/STORED**.

/TSC_PRIORITY=*value*

This qualifier corresponds to and updates the ACMSGEN field TSC_PRIORITY. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/TSC_USERNAME=*user-name*

This qualifier corresponds to and updates the ACMSGEN field TSC_USERNAME. Because this is a nondynamic ACMSGEN field, the **/ACTIVE** qualifier cannot be specified with this qualifier. The **/STORED** qualifier causes the specified value to be stored in the current ACMSGEN file.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command provides the ability to remotely update either the running ACMS system or the current ACMSGEN file.

The **/ACTIVE** and **/STORED** qualifiers control how updates are posted to ACMSGEN.

Examples

- `$ ACMSMGR SET TSC /NODE=SPARKS/MAX_LOGINS=500/ACTIVE`

  This command modifies the ACMSGEN field max_logins on node SPARKS and updates the active system only. The change is not saved in the ACMSGEN file.

- `$ ACMSMGR SET TSC /NODE=SPARKS/MAX_LOGINS=500/STORED`

  This command modifies the ACMSGEN field max_logins on node SPARKS and saves the change in the ACMSGEN file. The active system is not updated.

11.26. ACMSMGR SHOW ACC

ACMSMGR SHOW ACC

ACMSMGR SHOW ACC — Displays information about an ACC on one or more remote nodes.

Format

ACMSMGR SHOW ACC [/*qualifiers*]

Command Qualifier	Default
/[BRIEF,FULL]	/FULL if the CLASS=* (all). /BRIEF, otherwise.
/CLASS=*keyword*	*
/ACTIVE	See Notes.
/[BRIEF,FULL]	/FULL if no class qualifier (/CONFIG, /ID, /POOL or /RUNTIME) is specified. Otherwise, /BRIEF.
/[CONFIG,ID,POOL,RUNTIME]	* (all)
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE

Command Qualifier	Default
/OUT=*file-name*	None
/STORED	See Notes.
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/ACTIVE

When specified with the **/BRIEF** qualifier, this qualifier causes active ACMSGEN field values to be displayed. **/ACTIVE** is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. If no class qualifier (**/CONFIG**, **/ERROR**, **/ID**, **/POOL**, or **/RUNTIME**) is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**). Note that OpenVMS process quota and SYSGEN parameter information is only shown when **/FULL** is specified.

/[CONFIG,ERROR,ID,POOL,RUNTIME]

This qualifier causes data for only the specified class to be displayed. The default is to display information for all classes.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/STORED

When specified with the **/BRIEF** qualifier, this qualifier causes field values from the ACMSGEN file (not those active in memory) to be displayed. The **/STORED** qualifier is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays information about the ACC on the target node. The **/BRIEF** and **/FULL** qualifiers control the format of information to be displayed. To display OpenVMS process quota and SYSGEN parameter information, use the **/FULL** qualifier.

See *Section 9.2, "ACC Table"* for a discussion of each field displayed.

Note that some information may not be current, depending on whether the class to which the data belongs has been enabled for the ACC. The Config Class field indicates whether or not information is being collected for that class.

Examples

1.
```
$ ACMSMGR SHOW ACC /CONFIG /NODE=GOCROW,VLCROW
ACMS Remote Management - Command line utility  ACMS V4.3-0  ACC Table
 Display                     Time: 25-AUG-1999 13:59:06.99
         A Collect  Audit  Max   Mss    Mss Proc  Mss     Mss     WS
  TWS
   Node  S  State   State  Appl Max Obj   Pool   Poolsize Maxbuf
 Poolsize Poolsize
  ------ - ------- ------- ---- ------- -------- -------- ------
 -------- --------
  gocrow A enabled enabled 10   1006    512      4096     1544   256
  1440
  vlcrow A enabled enabled 10   1006    512      2048     1544   256
  1440
```

   This command displays ACC configuration information from nodes GOCROW and VLCROW. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or

by proxy if the logical is not defined. Only summary configuration information is displayed because neither the **/BRIEF** nor **/FULL** qualifier was supplied.

2. $ **ACMSMGR SHOW ACC /NODE=VLCROW /USER=JONES**

```
ACMS Remote Management - Command line utility
ACMS V4.3-0  ACC Table Display                Time: 25-AUG-1999
 13:59:04.38
 ================================================================================
 Node     IDENTIFICATION
 --------
 ----------------------------------------------------------------
 vlcrow   Id Class Collection State         enabled
          Version                           V4.3-0
          Process Name                      ACMS01ACC001000
          PID                               37C0024F
          User Name                         LT$ACC_V31
          Start Time                        24-AUG-1999 14:49:15.76
          End Time                          (null)
 Node     CONFIGURATION                     Active          Stored
 -------- --------------------------------- ---------------
 -------------
 vlcrow   Config Class Collection State     enabled
          System Auditing State             enabled
          ACC Running State                 started
          ACC Username                      LT$ACC_V31      LT$ACC_V31
          ACC Base Priority                 4               4
          Max Applications                  10              10
          MSS Max Objects                   1006            1006
          MSS Maxbuf (bytes)                1544            1544
          MSS Poolsize (pagelets)           2048            2048
          MSS Process Pool (pagelets)       512             512
          MSS Net Retry Timer (D) (seconds) 10              10
          Username Default (D)              LTU_ACMSDEF     LTU_ACMSDEF
          Node Name (ACMSGEN) (DECnet node) VLCROW          VLCROW
          WS Poolsize  (pagelets)           256             256
          WSC Poolsize (pagelets)           128             128
          TWS Poolsize (pagelets)           1440            1440
          TWSC Poolsize (pagelets)          169             169
 Node     RUNTIME
 --------
 ---------------------------------------------------------------------------
 vlcrow   Runtime Class Collection State    enabled
          DECnet Object                     started
          Gauges                            Current  Max    Limit   Max
 Time
          --------------------------------- -------- ------ -------
 ----------------------
          Users: Total                      4        85
 (null)
          Users: Local                      4        20
 (null)
          Users: Remote                     0        0
 (null)
          Applications                      1        1      20      24-
AUG-1999 14:49:49.19

          Number of application starts      6
```

```
         Process Quotas               Current      Max          Limit
Max Time
         -------------------------- -----------  ----------  -------
------------------------
         Working Set Size              6544        6544           300000
18-APR-2001 14:49:15.76
         AST Limit                        5 (  0%)     5 (  0%)      500
18-APR-2001 14:49:15.76
         Byte Limit                    1792 (  0%)  3648 (  0%) 1775409
18-APR-2001 14:49:15.76
         Direct I/O Limit                 0 (  0%)     1 (  0%)   15000
18-APR-2001 14:49:15.76
         Buffered I/O Limit               2 (  0%)     3 (  0%)   10000
18-APR-2001 14:49:15.76
         Enqueue Limit                    2 (  0%)     2 (  0%)    2000
18-APR-2001 14:49:15.76
         File Limit                       3 (  0%)     3 (  0%)    1001
18-APR-2001 14:49:15.76
         Page File Quota               7056 (  1%)  7056 (  1%)  500000
18-APR-2001 14:49:15.76
         Timer Queue Limit                4 (  0%)     4 (  0%)     500
18-APR-2001 14:49:15.76
         Channel Count                   29 ( 11%)    31 ( 12%)     256
18-APR-2001 14:49:15.76
Node     POOL
--------
--------------------------------------------------------------------------------
vlcrow   Pool Class Collection State       enabled
         Objects                           Current   Max    Time
         --------------------------------- --------- ------
------------------------
         MSS Objects                       252       432    25-AUG-1999
13:59:03.86
         Process Pool                                        Pct    Time
         ----------------------------------------- ------
------------------------
         Pool Size                         262144
         Current Free (bytes)              255312    (97%)
         Minimum Free (bytes)              255056    (97%)  24-AUG-1999
15:00:17.30
         Largest Current Free Block (bytes) 65536
         Minimum Largest Free Block (bytes) 65536           25-AUG-1999
13:59:03.25
         Allocation Failures                0
         Garbage Collections                0
         Shared Pool                                         Pct    Time
         ----------------------------------------- ------
------------------------
         Pool Size                         1048576
         Current Free (bytes)              973176    (92%)
         Minimum Free (bytes)              948480    (90%)  24-AUG-1999
16:s22:58.01
         Largest Current Free Block (bytes) 65536
         Minimum Largest Free Block (bytes) 65536           25-AUG-1999
13:59:03.25
         Allocation Failures                0
         Garbage Collections                0
```

This command displays all ACC management information from node VLCROW. Authorization is performed for user JONES. Since no class qualifiers (**/ID**, **/CONFIG**, **/RUNTIME**, **/POOL**) were specified, information is returned for all classes by default.

11.27. ACMSMGR SHOW AGENT

ACMSMGR SHOW AGENT

ACMSMGR SHOW AGENT — Displays Collection table data for one or more Agents.

Format

ACMSMGR SHOW AGENT [/*qualifiers*]

Command Qualifier	Default
/ACTIVE	/ACTIVE (CONFIG class brief display)
/ALL	Inactive data not displayed.
/BRIEF	Default when a class is specified.
/[CONFIG,ERROR,ID,POOL,RUNTIME]	All are shown when no class is specified.
/FULL	Full when no class is specified.
/INTERVAL=*value*	None
/OUT=[*filespec*,*logical_name*]	SYS$OUTPUT
/PROCESS_NAME=*proc_name*	*
/STORED	/ACTIVE
/NODE=*value*	None
/USER=*value*	None

Parameters

None.

Qualifiers

/ACTIVE

When specified with the **/BRIEF** qualifier, this qualifier causes active ACMSGEN field values to be displayed. **/ACTIVE** is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/ALL

This qualifier displays all available application data, even data for applications that may no longer be running. When applications are stopped, the table row they were occupying is marked for reuse. If the row has not been reused, the data remains available for display. This qualifier allows that data to be displayed. Inactive rows are flagged with an asterisk (*) in the output.

To inhibit the display of old data, do not specify this qualifier. By default, only data for currently running processes is displayed.

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. If no class qualifier (**/CONFIG**, **/ERROR**, **/ID**, **/POOL**, or **/RUNTIME**) is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**). Available OpenVMS process quota and SYSGEN parameter information is only displayed when **/FULL** is specified.

/[CONFIG,ERROR,ID,POOL,RUNTIME]

This qualifier causes data for only the specified class to be displayed. If this qualifier is omitted, the default is to display information for all classes.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/PROCESS_NAME=*process-name*

This qualifier causes data for only the specified process to be displayed. If this qualifier is omitted, the default is to display information for all Agent processes.

/STORED

When specified with the **/BRIEF** qualifier, this qualifier causes field values from the ACMSGEN file (not those active in memory) to be displayed. The **/STORED** qualifier is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Examples

```
$ ACMSMGR SHOW AGENT

ACMS Remote Management -- Command line utility
ACMS RM Client V5.0    Agent Table Display    Time: 9-FEB-2005 10:00:15.32
 ================================================================================
 Node         IDENTIFICATION
 ------------
 ----------------------------------------------------------------
OHMARY        ID Class Collection State            enabled
              Process Name                         _FTA37:
              PID                                  00014D60
              User Name                            HALL
              Start Time                            9-FEB-2005 09:58:47.12
              End Time                             (null)

 Node         CONFIGURATION                      Active         Stored
 ------------ ---------------------------------- --------------
 --------------
OHMARY        Config Class Collection State      enabled
              Working Set Default                65536          80000
              Working Set Extent                 262144         90000
              Working Set Quota                  65536          65535
              AST Limit                          1024           1024
              Byte Limit                         347184         350000
              Direct I/O Limit                   500            500
              Buffered I/O Limit                 500            500
              Enqueue Limit                      16776959       32767
              File Limit                         5000           5000
              Page File Quota                    1000000        1000000
              Timer Queue Limit                  150            150

 Node         RUNTIME
 ------------
 ----------------------------------------------------------------
OHMARY        Runtime Class Collection State     enabled
              Agent Running State                started
              DECnet Object                      stopped

         Terminals/Tasks/User Defined  Current  Max       Max Time
------------------------------------ -------- --------- -----------
         Attached Terminals            0        0         (null)
         Active Task Calls             0        1         9-FEB-2005
 09:58:50.6
         Total Tasks Executed          258
         User1 Time                     (null)
         User2 Time                     (null)
         User3 Time                     (null)
         User1 Data                    0
         User2 Data                    0
         User3 Data                    0
         User4 Data                    0
         User5 Data                    0
```

```
        User6 Data                    0

        TDMS                                Current  Max       Max Time
        ----------------------------------- -------- --------- -----------
        Active TDMS                         0        0         (null)
        Active TDMS Menu Requests           0        0         (null)
        Active TDMS Read Messages           0        0         (null)
        Active TDMS Write Messages          0        0         (null)
        Active TDMS Cancels                 0        0         (null)
        Total TDMS Requests                 0
        Total TDMS Menu Requests            0
        Total TDMS Read Messages            0
        Total TDMS Write Messages           0
        Total TDMS Cancels                  0

        DECforms                            Current  Max       Max Time
        ----------------------------------- -------- --------- -----------
        Active DECforms                     0        0         (null)
        Active DECforms Menu Requests       0        0         (null)
        Active DECforms Enables             0        0         (null)
        Active DECforms Disables            0        0         (null)
        Active DECforms Cancel              0        0         (null)
        Active DECforms Send Requests       0        0         (null)
        Active DECforms Receive Requests    0        0         (null)
        Active DECforms Transceive Reqsts   0        0         (null)
        Total DECforms Requests             0
        Total DECforms Menu Requests        0
        Total DECforms Cancel Requests      0
        Total DECforms Send Requests        0
        Total DECforms Receive Requests     0
        Total DECforms Transceive Reqsts    0
        Total DECforms Enables              0
        Total DECforms Disables             0

        Process Quotas      Current   Max         Limit     Max Time
        ------------------  -------   ---------   ------
-------------------
        Working Set Size    65536     65536       262144    9-FEB-2005
10:00:07.13
        AST Limit           11 (1%)   11(1%)      1024      9-FEB-2005
10:00:07.13
        Byte Limit          9024(2%)  9024(2%)    347184    9-FEB-2005
10:00:07.13
        Direct I/O Limit    0 (0%)    0(0%)       500       9-FEB-2005
10:00:07.13
        Buffered I/O Limit  2 (0%)    2(0%)       500       9-FEB-2005
10:00:07.13
        Enqueue Limit       1 (0%)    1(0%)       16776959  9-FEB-2005
10:00:07.13
        File Limit          5 (0%)    5(0%)       5000      9-FEB-2005
10:00:07.13
        Page File Quota     25904(2%) 25904(2%)   1000000   9-FEB-2005
10:00:07.13
        Timer Queue Limit   5 (3%)    5 (3%)      150       9-FEB-2005
10:00:07.13
        Channel Count       29        29                    9-FEB-2005
10:00:07.13
```

```
Node    POOL
-------
------------------------------------------------------------------
OHMARY  Pool Class Collection State    enabled
        MSS Process Pool                          Pct      Time
        ----------------------------------        -------  -----------
        Pool Size (bytes)                  7680000
        Current Free (bytes)               7669200  (99%)
        Minimum Free (bytes)               7669200  (99%)    9-FEB-2005
10:00:07.13
        Largest Current Free Block (bytes) 65536
        Minimum Largest Free Block (bytes) 65536             9-FEB-2005
10:00:07.13
        Allocation Failures 0
        Garbage Collections 0

Node    ERROR
-------- ---------------------------------------------------------------
OHMARY Error Class Collection State    enabled
       Error Count                     0
       Last Error Message              0
       Time of Last Error              (null)
```

This command displays the contents of the Collection table for all Agents that are running.

11.28. ACMSMGR SHOW COLLECTION

ACMSMGR SHOW COLLECTION

ACMSMGR SHOW COLLECTION — Displays Collection table data from one or more remote nodes.

Format

ACMSMGR SHOW COLLECTION [/*qualifiers*]

Command Qualifier	Default
/[BRIEF,FULL]	/BRIEF
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed.

Note that storage start and end times for data snapshots are only visible when **/FULL** is provided. When not specified, the resulting summary display may contain truncated values for some of the longer fields (such as entity name and storage location).

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays data-collection configuration information on the target node.

See *Section 9.7, "EXC Table"* for a discussion of each field displayed. See *Section 5.1, "Managing Data Collection"* for a discussion of collections.

Examples

```
$ ACMSMGR SHOW COLL/NODE=VLCROW
```

```
ACMS Remote Management - Command line utility
ACMS V5.0  Entity/Collection table Display     Time: 18-APR-2001
 13:59:11.31

            Entity   Entity Collect Collect                       Storage
 Storage
Node     Wt Type     Name   Class   State    Storage Location     State
 Interval
-------- -- -------- -----  ------- -------- ------------------   --------
 --------
vlcrow   2  *        *      id      enabled  acms$mgmt_snapshot   enabled
 3600
vlcrow   2  *        *      config  enabled  acms$mgmt_snapshot   disabled
 3600
vlcrow   2  *        *      runtime enabled  acms$mgmt_snapshot   disabled
 10
vlcrow   2  *        *      pool    enabled  acms$mgmt_snapshot   disabled
 10
vlcrow   2  *        *      error   enabled  acms$mgmt_snapshot   disabled
 10
```

This command displays the contents of the Collection table on node VLCROW, where all collections have been enabled for all entities. Authorization is performed for the user specified by the logical ACMS$MGMT_USER, or by proxy if the logical is not defined. Data snapshots for the ID class have been enabled and are set to occur every 5 minutes (3600 seconds). The data files are stored in the file specified by the logical ACMS$MGMT_SNAPSHOT.

11.29. ACMSMGR SHOW CP

ACMSMGR SHOW CP

ACMSMGR SHOW CP

Format

ACMSMGR SHOW CP [*/qualifiers*]

Command Qualifier	Default
/ACTIVE	See Notes.
/ALL	Current data only
/[BRIEF,FULL]	/FULL if no class qualifier (/ID, /POOL, or /RUNTIME) is specified. Otherwise, /BRIEF.
/[*class-name*]	* (all)
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/PROCESS_NAME=*process-name*	* (all)
/STORED	See Notes.

Command Qualifier	Default
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/ACTIVE

When specified with the **/BRIEF** qualifier, this qualifier causes active ACMSGEN field values to be displayed. **/ACTIVE** is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/ALL

This qualifier displays all available application data, even data for applications that are no longer running. When applications are stopped, the CP table row they were occupying is marked for reuse. If the row has not been reused, the data remains available for display. This qualifier allows that data to be displayed. Inactive rows are flagged with an asterisk (*) in the output.

To inhibit the display of old data, do not specify this qualifier. By default, only data for currently running processes is displayed.

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. If no class qualifier (**/CONFIG**, **/ERROR**, **/ID**, **/POOL**, or **/RUNTIME**) is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**). Available OpenVMS process quota and SYSGEN parameter information is only displayed when **/FULL** is specified.

/[CONFIG, ERROR, ID, POOL, RUNTIME]

This qualifier causes data for only the specified class to be displayed. If this qualifier is omitted, the default is to display information for all classes.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/PROCESS_NAME=*process-name*

This qualifier causes data for only the specified process to be displayed. If this qualifier is omitted, the default is to display information for all CP processes.

/STORED

When specified with the **/BRIEF** qualifier, this qualifier causes field values from the ACMSGEN file (not those active in memory) to be displayed. The **/STORED** qualifier is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays information about ACMS command processes (CPs) on the target node. The **/BRIEF** and **/FULL** qualifiers control the format of information to be displayed.

See *Section 9.5, "CP Table"* for a discussion of each field displayed.

Note that some information may not be current, depending on whether the class to which the data belongs has been enabled for the CP. The Collect State field indicates whether or not information is being collected for that class.

Examples

```
$ ACMSMGR SHOW CP/RUNTIME/NODE=VLCROW
ACMS Remote Management -- Command line utility
ACMS V5.0 CP Table Display                  Time: 18-APR-2001 13:59:13.39
                              Active  Active   Total   Total   Total
```

```
       Runtime   Process      Attached  Task  DECforms  TDMS    Task
 Dataset
Node   Class       Name        Terms   Calls  Requests Requests Calls
 Hangups
------ ------- -------------- -------- ------ -------- -------- -----
 -------
vlcrow enabled ACMS01CP001000 0        0      0        0        20    0
vlcrow enabled ACMS01CP002000 1        1      0        0        20    0
vlcrow enabled ACMS01CP003000 0        0      0        0        20    0
vlcrow enabled ACMS01CP004000 1        1      0        0        20    0
```

This command displays summary RUNTIME class information for CPs on node VLCROW. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.30. ACMSMGR SHOW ERROR

ACMSMGR SHOW ERROR

ACMSMGR SHOW ERROR — Displays the errors recorded in the Remote Manager error log file. This command (and its qualifiers) is only for use with systems running ACMS Version 4.4 or higher.

Format

ACMSMGR SHOW ERROR [/*qualifiers*]

Command Qualifier	Default
/BEFORE=*time*	End of file
/BRIEF	/BRIEF
/FILENAME=*file-name*	Translation of logical ACMS$MGMT_ERR_LOG
/FULL	/BRIEF
/INTERVAL=*interval*	Command is executed once.
/LOCAL	Remote
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/SEVERITY=[I,W,E,F]	All
/SINCE=*time*	Beginning of file
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/BEFORE=*time*

This qualifier causes only those error log entries with a timestamp less than or equal to the *time* specified by time to be returned and displayed. The format of *time* is *DD-MMM-YY:HH:MM:SS.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. If this qualifier is not specified, the search ends when the end of the audit file is reached.

/BRIEF

The **/BRIEF** and **/FULL** qualifiers control the amount of information shown for any errors. ACMSMGR SHOW ERROR defaults to the **/BRIEF** display.

/FILENAME=*file-name*

This qualifier allows log records to be displayed from a file other than the current error log file. Specify a fully- or partially-qualified file specification.

/FULL

The **/BRIEF** and **/FULL** qualifiers control the amount of information shown for any errors. ACMSMGR SHOW ERROR defaults to the **/BRIEF** display.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/LOCAL

This qualifier causes the ACMSMGR to open and read the error log on the local node directly. You can use this qualifier if the Remote Manager process is not started. The **/LOCAL** qualifier overrides the **/NODE** qualifier and the ACMS$MGMT_SERVER_NODE logical.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/SEVERITY=[I, W, E, F]

This qualifier causes only error log entries with matching severities to be displayed. Valid severities are informational (I), warning (W), error (E), and fatal (F). If this qualifier is not specified, all severities are returned.

/SINCE=*time*

This qualifier causes only error log entries with a timestamp greater than or equal to the time specified by *time* to be returned and displayed. The format of *time* is *DD-MMM-YY:HH:MM:SS.nn*. Partial dates and times (for example, 10- OCT or 09:00) are supported. If this qualifier is not specified, the search begins at the beginning of the audit file.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays Remote Manager error log information. The format of the error log entries displayed is:

```
node time: severity: error-text
```

In this format:

- *node* is the node from which the information was obtained.
- *time* is the time the error was logged.
- *severity* is the severity of the error.
- *error-text* is the details of the error.

Examples

```
$ ACMSMGR SHOW ERROR /NODE=VLCROW
ACMS Remote Management -- Command line utility

ACMS V5.0  Log Display                Time: 18-APR-2001 13:59:13.39
 Node   Message
vlcrow  : 17-APR-2001 10:40:41.04 : %ACMSACC-I-EVENT, Event
                                  : -ACMSACC-E-ERRSTARTA, Error occurred
 starting application
                                  : -ACMSEXC-ENO_
TDB, Error opening TDB file !AS for task group !AS
vlcrow : 16-APR-2001 14:26:01.34 : %ACMSMSS-E-ERRNETCRE, Error creating
 DECnet object
```

```
                                    : -ACMSMSS-ENODEMISMATCH,
NODE_Name is ACMSPAR does not match DECnet node name
```

This command displays entries from the Remote Manager error log on node VLCROW. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.31. ACMSMGR SHOW EXC

ACMSMGR SHOW EXC

ACMSMGR SHOW EXC — Displays information about an application on one or more remote nodes.

Format

ACMSMGR SHOW EXC [/*qualifiers*]

Command Qualifier	Default
/ACTIVE	See Notes.
/ALL	Current applications only.
/APPLICATION=*application_name*	* (all)
/[BRIEF,FULL]	/FULL if no class qualifier (/CONFIG, /ID, /POOL, or /RUNTIME) is specified. Otherwise, /BRIEF
/[*class-name*]	* (all)
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/STORED	See Notes.
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/ACTIVE

When specified with the **/BRIEF** qualifier, this qualifier causes active ACMSGEN field values to be displayed. **/ACTIVE** is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/ALL

This qualifier displays all available application data, even data for applications that may no longer be running. When applications are stopped, the EXC table row they were occupying is marked for reuse. If the row has not been reused, the data remains available for display. This qualifier allows that data to be displayed. Inactive rows are flagged with an asterisk (*) in the output.

To inhibit the display of old data, do not specify this qualifier. By default, only data for currently running processes is displayed.

/APPLICATION=*application-name*

This qualifier specifies a particular ACMS application to display. Wildcard matching is performed on the name provided; use of asterisks (*) is allowed.

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. If no class qualifier (**/CONFIG**, **/ERROR**, **/ID**, **/POOL**, or **/RUNTIME**) is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**). Available OpenVMS process quota and SYSGEN parameter information is only displayed when **/FULL** is specified.

/[CONFIG,ERROR,ID,POOL,RUNTIME]

This qualifier causes data for only the specified class to be displayed. If this qualifier is omitted, the default is to display information for all classes.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically every interval seconds. Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/STORED

When specified with the **/BRIEF** qualifier, this qualifier causes field values from the ACMSGEN file (not those active in memory) to be displayed. The **/STORED** qualifier is effective only when

used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays information about ACMS applications on the target node. The **/BRIEF** and **/FULL** qualifiers control the format of information to be displayed.

See *Section 9.7, "EXC Table"* for a discussion of each field displayed.

Note that some information may not be current, depending on whether the class to which the data belongs has been enabled for the application. The Collect State field indicates whether or not information is being collected for that class.

Examples

```
$ ACMSMGR SHOW EXC/ID
ACMS Remote Management -- Command line utility
ACMS V5.0  EXC Table Display                    Time: 18-APR-2001
 13:59:09.33
        ID
  Node Class PID Process Name Start Time Application Name
------ ------- -------- --------------- ----------------------- ----------
vlcrow enabled 37C0025A ACMS01EXC001000 18-APR-2001 14:49:49.22 LDT_APPL_A
gocrow enabled 38000249 ACMS01EXC001000 18-AUG-2001 15:07:23.51 LDT_APPL_B
```

This command displays summary IDENTIFICATION class information for all applications on the nodes specified by the logical name ACMS$MGMT_SERVER_ NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.32. ACMSMGR SHOW FILTER

ACMSMGR SHOW FILTER

ACMSMGR SHOW FILTER — Displays the errors currently being filtered on one or more nodes. This command (and its qualifiers) is only for use with systems running ACMS Version 4.4 or higher.

Format

ACMSMGR SHOW FILTER [/*qualifiers*]

Command Qualifier	Default
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays all system errors currently being filtered for the specified node or nodes.

Examples

```
$ ACMSMGR SHOW FILTER /NODE=VLCROW
ACMS Remote Management -- Command line utility
ACMS V5.0  ACMS Error Filter Table Display     Time: 18-APR-2001
 13:59:09.33
 Node   Filtered Message Name (Code)
------
 ------------------------------------------------------------------
VLCROW ACMSACC-W-AUDSYSSTARTS ( FD8748)
VLCROW SYSTEM-W-TOOMUCHDATA ( 298)
VLCROW SYSTEM-W-NOMOREREG ( AE8)
```

This command displays the current errors being filtered for node VLCROW. When generated by an ACMS process on node VLCROW, these errors are not relayed to the Remote Manager.

11.33. ACMSMGR SHOW GROUP

ACMSMGR SHOW GROUP

ACMSMGR SHOW GROUP — Displays information about one or more ACMS task groups on one or more nodes.

Format

ACMSMGR SHOW GROUP [*/qualifiers*]

Command Qualifier	Default
/APPLICATION=*application_name*	* (all applications)
/[BRIEF,FULL]	/FULL if no class qualifier (/ID or /POOL) is specified. Otherwise, /BRIEF.
/GROUP=*group-name*	* (all groups)
/[ID,POOL]	* (all)
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/APPLICATION=*application-name*

This qualifer specifies a particular ACMS application to display. Wildcard matching is performed on the name provided; use of asterisks (*) is allowed.

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. If no class qualifier (**/ID** or **/POOL**) is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**). If a class qualifier is used on the command, the default action provides a brief format display. Using **/FULL** with a class qualifier produces a full format display.

/GROUP=*group-name*

This qualifier specifies a particular ACMS task group to display. Wildcard matching is performed on the name provided; use of asterisks (*) is allowed.

/[ID, POOL]

This qualifier causes data for only the specified class to be displayed. If this qualifier is omitted, the default is to display information for all classes.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays information about ACMS application task groups on the target node. The **/BRIEF** and **/FULL** qualifiers control the format of information to be displayed.

See *Section 9.13, "Task Group Table"* for a discussion of each field displayed.

Note that some information may not be current, depending on whether the class to which the data belongs has been enabled for the task group. The Collect State field indicates whether or not information is being collected for that class.

Examples

```
$ ACMSMGR SHOW GROUP/POOL

ACMS Remote Management -- Command line utility
ACMS V5.0  Task Group Table Display           Time: 18-APR-2001 13:59:36.35
        Pool      Application                   -TWS Pool  Free-     -TWSC
 Pool Free-
 Node   Class     Name         Task Group Name  Current   Minimum
 Current  Mininum
------  --------  -----------  ---------------  --------  ---------
 -------  ---------
VLCROW  enabled   LDT_APPL_A   TEST_GRP01       99 %      99 %       99 %
   98 %
VLCROW  enabled   LDT_APPL_A   TLOAD001_GRP     97 %      97 %       98 %
   98 %
VLCROW  enabled   LDT_APPL_A   DBMS_LOAD_GRP    99 %      99 %       99 %
   99 %
VLCROW  enabled   LDT_APPL_A   RDB_LOAD_GRP     99 %      99 %       99 %
   98 %
VLCROW  enabled   LDT_APPL_A   RMSR_GRP         99 %      99 %       99 %
   98 %
VLCROW  enabled   LDT_APPL_A   TCT_ LDT_GROUP   99 %      90 %       98 %
   95 %
VLCROW  enabled   LDT_APPL_A   AT_TT_GROUP      99 %      99 %       99 %
   98 %
VLCROW  enabled   LDT_APPL_A   RI_FMS           99 %      99 %       99 %
   98 %
VLCROW  enabled   LDT_APPL_A   RI_SMG           99 %      99 %       99 %
   99 %
VLCROW  enabled   LDT_APPL_A   VF_GROUP         99 %      99 %       99 %
   98 %
VLCROW  enabled   LDT_APPL_A   CS_GROUP         99 %      99 %       99 %
   97 %
VLCROW  enabled   LDT_APPL_A   DT_GROUP         96 %      95 %       92 %
   89 %
VLCROW  enabled   LDT_APPL_A   VF_V32_GROUP     99 %      99 %       98 %
   97 %
VLCROW  enabled   LDT_APPL_A   DCL_CLI_GROUP    99 %      99 %       99 %
   98 %
VLCROW  enabled   LDT_APPL_A   DETASK_GROUP     99 %      99 %       99 %
   98 %
```

This command displays summary POOL class information for all task groups in all applications on the node specified by the logical name ACMS$MGMT_ SERVER_NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.34. ACMSMGR SHOW INTERFACE

ACMSMGR SHOW INTERFACE

ACMSMGR SHOW INTERFACE — Displays Remote Manager interface (RPC or SNMP) information for a Remote Manager process on one or more nodes.

Format

ACMSMGR SHOW INTERFACE [/*qualifiers*]

Command Qualifier	Default
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

The ACMS Remote Manager supports two interfaces: RPC and SNMP. This command displays the running and enabled states of each interface, along with some counter and status information. See *Section 9.8, "Interfaces Table"* for a discussion of each field displayed.

Examples

```
$ ACMSMGR SHOW INTERFACE /NODE=VLCROW,GOCROW /USER=JONES

ACMS Remote Management -- Command line utility
ACMS V5.0  Interfaces Display                Time: 18-APR-2001 13:59:15.51
                 Enabled Running Get      Set      Alarms Time Last
Node   Interface State   State   Requests Requests Sent   Alarm Sent
------ --------- ------- ------- -------- -------- ------
 ------------------------
vlcrow rpc       enabled started 987      0        0      17-NOV-1858
 00:00:00.00
vlcrow snmp      enabled started 0        0        0      17-NOV-1858
 00:00:00.00
gocrow rpc       enabled started 964      0        0      17-NOV-1858
 00:00:00.00
gocrow snmp      enabled started 0        0        0      17-NOV-1858
 00:00:00.00
```

This command displays information for the Remote Manager interfaces on nodes VLCROW and GOCROW. Authorization is performed for user JONES.

11.35. ACMSMGR SHOW LOG

ACMSMGR SHOW LOG

ACMSMGR SHOW LOG — Displays Remote Manager log entries for a server on one or more nodes.

Format

ACMSMGR SHOW LOG [/*qualifiers*]

Command Qualifier	Default
/BEFORE=*time*	End of file
/FACILITY=facility	All
/FILENAME=*file-name*	Translation of logical ACMS$MGMT_LOG
/INTERVAL=*interval*	Command is executed once.
/LOCAL	Remote
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/SEVERITY=[I,W,E,F]	All
/SINCE=*time*	Beginning of file
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/BEFORE=*time*

This qualifier causes only audit log entries with a timestamp less than or equal to the time specified by *time* to be returned and displayed. The format of *time* is *DD-MMM-YY:HH:MM:SS.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported. If this qualifier is not specified, the search ends when the end of the audit file is reached.

/FACILITY=[DCL, LOG, MGR, MSG_PROC, PROCMON, RPC, SEC, SNAP, SNMP, TRAP]

This qualifier causes only audit log entries with matching facilities to be displayed. If this qualifier is not specified, all facilities are returned.

/FILENAME=*file-name*

This qualifier allows log records to be displayed from a file other than the current log file. Specify a fully or partially qualified file specification.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/LOCAL

This qualifier causes the ACMSMGR to open and read the audit log on the local node directly. You can use this qualifier if the Remote Manager process is not started. The **/LOCAL** qualifier overrides the **/NODE** qualifier and the ACMS$MGMT_SERVER_NODE logical.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/SEVERITY=[I, W, E, F]

This qualifier causes only audit log entries with matching severities to be displayed. Valid severities are informational (I), warning (W), error (E), and fatal (F). If this qualifier is not specified, all severities are returned.

/SINCE=*time*

This qualifier causes only audit log entries with a timestamp greater than or equal to the time specified by *time* to be returned and displayed. The format of *time* is *DD-MMM-YY:HH:MM:SS.nn*. Partial dates and times (for example, 10- OCT or 09:00) are supported. If this qualifier is not specified, the search begins at the beginning of the audit file.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access

Notes

This command displays Remote Manager audit file information. The format of the audit entries displayed is:

```
node  time: facility: severity: error-text
```

In this format:

- *node* is the node from which the information was obtained.
- *time* is the time the entry was logged.
- *facility* is the facility that generated the entry.

- *severity* is the severity of the entry.
- *error-text* is the details of the entry.

Examples

- ```
  $ ACMSMGR SHOW LOG /NODE=VLCROW /SINCE="20-MAR-2001 11:00" -
  _$ /BEFORE="20-MAR-2001 12:00"
  ACMS Remote Management -- Command line utility
  ACMS V5.0 Log Display Time:                  18-APR-2001 13:59:17.45
  Node Message
  vlcrow 20-MAR-2001 11:00:37.15 : log: i : Log opened
  vlcrow 20-MAR-2001 11:02:29.43 : msg_proc: i : EXC shutdown. Attempting
   to
          unmap application global section for ACMS01EXC001000
  vlcrow 20-MAR-2001 11:02:29.43 : msg_proc: i : Application global
   section
          unmapped for ACMS01EXC001000
  vlcrow 20-MAR-2001 11:02:29.43 : msg_proc: i : EXC shutdown processing
          complete for ACMS01EXC001000
  vlcrow 20-MAR-2001 11:02:35.18 : msg_proc: i : CP process shutdown
   message
          received for ACMS01CP001000
  vlcrow 20-MAR-2001 11:02:38.40 : msg_proc: i : CP process shutdown
   message
          received for ACMS01CP004000
  vlcrow 20-MAR-2001 11:02:38.40 : msg_proc: i : CP process shutdown
   message
          received for ACMS01CP003000
  vlcrow 20-MAR-2001 11:02:38.40 : msg_proc: i : CP process shutdown
   message
          received for ACMS01CP002000
  vlcrow 20-MAR-2001 11:02:39.23 : msg_proc: i : TSC process shutdown
   message
          received for ACMS01TSC001000
  vlcrow 20-MAR-2001 11:02:41.93 : msg_proc: i : ACC process shutdown
   message
          received for ACMS01ACC001000
  vlcrow 20-MAR-2001 11:02:42.05 : procmon: w : ACC process is absent
   after
          being present.
  vlcrow 20-MAR-2001 11:03:00.23 : msg_proc: i : ACC process startup
   message
          received for ACMS01ACC001000
  vlcrow 20-MAR-2001 11:03:00.23 : msg_proc: e : Failure getting current
          collection states. Ignoring process ACMS01ACC001000
  vlcrow 20-MAR-2001 11:03:03.93 : msg_proc: i : TSC process startup
   message
          received for ACMS01TSC001000
  vlcrow 20-MAR-2001 11:03:10.03 : msg_proc: i : CP process startup
   message
          received for ACMS01CP001000
  vlcrow 20-MAR-2001 11:03:10.04 : msg_proc: i : CP process startup
   message
          received for ACMS01CP002000
  vlcrow 20-MAR-2001 11:03:10.04 : msg_proc: i : CP process startup
   message
          received for ACMS01CP004000
  ```
  ```

```
vlcrow 20-MAR-2001 11:03:10.05 : msg_proc: i : CP process startup
 message
         received for ACMS01CP003000
vlcrow 20-MAR-2001 11:03:31.87 : msg_proc: i : EXC startup. Attempting
 to map
         application global section for ACMS01EXC001000
vlcrow 20-MAR-2001 11:03:31.87 : msg_proc: i : EXC startup processing
 complete
         for ACMS01EXC001000
vlcrow 20-MAR-2001 11:03:51.28 : sec: w : Operator access attempt by
 user
         vlcrow.zko.dec.com::LT_SUT [305,3] for function ACMSMGMT_STOP
vlcrow 20-MAR-2001 11:03:51.28 : rpc: e : Call to mgmt_shutdown complete
vlcrow 20-MAR-2001 11:03:51.28 : rpc: f : svc_run returned!
vlcrow 20-MAR-2001 11:03:51.48 : procmon: f : Failure waiting on
         mgmt$x_proc_mon_cond_var
vlcrow 20-MAR-2001 11:03:51.50 : mgr: w : Rejected request to stop RPC
         interface when it is already stopped.
vlcrow 20-MAR-2001 11:04:00.12 : log: i : Log opened
vlcrow 20-MAR-2001 11:04:02.08 : procmon: e : Failure obtaining current
         collection states. Bypassingqti
vlcrow 20-MAR-2001 11:04:02.63 : msg_proc: i : Message proc thread
         initializing
vlcrow 20-MAR-2001 11:04:02.63 : msg_proc: i : Message proc thread
 starting
vlcrow 20-MAR-2001 11:04:02.63 : msg_proc: i : Message proc thread
 executing
vlcrow 20-MAR-2001 11:04:03.14 : sec: e : Failure obtaining uaf info for
         ACMS$SNMP
vlcrow 20-MAR-2001 11:04:03.20 : sec: e : %RMS-E-RNF, record not found
vlcrow 20-MAR-2001 11:04:03.20 : sec: e : Account verification failed
 for
         ACMS$SNMP user
vlcrow 20-MAR-2001 11:25:53.20 : msg_proc: i : CP process shutdown
 message
         received for ACMS01CP001000
vlcrow 20-MAR-2001 11:25:53.72 : msg_proc: i : CP process shutdown
 message
         received for ACMS01CP003000
vlcrow 20-MAR-2001 11:25:53.72 : msg_proc: i : CP process shutdown
 message
         received for ACMS01CP002000
vlcrow 20-MAR-2001 11:25:53.72 : msg_proc: i : CP process shutdown
 message
         received for ACMS01CP004000
vlcrow 20-MAR-2001 11:26:07.51 : msg_proc: i : CP process shutdown
 message
         received for ACMS01CP002000
vlcrow 20-MAR-2001 11:26:38.34 : msg_proc: i : TSC process shutdown
 message
         received for ACMS01TSC001000
```

This command displays entries from the Remote Manager log on node VLCROW. Only entries that were logged between 11:00 AM and 12:00 PM on March 20, 2001, are displayed. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

- `$ ACMSMGR SHOW LOG /SINCE=20-MAR /BEFORE=21-MAR`

```
ACMS Remote Management -- Command line utility
ACMS V5.0 Log Display                          Time: 18-APR-2001
 13:59:20.12
Node  Message
local 20-MAR-2001 15:13:23.47 : msg_proc: i : CP process shutdown
 message
        received for ACMS01CP001000
local 20-MAR-2001 15:13:25.57 : msg_proc: i : CP process shutdown
 message
        received for ACMS01CP004000
local 20-MAR-2001 15:13:25.88 : msg_proc: i : CP process shutdown
 message
        received for ACMS01CP002000
local 20-MAR-2001 15:13:41.77 : msg_proc: i : CP process shutdown
 message
        received for ACMS01CP003000
local 20-MAR-2001 15:14:14.16 : msg_proc: i : CP process startup message
        received for ACMS01CP001000
local 20-MAR-2001 15:14:14.87 : msg_proc: i : CP process startup message
        received for ACMS01CP002000
local 20-MAR-2001 16:55:50.92 : log: i : Log opened
local 20-MAR-2001 16:55:53.44 : msg_proc: i : Message proc thread
        initializing
local 20-MAR-2001 16:55:53.44 : msg_proc: i : Message proc thread
 starting
local 20-MAR-2001 16:55:53.44 : msg_proc: i : Message proc thread
 executing
local 20-MAR-2001 16:55:54.49 : sec: f : ACMS$SNMP user has been granted
        no rights.
local 20-MAR-2001 16:56:44.20 : sec: w : User does not hold the proper
        rights identifer -> vlcrow.zko.dec.com::LT_SUT [305,3]
local 20-MAR-2001 16:56:44.20 : rpc: w : Security check failed
local 20-MAR-2001 17:02:35.17 : sec: w : User does not hold the proper
        rights identifer -> vlcrow.zko.dec.com::LT_SUT [305,3]
local 20-MAR-2001 17:02:35.17 : rpc: w : Security check failed
local 20-MAR-2001 17:05:16.46 : sec: w : User does not hold the proper
        rights identifer -> vlcrow.zko.dec.com::LT_SUT [305,3]
local 20-MAR-2001 17:05:16.46 : rpc: w : Security check failed
local 20-MAR-2001 17:05:20.53 : sec: w : User does not hold the proper
        rights identifer -> vlcrow.zko.dec.com::LT_SUT [305,3]
local 20-MAR-2001 17:05:20.53 : rpc: w : Security check failed
local 20-MAR-2001 17:46:22.40 : msg_proc: i : ACC process startup
 message
        received for ACMS01ACC001000
local 20-MAR-2001 17:46:22.42 : msg_proc: e : Failure getting current
        collection states. Ignoring process ACMS01ACC001000
local 20-MAR-2001 17:46:22.85 : procmon: e : Failure obtaining current
        collection states. Bypassingtsc
local 20-MAR-2001 17:46:22.85 : procmon: e : Failure obtaining current
        collection states. Bypassingqti
local 20-MAR-2001 17:46:25.92 : msg_proc: i : TSC process startup
 message
        received for ACMS01TSC001000
```

This command displays entries from the Remote Manager log on node VLCROW. Only entries that were logged between March 20 and 21, 2001, are displayed. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.36. ACMSMGR SHOW MANAGER

ACMSMGR SHOW MANAGER

ACMSMGR SHOW MANAGER — Displays run-time information about a Remote Manager on one or more nodes.

Format

ACMSMGR SHOW MANAGER [/*qualifiers*]

Command Qualifier	Default
/[BRIEF,FULL]	/BRIEF
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/[BRIEF,FULL]

Specifies the format of the data displayed. **/BRIEF** is the default. **/FULL** qualifier displays timer information in addition to the information displayed in the brief display.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

See *Section 9.9, "Manager Status Table"* for a discussion of each field displayed.

Examples

```
$ ACMSMGR SHOW MANAGER
ACMS Remote Management -- Command line utility
ACMS V5.0 Remote Manager Status Table Display Time: 18-APR-2001 13:59:22.76
 Node         Fields
------------- ------------------------------------------------------------
 VLCROW       Collection Count                  0
 VLCROW       Interfaces Count                  2
 VLCROW       Trap Count                        0
 VLCROW       RPC UDP State                     1
 VLCROW       RPC TCP State                     1
 VLCROW       Timer Count                       0
```

This command displays summary information about the Remote Manager on the node specified by the logical name ACMS$MGMT_SERVER_NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.37. ACMSMGR SHOW PARAMETER

ACMSMGR SHOW PARAMETER

ACMSMGR SHOW PARAMETER — Displays Remote Manager configuration parameters for a server on one or more nodes.

Format

ACMSMGR SHOW PARAMETER [/*qualifiers*]

Command Qualifier	Default
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

See *Section 9.10, "Parameter Table"* for a description of each parameter.

Examples

```
$ ACMSMGR SHOW PARAMETER /NODE=VLCROW
ACMS Remote Management -- Command line utility
ACMS V5.0  Parameters Table Display           Time: 18-APR-2001 13:59:24.79
 Node     Parameter               Value    Default   Min     Max
 Units
------------------------------------------------------------------------------
VLCROW    dcl_audit_level          E        E         0                F
 (D)
VLCROW    dcl_mgr_priority         5        5         1               10
VLCROW    dcl_stacksize          300      300         1       2147483647
 k (Vax), 8k (Alpha)
VLCROW    event_log_priority       5        5         1               10
VLCROW    error_interval          10       10         1        863999999
 seconds (D)
VLCROW    log_stacksize          300      300         1       2147483647
 K (Vax), 8k (Alpha)
VLCROW    login_creds_lifetime    60       60         1         14399999
 minutes (D)
VLCROW    max_logins              20       20         1       2147483647
 (D)
VLCROW    max_rpc_return_recs     20       20         1       2147483647
VLCROW    mgr_audit_level          E        E         0                F
 (D)
VLCROW    msg_proc_audit_level     E        E         0                F
 (D)
VLCROW    msg_proc_priority        5        5         1               10
VLCROW    msg_proc_stacksize     300      300         1       2147483647
 k (Vax), 8k (Alpha)
VLCROW    mss_coll_interval       10       10         1        863999999
 seconds (D)
VLCROW    proc_mon_audit_level     E        E         0                F
 (D)
VLCROW    proc_mon_interval        3       30         1         14399999
 seconds (D)
VLCROW    proc_mon_priority        5        5         1               10
VLCROW    proc_mon_stacksize     300      300         1       2147483647
 K (Vax), 8k (Alpha)
VLCROW    proxy_creds_lifetime    60       60         1         14399999
 minutes (D)
VLCROW    rpc_audit_level          E        E         0                F
 (D)
VLCROW    rpc_priority             5        5         1               10
VLCROW    rpc_stacksize           30      300         1       2147483647
 k (Vax), 8k (Alpha)
VLCROW    security_audit_level     E        E         0                F
 (D)
VLCROW    vsnap_audit_level        E        E         0                F
 (D)
VLCROW    vsnap_priority           5        5         1               10
VLCROW    snap_stacksize         300      300         1       2147483647
 k (Vax), 8k (Alpha)
```

```
VLCROW    snmp_agent_time_out    10       10         1        863999999
 seconds
VLCROW    snmp_are_you_there    300      300         2        863999999
 seconds
VLCROW    snmp_audit_level        E        E         0                F
 (D)
VLCROW    snmp_priority           5        5         1               10
VLCROW    snmp_sel_time_out       5        5         1        863999999
 seconds
VLCROW    snmp_stacksize        300      300         1       2147483647
 k (Vax), 8k (Alpha)
VLCROW    tcp_enabled             1        1         0                1
 [0,1] 1=enabled
VLCROW    timer_audit_level       E        E         0                F
 (D)
VLCROW    timer_interval         30       30         1        863999999
 seconds (D)
VLCROW    timer_priority          5        5         1               10
VLCROW    timer_stacksize       300      300         1       2147483647
 k (Vax), 8k (Alpha)
VLCROW    total_entity_slots     20       20         1       2147483647
VLCROW    trace_msg_wait_time     5        5         1         14399999
 seconds (D)
VLCROW    trace_start_wait_time   5        5         1         14399999
 seconds (D)
VLCROW    trap_audit_level        E        E         0                F
 (D)
VLCROW    trap_priority           5        5         1               10
VLCROW    trap_stacksize        300      300         1       2147483647
 k (Vax), 8k (Alpha)
VLCROW    udp_enabled             1        1         0                1
 [0,1] 1=enabled
VLCROW    vms_coll_interval      10       10         0        863999999
 seconds (D)
VLCROW    wksp_coll_interval     10       10         1        863999999
 seconds (D)
VLCROW    max_agents              2        2         1       2147483647
```

This command displays data from the Remote Manager Parameter table on node VLCROW. Authorization is performed on the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.38. ACMSMGR SHOW PROCESS

ACMSMGR SHOW PROCESS

ACMSMGR SHOW PROCESS — Displays summary data collection information for processes on one or more nodes.

Format

ACMSMGR SHOW PROCESS [/*qualifiers*]

Command Qualifier	Default
/ALL	Displays only currently running processes.

Command Qualifier	Default
/[BRIEF,FULL]	Controls the format of data displayed.
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/ALL

This qualifier displays all available application data, even data for applications that may no longer be running. When applications are stopped, the table row they were occupying is marked for reuse. If the row has not been reused, the data remains available for display. This qualifier allows that data to be displayed. Inactive rows are flagged with an asterisk (*) in the output.

To inhibit the display of old data, do not specify this qualifier. By default, only data for currently running processes is displayed.

/[BRIEF,FULL]

Specifies the format of the data displayed. **/BRIEF** is the default and displays data in tabular format.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays the current data-collection states for all process entry slots. Both active and inactive process data is displayed. Inactive data is flagged with an asterisk (*) in the first column. The following fields are displayed:

- Server Node: The node from which the information was fetched.
- Entity Type: The ACMS entity type to which the data belongs.
- PID: The OpenVMS PID of the process.
- Process Name: The process name for the process.
- Collection States: The current collection state of the process for each class (Id, Cfg, RT and POOL). Collection states are displayed as binary values.
  - 0 = Collection is currently disabled.
  - 1 = Collection is currently enabled.

Previously, the **SHOW PROCESS/BRIEF** and **SHOW PROCESS/FULL** commands displayed all collection states for entities even when some were not applicable. You can enable/disable collection states for all classes and all entities. However, some combinations have no effect. These states are now shown in the **SHOW PROCESS/BRIEF** display as a "-", and as "N/A" for the **SHOW PROCESS/FULL** command. The display for the **SHOW PROCESS** command in the WEB display has also been modified.

For Task Groups, the following classes are not applicable: CONFIG, RUNTIME, ERROR. For Servers, the following classes are not applicable: POOL, ERROR.

Examples

```
$ ACMSMGR SHOW PROCESS
ACMS Remote Management -- Command line utility
ACMS V5.0  Process Table Display              Time: 18-APR-2001
 13:59:26.77
Server Entity          Process Name -or- Application.          Collection
 States
```

```
Node   Type    PID     [server_name, task_group_name]          ID   Cfg  RT
   Pool Err
------ ------ -------- ------------------------------------  ---- ----
 ---- ---- ----
VLCROW acc    37C0024F ACMS01ACC001000                        1    1    1
   1    1
VLCROW tsc    37C00251 ACMS01TSC001000                        1    1    1
   1    1
VLCROW cp     37C00252 ACMS01CP001000                         1    1    1
   1    1
VLCROW cp     37C00253 ACMS01CP002000                         1    1    1
   1    1
VLCROW cp     37C00254 ACMS01CP003000                         1    1    1
   1    1
VLCROW cp     37C00255 ACMS01CP004000                         1    1    1
   1    1
VLCROW exc    37C0025A ACMS01EXC001000                        1    1    1
   1    1
VLCROW          server LDT_APPL_A.TESTSRV01                   1    1    1
   1    1
VLCROW           group LDT_APPL_A.TEST_GRP01                  1    1    0
   1    1
```

This command displays Collection state information on a per-process basis from the node specified by the logical name ACMS$MGMT_SERVER_ NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.39. ACMSMGR SHOW QTI

ACMSMGR SHOW QTI

ACMSMGR SHOW QTI — Displays information about QTIs on one or more remote nodes.

Format

ACMSMGR SHOW QTI [/*qualifiers*]

Command Qualifier	Default
/ACTIVE	See Notes.
/ALL	Current process data only.
/[BRIEF,FULL]	/FULL if no class /CLASS=*keyword*qualifier (/CONFIG, /ID, /POOL, or /RUNTIME) is specified. Otherwise, /BRIEF.
/[*class-name*]	* (all)
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/STORED	See Notes.

Command Qualifier	Default
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/ACTIVE

When specified with the **/BRIEF** qualifier, this qualifier causes active ACMSGEN field values to be displayed. **/ACTIVE** is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/ALL

This qualifier displays all available QTI data, including data for processes that are no longer running. When QTI processes are stopped, the QTI table row they were occupying is marked for reuse. If the row has not been reused, the data remains available for display. This qualifier allows that data to be displayed. Inactive rows are flagged with an asterisk (*) in the output.

To inhibit the display of old data, do not specify this qualifier. By default, only data for currently running processes is displayed.

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. If no class qualifier (**/CONFIG**, **/ERROR**, **/ID**, **/POOL**, or **/RUNTIME**) is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**). Available OpenVMS process quota and SYSGEN parameter information is only displayed when **/FULL** is specified.

/[*class-name*]

This qualifier causes data for only the specified class to be displayed. If this qualifier is omitted, the default is to display information for all classes.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/STORED

When specified with the **/BRIEF** qualifier, this qualifier causes field values from the ACMSGEN file (not those active in memory) to be displayed. The **/STORED** qualifier is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays information about the QTIs on the target nodes. The **/BRIEF** and **/FULL** qualifiers control the format of information to be displayed.

See *Section 9.11, "QTI Table"* for a discussion of each field displayed.

Note that some information may not be current, depending on whether the class to which the data belongs has been enabled for the QTI. The Collect State field indicates whether or not information is being collected for that class.

Examples

```
$ ACMSMGR SHOW QTI /NODE=KAZONS /RUNTIME
ACMS Remote Management -- Command line utility
ACMS V5.0  QTI  Table Display                    Time: 18-APR-2001
 13:41:11.09
       Runtime  Started Current   Task    Tasks   Tasks    Current  Errors
Node   Class    Queues  Tasks   Retries  Success  Failed   Submitrs Queued
------ -------- ------- ------- -------- -------- -------- --------
 ---------
kazons enabled  3       17      4361     14859    5783     5        1329
```

This command displays summary run-time information for the QTI on node KAZONS. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.40. ACMSMGR SHOW SERVER

ACMSMGR SHOW SERVER

ACMSMGR SHOW SERVER — Displays information about one or more ACMS application server types on one or more nodes.

Format

ACMSMGR SHOW SERVER [/*qualifiers*]

Command Qualifier	Default
/APPL=*application-name*	* (all applications)
/[BRIEF,FULL]	/FULL if no class qualifier (/CONFIG, /ID, or /RUNTIME) is specified. Otherwise, /BRIEF.
/[*class-name*]	* (all applications)
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/CLASS=*keyword*	* (all)
/SERVER=*server-name*	* (all servers)
/USER=*user-name*	Translation of logical ACMS$MGMT_USER.

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/APPL=*application-name*

Use this qualifer to specify a particular ACMS application to display. Wildcard matching is performed on the name provided; use of asterisks (*) is allowed.

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. If no class qualifier (**/CONFIG**, **/ERROR**, **/ID**, or **/RUNTIME**) is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**).

/[*class-name*]

This qualifier causes data for only the specified class to be displayed. If this qualifier is omitted, the default is to display information for all classes.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/SERVER=*server-name*

This qualifier specifies a particular ACMS application server to display. Wildcard matching is performed on the name provided; use of asterisks (*) is allowed.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays information about ACMS application servers on the target node. The **/BRIEF** and **/FULL** qualifiers control the format of information to be displayed.

See *Section 9.12, "Server Table"* for a discussion of each field displayed.

Note that some information may not be current, depending on whether the class to which the data belongs has been enabled for the server. The Collect State field indicates whether or not information is being collected for that class.

Examples

```
$ ACMSMGR SHOW SERVER /NODE=VLCROW /RUNTIME
ACMS Remote Management -- Command line utility
ACMS V5.0  SER Table Display                    Time: 18-APR-2001
 13:59:33.60
       Runtime Application                      Current Waiting Server
 Server
Node   Class   Name        Server Name          Servers Tasks   Starts
 Failures
------ ------- ----------- -------------------- ------- -----   ------
 --------
VLCROW enabled LDT_APPL_A  TESTSRV01            2       0       2      0
VLCROW enabled LDT_APPL_A  TESTSRV1D            0       0       3419   0
VLCROW enabled LDT_APPL_A  TESTSRV2D            1       0       19     0
VLCROW enabled LDT_APPL_A  TESTSRV3D            1       0       1      0
VLCROW enabled LDT_APPL_A  TESTSRV05            0       0       0      0
VLCROW enabled LDT_APPL_A  TLOAD001S            4       0       6      0
VLCROW enabled LDT_APPL_A  TLOAD002S            1       0       1      0
VLCROW enabled LDT_APPL_A  DBMSLSRV1            1       0       1      0
VLCROW enabled LDT_APPL_A  DBMSLSRV2            1       0       1      0
VLCROW enabled LDT_APPL_A  RDBLSRV1             1       0       44     0
VLCROW enabled LDT_APPL_A  RMSRSERVR            1       0       33     0
VLCROW enabled LDT_APPL_A  TCT_LDT_PROC_SERVER  1       0       1      0
VLCROW enabled LDT_APPL_A  TCT_LDT_CHAIN_SERVER 1       0       1      0
VLCROW enabled LDT_APPL_A  TCT_LDT_DCL_SERVER   0       0       0      0
VLCROW enabled LDT_APPL_A  RI_DCL_SERVER        2       0       9      0
VLCROW enabled LDT_APPL_A  RI_V3016_FMS_SERVER  1       0       1      0
VLCROW enabled LDT_APPL_A  RI_V3016_RI_SERVER   1       0       1      0
VLCROW enabled LDT_APPL_A  VF_V3111_SERVER      0       0       0      0
VLCROW enabled LDT_APPL_A  LDT_CS_V3111_SERVER  3       0       3      0
VLCROW enabled LDT_APPL_A  TESTV32_RMS_SERVER   8       0       28     0
VLCROW enabled LDT_APPL_A  TESTV32_RDB_SERVER   8       0       375    0
VLCROW enabled LDT_APPL_A  TESTV32_DBMS_SERVER  8       0       47     0
VLCROW enabled LDT_APPL_A  TESTV32_SQL_SERVER   8       0       405    0
VLCROW enabled LDT_APPL_A  TESTV32_RM_SERVER    4       0       4      0
VLCROW enabled LDT_APPL_A  GEN_INPUT_SERVER     4       0       24     0
VLCROW enabled LDT_APPL_A  NOOP_SERVER          4       0       4      0
VLCROW enabled LDT_APPL_A  UNUSED_VF_V32        0       0       0      0
VLCROW enabled LDT_APPL_A  V_SERVER_W_DCL       1       0       1      0
VLCROW enabled LDT_APPL_A  I_SERVER_W_DCL       1       0       1      0
VLCROW enabled LDT_APPL_A  DETASK_SERVER        2       0       3      0
```

This command displays summary run-time information for all servers on node VLCROW. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.41. ACMSMGR SHOW TRAP

ACMSMGR SHOW TRAP

ACMSMGR SHOW TRAP — Displays SNMP trap configurations for one or more nodes.

Format

ACMSMGR SHOW TRAP [/*qualifiers*]

Command Qualifier	Default
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays Remote Manager SNMP trap configuration information. SNMP traps are generated only if the SNMP interface is started. Changes to this table take effect immediately after they are processed.

See *Section 9.14, "Trap Table"* for a description of each field displayed.

Examples

```
$ ACMSMGR SHOW TRAP
ACMS Remote Management -- Command line utility
ACMS V5.0  Trap Table Display                  Time: 18-APR-2001
 13:59:38.69
Node   Entity  Entity Name Parameter Min Max Sev Alarms Alarm Time
------ ------- ----------- --------- --- --- --- ------
 -----------------------
VLCROW *       *           exists    1   -1  I   0      17-NOV-1858
 00:00:00.00
```

This command displays SNMP traps that have been configured on the node specified by the logical name ACMS$MGMT_SERVER_NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined.

11.42. ACMSMGR SHOW TSC

ACMSMGR SHOW TSC

ACMSMGR SHOW TSC — Displays information about TSCs on one or more remote nodes.

Format

ACMSMGR SHOW TSC [/*qualifiers*]

Command Qualifier	Default
/ACTIVE	See Notes.
/[BRIEF,FULL]	/FULL if no class qualifier (/CONFIG, /ID, or /POOL) is specified. Otherwise, /BRIEF.
/[*class-name*]	* (all)
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/STORED	See Notes.
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/ACTIVE

When specified with the **/BRIEF** qualifier, this qualifier causes active ACMSGEN field values to be displayed. **/ACTIVE** is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/ALL

This qualifier displays all available TSC data, including data for processes that are no longer running. When TSC processes are stopped, the TSC table row they were occupying is marked for reuse. If the row has not been reused, the data remains available for display. This qualifier allows that data to be displayed. Inactive rows are flagged with an asterisk (*) in the output.

To inhibit the display of old data, do not specify this qualifier. By default, only data for currently running processes is displayed.

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed. If no class qualifier (**/CONFIG**, **/ERROR**, **/ID**, **/POOL**, or **/RUNTIME**) is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**). Available OpenVMS process quota and SYSGEN parameter information is only displayed when **/FULL** is specified.

/[*class-name*]

This qualifier causes data for only the specified class to be displayed. If this qualifier is omitted, the default is to display information for all classes.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/STORED

When specified with the **/BRIEF** qualifier, this qualifier causes field values from the ACMSGEN file (not those active in memory) to be displayed. The **/STORED** qualifier is effective only when used with the **/CONFIG** qualifier. If **/BRIEF** is not specified, or if **/FULL** is specified, this qualifier has no effect (both active and stored values are displayed).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays information about the TSCs on the target nodes. The **/BRIEF** and **/FULL** qualifiers control the format of information to be displayed.

/ACTIVE is the default when the CONFIG class is being displayed.

See *Section 9.15, "TSC Table"* for a discussion of each field displayed.

Note that some information may not be current, depending on whether the class to which the data belongs has been enabled for the TSC. The Collect State field indicates whether or not information is being collected for that class.

Examples

- ```
  $ ACMSMGR SHOW TSC /NODE=VLCROW /CONFIG
  ACMS Remote Management -- Command line utility
  ACMS V5.0  TSC Table Display                    Time: 18-APR-2001
   13:59:43.00
               A  Config    Run                  CP    Max    Max TTS Perm
    Min
     Node      S   Class    State    User Name   Slots Logins Per CP  CPs
    CPIs
  ------------ -- -------- -------- ------------ ----- ------ -------
   ----- -----
  VLCROW       A  enabled  started  LT$TSC_V31   20    400    20      4
    0
  ```

  This command displays summary Configuration class information for the TSC on node VLCROW. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or
  ```

by proxy if the logical is not defined. Since the **/FULL** qualifier was not supplied, only summary information is displayed.

- ```
  $ ACMSMGR SHOW TSC /NODE=VLCROW
  ACMS Remote Management -- Command line utility
  ACMS V5.0  TSC Table Display                    Time: 18-APR-2001
   13:59:40.62
  ================================================================================
  Node   IDENTIFICATION
  ------
   ----------------------------------------------------------------
  VLCROW ID Class Collection State          enabled
  Process Name                              ACMS01TSC001000
  PID                                       37C00251
  User Name                                 LT$TSC_V31
  Start Time                                18-APR-2001 14:49:18.98
  End Time                                  (null)

  Node   CONFIGURATION                      Active          Stored
  ------ ---------------------------------- --------------- -------------
  VLCROW Config Class Collection State      enabled
  TSC Running State                         started
  TSC Username                              LT$TSC_V31      LT$TSC_V31
  TSC Base Priority                         4               4
  CP Username                               LT$CP_V40       LT$CP_V40
  CP Base Priority                          4               4
  CP Slots                                  20              20
  Max Logins (D)                            400             400
  Max TTS per CP (D)                        20              20
  Permanent CPs (D)                         4               4
  Min CP slots (D)                          0               0
  Working Set Default                       65008           65001
  Working Set Extent                        322992          90000
  Working Set Quota                         65536           65536
  AST Limit                                 1999            1999
  Byte Limit                                1775409         1777777
  Direct I/O Limit                          15000           15000
  Buffered I/O Limit                        10000           10000
  Enqueue Limit                             10000           10000
  Page File Quota                           500000          500000
  Timer Queue Limit                         500             500

  Node   RUNTIME
  ------ ------------------------------------------------------------------
  VLCROW Runtime Class Collection State     enabled
  Gauges                             Current   Max     Limit   Max Time
  ---------------------------------- --------- ------- -------
   ------------------------
  Logins                                2         4       60    18-APR-2001
   15:49:55.13
  CP Slots Used                         4         4        4    18-APR-2001
   18:21:19.78
  Terminals per CP (avg)                1         1       20    14-APR-2001
   15:50:19.34

  Process Quotas                 Current    Max      Limit   Max Time
  ------------------------------ ---------- -------- ------- -------
   ------------------------
  ```
  ```

```
Working Set Size               65008      65008    322992  18-APR-2001
 18:21:19.78
AST Limit                      4 ( 0%)    5 ( 0%)  1999    11-APR-2001
 15:50:19.34
Byte Limit                     0 ( 0%)    0 ( 0%) 1775409  18-APR-2001
 18:21:19.78
Direct I/O Limit               0 ( 0%)    0 ( 0%)   15000  18-APR-2001
 18:21:19.78
Buffered I/O Limit             2 ( 0%)    2 ( 0%)   10000  18-APR-2001
 18:21:19.78
Enqueue Limit                  0 ( 0%)    0 ( 0%)   10000  18-APR-2001
 18:21:19.78
File Limit                     1 ( 0%)    1 ( 0%)    1001  18-APR-2001
 18:21:19.78
Page File Quota             6704 ( 1%) 6704 ( 1%)  500000  18-APR-2001
 18:21:19.78
Timer Queue Limit              3 ( 0%)    3 ( 0%)     500  11-APR-2001
 09:14:55.49
Channel Count                 15         15

Node   POOL
------ ----------------------------------------------------------------
VLCROW Pool Class Collection State        enabled

       Process Pool                                Pct   Time
       ------------------------------------------ -----
 ----------------------
       Pool Size                           262144
       Current Free (bytes)                259760 (99%)
       Minimum Free (bytes)                259760 (99%) 19-APR-2001
 13:59:32.98
       Largest Current Free Block (bytes) 65536
       Minimum Largest Free Block (bytes) 65536        19-APR-2001
 13:59:32.98
       Allocation Failures                 0
       Garbage Collections                 0

Node   ERROR
------ ----------------------------------------------------------------
VLCROW Error Class Collection State       enabled
       Error Count                         0
       Last Error Message                  0
       Time of Last Error                  (null)
```

This command displays all Configuration class information for the TSC on node VLCROW. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined. Since neither the **/FULL** nor the **/BRIEF** qualifier was supplied, and no specific class was selected, all TSC information is displayed.

11.43. ACMSMGR SHOW USER

ACMSMGR SHOW USER

ACMSMGR SHOW USER — Displays Remote Manager user information for a server on one or more nodes. Information about both proxy and nonproxy users is displayed.

Format

ACMSMGR SHOW USER [/*qualifiers*]

Command Qualifier	Default
/[BRIEF,FULL]	/BRIEF
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ

Parameters

None.

Qualifiers

/[BRIEF,FULL]

This qualifier causes detailed information about each user to be displayed. When the qualifier is omitted, only summary information is displayed.

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command displays information about each user currently logged in to the Remote Manager server. The following fields are displayed:

- Server Node: The node from which the information was obtained.
- Client Id: A unique identifier for each client.
- Username: The name of the local (server node) account to which the user is logged in. This is the user name that is being used for authorization.
- Proxy: A flag indicating whether the user is logged in using an ACMS proxy. A value of zero (0) indicates that the login is not proxy based; a value of 1 indicates that the login is proxy based.
- Login Node: The node from which the client logged in. This node may not be the same as the server node.

If the **/FULL** qualifier is specified, the following additional information is displayed:

- Credentials expiration: The date and time at which the user's credentials will expire.
- UIC: The UIC of the account from which the login was initiated.
- Proxy UIC: The UIC of the account on the server node that is used for authorization.

Examples

- ```
  $ ACMSMGR SHOW USERS
  ACMS Remote Management -- Command line utility
  ACMS V5.0  User Table Display                  Time: 18-APR-2001
   13:59:45.09
   Server      Client   Login
   Node        Id       Username     Proxy Node
  ------------ -------- ------------ ----- ------------------------------
   gocrow      16       LT_SUT       1     gocrow.zko.dec.com
   gocrow      17       LT_SUT       1     vlcrow.zko.dec.com
   vlcrow      16       LT_SUT       1     vlcrow.zko.dec.com
   vlcrow      20       LT_SUT       1     vlcrow.zko.dec.com
  ```

  This command displays summary information about users who have logged in to Remote Manager on the node specified by the logical name ACMS$MGMT_ SERVER_NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined. In this example, all users logged in using proxy (Proxy = 1).
  ```

- ```
  $ ACMSMGR SHOW USERS /FULL /NODE=VLCROW
  ACMS Remote Management -- Command line utility
  ACMS V5.0  User Table Display                  Time: 18-APR-2001
   13:59:47.07
   Node         User Information
  ------------ ------------------------------------------------------
   VLCROW       Client id              16
                Username               LT_SUT
                Login Node             vlcrow.zko.dec.com
                Credentials expiration 18-APR-2002 13:59:47.07
                Proxy Flag             1
                UIC                    [208,40]
                Proxy UIC              [197,3]

   Node         User Information
  ------------ ------------------------------------------------------
   VLCROW       Client id              18
                Username               LT_SUT
                Login Node             vlcrow.zko.dec.com
                Credentials expiration 18-APR-2001 14:44:01.02
                Proxy Flag             1
                UIC                    [197,3]
                Proxy UIC              [197,3]
  ```

  This command displays all information about users who have logged in to Remote Manager on the node specified by the logical name ACMS$MGMT_ SERVER_NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or by proxy if the logical is not defined. In this example, both users logged in using proxy (Proxy = 1).

11.44. ACMSMGR SHOW VERSION

ACMSMGR SHOW VERSION

ACMSMGR SHOW VERSION — Displays the current version of ACMSMGR and its related software components. This command (and its qualifiers) is only for use with systems running ACMS Version 4.4 or higher.

Format

ACMSMGR SHOW VERSION [/*qualifiers*]

Command Qualifier	Default
/INTERVAL=*interval*	Command is executed once.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/OUT=*file-name*	None
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_READ
  ```

Parameters

None.

Qualifiers

/INTERVAL=*interval*

This qualifier causes the command to be reissued automatically at a specified interval (in seconds). Use either **Ctrl/C** or **Ctrl/Y** to interrupt the command. If this qualifier is not specified, the command is executed only once.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Examples

```
$ ACMSMGR SHOW VERSION /NODE=GOCROW
ACMS Remote Management -- Command line utility
ACMS V5.0  ACMS Version Display         Time: 18-APR-2001 13:59:47.07
 Node Version Information
------------- ---------------------------------------------------
 local        ACMSMGR Version                ACMS V5.0
 GOCROW       ACMS Version                   V5.0
              MGMT Header Version            2
              MGMT EXC Header Version        2
              MGMT Config File Version       2
```

This command displays the current version of the ACMSMGR installed locally as well as the location and version of the related software components.

11.45. ACMSMGR START EXC

ACMSMGR START EXC

ACMSMGR START EXC — Starts an ACMS application on one or more remote nodes.

Format

ACMSMGR START EXC [/*qualifiers*]

Command Qualifier	Default
/APPLICATION=*application-name*	Qualifier is required.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/APPLICATION=*application-name*

This required qualifier specifies a particular ACMS application to start. The full application name must be specified.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/START APPL**. This command is executed synchronously. If the command completes successfully, the application has been started on the target nodes. If not, error messages are displayed.

Examples

```
$ ACMSMGR START EXC/APPL=VR_APPL/NODE=SPARKS
```

This command starts the VR_APPL application on node SPARKS.

11.46. ACMSMGR START QTI

ACMSMGR START QTI

ACMSMGR START QTI — Starts an ACMS Queued Task Initiator (QTI) on one or more remote nodes.

Format

ACMSMGR START QTI [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/START QTI**. This command is executed synchronously. If the command completes successfully, the application has been started on the target nodes. If not, error messages are displayed.

Examples

```
$ ACMSMGR START QTI/NODE=SPARKS
```

This command starts the QTI on node SPARKS.

11.47. ACMSMGR START SYSTEM

ACMSMGR START SYSTEM

ACMSMGR START SYSTEM — Starts an ACMS run-time system on one or more remote nodes.

Format

ACMSMGR START SYSTEM [/*qualifiers*]

Command Qualifier	Default
/NOAUDIT	Start system with auditing enabled.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/QTI	Start system without QTI running.
/NOTERMINALS	Start system with TSC and CPs running.
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/NOAUDIT

This qualifier starts the system with auditing disabled. If this qualifier is not specified, the system is started with auditing enabled.

There is no **/AUDIT** qualifier to start ACMS with auditing enabled. Rather as the notes below describe, just start ACMS without the **/NOAUDIT** qualifier in order for ACMS to start with auditing enabled.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/QTI

This qualifier starts the system with QTI running. If this qualifier is not specified, the system is started with QTI in stopped state.

There is no **/NOQTI** qualifier to start ACMS with QTI stopped. Rather as the notes below describe, just start ACMS without the **/QTI** qualifier in order for ACMS to start with QTI stopped.

/NOTERMINALS

This qualifier starts the system without the TSC and CPs running. If this qualifier is not specified, the system is started with the TSC and CPs running.

There is no **/TERMINALS** qualifier to start ACMS with the TSC and CPs running. Rather as the notes below describe, just start ACMS without the **/NOTERMINALS** qualifier in order for ACMS to start without the TSC and CPs running.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/START SYS**. This command is executed synchronously. If the command completes successfully, the system has been started on the target nodes. If not, error messages are displayed.

If no qualifiers are specified with this command, the equivalent ACMSOPER command is **ACMS/START SYS/NOQTI/TERMINALS/AUDIT**. In other words, the default values for the ACMSMGR **START SYS** command are **/AUDIT**, **/TERMINALS**, and **/NOQTI**. If you want to override a default you may use **/NOAUDIT**, **/NOTERMINALS**, or **/QTI** however you may not specify any of the defaults directly. You may only select them by leaving out the qualifier and taking the default.

Examples

```
$ ACMSMGR START SYSTEM/NODE=SPARKS/NOTERMINALS
```

This command starts the ACMS run-time system on node SPARKS, without the QTI, TSC, or CPs running.

11.48. ACMSMGR START TERMINALS

ACMSMGR START TERMINALS

ACMSMGR START TERMINALS — Starts an ACMS TSC and any associated CPs on one or more remote nodes.

Format

ACMSMGR START TERMINALS [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/START TERMINALS**. This command is executed synchronously. If the command completes successfully, the application has been started on the target nodes. If not, error messages are displayed.

Examples

```
$ ACMSMGR START TERMINALS/NODE=SPARKS
```

This command starts the TSC on node SPARKS.

11.49. ACMSMGR START TRACE_MONITOR

ACMSMGR START TRACE_MONITOR

ACMSMGR START TRACE_MONITOR — This command requests the Remote Manager on the target nodes to start the ACMS$TRACE_MON process. The ACMS$TRACE_MON process is an intermediate process used by the Remote Manager to communicate with ACMS run-time processes to enable and disable collections.

Format

ACMSMGR START TRACE_MONITOR [/*qualifiers*]

Command Qualifier	Default

/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command requests the Remote Manager to start the ACMS$TRACE_MON process on the target node. The ACMS$TRACE_MON process is an intermediate process used by the Remote Manager to communicate with ACMS run-time processes to enable and disable collections.

In general, external entities do not require a startup or shutdown request of the trace monitor process. The Remote Manager starts the trace monitor during process initialization and stops it during process shutdown. Additionally, the Remote Manager starts the trace monitor anytime it is needed if it is not already started. Once started, the trace monitor continues running until the Remote Manager shuts down.

After issuing the start command to the trace monitor, the Remote Manager waits for a period of up to `trace_start_wait_time` (a Parameter table parameter that is dynamic and expressed in seconds). If the trace monitor fails to start during that period, the ACMSMGR command returns an error.

Examples

```
$ ACMSMGR START TRACE_MONITOR
```

This command starts the ACMS$TRACE_MON process on the node specified by the logical name ACMS$MGMT_SERVER_NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or based on an ACMS proxy on the target node if the logical is not defined. If the process is successfully started, no messages are displayed.

11.50. ACMSMGR STOP EXC

ACMSMGR STOP EXC

ACMSMGR STOP EXC — Stops an ACMS application on one or more remote nodes.

Format

ACMSMGR STOP EXC [/*qualifiers*]

Command Qualifier	Default
/APPLICATION=*application-name*	Qualifier is required.
/CANCEL	Wait for executing tasks to complete.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE.
/USER=*user-name*	Translation of logical ACMS$MGMT_USER.

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/APPLICATION=*application-name*

This required qualifier specifies a particular ACMS application to start. The entire application name must be specified.

/CANCEL

This qualifier stops the application without waiting for currently executing tasks to complete. If this qualifier is omitted, any tasks currently executing are allowed to complete before the application is stopped.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/STOP APPL**. This command is executed synchronously. If the command completes successfully, the application has been stopped on the target nodes. If not, error messages are displayed.

Examples

```
$ ACMSMGR STOP EXC/APPL=VR_APPL/NODE=SPARKS
```

This command stops the VR_APPL application on node SPARKS.

11.51. ACMSMGR STOP MANAGER

ACMSMGR STOP MANAGER

ACMSMGR STOP MANAGER — Stops the Remote Manager on the target nodes.

Format

ACMSMGR STOP MANAGER [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command requests an orderly shutdown of the Remote Manager process. This command may take several minutes to complete if any of the interfaces is in a noninterruptible state when the command is issued.

If the command fails to complete successfully, an alternative means of stopping the Remote Manager is to use the DCL command **STOP/ID**.

The Remote Manager can be restarted only by logging in to the target node and running the ACMS$MGMT_STARTUP command procedure.

Examples

```
$ ACMSMGR STOP MANAGER
```

This command stops the Remote Manager on the node specified by the logical name ACMS$MGMT_SERVER_NODE. Authorization is performed for the user specified by the logical ACMS$MGMT_USER, or is based on an ACMS proxy on the target node if the logical is not defined. If the server is successfully stopped, no messages are displayed.

11.52. ACMSMGR STOP QTI

ACMSMGR STOP QTI

ACMSMGR STOP QTI — Stops an ACMS Queued Task Initiator (QTI) on one or more remote nodes.

Format

ACMSMGR STOP QTI [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/STOP QTI**. This command is executed synchronously. If the command completes successfully, the QTI has been stopeed on the target nodes. If not, error messages are displayed.

Examples

```
$ ACMSMGR STOP QTI/NODE=SPARKS
```

This command stops the QTI on node SPARKS.

11.53. ACMSMGR STOP SYSTEM

ACMSMGR STOP SYSTEM

ACMSMGR STOP SYSTEM — Stops an ACMS run-time system on one or more remote nodes.

Format

ACMSMGR STOP SYSTEM [/*qualifiers*]

Command Qualifier	Default
/CANCEL	Wait for executing tasks to complete.
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/CANCEL

This qualifier stops the ACMS run-time system without waiting for currently executing tasks to complete. If not specified, any tasks currently executing are allowed to complete before the application is stopped.

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/STOP SYSTEM**. This command is executed synchronously. If the command completes successfully, the application has been stopped on the target nodes. If not, error messages are displayed.

Examples

```
$ ACMSMGR STOP SYS/NODE=SPARKS/CANCEL
```

This command stops the ACMS run-time system on node SPARKS. All currently executing tasks, servers, and users are canceled.

11.54. ACMSMGR STOP TERMINALS

ACMSMGR STOP TERMINALS

ACMSMGR STOP TERMINALS — Stops the TSC and any related CPs on one or more remote nodes.

Format

ACMSMGR STOP TERMINALS [/*qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command is equivalent to the ACMSOPER command **ACMS/STOP TERMINALS**. This command is executed synchronously. If the command completes successfully, the terminal subsystem has been stopped on the target nodes. If not, error messages are displayed.

Examples

```
$ ACMSMGR STOP TERMINALS/NODE=SPARKS/CANCEL
```

This command stops the ACMS terminal subsystem on node SPARKS.

11.55. ACMSMGR STOP TRACE_MONITOR

ACMSMGR STOP TRACE_MONITOR

ACMSMGR STOP TRACE_MONITOR — Stops the ACMS$TRACE_MON process on the target node.

Format

ACMSMGR STOP TRACE_MONITOR [*/qualifiers*]

Command Qualifier	Default
/NODE=*node-name*	Translation of logical ACMS$MGMT_SERVER_NODE
/USER=*user-name*	Translation of logical ACMS$MGMT_USER

Privileges Required

ACMS$MGMT_OPER

Parameters

None.

Qualifiers

/NODE=*node-name*

This qualifier specifies a fully- or partially-qualified TCP/IP host name. This name must match the current DECnet host name. IP addresses and host names (or aliases) that exceed six characters or include mixed case are not allowed.

If the **/NODE** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_SERVER_NODE. If the logical is defined, the value of the logical is used by default.

In order for the command to execute, either the **/NODE** qualifier must be provided on the command line, or the ACMS$MGMT_SERVER_NODE logical must be defined.

/USER=*user-name*

This qualifier specifies the name of the OpenVMS account on the server node to be used for authorization. If this qualifier is specified, an explicit login must already have been completed successfully (see *Section 11.10, "ACMSMGR LOGOUT"*).

If the **/USER** qualifier is not specified, the ACMSMGR utility checks for the presence of the logical name ACMS$MGMT_USER. If the logical is defined, the value of the logical is used by default.

If the **/USER** qualifier is not specified and the ACMS$MGMT_USER logical is not defined, the ACMSMGR utility attempts proxy access. See *Section 4.4.1.2, "Proxy Accounts"* for a discussion of proxy access.

Notes

This command requests the Remote Manager to stop the ACMS$TRACE_MON process on the target node. The ACMS$TRACE_MON process is an intermediate process used by the Remote Manager to communicate with ACMS run-time processes to enable and disable collections.

In general, external entities do not require a startup or shutdown request of the trace monitor process. The Remote Manager starts the trace monitor during process initialization and stops it during process shutdown. Additionally, the Remote Manager starts the trace monitor anytime it is needed if it is not already started. Once started, the trace monitor continues running until the Remote Manager shuts down.

After issuing the **STOP** command to the trace monitor, the Remote Manager waits for a period of up to `trace_start_wait_time` (a Parameter table parameter that is dynamic and expressed in seconds). If the trace monitor fails to stop during that period, the ACMSMGR command returns an error.

Examples

```
$ ACMSMGR STOP TRACE_MONITOR
```

This command stops the ACMS$TRACE_MON process on the node specified by the logical name ACMS$MGMT_SERVER_NODE. Authorization is performed for the user specified by the logical name ACMS$MGMT_USER, or based on an ACMS proxy on the target node if the logical is not defined. If the process is successfully stopped, no messages are displayed.

Chapter 12. ACMSSNAP Commands

This chapter provides reference information about ACMSSNAP utility commands. Note that this utility and its commands are designed to run locally on a Remote Manager Version 4.4 or higher system.

12.1. ACMSSNAP Overview

The ACMSSNAP utility is used to display information collected in an ACMS Remote Manager data snapshot file. ACMSSNAP runs locally and requires local access to data snapshot files. This means that although a data snapshot file may have been created on a remote node, that file must be directly accessible to the local process running the ACMSSNAP utility.

The ACMSSNAP utility only displays information from the data snapshot file. It does not modify the data in the file or make modifications to the ACMS run-time system.

12.1.1. Command Format

The format for ACMSSNAP commands is as follows:

```
ACMSSNAP> verb object qualifiers
```

The following verbs are supported:

- **CLOSE**
- **EXIT**
- **HELP**
- **NEXT**
- **OPEN**
- **PREV**
- **QUIT**
- **RESET**
- **SHOW**
- **TRACE**

Most verbs have associated objects and qualifiers. The following sections list the objects and any qualifiers for each ACMSSNAP command.

12.1.2. Command Objects and Qualifiers

The objects and qualifiers for the ACMSSNAP commands are summarized in the table below.

Table 12.1. ACMSSNAP Command Objects and Qualifiers

Objects	Qualifiers
CLOSE Command	
None	None
EXIT Command	
None	None
HELP Command	
None	None
NEXT Command	
[*number*]	None
OPEN Command	
[*file-name*]	/AT, /CP_SLOTS, /MAX_AGENTS, /MAX_APPL, /SUMMARY
PREV Command	
[*number*]	None
QUIT Command	
None	None
RESET Command	
None	/ALL
SHOW Command	
ACC	/AT, /BRIEF, /CONFIG, /FULL, /ID, /NEXT, /OUT, /POOL, /PREV, /RUNTIME
AGENT	/AT, /BRIEF, /CONFIG, /FULL, /ID, /NEXT, /OUT, /POOL, /PREV, /RUNTIME
CP	/AT, /BRIEF, /CONFIG, /FULL, /ID, /NEXT, /OUT, /POOL, /PREV, /RUNTIME
EXC	/APPL, /AT, /BRIEF, /CONFIG, /FULL, /ID, /NEXT, /OUT, /POOL, /PREV, /RUNTIME
GROUP	/APPL, /AT, /BRIEF, /CONFIG, /FULL, /ID, /NEXT, /OUT, /POOL, /PREV, /RUNTIME
QTI	/AT, /BRIEF, /CONFIG, /FULL, /ID, /NEXT, /OUT, /POOL, /PREV, /RUNTIME
SERVER	/APPL, /AT, /BRIEF, /CONFIG, /FULL, /ID, /NEXT, /OUT, /POOL, /PREV, /RUNTIME
TSC	/AT, /BRIEF, /CONFIG, /FULL, /ID, /NEXT, /OUT, /POOL, /PREV, /RUNTIME
TRACE Command	
None	None

12.2. ACMSSNAP CLOSE Command

ACMSSNAP CLOSE

ACMSSNAP CLOSE — Closes the current data snapshot file.

Format

ACMSSNAP> CLOSE

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

Only one snapshot file can be open at a time.

Examples

```
ACMSSNAP> CLOSE
```

This command closes the current data snapshot file.

12.3. ACMSSNAP EXIT Command

ACMSSNAP EXIT

ACMSSNAP EXIT — Ends the current ACMSSNAP session.

Format

ACMSSNAP> EXIT

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

This command ends the current ACMSSNAP session and returns control to DCL. It is equivalent to the **QUIT** command.

Examples

```
ACMSSNAP> EXIT
```

This command ends the current ACMSSNAP session.

12.4. ACMSSNAP HELP Command

ACMSSNAP HELP

ACMSSNAP HELP — Displays help information about the ACMS Remote Manager Data Snapshot Utility (ACMSSNAP) and its commands.

Format

ACMSSNAP> HELP

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

Online help is available for each ACMSSNAP command. Each help topic summarizes the valid syntax, abbreviations, parameters, and qualifiers for a particular command and also indicates all default and required values.

For a comprehensive list of ACMS utilities that offer online help or for further instructions on how to invoke help, see ACMS Help.

Examples

```
ACMSSNAP> HELP
```

This command invokes online help for the ACMSSNAP utility and displays a list of available topics.

12.5. ACMSSNAP NEXT Command

ACMSSNAP NEXT

ACMSSNAP NEXT — Reads the next sequence of snapshot records.

Format

ACMSSNAP> NEXT [*number*]

Privileges Required

None.

Parameters

number

A numeric value that indicates the number of records to be read. If a value is not specified, the default value of 1 record is used.

Qualifiers

None.

Notes

Use the **NEXT** command to move forward incrementally in a data snapshot file. When the **NEXT** command is issued, the ACMSSNAP utility reads the next record or series of records from the currently open snapshot file in chronological order. If a number is not specified with this command, **NEXT** moves forward one record at a time.

Note that the **NEXT** and **PREV** commands are not intended to be the primary means of navigation through a data snapshot file. Use the ACMSSNAP **SHOW/AT** command to first identify an approximate timeframe for the ACMS activity that you want to view. Then, use the **NEXT** or **PREV** command to move forward or backward incrementally from that timeframe.

If tracing is turned on (**TRACE** command), header information is displayed for each record read.

Examples

```
ACMSSNAP> NEXT 10
```

This command reads the next 10 records in the data snapshot file. These records overlay any previous records for this entity and class.

12.6. ACMSSNAP OPEN Command

ACMSSNAP OPEN

ACMSSNAP OPEN — Opens the specified data snapshot file.

Format

ACMSSNAP> OPEN *file-name* [/*qualifiers*]

Command Qualifier	Default
/AT=*time*	None
/CP_SLOTS=*value*	The value of ACMS$MGMT_SNAP_CP_SLOTS, or if not defined, 3.
/MAX_AGENTS=*value*	The value of ACMS$MGMT_SNAP_MAX_AGENTS, or if not defined, 2.
/MAX_APPLS=*value*	The value of ACMS$MGMT_SNAP_CP_SLOTS, or if not defined, 10.
/SUMMARY	None

Privileges Required

None.

Parameters

file-name

This required parameter specifies an OpenVMS file specification or logical that indicates the name and location of the data snapshot file.

Qualifiers

/AT=*time*

This qualifier moves through the open data snapshot file to the first record equal to or greater than the specified time. The format of time is *DD-MMMYY: hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported.

/CP_SLOTS=*value*

This qualifier reserves space for CP records in a statically-sized internal table. CP records are stored in the table by process name. These records are not removed from the table until the snapshot file is closed. If there is insufficient space in the table, data for subsequent CP processes is discarded. A warning message is issued if the CP table becomes full and **TRACE** is turned on. Specifying this qualifier overrides the value of the logical name ACMS$MGMT_SNAP_CP_ SLOTS, if defined. The default value is 3.

/MAX_AGENTS=*value*

This qualifier determines the number of different AGENT processes that can be read while the file is open. AGENT data is stored internally in a statically sized table, with one row for each unique AGENT process name found. If there is insufficient space in the table to hold all the AGENT records in the file, AGENT records are discarded. This qualifier overrides

the ACMS$MGMT_SNAP_MAX_ AGENTS logical name, which is an alternative way of specifying this value. Specifying this qualifier overrides the value of the logical name ACMS$MGMT_ SNAP_MAX_AGENTS, if defined. The default value is 2.

/MAX_APPLS=*value*

This qualifier reserves space for EXC records in a statically-sized internal table. EXC records are stored in the table by application name. These records are not removed from the table until the snapshot file is closed. If there is insufficient space in the table, data for subsequent EXC processes is discarded. A warning message is issued if the EXC table becomes full and **TRACE** is turned on. Specifying this qualifier overrides the value of the logical name ACMS$MGMT_ SNAP_MAX_APPL, if defined. The default value is 10.

/SUMMARY

This qualifier scans the entire data snapshot file and displays a summary report that shows the total number of records written (per entity, class, and file) as well as the time when the first and last record was written (per entity).

Notes

Data snapshot files are RMS indexed files. You can open a snapshot file with the ACMSSNAP utility even if it is currently being written to by an ACMS Remote Manager process.

Examples

```
ACMSSNAP>  OPEN ACMS$MGMT_SNAPSHOT /SUMMARY /CP_SLOTS=10 /MAX_APPL=5
ACMS Remote Management -- Snapshot utility
 Compiling summary statics ...

Entity # Recs First
              Record                  Last Record            All  Id  Cfg Rt
 Pool Error
------ ------ ---------------------- ---------------------- ---- --- --- --
 ---- -----
acc    42     7-JUN-2001 14:00:56.69 7-JUN-2001 14:21:32.19 42   0   0   0
 0    0
tsc    42     7-JUN-2001 14:00:56.69 7-JUN-2001 14:21:32.19 42   0   0   0
 0    0
qti    0                                                    0    0   0   0
 0    0
cp     184    7-JUN-2001 14:00:56.69 7-JUN-2001 14:21:32.19 184  0   0   0
 0    0
exc    204    7-JUN-2001 14:01:28.01 7-JUN-2001 14:21:32.19 204  0   0   0
 0    0
server 6496   7-JUN-2001 14:01:28.01 7-JUN-2001 14:21:32.19 6496 0   0   0
 0    0
group  3032   7-JUN-2001 14:01:28.01 7-JUN-2001 14:21:32.19 3032 0   0   0
 0    0
mgr       0                                                 0    0   0   0
 0    0

 10000 Records Read
MAX_APPL = 10 (use /MAX_APPL on OPEN or define ACMS$MGMT_SNAP_MAX_APPL to
 change)
```

```
CP_SLOTS = 10 (use /CP_SLOTS on OPEN or define ACMS$MGMT_SNAP_CP_SLOTS to
 change)
```

This command opens the data snapshot file referenced by the ACMS$MGMT_ SNAPSHOT logical, scans all the records, and displays a summary report. The CP_SLOTS and MAX_APPLS values are explicitly set.

12.7. ACMSSNAP PREV Command

ACMSSNAP PREV

ACMSSNAP PREV — Scans the previous sequence of snapshot records.

Format

ACMSSNAP> PREV [*number*]

Privileges Required

None.

Parameters

number

A numeric value that indicates the number of records to be read. If a value is not specified, the default value of 1 record is used.

Qualifiers

None.

Notes

Use the **PREV** command to move backward incrementally in a data snapshot file. When the **PREV** command is issued, the ACMSSNAP utility reads the previous record or series of records from the currently open snapshot file in reverse chronological order. If a number is not specified with this command, **PREV** moves backward one record at a time.

Note that the **NEXT** and **PREV** commands are not intended to be the primary means of navigation through a data snapshot file. Use the **SHOW/AT=*date-time*** command to first identify an approximate timeframe for the ACMS activity that you want to view. Then, use the **NEXT** or **PREV** command to move forward or backward incrementally from that timeframe.

If tracing is turned on (**TRACE** command), header information is displayed for each record read.

Examples

```
ACMSSNAP> PREV 10
```

This command scans and reads the previous 10 data snapshot records. These records overlay any previous records for this entity and class.

12.8. ACMSSNAP QUIT Command

ACMSSNAP QUIT

ACMSSNAP QUIT — Ends the current ACMSSNAP session.

Format

ACMSSNAP> QUIT

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

This command ends the current ACMSSNAP session and returns control to DCL. It is equivalent to the **EXIT** command.

Examples

```
ACMSSNAP> QUIT
```

This command ends the current ACMSSNAP session.

12.9. ACMSSNAP RESET Command

ACMSSNAP RESET

ACMSSNAP RESET — Clears the local memory tables.

Format

ACMSSNAP> RESET [/*qualifiers*]

Command Qualifier	Default
/ALL	None

Privileges Required

None.

Parameters

None.

Qualifiers

/ALL

This qualifier instructs the ACMSSNAP utility to place the file pointer at the beginning of the data snapshot file. If this qualifier is not specified, the local memory tables are cleared; however, the file pointer remains at the current position within the data snapshot file.

Notes

Use the **RESET** command to remove all data snapshot records from internal data tables and clear the local memory buffer.

Specifying the **/ALL** qualifier is equivalent to closing and reopening the data snapshot file; the position marker is moved to the beginning of the data snapshot file. Without the **/ALL** qualifier, the internal tables are cleared, but the location of the position marker is not changed.

Examples

```
ACMSSNAP> RESET /ALL
```

This command clears the local buffer of all data snapshot records and places the position marker at the beginning of the data snapshot file.

12.10. ACMSSNAP SHOW Command

ACMSSNAP SHOW

ACMSSNAP SHOW — Locates and displays information from one or more data snapshot records.

Format

ACMSSNAP> SHOW *entity* [/*qualifiers*]

Command Qualifier	Default
/APPL=*application-name*	None. Valid for EXC, SERVER, and Task Group only.
/AT=*time*	None
/[BRIEF,FULL]	/FULL if no class qualifier (/CONFIG, /ID, /POOL, or /RUNTIME) is specified. Otherwise, /BRIEF.
/[*class-name*]	* (all)
/GROUP=*group-name*	Valid for Task Group only.
/NEXT=*value*	None
/OUT=*file-name*	None

Command Qualifier	Default
/PREV=*value*	None
/SERVER=*server-name*	Valid for Server only.

Privileges Required

None.

Parameters

entity

This parameter can be one of the following ACMS entities: ACC, AGENT, CP, EXC, GROUP, QTI, SERVER, or TSC.

Qualifiers

/APPL=*application-name*

This qualifier specifies the application for which you want to view information. This qualifier is only valid when specified with the EXC, SERVER, or GROUP entity.

/AT=*time*

This qualifier moves through the available data snapshot records and displays the first record equal to or greater than the specified time. The format of time is *DD-MMM-YY:hh:mm:ss.nn*. Partial dates and times (for example, 10-OCT or 09:00) are supported.

/[BRIEF,FULL]

This qualifier causes either summary (**/BRIEF**) or detailed (**/FULL**) information to be displayed.

/[CONFIG,ID,POOL,RUNTIME]

This qualifier causes data for the class to be displayed. If no class qualifier is specified, this qualifier is ignored and all details are displayed (equivalent to **/FULL**).

/GROUP=*group-name*

This qualifier specifies the ACMS task group for which you want to view information. Wildcard matching is performed on the name provided. This qualifier is only valid when specified with the GROUP entity.

/NEXT=*value*

Use this qualifier to display the specified number of records (in chronological order) for an entity. Data is displayed for each record found. If **/FULL** is not specified, a timestamp that indicates when the record was created is also displayed. A value is required with this qualifier.

/OUT=*file-name*

This qualifier causes output to be written to the specified file. If this qualifier is not specified, output is displayed to the terminal (SYS$OUTPUT).

/PREV=*value*

Use this qualifier to display the specified number of records (in reverse chronological order) for an entity. Data is displayed for each record found. If **/FULL** is not specified, a timestamp that indicates when the record was created is also displayed. A value is required with this qualifier.

/SERVER=*server-name*

This qualifier specifies the ACMS procedure server for which you want to view information. Wildcard matching is performed on the name provided. This qualifier is only valid when specified with the SERVER entity.

Notes

The data shown for each entity parallels the format of the equivalent ACMSMGR command with the following exceptions:

- A timestamp is appended within square brackets to the output of **/BRIEF** displays. This is provided as a navigational aid.
- The node name is derived from the translation of the UCX$INET_HOST logical on the system the snapshot file was created. In ACMSMGR, the node name is taken from whatever was specified by the client when the ACMSMGR command was issued (such as from the **/NODE** qualifier or from the ACMS$MGMT_SERVER_NODE logical).

Data is written to snapshot files based on entries in the Collection table. As a result, only specific classes of information for a given entity may have been stored. Show commands for entities and classes that were not stored may display their default collection values along with a state value of disabled. The storage state for the collection row must be enabled for the actual data to be stored.

The **/AT** qualifier is intended to be the initial means of navigation by locating a specific entity record for a point in time. The ACMSSNAP utility uses the date given with the **/AT** qualifier and reads either backwards or forwards through the file until it finds a record for the specified entity. If the timestamp of the record is greater than the one specified, the utility begins reading backwards through the file until it finds an entity record with a time stamp equal to or less than the one specified.

If end or beginning of file is reached first, the search ends and an end-of-file message is displayed. Otherwise, the requested data is displayed. The end result is that when the command completes, a record is shown which is either at the exact time requested, or is the record just before or just after the time requested. You can then use the **/NEXT** or **/PREV** qualifier to navigate chronologically through adjacent records.

Special timestamps are used to deal with beginning and end of file conditions. If the beginning of file is reached, the current timestamp is forced to be NULL (17-NOV-1958 00:00:00.00). If the end of file is reached, the current timestamp is forced to be 17-NOV-3000 00:00:00.00. To recover from these situations, a single **NEXT** or **PREV** command will read either the first or last record in the file.

Examples

- ```
  ACMSSNAP> SHOW ACC /RUNTIME /AT="7-JUN-2001 14:21"
  ACMS Remote Management -- Snapshot utility
           Runtime  DECnet  ---- Users ---- --Applications- Application
   Node    Class    Object  Current Maximum Current Maximum Starts
   ------- -------- ------- ------- ------- ------- ------- ------------
   sparks  enabled  started 97      100     5       5       5 [7-
  JUN-2001-14:20:31.98]
  ```
  ```

This command displays ACC runtime information for the record written on June 7, 2001 at 14:20:31.98.

- ```
  ACMSSNAP> SHOW ACC /POOL /FULL
  ACMS Remote Management -- Snapshot utility
   ================================================================================
   Node         POOL
   ------------
   ------------------------------------------------------------------------
  sparks        Pool Class Collection State         enabled
                MSS Gauge             Current    Max       Time
                --------------------
   ------------------------------------------------
                MSS Objects           1859       1881      7-JUN-2001
   14:19:12.45
                MSS Maxbuf Message Counters         Current           Time
                ---------------------------------  -----------------
   ------------------
                MSS Msg Size     0 to 1024 bytes                13927
                MSS Msg Size  1025 to 2048 bytes                   94
                MSS Msg Size  2049 to 4096 bytes                   15
                MSS Msg Size  4097 to 8192 bytes                   41
                MSS Msg Size  8193 to 16384 bytes                   0
                MSS Msg Size 16385 to 32768 bytes                   0
                MSS Msg Size 32769 to 65536 bytes                   0
                MSS Message Counter Overflow Resets                 0
   (null)
                MSS Process Pool                              Pct  Time
                ----------------------------------------- -----
   ----------------------
                Pool Size (bytes)                524288
                Current Free (bytes)             516688   (98%)
                Minimum Free (bytes)             515664   (98%) 7-
  JUN-2001 14:18:16.00
                Largest Current Free
                Block (bytes)                    65536
                Minimum Largest Free
                Block (bytes)                    65536          7-
  JUN-2001 14:20:56.00
                Allocation Failures               0
                Garbage Collections               0
                MSS Shared Pool                                Pct  Time
                ----------------------------------------- ----
   ------------------------
                Pool Size (bytes)                 33792000
                Current Free (bytes)              33624344  (99%)
                Minimum Free (bytes)              33620712  (99%)  7-
  JUN-2001 14:14:25.98
                Largest Current Free
                Block (bytes)                    65536
                Minimum Largest Free
                Block (bytes)                    65536           7-
  JUN-2001 14:20:56.00
                Allocation Failures              0
                Garbage Collections              0
                WS/TWS Pools (for all EXCs)      Current   Max
   Time
  ```
  ```

```
                ---------------------------------- --------- ---------
 ----------------------
                TWS Pool Size Total (pagelets)     562800    562800    7-
JUN-2001 14:20:55.99
                TWSC Pool Size Total (pagelets)    22500     22500     7-
JUN-2001 14:20:55.99
                WS Pool Largest Used (bytes)       536       536       7-
JUN-2001 14:20:55.99
                WSC Pool Largest Used (bytes)      848       848       7-
JUN-2001 14:20:55.99
                TWS Pool Largest Used (bytes)      73728     73728     7-
JUN-2001 14:20:55.99
                TWSC Pool Largest Used (bytes)     1792      1792      7-
JUN-2001 14:20:55.99
                WS/TWS Pools (for all EXCs)        Current   Min
 Time
                ---------------------------------- --------- ---------
 ------------------------
                WS Pool Minimum Free (bytes)       130536    130536    7-
JUN-2001 14:20:55.99
                WSC Pool Minimum Free (bytes)      64688     64688     7-
JUN-2001 14:20:55.99
                TWS Pool Minimum Free (bytes)      3809280   3809280   7-
JUN-2001 14:20:55.99
                TWSC Pool Minimum Free (bytes)     152704    152704    7-
JUN-2001 14:20:55.99
```

This command displays ACC pool information from the same record.

12.11. ACMSSNAP TRACE Command

ACMSSNAP TRACE

ACMSSNAP TRACE — Toggles tracing information on and off.

Format

ACMSSNAP> TRACE

Privileges Required

None.

Parameters

None.

Qualifiers

None.

Notes

The main function of tracing is to display the node, timestamp, entity, and class type for the record being read. Tracing also provides some warning information when the internal CP or EXC tables become full.

The **TRACE** command either turns tracing on or off, depending on the current state. A status message is displayed when the command is issued that indicates which action was performed.

Examples

```
ACMSSNAP> TRACE
```

This command activates tracing for the current ACMSSNAP session.

Appendix A. Remote Manager Logical Names

This appendix contains information about the Remote Manager logical names used by the Remote Manager server and the Remote Manager client (ACMSMGR utility).

A.1. Remote Manager Server

The following list describes the logical names used by the Remote Manager server. If these logical names are present, they must be defined at a level that will be translated by the Remote Manager server. In general, these should be defined as system logicals.

- ACMS$MGMT_ALLOW_PROXY_ACCESS

  If defined as 1 or TRUE, the Remote Manager will perform proxy authorization using the ACMSPROXY.DAT file. If not defined, only explicit (user name and password) authorization is allowed.

- ACMS$MGMT_CONFIG

  File specification for the configuration file. If not defined, the default is SYS$SYSROOT:[SYSEXE]ACMS$MGMT_CONFIG.ACM. The default file extension is .ACM.

- ACMS$MGMT_TEMP

  Pointer to a directory that the Remote Manager server uses for writing and reading temporary command procedures used to modify the ACMS run-time system. If not defined when the Remote Manager server is started, the Remote Manager will define this as a system logical with a translation of SYS$MANAGER.

- ACMS$MGMT_LOG

  File specification for the Remote Manager log. If not defined, the default is ACMS$MGMT_LOG.LOG in the default directory of the Remote Manager.

A.2. Remote Manager Client (ACMSMGR Utility)

The following us a list of logical names used by the ACMSMGR utility:

- ACMS$MGMT_USER

  Defines the user name for Remote Manager authentication and authorization. This logical name should not be defined if proxy access is being used. If ACMS$MGMT_USER is not defined, the ACMSMGR utility either creates a user name (during login) or searches for the credentials file for this user. This logical name can be overriden by using the **/USER** qualifier on ACMSMGR commands.

- ACMS$MGMT_SERVER_NODE

Defines the fully or partially qualified TCP/IP node name. ACMS$MGMT_SERVER_NODE determines the nodes to which the ACMSMGR command is submitted. ACMS$MGMT_SERVER_NODE can be specified as a comma-separated list, in which case an attempt is made to execute the command on each node in the list serially. This logical name can be overridden by using the **/NODE** qualifier to ACMSMGR commands.

- ACMS$MGMT_CREDS_DIR

  Defines the directory in which the ACMSMGR stores and looks for credentials files. Credentials files are created by the ACMSMGR **LOGIN** command and are specific to a process on a node. See *Section 4.4.1.1, "Logging In"* for more information.

Appendix B. RPC Procedures and Corresponding Rights Identifiers

Table B.1, "RPC Procedures and Corresponding Rights Identifiers" lists RPC procedures and their corresponding rights identifiers.

Table B.1. RPC Procedures and Corresponding Rights Identifiers

Procedure	Rights Identifier
ACMSMGMT_ADD_COLLECTION_1	ACMS$MGMT_WRITE
ACMSMGMT_ADD_TRAP_1	ACMS$MGMT_OPER
ACMSMGMT_DELETE_COLLECTION_1	ACMS$MGMT_WRITE
ACMSMGMT_DELETE_TRAP_1	ACMS$MGMT_OPER
ACMSMGMT_GET_ACC_1	ACMS$MGMT_READ
ACMSMGMT_GET_MGR_STATUS_1	ACMS$MGMT_READ
ACMSMGMT_GET_PARAM_1	ACMS$MGMT_READ
ACMSMGMT_GET_QTI_1	ACMS$MGMT_READ
ACMSMGMT_GET_TSC_1	ACMS$MGMT_READ
ACMSMGMT_LIST_COLLECTIONS_1	ACMS$MGMT_READ
ACMSMGMT_LIST_CP_1	ACMS$MGMT_READ
ACMSMGMT_LIST_EXC_1	ACMS$MGMT_READ
ACMSMGMT_LIST_INTERFACES_1	ACMS$MGMT_READ
ACMSMGMT_LIST_LOG_1	ACMS$MGMT_READ
ACMSMGMT_LIST_PROC_1	ACMS$MGMT_READ
ACMSMGMT_LIST_SERVER_1	ACMS$MGMT_READ
ACMSMGMT_LIST_TG_1	ACMS$MGMT_READ
ACMSMGMT_LIST_TRAP_1	ACMS$MGMT_READ
ACMSMGMT_LIST_USERS_1	ACMS$MGMT_READ
ACMSMGMT_REPLACE_SERVER_1	ACMS$MGMT_OPER
ACMSMGMT_RESET_LOG_1	ACMS$MGMT_WRITE
ACMSMGMT_SET_ACC_1	ACMS$MGMT_OPER
ACMSMGMT_SET_COLLECTION_1	ACMS$MGMT_WRITE
ACMSMGMT_SET_EXC_1	ACMS$MGMT_OPER
ACMSMGMT_SET_INTERFACE_1	ACMS$MGMT_WRITE
ACMSMGMT_SET_PARAM_1	ACMS$MGMT_WRITE
ACMSMGMT_SET_QTI_1	ACMS$MGMT_OPER

Procedure	Rights Identifier
ACMSMGMT_SET_SERVER_1	ACMS$MGMT_OPER
ACMSMGMT_SET_TRAP_1	ACMS$MGMT_OPER
ACMSMGMT_SET_TSC_1	ACMS$MGMT_OPER
ACMSMGMT_START_ACC_1	ACMS$MGMT_OPER
ACMSMGMT_START_EXC_1	ACMS$MGMT_OPER
ACMSMGMT_START_QTI_1	ACMS$MGMT_OPER
ACMSMGMT_START_TRACE_MONITOR_1	ACMS$MGMT_OPER
ACMSMGMT_START_TSC_1	ACMS$MGMT_OPER
ACMSMGMT_STOP_1	ACMS$MGMT_OPER
ACMSMGMT_STOP_ACC_1	ACMS$MGMT_OPER
ACMSMGMT_STOP_EXC_1	ACMS$MGMT_OPER
ACMSMGMT_STOP_QTI_1	ACMS$MGMT_OPER
ACMSMGMT_STOP_TRACE_MONITOR_1	ACMS$MGMT_OPER
ACMSMGMT_STOP_TSC_1	ACMS$MGMT_OPER

Appendix C. RPC Procedures and Corresponding Rights Identifiers

This appendix contains the error messages related to the Remote Manager server process, as well as the ACMSMGR, ACMSCFG, and ACMSSNAP utilities.

C.1. Server Messages

The following error messages pertain to the ACMS Remote Manager server process.

Message: 2MANY_USERS, the maximum number of users has been reached

- **Explanation:** The user could not be logged in because the maximum number of concurrent users has been reached. This maximum is determined by the `max_logins` parameter, which is a dynamic parameter (that is, it can be changed dynamically).
- **User Action:** Either log some users out, or increase the value of the `max_logins` parameter. You can use the ACMSMGR **`SHOW USERS`** command to determine which users are already logged in to the Remote Manager. Note that in order to issue that command or to increase the `max_logins` parameter, you must be logged in.

Message: ACCTEXP, account is expired

- **Explanation:** This status is returned during user login if the account associated with the user name has expired.
- **User Action:** Either remove or modify the account expiration.

Message: ACMSPARFAIL, attempt to read ACMSPAR.ACM failed. See audit log for details.

- **Explanation:** This status is returned when the Remote Manager cannot access the ACMSGEN parameters file ACMSPAR.ACM.
- **User Action:** Examine the Remote Manager log file (using ACMSMGR **`SHOW LOG`**) and correct the problem.

Message: AUTHUPDFAIL, attempt to update the OpenVMS Authorization file failed. See audit log for details.

- **Explanation:** This status is returned when the Remote Manager is unable to write to the SYSUAF during an attempt to update an OpenVMS system parameter.
- **User Action:** Examine the Remote Manager log file (using ACMSMGR **`SHOW LOG`**) and correct the problem.

Message: BADDAY, network access is prohibited for this day of week for this account

- **Explanation:** This status is returned during user login if the UAF record for this user does not allow network access on this day of the week. Day-of-week restrictions on network access are set by system administrators or security personnel.
- **User Action:** Either wait until an authorized day of the week to access the Remote Manager, or modify the network access portion of the UAF for this user.

Message: BADHOUR, network access is prohibited for this hour for this account

- **Explanation:** This status is returned during user login if the UAF record for this user does not allow network access during this time of day. Time-of-day restrictions on network access are set by system administrators or security personnel.
- **User Action:** Either wait until an authorized hour to access the Remote Manager, or modify the network access portion of the UAF for this user.

Message: CREDS_DATA_ERR, credentials file is corrupt

- **Explanation:** The credentials file for this user has been corrupted. The file has been opened, but the client process cannot parse the contents of the file.
- **User Action:** The user should log in to the Remote Manager again. This will create a new credentials file.

Message: DISUSER, account is disusered

- **Explanation:** This status is returned during user login if the account associated with the user name has the DISUSER flag set.
- **User Action:** Clear the DISUSER flag on the UAF record for the account.

Message: DUPLICATE_ROW, table row exists

- **Explanation:** An attempt was made to add a row to either the Trap or the Collection table, but the row already exists.
- **User Action:** Either modify the existing row, or add a new row with unique data.

Message: ERRBOTHFLAGS, both current and active flags were set. They are mutually exclusive. No updates performed.

- **Explanation:** This status is returned when a request is made to modify Configuration data for an entity by setting both the `current_flag` (**`/CURRENT`**) and `active_flag` (**`/ACTIVE`**) parameters. These flags are mutually exclusive; the Remote Manager rejects the request.
- **User Action:** Resubmit the RPC call or ACMSMGR **`SET`** command setting only one flag per call or command.

Message: ERRSTOPSNP, attempt to stop snapshot thread failed. General internal error.

- **Explanation:** An attempt was made to stop a data snapshot thread by setting the `storage_state` parameter to DISABLED or deleting a row in the Collection table. The attempt failed due to a CMA exception.
- **User Action:** This is a non-recoverable error. Reissue the RPC call or ACMSMGR command. If the problem persists, restart the Remote Manager process.

Message: FLTRDBCORRUPT, the filter file database is corrupt.

- **Explanation:** This status is returned when the Remote Manager error filter database is non-readable.
- **User Action:** Shut down ACMS and the Remote Manager. Delete the file SYS$SYSTEM:ACMS$MGMT_ERROR_FILTER.DAT;*, and restart ACMS and the Remote Manager. If you have previously saved the contents of the error filter database (with ACMSMGR

SAVE FILTER), you can restore the database with the following command where *file-name* is a full OpenVMS file specification (`node::device:[directory]file.ext`) for the error filter file:

```
$ ACMSMGR ADD FILTER /FILE=file-name
```

Message: FLTRDBFULL, the filter database is fulls

- **Explanation:** This status is returned while attempting to add an error message filter record to a database that is at capacity.
- **User Action:** Delete one or more error filter records using the ACMSMGR **DELETE FILTER** command.

Message: FLTRDBINIT, the filter file global section is not initialized

- **Explanation:** This status is returned while attempting to access the error filter database without the global section initialized.
- **User Action:** Report this error to your VSI support representative.

Message: EVNT_MBX_FUL, event mailbox is full

- **Explanation:** This status code has not been implemented.
- **User Action:** No action is required.

Message: FAIL, operation failed

- **Explanation:** The function requested could not be performed.
- **User Action:** The appropriate action depends on the function being called. In general, additional information is displayed by the ACMSMGR command in conjunction with this error code. That information should be more indicative of the reason for failure. If this status was returned by an RPC, the failure occurred in the Remote Manager process and a second-level error code is returned in the output record.

Message: INFO, operation completed with information message

- **Explanation:** The ACMS Remote Manager service completed without error but has logged an informational message. Informational messages are for debugging and auditing purposes.
- **User Action:** No action is required.

Message: INTERNALERR, an internal error has occurred. See audit log for details

- **Explanation:** A request was to update the runtime ACMS system, but an unexpected error was returned by the DCL manager subsystem.
- **User Action:** Examine the Remote Manager log for related informational messages. If the problem persists, restart the Remote Manager process.

Message: INV_CLNTID, client id is invalid

- **Explanation:** A request was made to the Remote Manager with an invalid client id. The client id is a unique value assigned to each client and used to verify client authorization. If the client id is not known to the Remote Manager, it either belongs to an old log in that has expired and been purged, or it was never valid.

- **User Action:** The user should log in to the Remote Manager again.

Message: INVVAR, invalid variable value was provided

- **Explanation:** This status code is obsolete.
- **User Action:** No action required.

Message: LOGIN_EXPIRED, login credentials have expired, please log in again

- **Explanation:** The credentials you used to log in to the Remote Manager have expired. Credentials are granted when the user logs in and are valid for a period of time equal to the value of the `login_creds_lifetime` parameter at the time of login. After that time, the credentials expire and must be re-created by logging in to the Remote Manager again.

  Note that while proxy credentials also expire, they are automatically recreated at the end of the expiration period. Therefore, this status is never returned to proxy users.
- **User Action:** The user must log in to the Remote Manager again.

Message: MSGEXISTS, message already exists

- **Explanation:** An attempt was made to add an existing error message record to the filter database.
- **User Action:** No action required.

Message: NOACCFLTRDB, the filter database cannot be read or written

- **Explanation:** The memory address returned for error filter global section is not allowing read or write operations.
- **User Action:** Examine the OWNER and GROUP protections on the file SYS$SYSTEM:ACMS$MGMT_ERROR_FILTER.DAT. Check the user accounts for the related ACMS Remote Manager processes to verify that they have READ and WRITE access to the file. The file owner and the ACMS accounts should be in the same group. All members of this group should have read and write access to the file.

Message: NOAPPLQUAL, /APPL qualifier missing

- **Explanation:** The command ACMSMGR **`SET EXC`** requires that you use an **`/APPL`** qualifier to specify the application for which stored values are being set. If the **`/APPL`** qualifier is missing, the error NOAPPLQUAL will be returned.
- **User Action:** Re-enter the ACMSMGR **`SET EXC`** command with a **`/APPL`** qualifier.

Message: NO_CREDS_FILE, credentials file not found

- **Explanation:** The credentials file for the user either could not be found or could not be opened by the client process. The credentials file is created when a user explicitly logs in to the Remote Manager (that is, when the user supplies a user name and password). A separate credentials file is created for each node to which a particular process logs in. The logical name ACMS$MGMT_CREDS_DIR is used to point to the directory containing credentials files.
- **User Action:** Verify that the ACMS$MGMT_CREDS_DIR logical is pointing to a valid disk and directory in which a credentials file has been created; verify that the process has read access to the files in the directory. If necessary, the process may have to log in to the Remote Manager again to create a new credentials file.

Message: NOINTERVAL, storage interval not supplied. Cannot enable a thread without an internal. Resubmit with an interval.

- **Explanation:** An API call was made to either the ACMS$MGMT_SET_COLLECTION_2 or ACMS$MGMT_ADD_COLLECTION_2 function, but no value was specified for the storage interval.
- **User Action:** Reissue the call making sure that a valid `storage_interval` is specified prior to setting the `storage_state` to ENABLED.

Message: NOMEM, memory allocation failed

- **Explanation:** An internal memory allocation by the Remote Manager failed. This can occur during a request for data, a call to add a record to a table, or during server initialization while it is loading initial configuration information. In the first two instances, the Remote Manager continues to run; in the third, the Remote Manager exits.
- **User Action:** Increase the amount of memory available to the Remote Manager. If the problem is due to insufficient physical or virtual memory, try allocating more page or swap space. If physical and virtual memory are not exhausted, try increasing the memory quotas for the account in which the Remote Manager is running. Be sure to check SYSGEN PQL quotas to ensure that the quotas you grant to the Remote Manager are allowed by the system.

  After making more memory available to the Remote Manager, you must restart the Remote Manager process.

Message: NOMORE_DATA, no more data is available

- **Explanation:** There is no more data that satisfies the query. This message is provided on list RPCs that can return more than one buffer of data. If a list RPC is called and this status is not returned, then more data is available that satisfies the query criteria. If this status is returned, then there is no more data to retrieve for this query.
- **User Action:** No action is required. The query is complete.

Message: NOMSGINTBL, message not in filter database

- **Explanation:** An attempt was made to access an error message code that does not currently exist in the error filter database.
- **User Action:** Verify that the error message code is correct and that it exists in the filter database. To display the error message codes currently being filtered, use the ACMSMGR **`SHOW FILTER`** command.

Message: NONETACCESS, network access is prohibited for this account

- **Explanation:** This status is returned during user login if the account associated with the user name has not been granted network access. Network access is required, even if the user is logged in to same node on which the Remote Manager is running.
- **User Action:** Grant network access to the account.

Message: NO_NODELOGICAL, cannot translate logical UCX$INET_HOST to get node name

- **Explanation:** The logical name UCX$INET_HOST could not be translated by the client process. This logical is used to determine the current host name, which is used during client authentication.

It is defined by the UCX or TCP/IP layered product when it is started. If this logical is not defined, UCX or TCP/IP is not started; either a different TCP/IP networking package is being used, or something has gone wrong with the logical name.

- **User Action:** Verify that the UCX or TCP/IP layered product is started. If it is not, start it and then reissue the command. If it is started, contact your system administrator to determine why the logical is not defined.

Message: NOPROXY, proxy access is not enabled

- **Explanation:** This status is returned when a user attempts to access a Remote Manager function without explicitly logging in, and proxy access has not been enabled on the Remote Manager node. Proxy access is enabled on the node by defining the system logical ACMS$MGMT_ALLOW_PROXY_ACCESS to be TRUE, true, T, t, Y, y, or 1. The translation of this logical is cached by the Remote Manager when the RPC thread is started.

- **User Action:** If proxy access is not supposed to be enabled, then there is no action to perform. If proxy access is to be allowed, define the ACMS$MGMT_ALLOW_PROXY_ACCESS system logical, with a translation value of TRUE, true, T,t, Y, y or 1. Then, restart the Remote Manager and resubmit the RPC.

Message: NOREINIT, filter file cannot be re-initialized

- **Explanation:** The Remote Manager attempted to reinitialize the filter database. This is result of an internal consistency check that failed. The error should not have occurred.

- **User Action:** Report this error to your VSI support representative.

Message: NORIGHT, user does not hold the proper rights identifier

- **Explanation:** Access to Remote Manager functions is restricted by a set of rights identifiers; the account being used to access the function must have the appropriate rights identifier. If this status code is returned, the account does not have the appropriate rights identifier.

- **User Action:** Grant the appropriate rights identifier to the user's account. If a proxy account is being used, the rights identifier must be granted to the proxy account.

Message: NOSTOP, interface cannot be used to stop itself

- **Explanation:** An attempt was made to stop an interface by using that interface. The Remote Manager does not allow either the RPC or the SNMP interface to be used to stop themselves.

- **User Action:** If you need to stop the RPC interface and cannot use the ACMSMGR command, stop the Remote Manager either by using the DCL command **`STOP/ID`** or by using the UCX$SNMP_REQUEST (or TCPIP$SNMP_REQUEST) program. These programs are located in SYS$SYSTEM; the OID to use is 1.3.6.1.4.1.36.2.18.48.4.1.3.1.

  To stop the SNMP interface, you must use the ACMSMGR **`SET INTERFACE`** command, or you must stop the Remote Manager (using either the ACMSMGR **`STOP`** command or the DCL command **`STOP/ID`**).

Message: NOSUCHAPPL, /APPL application does not exist

- **Explanation:** The command ACMSMGR **`SET EXC`** requires that you use an **`/APPL`** qualifier to specify the application for which stored values are being set. NOSUCHAPPL is returned if the application you specify is not active.

- **User Action:** Re-enter the ACMSMGR **SET EXC** command specifying an active application in the **/APPL** qualifier.

Message: NOTENABLED, cannot disable storage if it is not enabled

- **Explanation:** An attempt was made to set the `storage_state` for a Collection record to DISABLED when it already was disabled. No action was performed by the Remote Manager.

- **User Action:** No action is required.

Message: NOTFILTERFILE, the file is not an error filter text file

- **Explanation:** This status is returned when the file specified in the ACMSMGR **ADD FILTER/FILER** command does not meet the format requirements for an error filter file.

- **User Action:** Verify that the file specification is correct. If the specification is correct, review ACMSMGR online help for details about the formatting requirements for an error filter file. The first line of the file must contain the string `%%ACMS Filter File V1.00`. If it does not, the Remote Manager will not consider it a valid error filter file.

Message: NOT_FOUND, record not found

- **Explanation:** An attempt was made to delete a row from the Collection table, but that row does not exist.

- **User Action:** Modify the request to include the proper identification information for the row (entity type, class, and name).

Message: NOT_MAPPED, ACMSMGMT global section is not available on node <nodename>

- **Explanation:** This status code is returned if a Remote Manager function was requested that requires access to the ACMSMGMT global section, but the global section does not exist. The global section is created by the ACMS ACC during system startup. This status code indicates that the ACMS ACC is not running or that it has not yet created the global section.

- **User Action:** Start the ACMS run-time system in order to create the global section. The ACMS run-time system can be started by using either the ACMSMGR **START SYSTEM** command or the **ACMS/START SYSTEM** command.

Message: NOT_VALID, entity data is stale, please resubmit query or wait until later

- **Explanation:** The entity record in the global section is not valid. If this status code is returned, it means that the entity has never been started on the Remote Manager node. If the entity had been running at one time but no longer is, a severity level of WARN is returned and the `record_state` is set to INACTIVE.

- **User Action:** No action is required. However, if the entity is started on the Remote Manager node, the data will become available.

Message: NO_UPD_CLS, class cannot be modified

- **Explanation:** An attempt was made to add, delete, or modify a Collection table record for the Id or Config class. These records cannot be modified.

- **User Action:** No action is required. There is no way to modify the default collection records for the Id or Config classes.

Message: PROXY_FAILED, proxy access attempt failed

- **Explanation:** An attempt to verify proxy information for this client failed. A more specific message indicating why the proxy failed is written to the Remote Manager log. Reasons for proxy authentication failure include:
  - No proxy record is in the ACMSPROXY.DAT file.
  - The proxy account is disusered.
  - There is a problem with the network access for the account (does not have network access allowed, is outside of the allowed network access days or times).
  - The proxy account has expired.
  - An internal error occurred during processing.
- **User Action:** First check the Remote Manager log for any additional information related to the login attempt. Then verify that none of the conditions listed are preventing the login from succeeding.

Message: PWDEXP, password has expired

- **Explanation:** The password entered by the user has expired in the UAF on the server node.
- **User Action:** The user must either change the password or have it unexpired by a system or security administrator.

Message: PWDFAIL, invalid password

- **Explanation:** The password entered for the user during user login does not match the one stored in the UAF for this user on the server node.
- **User Action:** Resubmit the login request for the user with the correct password.

Message: SECCHKFAIL, security check failed. You do not hold the ACMS$MGMT_SYSUPD rights identifier

- **Explanation:** A request was made to modify one or more OpenVMS SYSGEN parameters with the Remote Manager; however, the user does not have the proper rights identifier. The request was denied.
- **User Action:** Ask the ACMS system manager to grant the user account the ACMS$MGMT_SYSUPD rights identifier.

Message: SNPRUNNING, failure creating timer thread entry. Thread already running?

- **Explanation:** A request was made to set the `storage_state` for a Collection record to ENABLED; however, an internal error was raised.
- **User Action:** Examine the Remote Manager log for additional messages that describe the source of the problem. Reissue the command. If the problem persists, try deleting and adding the Collection record. If the problem remains, restart the Remote Manger process.

Message: SUCCESS, operation completed

- **Explanation:** The ACMS Remote Manager service completed without error.

- **User Action:** No action is required.

Message: TABLE_FULL, collection table is full. Non dynamic parameter `total_entity_slots` controls size

- **Explanation:** An attempt was made to add a row to the Collection table, but there are no empty slots. The maximum number of rows in the Collection table is determined by the nondynamic parameter `total_entity_slots`.

- **User Action:** To make the table bigger, modify this parameter in the configuration file (using the ACMSCFG command), and restart the ACMS run-time system. The Remote Manager can be left running.

  Alternatively, delete unneeded rows from the Collection table.

Message: THREADRUNNING, inconsistent state detected. Snapshot state is disabled but thread is running! Cannot enable running thread

- **Explanation:** A request was made to set the `storage_state` for a Collection record to ENABLED. While processing the row, the Remote Manager discovered that a snapshot thread had already been assigned to this record. This condition is most likely due to a previous, unsuccessful attempt to end a snapshot operation.

- **User Action:** Examine the Remote Manager log for additional messages that describe the source of the problem. Reissue the command. If the problem persists, try deleting and adding the Collection record. If the problem remains, restart the Remote Manger process.

Message: VRSNMISMAT, filter file version mismatch

- **Explanation:** The Remote Manager error filter file may be corrupted, or the format may have changed in this version of ACMS.

- **User Action:** Shut down ACMS and the Remote Manager. Delete the file SYS$SYSTEM:ACMS$MGMT_ERROR_FILTER.DAT;*, and restart ACMS and the Remote Manager. If you have previously saved the contents of the error filter database (with ACMSMGR **`SAVE FILTER`**), you can restore the database with the following command:

  ```
  $ ACMSMGR ADD FILTER /FILE=file-name
  ```

  Here, *file-name* is a full OpenVMS file specification (`node::device:[directory]file.ext`) for the error filter file.

Message: WARN, operation completed with warning, Not all operations completed successfully

- **Explanation:** The ACMS Remote Manager service did not complete successfully; some of the actions requested could not be completed. This status is returned in the following situations:

  - Multiple fields were specified on an update function, at least one of which failed. For example, a call may have been made to the set parameters function (`acmsmgmt_set_param_1`) to update more than one parameter, and one of the values specified was invalid. For these functions, a list of fields is returned with status codes for each field.

  - A call to start or stop an ACMS process was executed, and a warning was returned. Starting or stopping ACMS processes is performed by ACMS OPER, which may return warning messages. In this case, a set of messages is returned describing the cause for the warning.

- A call to display ACMS process information was made, and old (stale) data was returned. This can occur when an ACMS process is no longer running, and a show function requests data for that process. For instance, if the TSC was running and then had been stopped, and the `acmsmgmt_get_tsc_1` function is called, the tsc record is returned with WARN status.

- **User Action:** No action is required; however, the `record_state` field of any record returned should be checked. The Remote Manager flags old data with a `record_state` of MGMT$K_INACTIVE.

  If this status is returned as the result of an ACMSMGR command, old (inactive) records are flagged with an asterisk (*) preceding the node name.

Message: WRONG_NODE, current node does not match node in credentials file!

- **Explanation:** The node name stored in the credentials file does not match the node name on which the current process created it. Either the file is corrupt, or it has been tampered with.

- **User Action:** The user should log in to the Remote Manager again. This will create a new credentials file.

Message: WRONG_PID, current pid does not match pid in credentials file!

- **Explanation:** The PID stored in the credentials file does not match that of the current process. Either the file is corrupt, or it has been tampered with.

- **User Action:** The user should log in to the Remote Manager again. This will create a new credentials file.

Message: XLATE_FAILED, an attempt to translate a symbolic error name failed

- **Explanation:** An attempt was made to add an error filter record using the symbolic name for the error message. The translation of the symbolic name to its hexadecimal value failed.

- **User Action:** Verify that you have specified the symbolic name correctly. If the problem persists, try entering the record using the hexadecimal value of the error message that you want to filter.

C.2. ACMSMGR Messages

The following error messages pertain to the ACMSMGR process.

Message: 2MANYCLASSES, too many class qualifiers were specified.

- **Explanation:** Each ACMSMGR command allows either one class or all classes to be displayed at the same time. To display all classes, do not include a class qualifier with the command. To display a particular class, include that qualifier with the command. You cannot specify more than one class qualifier with a given command.

- **User Action:** Modify the command to include a maximum of one class qualifier, and resubmit the command. To learn more about the valid class qualifiers for a given command verb and object, use the ACMSMGMR **`HELP`** command.

Message: ACTUPDINV, /active was specified for <field>, but this value is not dynamic. No update was performed

- **Explanation:** An attempt was made to modify Configuration class data for an entity using ACMSMGR **SET *entity*/ACTIVE** command; however, one of the specified variables does not have an active value.

- **User Action:** Reissue the command either without the **/ACTIVE** qualifier or with only variables that have an active value. For a list of variables and their valid values, see the ACMSMGR online help.

Message: BADTIME, invalid time <time>

- **Explanation:** The time specification provided for the command qualifier could not be parsed. Time specifications can include date and time, date only or time only. The date and time should be specified as a quoted string in the format *DDMMMYYYY HH:MM:SS.hh*. Partial strings are accepted (for example, "1NOV" or "10:00").

- **User Action:** Modify the command to include a valid time specification. To learn more about valid time values, use the ACMSMGR **HELP** command.

Message: BADVALUE, invalid qualifier, cannot interpret value <value>

- **Explanation:** The value provided for a command qualifier is invalid. It is either out of the range of acceptable values or is an invalid type.

- **User Action:** Modify the command qualifier to include a valid value. To learn more about valid values for a given qualifier, use the ACMSMGR **HELP** command.

Message: CREDSFOUND, credentials for node <node> found, they will be used

- **Explanation:** This message is obsolete.

- **User Action:** No action is required.

Message: ENCRYPTFAIL, encryption routine failure on <username>

- **Explanation:** An attempt to encrypt the user name indicated for client authentication failed.

- **User Action:** Make sure the user name is correct and reenter it.

Message: END_TOO_SOON, end time must be greater than begin time

- **Explanation:** While adding or modifying a row in the Collection table, an attempt was made to specify a `storage_end_time` earlier than the current `storage_begin_time`.

- **User Action:** Reissue the command, ensuring that `storage_end_time` is later than the `storage_begin_time`; or omit `storage_end_time` or `storage_begin_time` to accept the default value.

Message: FILEOPENERR, could not open file <filename> for write

- **Explanation:** This message is displayed when the **/OUT** qualifier to ACMSMGR command specifies an invalid or inaccessible file specification. The reason could be an invalid disk or device name, an incorrect logical name in the file specification, insufficient privileges for writing to the directory or file, or a full device.

- **User Action:** Ensure that the client process can access the directory and file in the file specification, and resubmit the command.

Message: HOSTNAMEFAIL, can't translate UCX$INET_HOST name, aborting...

- **Explanation:** The logical name UCX$INET_HOST could not be translated by the client process. This logical is used to determine the current host name, which is used during client authentication. The logical is defined by the UCX or TCP/IP layered product when it is started. If the logical is not defined, either UCX or TCP/IP is not started, a different TCP/IP networking package is being used, or something has gone wrong with the logical name.
- **User Action:** Verify that the UCX or TCP/IP layered product is started. If it is not, start it and then reissue the command. If it is started, contact your system administrator to determine why the logical is not defined.

Message: INACTDATA, some or all data is old and may not accurately reflect the running system

- **Explanation:** The data being displayed may contain information about processes that are no longer running. Old, or stale, data is displayed only if the **/ALL** qualifier was included with the command. The old records are flagged with an asterisk (*) in the first character of the node name.
- **User Action:** No action is required. Remove the **/ALL** qualifier if you do not want to see old data.

Message: LOGIN_FAIL, login failed

- **Explanation:** The attempt to log in failed. The reason might be an invalid user name or password. The Remote Manager log will contain more information about the reason for failure.
- **User Action:** Consult the Remote Manager log on the target node for more information about the failure (using the ACMSMGR **SHOW LOG** command). Correct the problem and resubmit the login request.

Message: NAME2BIG, username is too long, please reenter

- **Explanation:** The user name specified exceeds the maximum allowed length of 12 characters.
- **User Action:** Modify the user name to be no longer than 12 characters and reenter it.

Message: NOCLIENTS, no clients created, cannot continue

- **Explanation:** No client attaches were successful. Previous messages will have been displayed indicating the particular reasons that the client attaches failed. Without attaching a client to a server, work can not be performed.
- **User Action:** Determine and remedy the reason for the client attach failures then resubmit the command.

Message: NOCLNT_ATTACH, cannot create client for node <node-name>

- **Explanation:** An attempt to attach to the server on the node indicated in the command failed. If more than one server name was in the list to be processed, the next server will be tried. No further attempt will be made to submit commands to the node for which the attached client failed.
- **User Action:** Verify that the node name is correct and that the Remote Manager is running on the node indicated. In some networks, it may be necessary to use a fully qualified TCP/IP node name. If the Remote Manager is running, verify that the Portmapper is running on the node and that the RPC interface in the Remote Manager has been started.

Message: NOCOMPND, compound name is not allowed for this entity (only server and group).

- **Explanation:** An attempt was made to add a collection record using a compound name for an entity that is not a server or group. Compound names (that is, names that contain an application specification and a process specification) are valid for only servers and groups.

- **User Action:** Modify the command so it does not include a compound name. For more help about adding collection records, use the ACMSMGR **HELP ADD COLLECTION** command.

Message: NOCREDS, could not get credentials for <node::user>

- **Explanation:** The credentials file for the user either could not be found or could not be opened by the client process. The credentials file is created when a user explicitly logs in to the Remote Manager (that is, the user supplies a user name and password). A separate credentials file is created for each node to which a particular process logs in. The logical name ACMS$MGMT_ CREDS_DIR is used to point to the directory containing credentials files.

  Note that credentials files are process-specific (PID) and node specific.

- **User Action:** Verify that the ACMS$MGMT_CREDS_DIR logical is pointing to a valid disk and directory in which a credentials file has been created; verify that the process has read access to the files in the directory. The process may have to log in to the Remote Manager again to create a new credentials file.

Message: NODATA, no <entity-type> data was found for <node-name>

- **Explanation:** The request to get information about the entity type indicated in the message returned no data. This message occurs when no instances of the particular process are running.

- **User Action:** No action is required. If the process had been running previously, you may be able to see the information by resubmitting the command with the **/ALL** qualifier.

Message: NOFACILITY, facility must be specified when setting trace level

- **Explanation:** This message is obsolete.

- **User Action:** No action is required.

Message: NONODE, node must be specified as a logical or an argument

- **Explanation:** Each ACMSMGR command needs a node name to be specified. The node name can be specified either as a command qualifier (for example, **/NODE=*mynode***) or by the logical name ACMS$MGMT_SERVER_NODE. Multiple nodes can be specified in a comma-separated list (for example, **/NODE=*node1,node2***). To specify a list of nodes when defining the logical name ACMS$MGMT_SERVER_NODE, enclose the entire list in double quotation marks. For example:

  ```
  $ DEFINE ACMS$MGMT_SERVER_NODE "NODE1,NODE2"
  ```

- **User Action:** Modify the command to include the **/NODE** qualifier, or define the logical ACMS$MGMT_SERVER_NODE to include at least one node name.

Message: NORMAL, operation completed

- **Explanation:** The command completed successfully.

- **User Action:** No action is required.

Message: NOVAL, qualifier <qualifier> requires a value string

- **Explanation:** The qualifier indicated requires a value string, but none was provided.
- **User Action:** Modify the command to include a valid value string and resubmit the command. To learn more about the valid values for a given qualifier, use the ACMSMGR **HELP** command.

Message: OLDDATA, this data is old and may not accurately reflect the running system

- **Explanation:** The data being displayed may contain information about processes that are no longer running. Old, or stale, data is displayed only if the **/ALL** qualifier was included with the command. The old records are flagged with an asterisk (*) in the first character of the node name.
- **User Action:** No action is required. Remove the **/ALL** qualifier if you do not want to see old data.

Message: PARAMFAIL, parameter update failed for parameter <parameter>. ACMS is not available or does not respond.

- **Explanation:** A request to update an interval value failed because the Remote Manager could not pass the changed value to the ACMS process.
- **User Action:** Examine the Remote Manager log for additional messages that might provide more information about the failure, and ensure that the ACMS trace monitor is running (process **ACMS$TRACE_MON**). If the trace monitor is not running, check whether the trace monitor logical **ACMS$TRACE_MBX** is defined.

  Also, ensure that the ACMS Remote Manager has been started (ACMSMGR **START SYS**). If the problem persists, restart ACMS and the Remote Manager.

Message: PASS2BIG, password is too long, please reenter

- **Explanation:** The password specified exceeds the maximum allowed length of 32 characters.
- **User Action:** Modify the password to be no longer than 32 characters and reenter it.

Message: QUALCONFLICT, qualifier <qualifier> is not supported in combination with other qualifier(s) on this command line

- **Explanation:** Multiple, mutually exclusive qualifiers were provided for an ACMSMGR command.
- **User Action:** Review the qualifiers and determine which are mutually exclusive. Reissue the command with the appropriate combination of qualifiers. For a list of variables and their valid values, see the ACMSMGR online help.

Message: STORUPDINV, /stored was specified for <field>, but this value cannot be stored. No update was performed

- **Explanation:** An attempt was made to modify Configuration class data for an entity using ACMSMGR **SET *<entity>* /STORED** command; however, one of the specified variables does not have a stored value.
- **User Action:** Reissue the command either without the **/STORED** qualifier or with only variables that have a stored value. For a list of variables and their valid values, see the ACMSMGR online help.

Message: UNKCMD, unrecognized command <command>

- **Explanation:** The command verb indicated in the message is not valid.

- **User Action:** Modify the command to include a valid command verb. To learn more about valid verbs, use the ACMSMGR **HELP** command.

Message: UNKCMDOBJ, unrecognized object <command object> for this command

- **Explanation:** The command object indicated in the message is not valid.
- **User Action:** Modify the command to include a valid command object. To learn more about valid command objects, use the ACMSMGR **HELP** command.

Message: USE_PROXY, username not supplied, proxy access will be attempted

- **Explanation:** This message is obsolete.
- **User Action:** No action is required.

Message: WRONGPARAM, parameter type is not valid for this entity type, please correct and reexecute command

- **Explanation:** The parameter type in the command is not valid for the entity type specified. Each parameter is valid for only a particular set of entity types; the combination specified is not a valid pair.
- **User Action:** Modify the command to include a valid combination of entity type and parameter. To learn more about valid combinations, use the ACMSMGR **HELP** command.

Message: WRONGQUAL, qualifier <qualifier> is not supported for this verb and object, please correct and re-execute command

- **Explanation:** A command qualifier was specified that is not valid for the verb/object combination specified.
- **User Action:** Modify the command to include a valid qualifier. To learn more about the valid qualifiers for a verb/object combination, use the ACMSMGR **HELP** command.

C.3. ACMSCFG Messages

The following error messages pertain to the ACMSCFG process.

Message: ADDREC, record does not exist, creating new record from defaults

- **Explanation:** This message is obsolete.
- **User Action:** No action is required.

Message: BAD_TIME, invalid time <time>

- **Explanation:** This status code has not been implemented yet.
- **User Action:** No action is required.

Message: BAD_VALUE, invalid qualifier, cannot interpret value <value>

- **Explanation:** A null or invalid value was provided to a qualifier.
- **User Action:** Correct the value and resubmit the command. To learn more about ACMSCFG qualifiers and valid values, use the ACMSCFG **HELP** command.

Message: CLASS_REQ, collection class is required, please resubmit command

- **Explanation:** This status code has not been implemented yet.
- **User Action:** No action is required.

Message: CONFIG_NORMAL, operation completed

- **Explanation:** The ACMSCFG command completed successfully.
- **User Action:** No action is required.

Message: END_TOO_EARLY, end time must be greater than begin time

- **Explanation:** This status code has not been implemented yet.
- **User Action:** No action is required.

Message: ENTITY_REQ, entity type is required, please resubmit command

- **Explanation:** A command was issued that requires an entity type to be specified.
- **User Action:** Resubmit the command, including the **/ENTITY=<*entity_type*>** qualifier and the appropriate entity type.

Message: ERRADD, error adding record

- **Explanation:** This message is obsolete.
- **User Action:** No action is required.

Message: ERRDEL, error deleting record

- **Explanation:** An error occurred while a record was being deleted. This message is preceded by a status message returned by RMS describing the error. The problem is usually an environmental one – for example, a locked file, insufficient privileges, and so on.
- **User Action:** Refer to the message immediately preceding this one that describes the error returned from RMS.

Message: ERROR_OPEN, could not open file <filename>

- **Explanation:** This message is obsolete.
- **User Action:** No action is required.

Message: ERRUP, error updating record

- **Explanation:** An error occurred while a record was being updated. This message is preceded by a status message returned by RMS describing the error. The problem is usually an environmental one – for example, a locked file, insufficient privileges, and so on.
- **User Action:** Refer to the message immediately preceding this one that describes the error returned from RMS.

Message: ID_REQ, entity name is required, please resubmit command

- **Explanation:** This status code has not been implemented yet.
- **User Action:** No action is required.

Message: IF_INVAL, invalid interface <interface>

- **Explanation:** The interface name indicated is unrecognizable.
- **User Action:** Modify the **/INTERFACE** qualifier to include a valid interface type. To learn about valid interface types, use the ACMSCFG **HELP** command.

Message: IF_REQ, interface type is required, please resubmit command

- **Explanation:** The interface type is required in order to update the correct record. The command that was submitted did not specify the interface type.
- **User Action:** Modify the command to include the **/INTERFACE** qualifier, along with the desired interface type. To learn more about updating interfaces, use the ACMSCFG **HELP** command.

Message: INSUF_ARGS, insufficient arguments

- **Explanation:** An insufficient number of arguments was passed to the ACMSCFG utility. At least one argument is required.
- **User Action:** Modify the command to include at least one argument. Use the ACMSCFG **HELP** command to learn about the various ACMSCFG commands.

Message: INVPARAMTYPE, parameter type <parameter> is not valid for this entity type, please correct and re-execute the command

- **Explanation:** The parameter type indicated is not valid for the entity type specified. Each parameter is valid for only a particular set of entity types; the combination specified is not a valid pair.
- **User Action:** Modify the command to include a valid combination of entity type and parameter. To learn more about valid combinations, use the ACMSCFG **HELP** command.

Message: INVSTATE, invalid state <state>

- **Explanation:** The state indicated is unrecognizable.
- **User Action:** Modify the qualifier to include a valid state. To learn about valid states, use the ACMSCFG **HELP** command.

Message: INVVALUE, qualifier <value> contains an invalid value, please correct and resubmit

- **Explanation:** A qualifier was specified with an invalid value.
- **User Action:** Modify the command to include a valid value with the qualifier. To learn more about valid values for the qualifier, use the ACMSCFG **HELP** command.

Message: NAME_REQ, entity name is required, please resubmit command

- **Explanation:** This message is obsolete.
- **User Action:** No action is required.

Message: NODEFADD, records for this class cannot be added

- **Explanation:** An attempt was made to add a record for the Id and Config classes, which is not allowed. New records for Id and Config classes cannot be added.
- **User Action:** Any attempt to add records for Id and Config classes will fail.

Message: NODEFDELETE, this record cannot be deleted, it is a mandatory default

- **Explanation:** An attempt was made to delete a record that cannot be deleted. Default collection records for Id and Config classes cannot be deleted.
- **User Action:** Any attempt to delete these records will fail.

Message: NODEFDISABLE, collection cannot be disabled for this entity and class

- **Explanation:** An attempt was made to modify the default Id and Config class records, which is not allowed. Default collection records for Id and Config classes cannot be deleted or modified.
- **User Action:** Any attempt to modify records for Id and Config classes will fail.

Message: NOFILE, unable to open file <filename>

- **Explanation:** The configuration file could not be opened. The specification for the file to be opened is determined by the translation of the logical name ACMS$MGMT_CONFIG, or, if the logical name is not defined, the default file name is SYS$SYSROOT:[SYSEXE]ACMS$MGMT_CONFIG.ACM. The ACMSCFG utility will ask you whether or not you want to create a new file.
- **User Action:** If the file does not exist and you would like to have a new file created with default values, respond to the prompts. If the file does exist, investigate why it could not be opened by the ACMSCFG utility.

Message: NO_QUAL, insufficient arguments – no qualifiers, please correct and resubmit

- **Explanation:** A command that requires at least one qualifier was submitted without any qualifiers.
- **User Action:** Modify the command to contain at least one qualifier. To learn more about ACMSCFG commands their qualifiers, use the ACMSCFG **`HELP`** command.

Message: NORECDEL, record does not exist, no record deleted

- **Explanation:** An attempt was made to delete a record that does not exist.
- **User Action:** Correct the command to include the correct record identifiers. For collection records, entity type, class name, and entity name uniquely identify records; for trap records, entity type, entity name, and parameter name uniquely identify records.

Message: NORECUP, record does not exist, no record updated

- **Explanation:** An attempt was made to update a record that does not exist.
- **User Action:** Correct the command to include the correct record identifiers. For collection records, entity type, class name, and entity name uniquely identify records; for trap records, entity type, entity name, and parameter name uniquely identify records. For interface records, interface type uniquely identifies the record.

Message: NOT_ADDED, no record added

- **Explanation:** This message is obsolete.
- **User Action:** No action is required.

Message: NOUPDATES, no changes to make, no updates made

- **Explanation:** A command was issued to update the parameter record, but no parameters were specified for update. No changes were made to the record.
- **User Action:** Modify the command to include at least one parameter to be modified.

Message: NULLQUAL, null qualifier <qualifier>, nothing to do

- **Explanation:** A qualifier was provided that requires a value, but no value was specified.
- **User Action:** Modify the command to include a valid value with the qualifier. To learn more about valid values for the qualifier, use the ACMSCFG **HELP** command.

Message: PARAM_INVALID, parameter <parameter name> is not valid for this entity

- **Explanation:** This status code has not been implemented yet.
- **User Action:** No action is required.

Message: PARAM_REQ, parameter type is required, please resubmit command

- **Explanation:** This status code has not been implemented yet.
- **User Action:** No action is required.

Message: PAST_BEGIN, begin time must be in the future

- **Explanation:** This status code has not been implemented yet.
- **User Action:** No action is required.

Message: TIMERID_REQ, timer id is required, please resubmit command

- **Explanation:** This status code has not been implemented yet.
- **User Action:** No action is required.

Message: UNKCLASS, unrecognized class <class>

- **Explanation:** The class type indicated is unrecognizable.
- **User Action:** Modify the **/CLASS** qualifier to include a valid class. To learn about valid classes, use the ACMSCFG **HELP** command.

Message: UNKENTITY, unrecognized entity <entity>

- **Explanation:** The entity type indicated is unrecognizable.
- **User Action:** Modify the **/ENTITY** qualifier to include a valid entity type. To learn about valid entity types, use the ACMSCFG **HELP** command.

Message: UNKNOWN_OBJ, unknown obj <object>

- **Explanation:** The command object specified is not valid.
- **User Action:** Modify the command to include a valid command object. The valid command objects vary depending on the command verb. Use the ACMSCFG **HELP** command to learn more about valid command verbs and objects.

Message: UNKPARAM, unrecognized param <param>

- **Explanation:** The parameter indicated is unrecognizable.
- **User Action:** Modify the **/PARAMETER** qualifier to include a valid parameter name. To learn about valid parameters, use the ACMSCFG **HELP** command.

Message: UNKQUAL, unrecognized qualifier <qualifier>, please correct and re-execute command

- **Explanation:** The qualifier indicated is unrecognized.
- **User Action:** Modify the command to include a valid qualifier. To learn more about valid qualifiers, use the ACMSCFG **HELP** command.

Message: UNKVERB, unrecognized verb <verb>, please correct and resubmit

- **Explanation:** The verb indicated is unrecognized.
- **User Action:** Modify the command to include a valid verb. To learn more about valid verbs, use the ACMSCFG **HELP** command.

Message: VALTOOBIG, value is too large: <value>, please correct and resubmit

- **Explanation:** A value was provided that is greater than the allowed maximum.
- **User Action:** Modify the command to include a valid value with the qualifier. To learn more about valid values for the qualifier, use the ACMSCFG **HELP** command.

Message: VALTOOSMALL, value is too small: <value>, please correct and resubmit

- **Explanation:** A value was provided that is less than the allowed minimum.
- **User Action:** Modify the command to include a valid value with the qualifier. To learn more about valid values for the qualifier, use the ACMSCFG **HELP** command.

C.4. ACMSSNAP Messages

The following error messages pertain to the ACMSSNAP process.

Message: FILEISOPEN, a file is already open. Use the CLOSE command to close the current file

- **Explanation:** A request was made to process an ACMS data snapshot file that is already open. Only one file can be open at a time.
- **User Action:** No action is required.

Message: NOFILEOPEN, no file is open. Use the OPEN command to open a file first

- **Explanation:** A request was made to process an ACMS data snapshot file, but no file has been opened. A data snapshot file must be open in order for the command to be executed.

- **User Action:** Use the ACMSSNAP **OPEN** command to open the file; then reissue the command.