SG24-4873-00

RS/6000 SP Monitoring: Keeping It Alive

April 1997

IBM

International Technical Support Organization
Poughkeepsie Center

International Technical Support Organization

SG24-4873-00

RS/6000 SP Monitoring: Keeping It Alive

April 1997

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix I, "Special Notices" on page 259.

First Edition (April 1997)

This edition applies to PSSP Version 2, Release 2 for use with the AIX Version 4 Operating System.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

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

Contents

Figures . . . ix

Tables . . . xiii

Preface . . . xv
The Team That Wrote This Redbook . . . xv
Comments Welcome . . . xvi

Part 1. HA Infrastructure . . . 1

Chapter 1. Overview . . . 3
1.1 Topology Services . . . 3
1.1.1 Topology Services Terminology . . . 5
1.1.2 Adapter Membership Group Algorithm . . . 6
1.1.3 The Topology Services Daemon . . . 7
1.2 Group Services . . . 8
1.2.1 The Group Services Daemon . . . 8
1.2.2 The Group Services Domain . . . 10
1.2.3 Group Services Groups . . . 14
1.3 Event Management . . . 16
1.3.1 Event Management Subsystem . . . 19
1.3.2 The Event Management Daemon . . . 21

Part 2. Subsystems . . . 25

Chapter 2. Resource Monitors . . . 27
2.1 Resource Monitor Overview . . . 27
2.2 Resource Monitor Objectives . . . 29
2.3 What Are Resources? . . . 30
2.4 Resource Representation . . . 30
2.4.1 Resource Monitor Modes . . . 30
2.4.2 Resource Variable Types . . . 31
2.4.3 Resource Variable Data Types . . . 33
2.4.4 Information Exchange of Resource Variables . . . 35
2.4.5 Resource Variable Names . . . 38
2.4.6 Classes of Resource Variables . . . 42
2.4.7 Resource Variable Instance Vector . . . 44
2.4.8 Event Management SDR Classes . . . 46
2.5 Resource Variable Definition . . . 48
2.6 Pseudocode for a Resource Monitor . . . 50
2.7 Event Management Configuration . . . 51
2.8 Event Generation . . . 57
2.9 Event Registration and Notification . . . 58
2.10 Predicates . . . 61
2.11 PSSP 2.2 Resource Monitors . . . 63
2.11.1 External Resource Monitors . . . 64
2.11.2 Internal Resource Monitors . . . 65
2.12 Support for System Partitioning . . . 66
2.13 Extensibility . . . 67
2.14 Resource Monitor API . . . 67

2.15 Summary of Functional Flow 68

Chapter 3. Problem Management Subsystem 71
3.1 Introduction to the Problem Management Subsystem 71
3.1.1 What Can You Monitor? 72
3.1.2 What Can You Do When System Resources Are Changed? 72
3.1.3 What Kind of Monitoring Tools Can You Use? 72
3.2 Create Your Own Monitor Using pmand 72
3.2.1 Scenario 1 73
3.2.2 Get a Rough Idea 73
3.2.3 Find Resource Variables You Might Want 73
3.2.4 Find the Resource Variable You Want 74
3.2.5 Find the Instance Vector You Want 75
3.2.6 Use the pmandef Command 76
3.2.7 Try Your Own Monitor 77
3.3 Create Your Own Monitor Using pmand and pmanrmd 77
3.3.1 Scenario 2 78
3.3.2 Create a pmanrmd Configuration File 78
3.3.3 Find the Instance Vector You Want 80
3.3.4 Find the Type of Your Resource Variable 81
3.3.5 Find the Structured Byte String You Want 81
3.3.6 Use the pmandef Command 81
3.3.7 Try Your Own Monitor 82
3.3.8 Scenario 3 82
3.3.9 Use the pmandef Command 82
3.3.10 Try Your Own Monitor 82
3.4 Problem Determination 83
3.4.1 Are You Authorized? 83
3.4.2 Is the Problem Management Subsystem Active? 84
3.4.3 Is Your Event Subscribed? 86
3.4.4 Is Your Event Ready to Be Acted On? 87
3.4.5 Is Your Event Active and Correct? 88
3.5 Hints and Tips for Problem Management Subsystem Commands 90
3.5.1 Commands for The Problem Management Daemon 91
3.5.2 Commands for the Problem Management Resource Monitor Daemon 92
3.6 Short Examples 93
3.6.1 The File System Monitor 93
3.6.2 The Process Monitor 94
3.6.3 /etc File Changed Monitor 94
3.6.4 Server Monitor 94

Chapter 4. Application Program Interfaces (APIs) 97
4.1 Some Details Before We Start 97
4.2 Example EMAPI Programs 98
4.2.1 Utility Functions and Construction of EMAPI Clients 98
4.2.2 The lsemv Program 101
4.2.3 The getemv Program 111
4.2.4 The monemv Program 115
4.2.5 EMAPI Summary 119
4.3 Sample RMAPI Program 119
4.3.1 The httprtA Program 119
4.3.2 Resource Monitor Configuration 128
4.3.3 RMAPI Summary 132
4.4 Corrections and Clarifications 133

Part 3. Tools . . . 135

Chapter 5. SP Perspectives GUI . . . 137
5.1 Introduction . . . 137
5.1.1 What Tasks Can Perspectives Perform? . . . 137
5.1.2 How Is Perspectives Invoked? . . . 138
5.2 Using the Event Perspective . . . 139
5.2.1 Example 1: Monitoring Memory Usage . . . 140
5.2.2 Example 2: Monitoring File System Size . . . 148
5.2.3 More about Using the Event Perspective . . . 152
5.2.3.1 Some Terminology Considerations . . . 152
5.2.3.2 Where Is My Event Definition Stored? . . . 154
5.2.3.3 Can I Modify or Delete an Event Definition? . . . 156
5.2.3.4 Summary of Authorizations Needed . . . 156
5.2.3.5 SDR and Some Helpful SDR Commands . . . 156
5.2.3.6 Runtime Error Message Help Facility . . . 158
5.2.4 Example 3: Monitoring CPU Usage . . . 160
5.2.5 The pcount and xpcount Resource Variables . . . 162
5.2.6 Example 4: Monitoring a Daemon . . . 165
5.2.7 Example 5: Monitoring the Health of Your System . . . 167

Chapter 6. Problem Management SNMP Subagent . . . 173
6.1 Problem Management SNMP Support . . . 174
6.1.1 Understanding Network Management . . . 174
6.1.2 Understanding Simple Network Management Protocol (SNMP) . . . 175
6.1.3 Understanding the Management Information Base (MIB) . . . 175
6.1.4 Understanding the SNMP Multiplexor (SMUX) Protocol . . . 176
6.1.5 SP SNMP Multiplexor Agent . . . 176
6.1.6 How Problem Management Uses SNMP . . . 177
6.2 Installation and Configuration . . . 178
6.2.1 Configuring the Ports . . . 178
6.2.2 Configuring SNMP Agents . . . 179
6.2.2.1 Community Name . . . 179
6.2.2.2 SMUX Peer Configuration . . . 181
6.2.3 Configuring the SNMP Manager . . . 181
6.2.4 Configuring the MIB . . . 182
6.2.5 Configure Subagent sp_configd . . . 184
6.2.6 Customizing Traps . . . 186
6.2.7 Topological View of SP . . . 188
6.2.8 The ibmSP MIB . . . 189
6.2.9 Viewing the ibmSP MIB . . . 194
6.2.9.1 The snmpinfo Command . . . 194
6.2.9.2 NetView for AIX MIB Browser . . . 196
6.3 The AIX Error Log . . . 197
6.3.1 SNMP Subagent (sp_configd) Monitor of AIX Error Log . . . 197
6.3.2 Subsystem Logs . . . 201
6.3.2.1 High Availability Subsystem Logs . . . 201
6.3.2.2 SNMP Agent and sp_configd Subagent Logs . . . 201
6.3.2.3 SNMP Log . . . 203
6.4 Monitoring SP Resources: The Mechanics . . . 205
6.4.1 Basic Example - Monitoring Server Key Switch . . . 207

Appendix A. Overview of NetView for AIX Ruleset Editor . . . 221

Appendix B. Makefile for the Resource Monitor Examples 223

Appendix C. User Response Time 225
C.1 Avoiding Surprises by Monitoring User Response Times 225
C.2 Sample Response Time Monitor: HTTP Client 226
C.2.1 httprtB, an HTTP Response Time Server 226
C.2.2 How to Improve httprtB 233

Appendix D. Essence of the Event Management and Problem Management Subsystems 235
D.1 Event Management Subsystem SDR Class 235
D.2 Event Management Resource Monitor Class 235
D.2.1 What Can You Get? 235
D.2.2 Why Is It Useful? 236
D.3 Event Management Resource Class Class 237
D.3.1 What Can You Get? 237
D.3.2 Why Is It Useful? 238
D.4 Event Management Instance Vector Class 238
D.4.1 What Can You Get? 238
D.4.2 Why Is It Useful? 239
D.5 Event Management Resource Variable Class 240
D.5.1 What Can You Get? 240
D.5.2 Why Is It Useful? 240
D.6 Event Management Structured Byte String Class 241
D.6.1 What Can You Get? 241
D.6.2 Why Is It Useful? 241
D.7 Problem Management Daemon (pmand) 242
D.7.1 SDR Class 242
D.7.2 Commands 242
D.8 Problem Management Resource Monitor Daemon (pmanrmd) 243
D.8.1 SDR Class for pmanrmd 243
D.8.2 Configuration File 244
D.8.3 Resource Variables 244
D.8.4 Commands 244

Appendix E. The IBM.PSSP.Prog.pcount Resource Variable 245
E.1 Limitations 246
E.1.1 Instance Vector Wildcarding 246
E.2 Related Resource Variable IBM.PSSP.Prog.xpcount 246

Appendix F. The IBM.PSSP.Prog.xpcount Resource Variable 249
F.1 Limitations 250
F.1.1 Instance Vector Wildcarding 250
F.2 Related Resource Variable IBM.PSSP.Prog.pcount 250

Appendix G. SNMP-Related Request For Comments 253
G.1 How to Get RFCs 256

Appendix H. How to Get the Examples in This Book 257
H.1 Diskette Version 257
H.2 FTP Site 257
H.3 WWW Site 258

Appendix I. Special Notices 259

Appendix J. Related Publications 261
J.1 International Technical Support Organization Publications 261
J.2 Redbooks on CD-ROMs 261
J.3 Other Publications 261

How to Get ITSO Redbooks 263
How IBM Employees Can Get ITSO Redbooks 263
How Customers Can Get ITSO Redbooks 264
IBM Redbook Order Form 265

List of Abbreviations 267

Index 269

ITSO Redbook Evaluation 271

Figures

1. Topology Services Structure 4
2. Group Services Structure 8
3. Group Services Daemon Structure 9
4. Event Management Subsystem Structure 17
5. Event Management Basic Structure 28
6. Resource Monitor Types 31
7. Resource Variable Types 32
8. Resource Variable Types 33
9. Structured Byte String 34
10. Resource Variable Data Types 35
11. Information Exchange of Resource Variables 36
12. Set of PTX/6000 Statistics 37
13. Partial Listing from xmpeek 38
14. Resource Variable Names 39
15. Example for Creating Resource Variables 40
16. Example Message File rmapi_smp.msg 42
17. Resource Variable Classes 43
18. Example EM_Resource_Class Entries 44
19. Event Management Instance Vector 45
20. Instance Vector Example 46
21. SDR Classes 46
22. Event Management SDR Classes 47
23. Create EM_Resource_Monitor Entries 49
24. Pseudocode of a Daemon-Based Resource Monitor 50
25. Event Management Configuration Steps 52
26. Compile the SDR Data to EMCDB 53
27. Event Management Database-Related Subdirectories 53
28. Event Management Database Subdirectories with New EMCDB 54
29. Shutdown of the Event Management Daemons 54
30. Restart of the Event Management Daemons 55
31. List of the lssrc Command 56
32. Observation Interval Example 58
33. Event Perspective View Condition Dialog Box 60
34. Event Notification Log 60
35. Monitoring Resource Variables with PTX 61
36. External Resource Monitors 65
37. Internal Resource Monitors 66
38. Event Management Information Flow 68
39. Flow of Using the pmand Command 76
40. File pmanrmd.conf 79
41. File sysctl.pman.acl 84
42. Status of Problem Management Subsystem Daemons and pmanctrl Command Options 86
43. Flow of Problem Determination 90
44. Commands for the Problem Management Daemon 92
45. Commands for the Problem Management Resource Monitor Daemon . 93
46. The Makefile File for Constructing lsemv, getemv, and monemv 99
47. Excerpts from util.h Showing Utility Functions 100
48. Two Examples of the lsemv Program 101
49. Another Example of lsemv 102
50. The main() Function for lsemv 103

51. The processNames() Function for lsemv 104
52. The processResponses() Function for lsemv 106
53. The processOneResponse() Function for lsemv 108
54. The processQerrList() Function for lsemv 109
55. The processDefinedList() Function for lsemv 110
56. Using getemv to List All Instances of Names That Start with IBM . . . 111
57. Using getemv to List Specific Instances of One Resource Variable . . 111
58. The processName() Function for getemv 112
59. The processOneResponse() Function for getemv 113
60. The processValueList() Function for getemv 114
61. Example of monemv Monitoring Key Mode Switches on Nodes 115
62. Use of spmon to Alter Key Mode Switches on Nodes 116
63. The processResponses() Function for monemv 117
64. The processEventList() Function for monemv 118
65. The main() Function for httprtA 121
66. The Top of the DoIt() Function for httprtA 122
67. The Middle of the DoIt() Function for httprtA 123
68. The Bottom of the DoIt() Function for httprtA 124
69. The ConnectAndGo() Function for httprtA 125
70. The Top of the Monitor() Function for httprtA 126
71. The Bottom of the Monitor() Function for httprtA 127
72. The SendRMData() Function for httprtA 128
73. The Message Catalog File for httprtA 130
74. Part of the SDR Configuration Script for httprtA 131
75. Output for WWW.HTTP.ResponseTimeMon.delay from the lsemv Command 132
76. The Perspectives GUI Launch Pad 138
77. The Event Perspective Primary Window 140
78. The Definition Page of the Create Event Definition Notebook 142
79. The Definition Page after Selecting a Predefined Condition 143
80. The Completed Definition Page in Example 1 145
81. The View Event Notification Log Window 146
82. The Response Options Page 148
83. The Create Condition Dialog Box 149
84. Resource Variables Classes 150
85. Displaying the Resource Variables of a Resource Class 150
86. Interrelation of Event Perspective with the Underlying Subsystems . . 155
87. Deleting a Condition from the SDR 156
88. Displaying the Class Names in the SDR 157
89. Displaying the Objects and Attributes of a Class 158
90. The Event Perspective Pop-Up Error Window 159
91. The SP Perspectives Help Window with Message Help 159
92. The SP Perspectives Help Window with Detailed Error Information . . 160
93. Process Flags: p_flags Field in the /usr/include/sys/proc.h File 164
94. Sample Output of the ps Command 164
95. SP Distributed Management 173
96. Object Identifier Tree - MIB Structure 176
97. IBM Private SP MIB 177
98. Relationship between NetView for AIX and SNMP Agents 178
99. SNMP Configuration 182
100. Loading a MIB 183
101. Loading the MIB from File 183
102. snmp and sp_configd 184
103. Add the ibmSP Enterprise 186
104. Adding Predefined Traps 187

105. Topological Representation of the IBM SP 189
106. MIB Browser - ibmSPConfig 190
107. MIB Browser - ibmSPErrlogVars 192
108. MIB Browser - ibmSPEMVariables 193
109. MIB Browser 196
110. NetView Subagent Event (Page 1) 199
111. NetView Subagent Event (Page 2) 200
112. Condition 206
113. PMAN Subsystem 206
114. Registration for Event 208
115. Event Description 211
116. Ruleset Logic 212
117. Trap Configuration Window 213
118. NetView Control Desk - Events Application 213
119. Ruleset Editor 214
120. Block Event Behavior 214
121. Trap Settings 215
122. Thresholds Setup Window 216
123. Action Node 217
124. Dynamic Workspace 218
125. NetView for AIX Ruleset Editor 222
126. Interesting Part of httprtB′s main() Function 227
127. Function AcceptClient() in httprtB 228
128. Processing Loop After Client is Accepted in httprtB 230
129. Interesting Part of Function FetchIt() 231
130. First Half of GetIt() Preparing to Get HTTP Object 232
131. Bottom Half of GetIt() Where HTTP Object is Retrieved 233
132. Installing Examples from Diskette to the Recommended Location . . . 257
133. Installing Examples to the Recommended Location Using FTP 257

Tables

1. EM_Resource_Monitor 63
2. Equivalent Terms between the SDR and the Event Perspective 152
3. Equivalent Attribute Names between Problem Management and Event Perspective 154
4. IBM Default Predefined EM_Condition Class Objects 170
5. Event Management Resource Monitor Class (1 of 3) 236
6. Event Management Resource Monitor Class (2 of 3) 236
7. Event Management Resource Monitor Class (3 of 3) 236
8. Event Management Resource Class Class 237
9. Event Management Instance Vector Class 239
10. Event Management Resource Variable Class 240
11. Event Management Structured Byte String Class 241

Preface

This redbook contains detailed information about the configuration and use of the new RS/6000 SP High Availability Infrastructure. It also includes examples with NetView for AIX in order to integrate the SP in an environment managed by a central manager, such as NetView. Programming examples that use the new APIs available with POWERparallel System Support Programs Version 2, Release 2 are provided to help you develop your own code.

This redbook is written in a how-to style, providing examples that illustrate the potential of the new features. It is divided into three parts: basic concepts and infrastructure, subsystems that make up this new infrastructure, and the front-end interfaces that allow you to manage your system. It is intended for system administrators who need to manage an SP system running PSSP 2.2. It is also a good starting point for anyone wanting to learn about the new capabilities of the High Availability Infrastructure.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Marcelo R. Barrios is an Advisory International Technical Support Organization (ITSO) Specialist for RS/6000 at the Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of RS/6000 SP. Before joining the ITSO, Mr. Barrios worked as an assistant professor in the Electronics Department of Santa Maria University, Valparaiso, Chile. In 1993, he joined IBM as a Marketing Specialist in the RS/6000 Unit, IBM Chile.

Paul G. Crumley is a Senior Programmer in IBM′s Research Division at the Center for Scalable Computing Solutions, the group that developed many parts of the SP. Before joining IBM, Paul worked at Transarc, the developer of DFS and Encina. Previous to that, he led various research projects at the Information Technology Center at Carnegie Mellon University. The ITC developed a number of distributed systems, including the AFS file system, the ATK multimedia GUI system, and AMS, a multimedia messaging system.

Abbas Farazdel is a consultant and SP specialist with the Scientific and Technical Systems and Solutions (STSS) group at the IBM Thomas J. Watson Research Center. He is responsible for generating worldwide business opportunities for IBM in the area of numerically intensive scientific and technical computing. Abbas Farazdel′s recent focus has been to help clients evaluate, plan for, and implement advanced highly and massively parallel high performance computing and open systems technologies. He holds an M.S. degree in Physics and a Ph.D. degree in Computational Chemistry from the University of Massachusetts.

Hajo Kitzhoefer is an SP specialist at the RS/6000 and AIX Competence Center, IBM Germany. He holds a Ph.D. degree in Electrical Engineering from the University of Bochum (RUB). He has worked at IBM for six years. His areas of expertise include RS/6000 SP, SMP, benchmarks, networking, PC integration, and AIX/ESA.

Vijji Korlipara is an SP specialist at the IBM Software Technical Center, IBM Sweden. She holds a degree in Joint Honours in Science (Statistics) from the University of Salford, Manchester. She joined IBM in 1988 as a Network Software Developer in the UK, and has since worked on distributed heterogeneous networks. She now specializes in the system and network management of distributed systems on the RS/6000 SP AIX platform. Vijji was the instructor for NetView for AIX, IBM UK education.

Yoshimichi Kosuge is an Advisory I/T Specialist at the AIX Technical Support Center, IBM Japan. He holds a degree in Electrical Engineering from Waseda University, and later studied Computer Science. He joined IBM Japan in 1982 as an LSI designer. He has worked on ES/9000 microcode, and in OS/2 and AIX programming. His current job is in AIX second-level support and professional service for customers.

Thanks to the following people for their invaluable contributions to this project:

Endy Chiakpo
International Technical Support Organization, Poughkeepsie Center

Peter Kes
International Technical Support Organization, Poughkeepsie Center

IBM PPS Lab Poughkeepsie
Mike Browne
Michael Schmidt
Stephen Tovcimak
James Gilman
Tim Race
Peter Badovinatz

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 271 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Home Pages at the following URLs:

For Internet users	`http://www.redbooks.ibm.com`
For IBM Intranet users	`http://w3.itso.ibm.com/redbooks`

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Part 1. HA Infrastructure

Chapter 1. Overview

POWERparallel System Support Programs Version 2.2 provides the foundation for the new High Availability Infrastructure on RS/6000 SP. Major code changes offer high performance *and* high availability, in an attempt to achieve a complete and robust commercial environment.

Many new features are included in this version. Perhaps the biggest changes are in the basic components, including Topology Services, Group Services, Resource Monitors, and Event Management. These components are the basic bricks used to build the new infrastructure; they are explained in the first two chapters.

The rest of the chapters will take you through the new tools available to monitor your system, such as Problem Management and Perspectives. An important feature not available in the previous versions of PSSP is the availability of Application Programming Interfaces (APIs). We include one chapter and appendices that will help you write your own code to tailor your system, or to create powerful new monitors.

We provide examples to make it easier for you to start working with these new tools. There are no extensive discussions of the concepts underlying the features, since the focus is on “how to” information, that is, how to create monitors, or how to monitor your system. The examples in this redbook can be retrieved via FTP or the Web, as explained in Appendix H, “How to Get the Examples in This Book” on page 257.

Enjoy the reading and do not poll your system, monitor it!

1.1 Topology Services

The Topology Services subsystem provides the foundation for the PSSP 2.2 High Availability Infrastructure. It is part of the internal components, also known as *Internal Phoenix Components*. These components provide essential function within the HA infrastructure to support the higher-level Group Services and Event Management subsystems. Their primary purpose is to maintain information about the topology of the communication networks, including accessibility of communication adapters and processors, functions usually referred to as “adapter membership” and “node membership.”

These components exist as a rough hierarchy of services:

1. At the “bottom” is *Adapter Membership*. This is the level that is responsible for passing *heartbeats* or *pulses* among all of the communications adapters to ensure that the available networks remain accessible from all nodes. Rather than a broadcast or all-to-all set of pulses, each node heartbeats only with a specified set of nodes. The combined coverage of the heartbeats results in all nodes and all networks being monitored on a regular basis.
2. Above this pulsing layer is the *Topology Manager*. This component controls the heartbeat function of Adapter Membership by specifying the topology of the network for heartbeating, and tells each node what the set of nodes is with which it should be concerned. It maintains a full graph of the system, and updates it as links fail or return.

3. *Node Membership* reads the graph generated by the Topology Manager to construct a graph of all accessible nodes in the system. It also interfaces with the SP hardware monitor to collect information about node hardware failure.

4. Finally, the *Reliable Messaging* component provides a communication facility to carry messages even across node and communication adapter failures. As part of this function, it uses information from Node Membership to build *routing maps* of the system. This allows it to use intermediate nodes to route messages if two nodes are not able to communicate.

To enhance the performance and scalability of these layers, the dynamic topology information is kept in a loosely-consistent manner. Each node is given the same static topology map from the System Data Repository (SDR), which is essentially a graph of the system configuration. However, each node independently constructs a dynamic (or connection) topology map over this that shows the accessibility from this node to all other nodes at that moment. As the state of the nodes and communications adapters changes over time, the nodes will gradually update their information as they attempt to communicate with a node, or receive updated information passed among the Topology Managers.

This is part of the reason why client subsystems cannot directly access these functions, because they would have to provide their own protocols for developing a consistent, ordered view of the system. That is done by Group Services, which maintains *System Membership Groups* (HostMembership and enMembership), which reflect an ordered view of the state of the nodes and communication adapters. Event Management uses these groups to generate events that signal when the states of these entities should change.

Figure 1 shows a diagram with the main components of Topology Services.

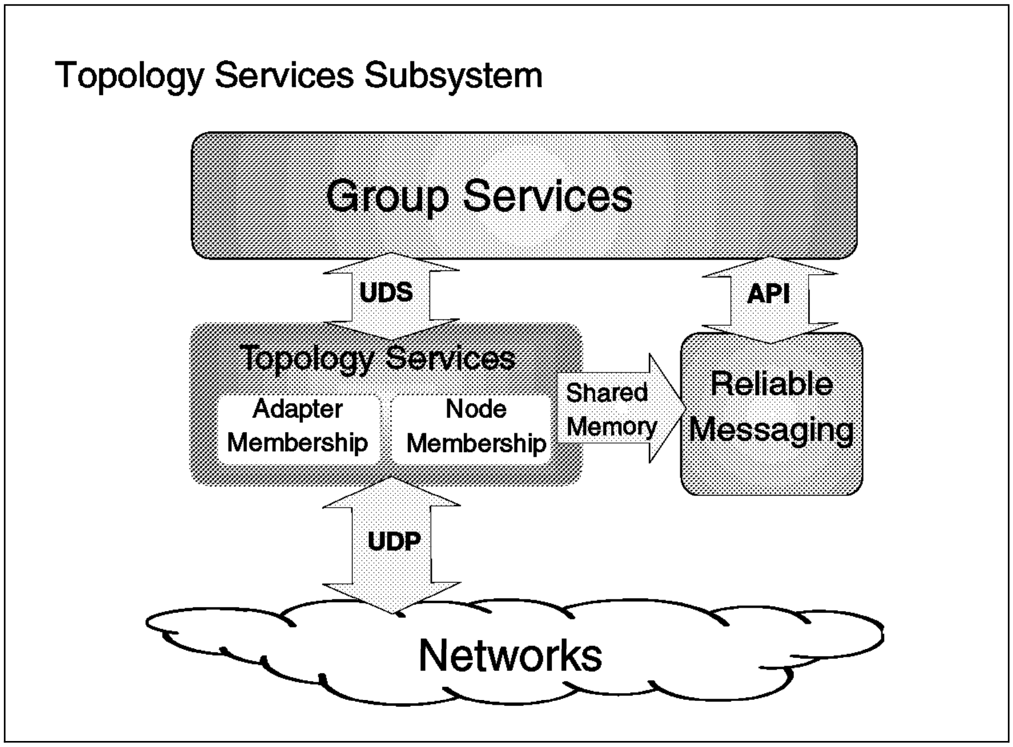


Figure 1. Topology Services Structure

The design of Topology Services is based on a route-discovering mechanism that provides a way to get information about a set of nodes that use different network interfaces and are connected to multiple networks at the same time. The current implementation of Topology Services uses the SP Ethernet and the Switch network to discover whether a node has failed or there is a problem trying to reach it. The information used to create the routing table is based on the SDR information, specifically the SP network and adapter information.

From the Topology Services perspective a system consists of a number of workstations or nodes connected through a number of networks. Each node can reach any other node through any number of networks. A node can have multiple network adapters and thereby be connected to several different networks.

The static configuration used to build the topology information is taken from the SDR, so Topology Services has a dependency on this subsystem, at least when it starts up. Also, Topology Services assumes that the information is available and static at the time it reads it from the SDR.

1.1.1 Topology Services Terminology

Before we continue, we must define some new terms used in Topology Services:

Adapter Membership Group (AMG)

This is an association of adapters connected to the same network. There is one AMG per network.

Node Membership Group (NMG)

There is one NMG per Topology Services domain or system partition, made up of all nodes belonging to that partition.

Group Leader (GL) The adapters within a network have a predefined priority ordering. The adapter with the highest IP address, and the highest priority, is the GL. It is responsible for maintaining the topology and connectivity information of a group, and distributing that information to the other members. The GL will also attempt to recruit other adapters into the group, or to merge with groups that have a GL with a higher IP address.

Crown Prince (CP) The CP monitors the availability of the GL and assumes the leadership role in case the GL fails. The CP has the second-highest IP address in the group.

Singleton This is a group with one member. All adapters initialize into a Singleton before they join a larger group.

Proclaim Until all the configured members of a network have joined the AMG, the leader is always in search of new adapters to join the group. This is accomplished by periodically broadcasting a *proclaim* message to all lower-priority adapters in its network that have not already joined the group.

As mentioned earlier, Topology Services maintains complete routing information on every node in the system. The following algorithm is used to maintain that topology information:

- If this is a node connected to multiple adapter groups, that is, multiple networks, it will periodically send a message to the GL in each adapter group telling it which other groups it can talk to. It also sends this message immediately whenever this information changes.
- If the node is a GL, then in addition to maintaining the information for every node in the group, it keeps track of the current adapter group membership. Periodically, and whenever this information changes, it will send it to all nodes in the group.
- Each node keeps the latest state message that it receives from the adapter group.
- If the node is connected to multiple adapter groups, it will forward this message to all those groups. Because each state message has a time stamp, only messages with a new time stamp will be forwarded.
- Each node receiving this message will incorporate the information in its internal data structures.

1.1.2 Adapter Membership Group Algorithm

An AMG is created and maintained for each network in the system by using the following algorithm:

1. Each node acquires the node list from the SDR on the Control Workstation.
2. Each adapter initializes itself into a Singleton group.
3. GLs periodically send proclaim messages to all lower-priority adapters.
4. Lower-priority GLs respond to proclaim messages by requesting to join that group.
5. The GL receives those requests and incorporates their group topology.
6. The GL notifies all members of the newly forming group to prepare to commit the new group topology.
7. Group members acknowledge this “prepare to commit” message.
8. The GL waits for all acknowledgements, then sends a message to commit the new topology.
9. Members acknowledge receipt of the commit message.
10. The GL then delivers the new topology to all members.
11. The members then determine the new GL, CP, and neighbors.
12. All members begin sending and receiving heartbeats to and from neighbors.

Once a group has been committed, each adapter computes the address of its upstream_neighbor, and its downstream_neighbor. Each adapter is responsible for listening for the heartbeats of its upstream_neighbor and sending heartbeats to its downstream_neighbor.

There are two exceptions to this statement: the GL, and the adapter with the lowest IP address. The GL receives heartbeats from the member with the lowest IP address, making the sequence circular. In this way, the traffic over the network is kept to a minimum, sending live messages only to neighbors.

> **Attention**
>
> It should be clear that neighbors are defined as such by their IP addresses, not by physical proximity.

1.1.3 The Topology Services Daemon

The Topology Services subsystem is provided by a daemon running in each node connected to, and part of, the topology domain. This daemon, called hatsd, is controlled as an SRC subsystem.

Topology Services is a partition-sensitive subsystem. Each partition is handled as a separate system, so the routing information is limited to each partition. Topology Services has a separate group of daemons, logs, and configuration files for each partition.

To query the status of all daemons running in the Control Workstation, you may use the following command:

```
[sp21en0:/]# lssrc -g hats
Subsystem         Group            PID     Status
 hats.sp21en0     hats             37412   active
 hats.sp21en1     hats             27430   active
[sp21en0:/]# ps -ef | grep hatsd
    root 13172 28638   2 14:32:44 pts/19  0:00 grep hatsd
    root 27430  3148   0   Sep 13      - 62:10 /usr/lpp/ssp/bin/hatsd
    root 37412  3148   0   Sep 13      - 68:16 /usr/lpp/ssp/bin/hatsd
```

This command will give you the list of daemons running on the system. Each daemon manages a different topology domain or partition.

The Topology Services daemon provides availability information to the Group Services subsystem through a UNIX Domain Stream Socket (UDS). You may find this special file in the /var/ha/soc/hats directory. In our example, we have two partitions. Therefore, there are two hats daemons running. For each daemon, there is one UNIX socket to connect its Group Services counterpart, as we can see in the following command output:

```
[sp21en0:/var/ha/soc/hats]# ls -l
total 0
srwxrwxrwx   1 root     system      0 Sep 13 10:28 server_socket.sp21en0
srwxrwxrwx   1 root     system      0 Sep 13 10:28 server_socket.sp21en1
```

In this example there are two sockets, one for each hats daemon running in a different partition.

Topology Services provides availability information to Group Services using the UNIX socket. This means the connection with Group Services is always local. In this way, Group Services may detect when Topology Services daemon has died prematurely. The hats daemon also provides routing information to the Reliable Messaging library, using a shared memory segment.

1.2 Group Services

Group Services (GS) is a distributed subsystem of the IBM POWERparallel System Support Programs (PSSP) on the RS/6000 SP. It is one of several subsystems in PSSP that provide a set of high availability services.

The function of the GS subsystem is to provide other subsystems with a distributed coordination and synchronization service. The structure of GS is described in Figure 2.

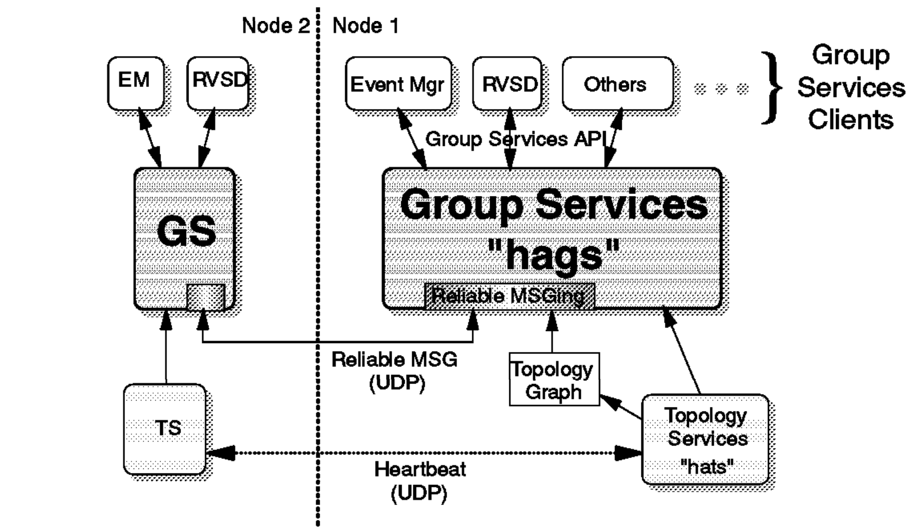


Figure 2. Group Services Structure

1.2.1 The Group Services Daemon

The GS daemon, called hagsd, is part of the GS subsystem and provides most of its services. One instance of this daemon executes on the Control Workstation for each system partition. Another executes on every node of a system partition. It is under the control of the System Resource Controller (SRC). Its structure is shown in Figure 3 on page 9.

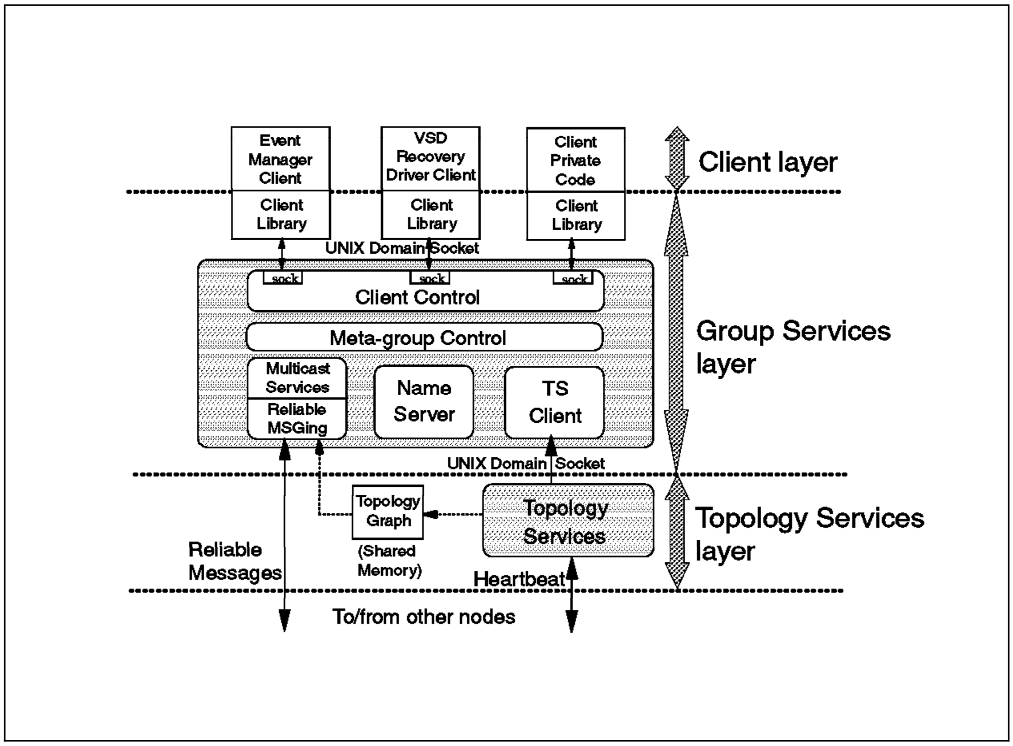


Figure 3. Group Services Daemon Structure

The GS daemon obtains information from Topology Services via a UNIX domain socket, located at /var/ha/soc/hagsd.<syspar_name>. It also communicates with the hagsd daemon on other nodes via a UDP/IP socket. The UDP port can be obtained by using the following command:

```
[sp21en0:/]# SDRGetObjects Syspar_ports
subsystem           port
hats                10000
hags                10001
haem                10002
```

The GS subsystem subscribes to the Adapter Membership and Node Membership group from Topology Services, locally using the UNIX Domain Stream Socket. If hatsd is not running, hagsd will not start. It will log an error in /var/ha/log/gs.default.<syspar_name> log file. Then it will log errors continually until Topology Services is available. The entries in the log file are as follows:

```
[sp21n01:/var/ha/log]# cat gs.default.sp21en0
hb_init:  hb_init_communication failed
hb_init:  hb_init_communication failed
hb_init:  hb_init_communication failed
hb_init:  hb_init_communication failed
hb_init:  hb_init_communication failed
status.sp21en0] previous incarnation [5] nothing special.
[TOD(Sep 13 10:43:56)]( TRACE_FYI ) in main:  Our NodeId is 1.6
[TOD(Sep 13 10:43:56)]( TRACE_FYI ) in Establish_Resource_Limits(void):
Successfully modified priority from [71] to [38] for pid [17350]
/usr/lpp/ssp/bin/hagsreap input arguments: hags sp21en0 Erase
hagsreap parsed arguments: hags sp21en0 Erase 5242880 1 /var/ha/log
/var/ha/run/hags.sp21en0
SizeLimit = 5242880
    n    CumSize       MB    weight       size name
    1       1060    0.001      0.00       1060 hags_1_6.sp21en0
    2       2551    0.002      1.00       1491 hags_1_5.sp21en0
    3      20044    0.019      2.00      17493 hags_1_4.sp21en0
    4      22536    0.021      3.00       2492 hags_1_3.sp21en0
    5     100213    0.096      4.00      77677 hags_1_2.sp21en0
    6     224021    0.214      5.00     123808 hags_1_1.sp21en0
    7     318856    0.304      6.00      94835 hags_1_0.sp21en0
Keep 7 logs, 318856 bytes, 0.304 MB; Trash 0 logs, 0 bytes, 0.000 MB.
7 total logs, 318856 total bytes, 0.304 total MB.
keep 7, trash 0, 0 MB.  rc = 0, 0x0
```

From this output you can also see that GS has a limit on the space used by the log files. This is why 5MB are needed for GS log files in /var/ha/log. The hagsd daemon keeps historical log files, and the space occupied by them cannot exceed 5MB. Core dump files are kept in the /var/ha/run/hags.<syspar_name> directory. The hagsreap script cleans up the core dump and log files to keep the space used below the limit.

1.2.2 The Group Services Domain

The set of nodes that is defined to GS is called a GS domain. It consists of the set of nodes that makes up a system partition. The subsystems that use GS are called *client subsystems*, and these can form groups by having their processes connected to GS using the Group Services Application Program Interface (GSAPI).

A key feature of GS is to provide a single group namespace across the SP. Actually, each SP partition is an isolated namespace that defines the complete scope of GS. All references to a specific group name within a group namespace will result in those references being directed to the same group. Any reference to that name in another group namespace refers to a completely independent group. Currently, there is no mechanism that allows a GS client within a group namespace to make any references outside of that namespace.

GS must be able to keep track of the group that its clients want to form. To do this, it establishes a GS name server within each domain, which is responsible for keeping track of all client groups that are created in that domain.

To ensure that only one node becomes a GS name server, GS uses the following protocol:

1. When a GS daemon is connected to the Topology Services subsystem, it waits for Topology Services to tell it which nodes are currently running in this system partition.

2. Based on the input from Topology Services, each daemon finds the lowest-numbered running node in the partition. It then compares its own node number to that number and performs one of the following:

   - If the daemon′s own node is the lowest-numbered node, it waits for all other running nodes to nominate it as the GS name server.
   - If the daemon′s node is not the lowest-numbered node, it starts sending nomination messages to the lowest-numbered node every 5 seconds.

3. Once all running nodes have nominated the GS name server-to-be and a variable-length timer has expired, the nominee sends an insert message to the nodes. All nodes must acknowledge this message. When they have done so, the nominee becomes the established GS name server and sends a commit message to all the nodes.
4. At this point, the GS domain is established, and requests by clients to join or subscribe to groups are processed.

Note that this is the case when all nodes are being booted simultaneously, such as at initial system power-on. Often, however, a GS daemon is already running in at least one node (for example, the Control Workstation) and the GS domain is already established, making the Control Workstation the GS name server. In that case, the GS name server will execute insert protocols when nodes start running. The list will be filled on a first-come, first-served basis.

The GS subsystem provides several commands that allow you to see the changes and information within a group. Since GS is an SRC subsystem, it can be controlled by SRC commands. For example, the following command will tell us whether the GS daemon is running:

```
[sp21en0:/]# lssrc -s hags.sp21en0
Subsystem         Group            PID     Status
hags.sp21en0      hags             24178   active
```

But if you want more information about the GS daemon, you can specify the `-l` flag as follows:

```
[sp21en0:/]# lssrc -ls hags.sp21en0
Subsystem         Group            PID     Status
 hags.sp21en0     hags             24178   active
 2 locally-connected clients.  Their PIDs:
 16856 19316
 HA Group Services domain information:
 Domain established by node 7.
 Number of groups known locally: 2
                    Number of   Number of local
 Group name         providers   providers/subscribers
 cssMembership            3           0           1
 ha_em_peers              6           1           0
```

This output tells us that there are two clients connected locally. These are defined by default and are part of the High Availability Infrastructure. We will discuss these clients and groups later in this chapter.

The important thing here is the line: Domain established by node 7. This indicates that a domain is established and the GS name server for that domain is Node 7.

Now if you want to see the list of nodes that are part of this domain, use the following command:

```
[sp21en0:/]# hagsmg -s hags.sp21en0
2.1 cssMembership: 1 5 7 6 0 8
1.1 ha_em_peers: 7 5 1 6 0 8
0.Nil ZtheNameServerXY: 7.1 5.Nil 1.Nil 6.Nil 0.2 8.2
0.Nil  theGROVELgroup:
```

This is the output you get from the Control Workstation. It shows us the members of each group, and more importantly, what their order is in the list. Take a closer look at the following line:

```
0.Nil ZtheNameServerXY: 7.1 5.Nil 1.Nil 6.Nil 0.2 8.2
```

This line specifies the list of nodes in the domain, which is called MetaGroup. It is made up of all the nodes in the partition that are running the GS daemon. The nomenclature in the list is quite simple: each pair is defined as node_number.incarnation_number, where incarnation number corresponds to a counter that tells us how many times the GS daemon has been started on that node. For example, 8.2 means that on Node 8, hagsd was started twice.

When the hagsd daemon wishes to start, it needs to determine the previous incarnation number. This is achieved by referencing the file hagsd.in<number>, where <number> is the previous incarnation number. This file is located in the /var/ha/lck/hags.tid.<syspar_name> directory; see the following example:

```
[sp21n01:/var/ha/lck/hags.tid.sp21en0]# ls -l
total 0
----------   1 root     system         0 Sep 13 10:48 hagsd.in7
```

In this example we are in the /var/ha/lck/hags.tid.sp21en0 directory on Node 1. The partition name is sp21en0, and the incarnation number is 7. This means that if the current GS daemon is up and running, it must be using this incarnation number. If not, the next time it starts it will use an incarnation number of 8.

This information is useful, because if the incarnation number is much larger than the number of times the node has been rebooted, then the daemon has been started several times, which indicates that something is wrong.

The incarnation number remains the same throughout the hagsd daemon′s lifetime. It will vary between the nodes and the control workstation, as subsystems will typically stop and restart at different times. Hence, to maintain a highly available system, we must limit the dependency of these subsystems.

In the previous output we saw entries like 5.Nil, which means the node where the command was executed does not know the incarnation number for Node 5. This is as it should be, because only the GS name server has all the information.

In this way, only the information that is needed is passed to the network. Take a look at the same output after executing the command on the GS name server, Node 7:

```
[sp21n07:/]# hagsmg -s hags
2.1 cssMembership: 1 5 7 6 0 8
1.1 ha_em_peers: 7 5 1 6 0 8
0.Nil ZtheNameServerXY: 7.1 5.1 1.1 6.3 0.2 8.2
0.Nil  theGROVELgroup:
```

You now see all the information. Also, notice that the subsystem is called hags only; that is because in the Control Workstation there are as many GS subsystems as partitions, but in the node there is only one, because a node can belong to only one partition at a time.

When the GS name server dies for any reason, the next node in the list takes its place. There is no conflict to decide which will be the next GS name server because GS guarantees that all the nodes have the same information, so all the nodes know which is next in the list. They send a nomination message to that node, and it becomes the new GS name server after it sends a commit message back to the nodes. When the previous GS name server comes back, it joins the MetaGroup and is put at the end of the list. For example, if the GS daemon dies in Node 7, Node 5 becomes the new GS name server. Let′s kill the daemon in Node 7:

```
[sp21n07:/]# ps -ef|grep hagsd
    root 12768  3542   0   Sep 09      -  0:17 /usr/lpp/ssp/bin/hagsd
    root 16460 11022   0 19:37:45  pts/3  0:00 grep hagsd
[sp21n07:/]# kill -9 12768
```

Now if we query GS, we get the new list:

```
[sp21en0:/]# hagsmg -s hags.sp21en0
2.1 cssMembership: 1 5 6 0 8 7
1.1 ha_em_peers: 5 1 6 0 8 7
0.Nil ZtheNameServerXY: 5.1 1.Nil 6.Nil 0.2 8.2 7.2
0.Nil  theGROVELgroup:
```

The new GS name server is Node 5, and Node 7 has been put at the end with an incarnation number of 2. If we now query the SRC subsystem, we get the following:

```
[sp21en0:/]# lssrc -ls hags.sp21en0
Subsystem         Group            PID     Status
 hags.sp21en0     hags             24178   active
 2 locally-connected clients.  Their PIDs:
 16856 19316
 HA Group Services domain information:
 Domain established by node 5.
 Number of groups known locally: 2
                   Number of   Number of local
 Group name        providers   providers/subscribers
 cssMembership          4          0          1
 ha_em_peers            6          1          0
```

This output tells us that the domain has been established and Node 5 is the GS name server. If we were fast enough, we could see an intermediate state, that is, when the domain is being recovered and Node 5 becomes the GS name server. We would see the line `Domain is recovering`.

1.2.3 Group Services Groups

Once the domain is established, processes from the nodes can start creating and joining groups. Any authorized process in a GS domain may create a new group, and ask to become a member of a group. Such a request is called a join request, or joining the group.

The process that has joined a group is called a provider. If this process only wants to monitor a group, without initiating any change to the group information, it is called a subscriber.

Each group is uniquely named within the domain, and GS guarantees that all processes that are joined to a group will see the same group information, and more importantly, it guarantees that they will see all changes to the group information in the same order.

The concept of membership is based on processes that are running in the nodes within the domain or partition. A group may have members on multiple nodes in the domain, and each node may have multiple members.

For each group, the GS subsystem maintains consistent group state data. A group′s state consists of two pieces of information:

- The membership list

  This is the list of providers in the group. Each provider is identified as follows:

  ```
  Identifier: [instance/node number]
  ```

  The instance number is passed to GS by the provider itself, when it wants to join the group. This identifier can be used by the client subsystem to specify different instances of a provider. A good example of this are NFS daemons. You may have several NFS daemons running in the same node, so they can be part of a group being identified by different instance numbers. The membership list is held on a first-come, first-served basis, so the first provider to join the group is at the head and the last one is at the end. All providers and subscribers in a group see the same ordering of the list.

- The group state value

  The state value of a group is defined by the application that is using the GSAPI and is controlled by the providers in a way that is meaningful to the application. This is a field of 256 bytes that is handled by the application and is not interpreted by GS.

The membership list of a group is modified by providers joining and leaving the group. Several protocols define how these processes can join or leave a group. Sometimes a process must leave the group involuntarily, due to a failure of the provider process itself or a failure of the node where the process provider is running. An involuntary leave is called a *failure leave* and is initiated by the GS subsystem.

A GS client that asks to become a provider or subscriber of a group must be admitted by the current members of that group. This is accomplished by a voting protocol, by which each member has to vote “approved” or “rejected.”

In PSSP 2.2, two groups are created by default in each GS domain. These groups are:

cssMembership This group is initiated by the hagsglsmd daemon, which is part of the GS subsystem and provides a general purpose facility for coordinating and monitoring changes to the state of an application that is running on a set of nodes. This daemon provides global synchronization services for the High Performance Switch adapter membership group.

ha_em_peers This group is initiated by the Event Manager daemon, haemd, which observes Resource Variable instances that are updated by Resource Monitors, and reports events to client programs.

The GS subsystem provides a command to query the group information in the GS domain. For example, to query the group information in a GS domain or system partition, you can use the following command:

```
[sp21en0:/usr/lpp/ssp/bin]# hagsgr -s hags.sp21en0
  Number of: groups: 6
Group slot # [0] Group name[HostMembership] group state[Idle |]
Providers[[0/5][0/1][0/6][0/0][0/8][0/7]]
Local subscribers[[10/0]]

Group slot # [1] Group name[ha_em_peers] group state[Inserted |Idle |]
Providers[[1/5][1/1][1/0][1/6][1/8][1/7]]
Local subscribers[]

Group slot # [2] Group name[!SwitchGroupie!] group state[Idle |]
Providers[]
Local subscribers[]

Group slot # [3] Group name[enMembership] group state[Idle |]
Providers[[0/0][0/1][0/5][0/6][0/8][0/7]]
Local subscribers[[10/0]]

Group slot # [4] Group name[cssMembership] group state[Inserted |Idle |]
Providers[[0/1][0/6][0/5][0/7]]
Local subscribers[[10/0]]

Group slot # [5] Group name[theSourceGroup] group state[Inserted |Idle |]
Providers[[100/6]]
Local subscribers[]
```

The first two lines show how many groups are present in this domain, followed by descriptions for each group that list its name, state, providers, and subscribers.

HostMembership and enMembership are two groups internal to GS. They are formed from the information provided by Topology Services. The group !SwitchGroupie! is a special group used to help build CSS adapter membership information, and joins to it are done internally only by GS.

The last group, theSourceGroup, was created by a GS client running on Node 6.

From the output you can see that one of the groups does not have a provider. In that case we could say that the group should not exist. That is partially true because, since the group exists “internally” to GS, the internal data structures in the daemon continue to persist unless all GS daemons that knew of that group fail or are intentionally stopped.

What does not exist is any indication of the former group′s provider membership list or its group state value. Therefore, the first provider to join the group again will see no current providers, and the group state value will have reverted to 0 (0x00000000). Also, the group attributes formerly used to establish the group are no longer used (and the new providers joining the group may use different group attributes to re-establish the group). If you try to subscribe to this group, you will get a `group does not exist` error.

Therefore, from an “external” viewpoint, the group really does not exist any longer. Internally, the data structures are maintained, so that when providers do start joining again, it will not be necessary to rebuild everything.

Since hagsgr is intended to dump out internal information, it will show these null groups sitting there.

1.3 Event Management

The Event Management subsystem is part of, and also a client of, the HA infrastructure. It is a distributed system that uses GS to provide event management and event notification to other client subsystems, such as the Problem Management subsystem. It can be used to monitor the system, subsystems, processes and application resources. Its applicability is intended for, but not limited to, SP platforms.

The structure of the Event Management subsystem can be seen in Figure 4 on page 17.

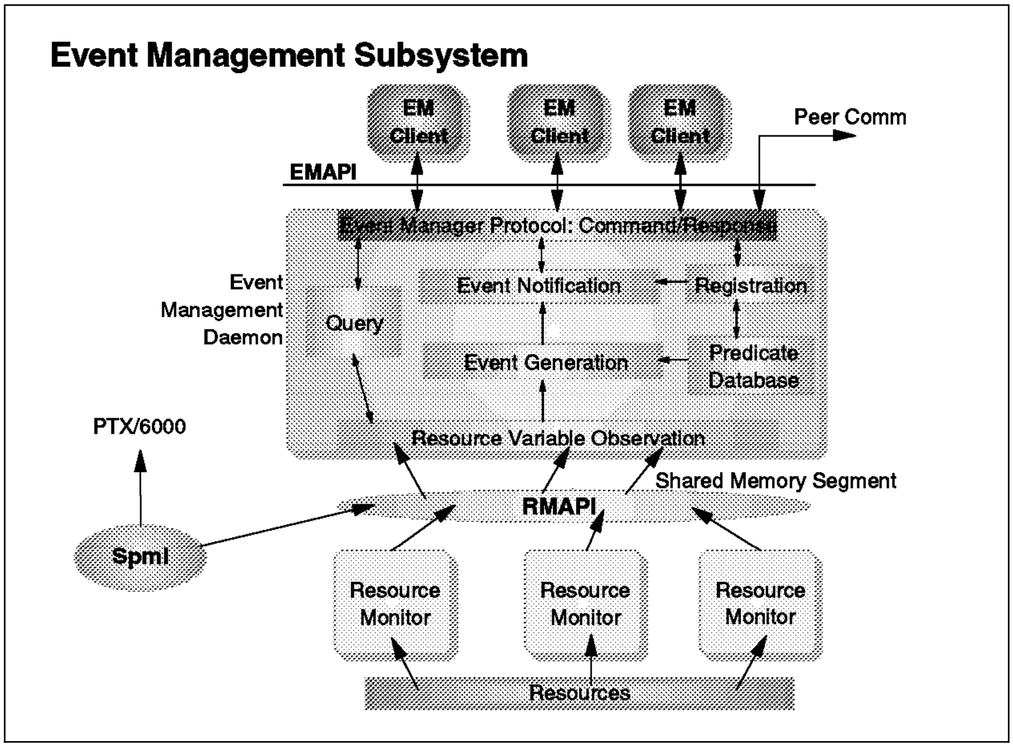


Figure 4. Event Management Subsystem Structure

In Figure 4 we can see the Event Management subsystem as three layers:

- Client programs (such as the Problem Management subsystem)
- Event Manager daemons
- Resource Monitors

Client programs are applications or other subsystems that wish to receive event information when resources change state. Client programs use the Event Manager Application Programming Interface (EMAPI) to register their interest in the events from a set of resources. The Event Manager daemon receives this request for notification through the API, and sends those events when they are produced. The events are created by the Event Manager based on the information reported by Resource Monitors. These may be daemons, or they may be included in the software components that manage the resources of interest.

Client programs communicate with an Event Manager daemon located on the same node. Resource Monitors provide data to an Event Manager daemon located on the same node. An Event Manager daemon communicates with Event Manager daemons on other nodes in order to provide events to its local clients. Therefore, a client program may register for and receive events about any resource in the SP system. In addition, a client program executing outside the SP system may remotely communicate with the Event Manager daemon located on the Control Workstation.

An event is defined as a change in state of a resource. The resource is represented by one or more variables. An event is further defined as the existence, at some point in time, of a particular relationship of the value of a variable to some other value, which may be a constant or a prior value of the Resource Variable. This relationship is called *predicate*. Each time a Resource

Variable is observed, the predicate is applied to the observed value. If the predicate is true, then an event is generated. For example, the percentage of free space in a file system may be a Resource Variable. The predicate could be P<10, where P is the name of the Resource Variable. Whenever P is observed and its value is less than 10, an event is generated.

A Resource Variable has a unique *name* and an *instantiation vector*. Since a Resource Variable represents a resource and resources typically have multiple copies or instances in the system, a Resource Variable also has multiple instances. The instantiation vector contains a sufficient and necessary number of elements to uniquely identify an instance of a Resource Variable in the system. For example, a variable that represents the number of free blocks in a file system requires a vector with three elements: node ID, volume group name and logical volume name. A variable that represents the power state of a node requires a single element: node ID.

There are three types of Resource Variables: *Counter, Quantity*, and *State*. The value of Counter increases monotonically. The value of Quantity fluctuates over time. The only semantics that Event Management assumes for these two types is that they may be updated more often than they are observed without loss of meaningful information with respect to event generation. The value of State fluctuates over time, also. However, a State Resource Variable must be observed every time it is updated in order to avoid missing potential events.

At the most elemental level, Event Management gathers Resource Variables, applies predicates to them and, if the predicates are true, generates events. These events are then delivered to clients that registered interest in them. A client is any program that uses the EMAPI to obtain events.[1] Events are generated by the Event Manager daemon from Resource Variables supplied by Resource Monitors. Refer to Chapter 2, “Resource Monitors” on page 27 for a complete description of these monitors.

An Event Manager daemon and one or more Resource Monitors are located on any AIX node that contains Resource Variables from which events of interest may be generated. An Event Manager daemon is also located on any AIX node where an Event Management client program is expected to be run. The AIX node must contain the appropriate level of AIX, Group Services, and Reliable Messaging.

A client registers interest in an event by passing the name and instantiation vector of the Resource Variable from which the event is to be generated, and by passing a predicate to be applied to that variable. If no predicate is provided by the client, a default predicate, configured with the Resource Variable, is used. The client may name more than one Resource Variable and optionally associate the instantiation vector. If each variable instance is listed, then a different predicate may be specified for each variable. If an instantiation vector is wildcarded, then the associated predicate applies to each variable instance found as a result of the wildcard.

If a Resource Variable specified by the client is located on other nodes, then the Event Manager assumes the role of a client and sends a registration request to the Event Managers on the appropriate nodes. In this case the Event Manager is

[1] Refer to Chapter 4 for examples of how to use the EMAPI.

a proxy acting on behalf of the client. A registration request from either a client or a proxy client, such as an Event Manager daemon, is processed the same way by the receiving Event Manager daemon.

When the Event Manager daemon receives the registration request, it ensures that the Resource Monitors that supply the named Resource Variable are running, assuming the monitors are not implemented as commands. If they are not running, and are of a type that can be started by the Event Manager, then they are started. If a Resource Monitor is not a command, it is then sent a request to start supplying data.

A Resource Monitor *sends* a Resource Variable by one of two methods: if the variable is of type Counter or Quantity, it is passed to the Event Manager through shared memory; if the variable is of type State, it is passed to the Event Manager as a message. Resource Variables of type Counter or Quantity are observed by the Event Manager daemon every *x* seconds, where *x* is configurable. Resource Variables of type State are observed when they are received by the Event Manager daemon. At each observation of a variable its associated predicates are applied. For each predicate that evaluates as true, an event is generated.

Once an event is generated it is sent to the client that registered for it, including proxy clients. An Event Manager daemon acting as a proxy forwards events it receives from the other daemons to the appropriate clients.

1.3.1 Event Management Subsystem

The Event Management subsystem is an SRC subsystem and can be controlled by SRC commands. For example, the following command will tell us whether the subsystem is running, and whether there is more than one daemon, in case the system is partitioned:

```
[sp21en0:/]# lssrc -g haem
Subsystem         Group           PID     Status
 haem.sp21en0      haem            36306   active
 haem.sp21en1      haem            29604   active
```

In this case, the system is partitioned and we have one daemon running for each partition.

If you want to know more details about the Event Management subsystem, you can specify the -l parameter with the SRC command, as follows:

```
[sp21en0:/]# lssrc -ls haem.sp21en0
Subsystem          Group            PID     Status
 haem.sp21en0      haem             36306   active

Trace flags set:  None

 Configuration Data Base version: 852569469,788246784,0(SDR)

 Daemon started on 01/06/97 at 11:51:31.044042496
    running 0 days, 0 hours, 57 minutes and 28 seconds
 Daemon connected to group services: TRUE
 Daemon has joined peer group:       TRUE
 Daemon communications enabled :     TRUE
 Peer count:                         8

 Logical Connection Information
  Type   LCID   FD  Node/PID  Start Time
 local      0   12    13456   Mon Jan  6 11:52:46 1997
 local      1   13    13456   Mon Jan  6 11:52:46 1997
 local      2   14    36404   Mon Jan  6 11:52:46 1997
 local      3   21    36092   Mon Jan  6 11:53:35 1997

 Resource Monitor Information
 Resource Monitor Name                 Type       FD    PID  Locked
 IBM.PSSP.harmld            server     16   31914    No
 IBM.PSSP.harmpd            server     17   13108    No
 IBM.PSSP.hmrmd             server     20   33660    No
 IBM.PSSP.pmanrmd           client     15      -2    No
 Membership                 internal   -1      -2    No
 Response                   internal   -1      -2    No
 aixos                      internal   -1      -2    No

 Highest file descriptor in use is 22

 Peer Daemon Status
    0 S S      1 I A      2 I A      3 - A      4 - A      5 I A
    6 I A      7 - A      8 - A      9 I A     10 I A     11 - A
   12 - A     13 I A     14 I A
```

The second part of the command output shows internal counters, CPU utilization by the daemons, and the data segment size:

```
Internal Daemon Counters
    GS init attempts =        11  GS join attempts =         1
    GS resp callback =        28  CCI conn rejects =         0
    RMC conn rejects =         0  HR conn rejects  =         0
    Retry req msg    =         0  Retry rsp msg    =         0
    Intervl usr util =         1  Total usr util   =        70
    Intervl sys util =         1  Total sys util   =        66
    Intervl time     =     12100  Total time       =    343865
    lccb′s created   =         4  lccb′s freed     =         0
    Reg rcb′s creatd =         1  Reg rcb′s freed  =         0
    Qry rcb′s creatd =         6  Qry rcb′s freed  =         6
    vrr created      =         1  vrr freed        =         0
    vqr created      =      1095  vqr freed        =      1095
    var inst created =       516  var inst freed   =         0
    Events regstrd   =         1  Events unregstrd =         0
    Insts assigned   =         9  Insts unassigned =         0
    Smem vars obsrv  =       114  State vars obsrv =      3129
    Preds evaluated  =         9  Events generated =         9
    Smem lck intrvl  =         0  Smem lck total   =         0
    PRM msgs to all  =         0  PRM msgs to peer =         0
    PRM resp msgs    =         0  PRM msgs rcvd    =         0
    PRM_NODATA       =         0  PRM_BADMSG errs  =         0
    xcb alloc′d      =        40  xcb freed        =        40
    xcb freed msgfp  =         0  xcb freed reqp   =         0
    xcb freed reqn   =         0  xcb freed rspc   =        23
    xcb freed rspp   =         0  xcb freed cmdrm  =        17
    xcb freed unkwn  =         0

Daemon Resource Utilization
       User:           0.010 secs    0.008% (last interval)
                       0.700 secs    0.020% (total)
       System:         0.010 secs    0.008% (last interval)
                       0.660 secs    0.019% (total)
       U+S:            0.020 secs    0.017% (last interval)
                       1.360 secs    0.040% (total)

       Date segment size:  1415K
```

1.3.2 The Event Management Daemon

The Event Management daemon, /usr/lpp/ssp/bin/haemd, runs on every node of a system partition and on the Control Workstation. If there is more than one partition, there will be multiple daemons running on the Control Workstation.

Each daemon operates in a different domain (partition) with a complete set of log files and sockets.

The log files are located on /var/ha/log, and there is one for each daemon running. In our case, these are the log files present in the /var/ha/log directory:

```
[sp21en0:/var/ha/log]# ls -l em.*
-rwxr-xr-x   1 root     system       339 Jan 06 11:52 em.default.sp21en0
-rwxr-xr-x   1 root     system         0 Jan 06 11:52 em.default.sp21en1
```

The Event Manager uses different types of communication:

- UDP packets for daemon-to-daemon communication

- TCP packets for daemon-to-client communication
- UNIX domain sockets for local communication between Event Management clients and Event Management daemons

The UDP port number used for daemon-to-daemon communication is stored in the SDR, class Syspar_ports:

```
[sp21en0:/]# SDRGetObjects Syspar_ports
subsystem    port
hats              10000
hags              10001
haem              10002
```

The TCP port number for daemon-to-client communication is also stored in the SDR, class SP_ports:

```
[sp21en0:/]# SDRGetObjects SP_ports
daemon       hostname     port
hardmon      sp21en0              8435
hb           ""                   4893
haemd        ""                  10000
```

The class contains the daemon, hostname, and port attributes. The hostname attribute is not used by the Event Management subsystem.

The UNIX domain sockets, which are connection-oriented, are used for local communication between the EM clients and the Event Manager daemon (EMAPI), and for local communication between the Event Manager daemon and the Resource Monitors (RMAPI). The following names are used:

```
[sp21en0:/var/ha/soc]# ls -l  em.*
srw-------   1 root  system   0 Jan 06 11:53 em.RMIBM.PSSP.harmld.sp21en0
srw-------   1 root  system   0 Jan 06 11:52 em.RMIBM.PSSP.harmpd.sp21en0
srw-------   1 root  system   0 Jan 06 11:52 em.RMIBM.PSSP.hmrmd.sp21en0
srw-------   1 root  system   0 Jan 06 11:53 em.RMIBM.PSSP.hmrmd.sp21en1
srw-rw-rw-   1 root  system   0 Jan 06 11:52 em.clsrv.sp21en0
srw-rw-rw-   1 root  system   0 Jan 06 11:52 em.clsrv.sp21en1
srw-rw-rw-   1 root  system   0 Jan 06 11:52 em.rmsrv.sp21en0
srw-rw-rw-   1 root  system   0 Jan 06 11:52 em.rmsrv.sp21en1
```

/var/ha/soc/em.clsrv.*syspar_name*

Used by the EMAPI to connect to the Event Manager daemon

/vsr/ha/soc/em.rmsrv.*syspar_name*

Used by Resource Monitors with a connection type of client to connect to the Event Manager daemon

/var/ha/soc/em.RM*rmname.syspar_name*

Used by the Event Manager daemon to connect to the Resource Monitor that is specified by *rmname*. This monitor has a connection type of server.

For more information refer to *IBM Parallel System Support Programs for AIX: Administration Guide*, GC23-3897.

Part 2. Subsystems

Chapter 2. Resource Monitors

This chapter gives an overview of Resource Monitors. It discusses what they are used for and how they are integrated into the new High Availability Infrastructure.

In PSSP 2.2, there are Resource Monitors that provide the information for the higher layers of the High Availability (HA) subsystem. Many of the resources in the system are already monitored by these integrated Resource Monitors.

To extend the HA infrastructure, you may need to write your own Resource Monitor; for example, you may have to supply new Resource Variables from your application to the HA subsystem. This chapter gives examples for doing this.

In the following sections, we refer to three examples of Resource Monitors, located in /usr/lpp/ssp/samples/haem/rmapi, that are included in the PSSP 2.2 code.

1. The first example is a command line-based Resource Monitor that simulates an adapter interface. From the command line you can mimic state changes, such as going from Unconfig → Config. These state changes can be used to initiate activities in the Problem Management subsystem.
2. The next example is a Resource Monitor daemon that can be triggered via a signal to send random numbers to the Event Management subsystem.
3. The third example is a Resource Monitor daemon that is implemented as a server. This daemon generates random numbers, and these can be used by the Event Management subsystem. As soon as some client of Event Management registers interest in this Resource Variable, the daemon starts to produce the random numbers.

In this chapter, we focus on the third example.

The source code for these examples can easily be used as a framework for writing your own Resource Monitor daemon. It consists mainly of setting up the Resource Monitor Application Programming Interface (RMAPI). The code generating the random numbers is only one line, which can easily be modified.

Information about Resource Monitors can be found in a number of documents, such as the Administration Guide, the Event Management Programming Guide and Reference, and *RS/6000 SP High Availability Infrastructure*. See Appendix J, “Related Publications” on page 261 for details. However, in order to be able to write a Resource Monitor, a sufficient set of basic information is needed. Therefore, some items you may already know from other sources may be repeated here.

2.1 Resource Monitor Overview

An RS/6000 SP must be highly available to the users of the SP. To provide such availability, all the resources that comprise a user′s application must be highly available. These resources include applications, subsystems such as transaction processing and databases, file systems, networks, processors, disks, and so forth. Not one resource in the system, from the application to the most basic hardware, may be a single point of failure. Each resource must be quickly recoverable to an available state without user intervention.

In a system that provides highly available resources, it is assumed that some recovery action is performed when a resource fails or becomes unavailable. In order to initiate recovery, the failure, or the *change in the state of the resource*, must be detected.

In order to support these requirements, a more universal infrastructure is needed to mask failures from the end user and recover from them automatically. That is the main reason why the HA subsystems are integrated into PSSP 2.2.

The components that make up the HA infrastructure have already been described. This chapter focuses on Resource Monitors, which are the software components that observe the state of specific system resources and transform this information into unambiguous variables. These Resource Variables are used to describe the state of the system. What these resources are and what kinds of Resource Variables exist are discussed.

Resource Monitors provide the Resource Variables to the next higher level of the HA subsystem, *Event Management*. Event Management observes these Resource Variables. The change of state of a variable is defined as an event. Events are then communicated to other subsystems, which are responsible for the recovery.

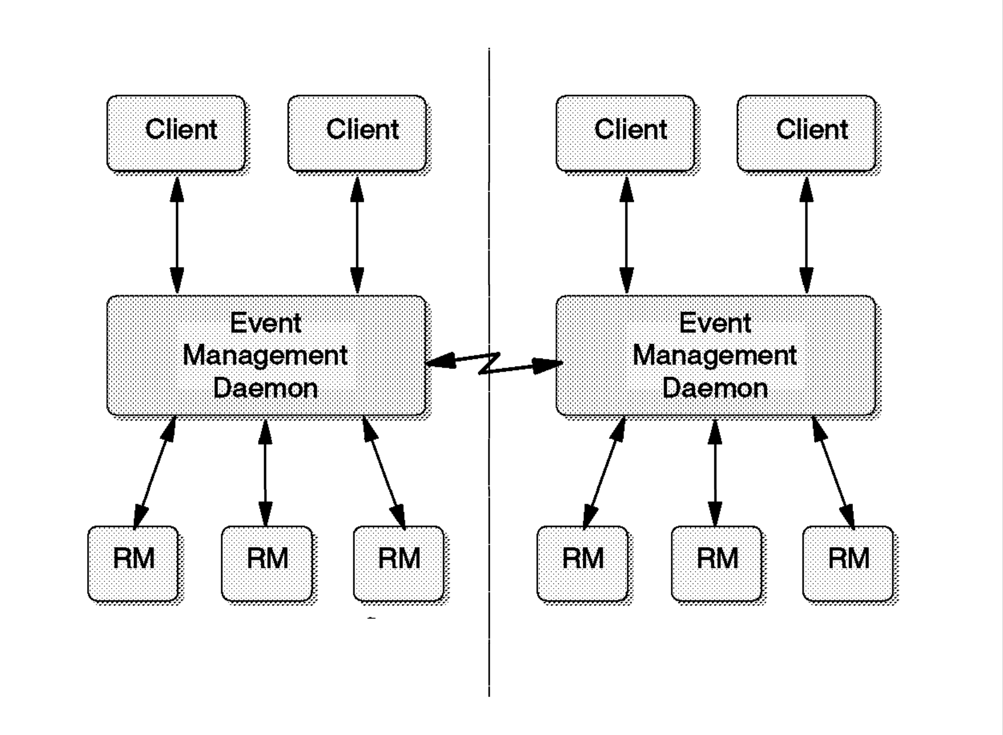


Figure 5. Event Management Basic Structure

Event Management is a distributed subsystem initially intended for, but not limited to, the SP platform. It provides comprehensive event detection by monitoring hardware and software resources. It is one of several subsystems in PSSP 2.2 that provides a set of high availability services, and consists of three types of components:

Client programs

Client programs are applications or other subsystems that wish to be notified when resources change state. Client programs use an Event Management Application Programming Interface (EMAPI) for communication with the Event Management daemon. An example of a client program is the Problem Management subsystem. For details, see Chapter 3, "Problem Management Subsystem" on page 71.

Event Management daemon

The Event Management daemon is the central component that receives data from the Resource Monitor, processes it, and generates events that are transferred to the clients. The Event Manager informs the interested parties (clients) whenever the state of a resource changes. The resource state change is reflected by a change in a resource attribute.

Resource Monitors

The Resource Monitors provide the state of a resource to the next higher level of control, which is the Event Management daemon. The Resource Monitors provide data to the Event Management daemon located on the same node. The data is provided to Event Management through the Resource Monitor Application Programming Interface (RMAPI). This interface is also the way Event Management daemon controls the Resource Monitor.

2.2 Resource Monitor Objectives

System administrators who monitor the hardware and software components of a system may find that the hardware seems different from what was expected, the software does not behave as expected, and the user applications do not use the system resources in an efficient manner. By monitoring these components and analyzing the results, the system administrator tries to optimize the system.

Monitoring the system also means to detect tendencies and to try to prevent performance bottlenecks and system down time by the timely changing of unhealthy system components.

The monitored results must be transferred to a higher level of software that processes the data (filtering, applying predicates, analyzing, and so on). The method of transferring this data to a higher level should be very efficient, otherwise the monitoring process itself puts more load on the system and the results are distorted.

The data obtained from a monitor program could be of interest to more than one higher-level analyzer. Therefore, the interface must be shareable among system components.

The shared interface must be standardized. Using a standard interface also allows independent software vendors to incorporate the monitoring functionality into their applications in order to make it more reliable and achieve the best possible system performance.

To summarize, the objectives of a Resource Monitor are to:

- Monitor hardware and software
- Transfer information to manager subsystems
- Use shared interfaces
- Communicate through a standard API

2.3 What Are Resources?

A resource is an entity in the system that is observed. Examples of resources include hardware entities such as processors, disk drives, memory, and adapters, and software entities such as databases, processes, and file systems. Each resource in the system has one or more attributes that define the state of the resource. The number of attributes is defined by the resource.

Overview of resources:

Hardware

- CPUs
- Memory
- Disk subsystems
- Adapters

Software

- System software
  - AIX subsystems
  - SP subsystems
- Application software
  - Databases
  - OLTP

Other System Resources

- File system space
- Networks

2.4 Resource Representation

A resource is represented by one or more variables that originate in, and are updated by, a Resource Monitor. The Event Manager daemon observes each variable as it changes, or at some periodic rate that is configurable.

2.4.1 Resource Monitor Modes

A Resource Monitor can operate in one of two modes. In the first mode of operation, a Resource Monitor *pushes* Resource Variables to the Event Manager. In the second mode of operation, a Resource Monitor *pulls* Resource Variables from the component being monitored and then transfers the information to Event Management.

Push mode

The push mode is implemented by incorporating Resource Monitor logic into the component that is the source of the Resource Variables. The component sends new values of its Resource Variables to the Event Management daemon as appropriate for the type of Resource Variable. A push mode Resource Monitor may also be implemented as a command. The Resource Monitor part is integrated into the application; in other words, the application is *Resource Monitor-enabled.* This type of application is also called a Resource Monitor client.

Pull mode

The pull mode is implemented by creating a daemon process that contains the Resource Monitor logic. The Resource Monitor daemon, using interfaces defined for the component being monitored, periodically polls the component for Resource Variable values. (These values may also be obtained in an asynchronous manner if the component supports this.) Again, as these values are obtained, the Resource Monitor sends them to the Event Manager daemon as appropriate for the particular Resource Variable. This type of application is also called a Resource Monitor server.

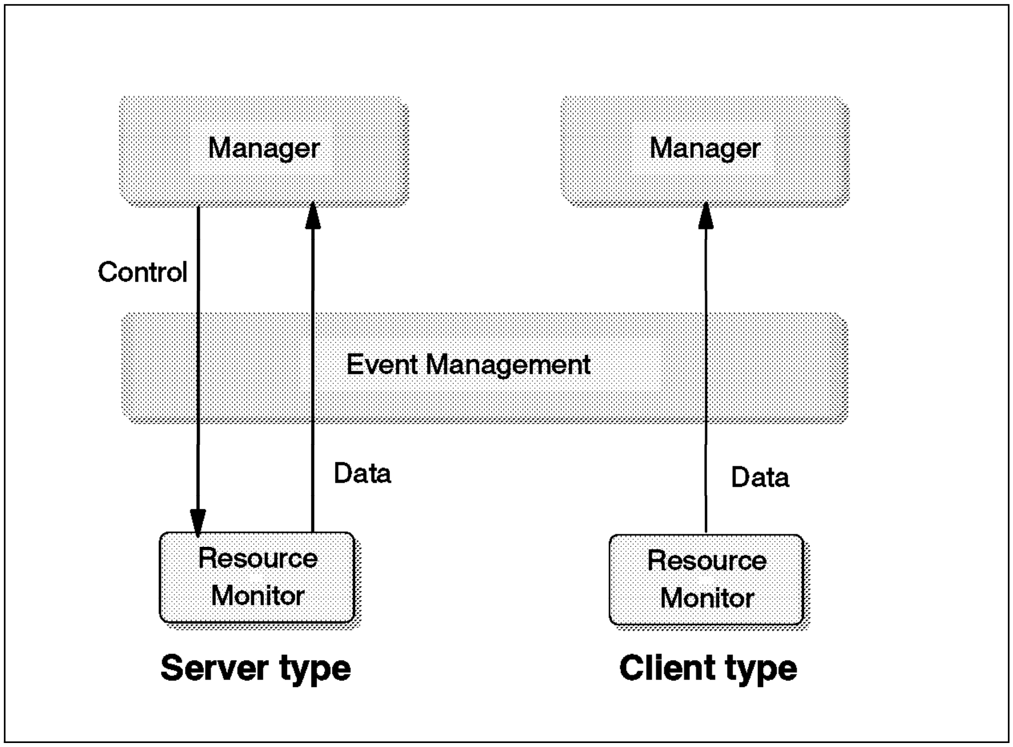


Figure 6. Resource Monitor Types

Resource Monitors that are implemented as daemons are started automatically by the Event Manager daemon whenever a client registers for events and when the related Resource Variables are provided by a Resource Monitor. Whenever resource information is required, Event Management requests that the Resource Monitor start supplying data, unless the Resource Monitor is a command.

2.4.2 Resource Variable Types

Resource Variables are one of the following three value types:

Counter Counter is a variable whose value increases monotonically; for example, the number of input packets from an Ethernet adapter.

Quantity Quantity is a variable whose value fluctuates over time; for example, the number of input packets per second from an Ethernet adapter.

State State is a variable whose value fluctuates over time also. However, every time it changes value it must be observed in order to not miss generating events. While the time between updates of a State variable may be very short, on average a State variable is expected to change at a relatively slow rate.

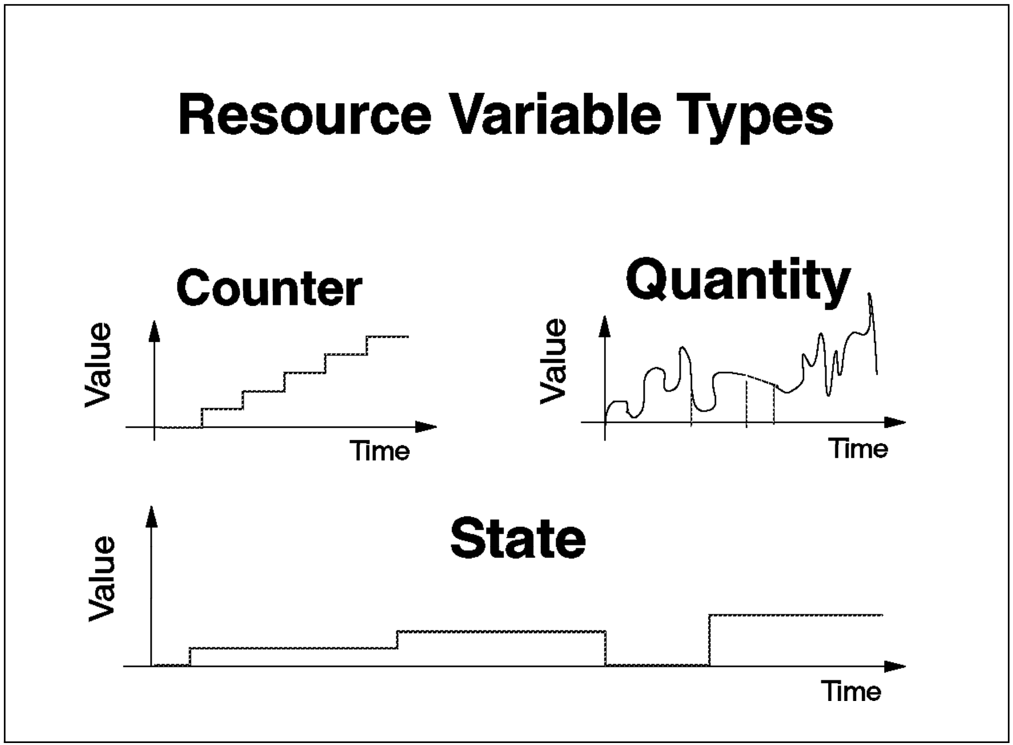


Figure 7. Resource Variable Types

The command in Figure 8 on page 33, shows Resource Variables and their related variable types.

```
# SDRGetObjects EM_Resource_Variable rvName rvValue_type

rvName                              rvValue_type
.
IBM.PSSP.CSS.bcast_rx_ok            Quantity
IBM.PSSP.CSS.bcast_tx_ok            Quantity
.
IBM.PSSP.SP_HW.Node.tempRange       State
IBM.PSSP.SP_HW.Node.type            State
.
IBM.PSSP.aixos.Mem.Real.%free       Quantity
IBM.PSSP.aixos.Mem.Real.%pinned     Quantity
.
IBM.PSSP.aixos.Mem.Virt.pagein      Counter
IBM.PSSP.aixos.Mem.Virt.pageout     Counter
.
IBM.PSSP.aixos.PagSp.%totalfree     Quantity
IBM.PSSP.aixos.PagSp.%totalused     Quantity
.
IBM.PSSP.Prog.pcount                State
IBM.PSSP.Prog.xpcount               State
.
```

Figure 8. Resource Variable Types

2.4.3 Resource Variable Data Types

The data types for Resource Variables are:

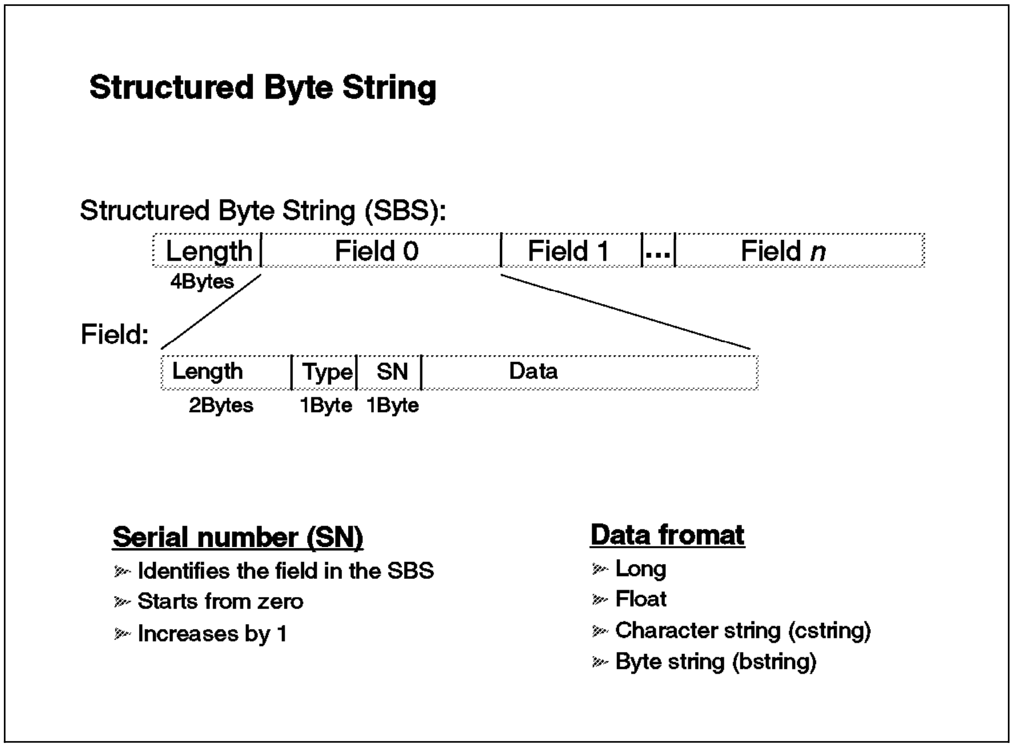
- Long
- Float
- Structured Byte String

A Resource Variable of type State can be of any of these data types. But Counter and Quantity Resource Variables can only be of type long or float.

The long and float formats are identical to the C-language types of the same name.

A Structured Byte String (SBS) is a string of bytes, where each byte may have any value from 0 through 255. The SBS starts with a 32-bit (4-byte) length field that specifies the total length of the structured fields (one or more) that follow.

Figure 9 on page 34 shows the format of an SBS.



Structured Byte String

Serial number (SN)

- Identifies the field in the SBS
- Starts from zero
- Increases by 1

Data fromat

- Long
- Float
- Character string (cstring)
- Byte string (bstring)

Figure 9. Structured Byte String

A structured field consists of a 4-byte header, followed by a value. The first two bytes of the header contain the length of the structured field. The third byte is a structured field data type, and the fourth byte is an 8-bit serial number.

A structured field is one of the following types:

- Long
- Float
- Character string
- Byte string

Long and float types are the same as in the C language.

A character string is some number of non-zero bytes terminated by a null byte. The null byte is included in the structured field length, like the C-language character string.

A byte string is some number of bytes, where each byte may have any value from 0 to 255.

The serial number is a unique value that identifies the structured field. It is defined by the Resource Monitor that supplies the SBS Resource Variable for each structured field. The set of serial numbers defined for the SBS starts with 0 and is contiguous.

The screen output in Figure 10 on page 35 shows some Resource Variables and their related data types.

```
# SDRGetObjects EM_Resource_Variable rvName rvValue_type rvData_type

rvName                          rvValue_type rvData_type
.
IBM.PSSP.CSS.bcast_rx_ok        Quantity     long
IBM.PSSP.CSS.bcast_tx_ok        Quantity     long
.
IBM.PSSP.SP_HW.Node.tempRange   State        long
IBM.PSSP.SP_HW.Node.type        State        long
.
IBM.PSSP.aixos.Mem.Real.%free   Quantity     long
IBM.PSSP.aixos.Mem.Real.%pinned Quantity     long
.
IBM.PSSP.aixos.Mem.Virt.pagein  Counter      long
IBM.PSSP.aixos.Mem.Virt.pageout Counter      long
.
IBM.PSSP.aixos.PagSp.%totalfree Quantity     float
IBM.PSSP.aixos.PagSp.%totalused Quantity     float
.
IBM.PSSP.Prog.pcount            State        SBS
IBM.PSSP.Prog.xpcount           State        SBS
.
```

Figure 10. Resource Variable Data Types

2.4.4 Information Exchange of Resource Variables

Resource Variables of type Counter or Quantity are useful not only to Event Management, but also to performance monitoring. Therefore, these variables are placed in a shared memory to make them accessible to other observers.

Figure 11 on page 36 shows how information is exchanged among Resource Variables.

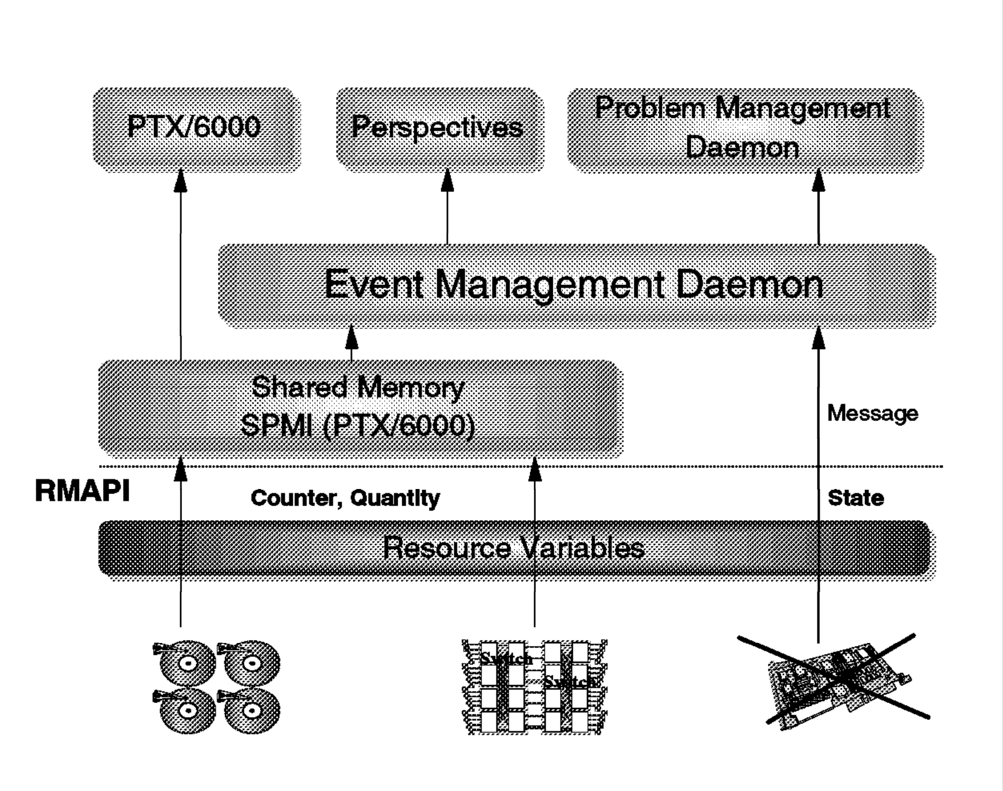


Figure 11. Information Exchange of Resource Variables

The Performance Toolbox for AIX (PTX/6000) provides a mechanism for easily obtaining Resource Variables. PTX/6000 also includes an API, called the System Performance Measurement Interface (SPMI), for obtaining these variables from shared memory and for placing additional variables in shared memory. This already-defined and available interface was chosen to enable monitoring of Resource Variables of type Counter and Quantity by other software packages, such as Performance Toolbox/6000 or Event Management.

Programs that supply additional values to the shared memory are called Dynamic Data Supplier (DDS) programs. This data can also be monitored using the PTX/6000 Manager.

It is important to know where this data can be found within the hierarchical information structure of PTX/6000. A model of this hierarchy appears in Figure 12 on page 37.

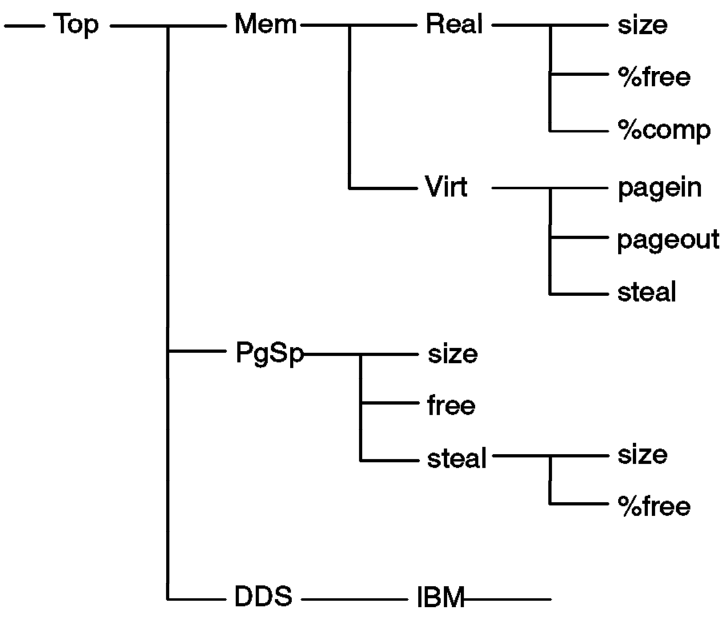


Figure 12. Set of PTX/6000 Statistics

Use `xmpeek -l` to get an overview of the resources that can be monitored. Figure 13 on page 38 shows an example of this.

```
.
/sp21n01/CPU/cpu0/kern          Time executing in kernel mode (percent)
.
/sp21n01/CPU/cpu0/syscall       Total system calls on this processor
.
/sp21n01/Mem/Real/size          Size of physical memory (4K pages)
.
/sp21n01/Mem/Real/%free         % memory which is free
.
/sp21n01/PagSp/%totalfree       Total free disk paging space (percent)
.
/sp21n01/Disk/hdisk0/rblk       512 byte blocks read from disk
.
/sp21n01/Disk/hdisk1/wblk       512 byte blocks written to disk
.
/sp21n01/LAN/ent0/xmitovfl      Count of transmit queue overflows
.
/sp21n01/Proc/5792biod/.....
.
/sp21n01/Syscall/total          Total system calls
.
/sp21n01/FS/rootvg/free         Free space in volume group, MB
.
/sp21n01/DDS/                                Dynamic Data Supplier Statistics
/sp21n01/DDS/IBM/                            IBM data suppliers.
/sp21n01/DDS/IBM/PSSP.SampleDaeMon/          RMAPI Sample Monitor
/sp21n01/DDS/IBM/PSSP.SampleDaeMon/StaticVars/static_var1      Example
               non-instantiable resource variable (Counter or Quantity
```

Figure 13. Partial Listing from xmpeek

Note: In Figure 13, a value has been added as subcontext to a context called DDS/IBM, and is named static_var1.

This information is referenced in 2.4.5, “Resource Variable Names.”

Resource Variables of type State send the data as a message using the communication path established between the Event Management daemon and the Resource Monitor.

The communication path for Resource Variables is one of the following:

- Shared memory
- Messages

2.4.5 Resource Variable Names

A Resource Variable name is a series of two or more components separated by a period. Each component is a character string that starts with an alphabetic character. Any component other than the first may also begin with a percent sign. The last component of a Resource Variable name is the resource attribute. All preceding components represent the name of the resource.

The name represents a hierarchical organization, from the general to the specific, similar to a fully qualified UNIX file name.

An example of a Resource Variable is as follows:

```
IBM.PSSP.Response.Host.state
IBM.PSSP.Response.Switch.state
IBM.PSSP.aixos.Disk.busy
IBM.PSSP.aixos.FS.%totused
IBM.PSSP.Prog.pcount
```

In order to avoid conflicts in the naming of Resource Variables, the first component of each variable name is the name of the vendor that supplied the subsystem containing the resource whose state is represented by the Resource Variable. For example, the vendor name for all IBM subsystems is IBM. The second component of each Resource Variable in an IBM subsystem is the name of the product containing the subsystem.

Example: IBM.PSSP...

To get an overview of all predefined Resource Variables, use the example shown in Figure 14.

```
# SDRGetObjects EM_Resource_Variable rvName rvClass | more

rvName                         rvClass
IBM.PSSP.Response.Host.state   IBM.PSSP.Response
IBM.PSSP.Response.Switch.state IBM.PSSP.Response
.
IBM.PSSP.aixos.Disk.busy       IBM.PSSP.aixos.Disk
.
IBM.PSSP.aixos.FS.%totused     IBM.PSSP.aixos.FS
.
IBM.PSSP.Prog.pcount           IBM.PSSP.Prog
.
```

Figure 14. Resource Variable Names

Currently, 367 Resource Variables are defined.

All Resource Variables and their related attributes are defined in the SDR Class EM_Resource_Variable. The attributes for a Resource Variable are:

rvName A string that contains the name of the Resource Variable.

rvDescription This is an index (ID) to the textual description of the variable.

rvValue_type This is the value type as described in 2.4.2, "Resource Variable Types" on page 31. The type must be one of Counter, Quantity or State.

rvData_type A string indicates the data type of the variable. It may be one of the following: long, float, or Structured Byte String (SBS).

rvInitial_value This is the Resource Variable′s initial value; it is the value that exists before the Resource Variable is observed for the first time.

rvClass All Resource Variables are grouped into classes (see 2.4.6, "Classes of Resource Variables" on page 42). These entries are pointers to the EM_Resource_Class SDR object.

rvPTX_name This name is used to read and write the variable on the Performance Toolbox/6000 shared memory (SPMI). It is only required for variables of type Counter and Quantity.

rvPTX_description

This is a string that contains a comma-separated list of message IDs that correspond to the components of the variable′s PTX name.

rvLocator This is a string that contains either an Instance Vector element name or a null string.

rvPredicate This is the default predicate, and its value is optional.

rvEvent_description

This is the ID of the message that contains a short description of an event.

rvDynamic_instance

This is a Boolean value. True (non-zero) indicates that instances of this Resource Variable are created dynamically by the Resource Monitor whenever an instance is referenced through the EMAPI.

rvIndex_vector This is a string that contains either an Instance Vector element name or a null string, and is optional.

Note: For more information, refer to Chapter 2 in the *RS/6000 SP: Event Management Programming Guide and Reference*, SC23-3996.

The example program generates three Resource Variables of type Quantity. The command shown in Figure 15 creates one of these three Resource Variables.

```
# SDRCreateObjects EM_Resource_Variable \
                   ′rvName=IBM.PSSP.SampleDaeMon.StaticVars.static_var1′  \
                   ′rvLocator=NodeNum′ \
                   ′rvDescription=3′ \
                   ′rvValue_type=Quantity′ \
                   ′rvInitial_value=0′ \
                   ′rvData_type=long′ \
                   ′rvPTX_name=StaticVars/static_var1′ \
                   ′rvPTX_description=7′ \
                   ′rvPTX_min=0′ \
                   ′rvPTX_max=500′ \
                   ′rvClass=IBM.PSSP.SampleDaeClass′ \
                   ′rvDynamic_instance=0′
```

Figure 15. Example for Creating Resource Variables

Let us have a closer look at each entry:

1. rvName

   This is the name of the first Resource Variable.

2. rvLocator

   If the Instance Vector for this resource implies the resource′s location, the value of this item is the name of the vector element whose value is the number of the node that contains the resource instance.

3. rvDescription

   This item identifies a test string ID within a message file. For details, see Figure 16 on page 42.

4. rvValue_type

   As already meantioned, the example program generates Resource Variables of type Quantity.

5. rvInitial_value

   The initial value in this case is zero.

6. rvData_type

   The data type for the Quantity variable is long.

7. rvPTX_name

   The last line of Figure 13 on page 38 is:
   /sp21n01/DDS/IBM/PSSP.SampleDaeMon/*StaticVars/static_var1*
   The highlighted part is exactly what is appended to the DDS variable by the rvPTX_name. PSSP.SampleDaeMon will be defined within the EM_Resource_Monitor class.

8. rvPTX_description

   This ID points to a message string within a message file.

9. rvPTX_min rvPRX_max

   These values define the plotting range for the Performance Monitor.

10. rvClass

    This is a link to the appropriate entry in the EM_Resource_Class.

11. rvDynamic_instance

    This value is set to false (rvDynamic_instance=0), which means this variable is static (see definition).

Having created the first Resource Variable, let us see how the message IDs are converted into text strings during run time. The message file for the example is shown in Figure 16 on page 42.

```
.
$ Lines beginning with a dollar sign are comments.  Lines
$ beginning with numbers are the messages.
$
1 ″IBM Data Suppliers″
$
2 ″RMAPI Sample Monitor″
$
3 ″Example non-instantiable resource variable (Counter or Quantity).″
$
4 ″Example dynamically instantiable resource variable.″
$
5 ″Example instantiable resource variable (Counter or Quantity).″
$
6 ″Example instantiable resource variable (State).″
$
7 ″Example non-instantiable variables.″
$
8 ″Example instantiable variables.″
$
9 ″Example ivector NodeNum. Used to instantiate variables as node \n\
specific and as the variable locator field.″
$
10 ″Example instance vector for creating multiple instances of variables.″
$
11 ″Example SBS field: Action. Last action requested by the sample ...″
$
12 ″Example SBS field: Options. Last options parameter passed to ...″
$
13 ″Example SBS field: StateChange. Success of last sample ...″
$
14 ″Example SBS field: State. State of the resource.″
$
```

Figure 16. Example Message File rmapi_smp.msg

Every ID in the Description areas points to a string in this message file, which is converted to the AIX internal message catalog format by using the command:

```
# runcat rmapi_smp rmapi_smp.msg
#
```

This will create the file rmapi_smp.cat, which needs to be copied to the NLS directory for the language set by your environment.

2.4.6 Classes of Resource Variables

As we have seen in Figure 14 on page 39, Resource Variables are combined into classes. Currently, 17 classes are defined. To get an overview of these, use the command shown in Figure 17 on page 43

```
# SDRGetObjects EM_Resource_Class

rcClass              Resource_monitor Observation_interval Reporting_int.
IBM.PSSP.CSS         IBM.PSSP.harmld         5                    5
IBM.PSSP.HARMLD      IBM.PSSP.harmld        30                   30
IBM.PSSP.LL          IBM.PSSP.harmld       250                  250
IBM.PSSP.Membership  Membership              0                    0
IBM.PSSP.PRCRS       IBM.PSSP.harmld     86400                86400
IBM.PSSP.Prog        IBM.PSSP.harmpd         0                    0
IBM.PSSP.Response    Response                0                    0
IBM.PSSP.SP_HW       IBM.PSSP.hmrmd          0                    0
IBM.PSSP.VSD         IBM.PSSP.harmld        10                   10
IBM.PSSP.aixos.CPU   aixos                  15                    0
IBM.PSSP.aixos.Disk  aixos                  30                    0
IBM.PSSP.aixos.FS    aixos                  60                    0
IBM.PSSP.aixos.LAN   aixos                  40                    0
IBM.PSSP.aixos.Mem   aixos                  15                    0
IBM.PSSP.aixos.PagSp aixos                  30                    0
IBM.PSSP.aixos.Proc  aixos                  60                    0
IBM.PSSP.pm          IBM.PSSP.pmanrmd        0                    0
#
```

Figure 17. Resource Variable Classes

Note: The Resource Class is used for grouping Resource Variables with the same resource characteristics, like the Resource Monitor, observation interval, and reporting frequency.

Following are the items included in the Resource Class definition:

rcClass

This class name represents a name that uniquely defines the resource class. It is referenced from the EM_Resource_Variable SDR class.

rcResource_monitor

This item specifies the Resource Monitor definition in the EM_Resource_Monitor class that supplies data to the variables of this class.

rcObservation_interval

The Observation Interval specifies the amount of time, in seconds, between each observation of any Counter or Quantity Resource Variable.

rcReporting_interval

The Reporting Interval is the amount of time in seconds between each update of any Counter or Quantity by the corresponding Resource Monitor. The Resource Monitor can ask the Event Management daemon for this value.

Note: The value of the observation interval must be greater than or equal to the value of the reporting interval.

For creating the appropiate EM_Resource_Class entries for the example, issue the command shown in Figure 18 on page 44.

```
# SDRCreateObjects EM_Resource_Class \
                   ′rcClass=IBM.PSSP.SampleDaeClass′ \
                   ′rcResource_monitor=IBM.PSSP.SampleDaeMon′  \
                   ′rcObservation_interval=10′  \
                   ′rcReporting_interval=10′
#
```

Figure 18. Example EM_Resource_Class Entries

The items are:

1. rcClass

   This is the name of the newly created Resource Class, IBM.PSSP.SampleDaeClass.

2. rcResource_monitor

   This is a pointer to the EM_Resource_Monitor class. It is covered in more detail in 2.5, “Resource Variable Definition” on page 48.

3. rcObservation_interval

   The values for the observation and reporting interval are equal.

4. rcReporting_interval

   For more information about the observation and reporting interval values see 2.8, “Event Generation” on page 57.

2.4.7 Resource Variable Instance Vector

Most resources in a system have multiple copies. For example, there is more than one disk per node, more than one logical volume per node, more than one CPU per node, more instances of a database, and certainly additional nodes.

The Resource Variables that represent the states of these resources also have multiple copies. Each of these copies is called an instance of the Resource Variable.

To uniquely identify each copy of a resource and all of its variables, each resource in the system has one, and only one, associated *Instance Vector*. An Instance Vector is a list of elements, where each element is a name/value pair. The name describes the element. The set of values in the Instance Vector uniquely identifies the copy of the resource in the system. By extension, these values also uniquely identify the copy of the Resource Variable in the system. If there is only one copy of the resource in the system, its Instance Vector is null.

Examples of Resource Variables and Instance Vectors are:

IBM.PSSP.Response.Host.state (NodeNum=15)
IBM.PSSP.Response.Switch.state (NodeNum=1)
IBM.PSSP.aixos.Disk.busy (NodeNum=5, Name=hdisk10)
IBM.PSSP.aixos.FS.%totused (NodeNum=3, VG=spdata, LV=lv00)
IBM.PSSP.Prog.pcount (NodeNum=2, UserName=″root″, ProgName=″dbserv″)

Note: A Resource Variable has a name and an Instance Vector.

In order to see how many elements an Instance Vector for a particular Resource Variable consists of, use the command in Figure 19 on page 45.

```
# SDRGetObjects EM_Instance_Vector

ivResource_name               ivElement_name    ivElement_description
IBM.PSSP.CSS                  NodeNum            30
IBM.PSSP.HARMLD               NodeNum            30
IBM.PSSP.LL.SCHEDD            NodeNum            30
IBM.PSSP.LL.SCHEDD            SCHEDD             33
IBM.PSSP.LL.STARTD            STARTD             32
IBM.PSSP.LL.STARTD            NodeNum            30
IBM.PSSP.Membership.LANAdapter NodeNum            3
IBM.PSSP.Membership.LANAdapter AdapterType        4
IBM.PSSP.Membership.LANAdapter AdapterNum         5
IBM.PSSP.Membership.Node      NodeNum             6
IBM.PSSP.PRCRS                NodeNum            30
IBM.PSSP.Prog                 ProgName            3
IBM.PSSP.Prog                 UserName            4
IBM.PSSP.Prog                 NodeNum             5
IBM.PSSP.Response.Host        NodeNum             3
IBM.PSSP.Response.Switch      NodeNum             4
IBM.PSSP.SP_HW.Frame          FrameNum          255
IBM.PSSP.SP_HW.Node           NodeNum           256
IBM.PSSP.SP_HW.Switch         SwitchNum         257
IBM.PSSP.VSD                  NodeNum            30
IBM.PSSP.VSD                  VSD                31
IBM.PSSP.VSDdrv               NodeNum            30
IBM.PSSP.aixos.CPU            NodeNum            42
IBM.PSSP.aixos.Disk           NodeNum            45
IBM.PSSP.aixos.Disk           Name               46
IBM.PSSP.aixos.FS             NodeNum            47
IBM.PSSP.aixos.FS             VG                 48
IBM.PSSP.aixos.FS             LV                 49
IBM.PSSP.aixos.LAN            Adapter            53
IBM.PSSP.aixos.LAN            NodeNum            52
IBM.PSSP.aixos.Mem.Kmem       Type               55
IBM.PSSP.aixos.Mem.Kmem       NodeNum            54
IBM.PSSP.aixos.Mem.Real       NodeNum            56
IBM.PSSP.aixos.Mem.Virt       NodeNum            57
IBM.PSSP.aixos.PagSp          NodeNum            60
IBM.PSSP.aixos.Proc           NodeNum            61
IBM.PSSP.aixos.VG             NodeNum            50
IBM.PSSP.aixos.VG             VG                 51
IBM.PSSP.aixos.cpu            CPU                44
IBM.PSSP.aixos.cpu            NodeNum            43
IBM.PSSP.aixos.pagsp          NodeNum            58
IBM.PSSP.aixos.pagsp          Name               59
IBM.PSSP.pm                   NodeNum             3
#
```

Figure 19. Event Management Instance Vector

For more detailed analysis, take a closer look at the screen output in Figure 19. If you want to know, for example, how many Instance Vector elements are required for the IBM.PSSP.aixos.FS Resource Variable, use the command shown in Figure 20 on page 46.

```
# SDRGetObjects EM_Instance_Vector | grep IBM.PSSP.aixos.FS

IBM.PSSP.aixos.FS            NodeNum          47
IBM.PSSP.aixos.FS            VG               48
IBM.PSSP.aixos.FS            LV               49
#
```

Figure 20. Instance Vector Example

From Figure 20, you can see that three Instance Vector elements are required to uniquely identify a Resource Variable. For example, look for an Instance Vector that describes the use of the /tmp file system, which is always located on logical volume hd3 and always belongs to the rootvg volume group. The Instance Vector for /tmp on Node 3 is:

```
IBM.PSSP.aixos.FS.%totused (NodeNum=3, VG=rootvg, LV=hd3)
```

2.4.8 Event Management SDR Classes

Up to now, we have seen that all Resource Monitor-related information is defined in the SDR. We have already used information from SDR classes such as EM_Resource_Class, EM_Resource_Variable and EM_Instance_Vector, but we did not describe in detail what these SDR classes are.

In this section, we provide details about the information contents of these classes and how they are related to each other. This knowledge is useful for understanding examples in later chapters.

To get an overview of which Event Management-related SDR classes are defined, use the command in Figure 21.

```
# SDRListClasses | grep EM

EM_Condition
EM_Instance_Vector
EM_Resource_Class
EM_Resource_Monitor
EM_Resource_Variable
EM_Structured_Byte_String
```

Figure 21. SDR Classes

More information about each of these classes follows:

EM_Condition

This class is only used by Perspectives to save predicates. For details see Chapter 5, “SP Perspectives GUI” on page 137. classes.

EM_Instance_Vector

This class describes the Instance Vectors for each resource and their associated Resource Variables.

EM_Resource_Class

This class contains information about the characteristics of obtaining Resource Variables. This class groups Resource Variables with the same basic characteristics into groups. It references the EM_Resource_Monitor for the Resource Monitor definitions.

EM_Resource_Monitor

This SDR class contains information about every Resource Monitor program.

EM_Resource_Variable

This class references the EM_Instance_Vector, EM_Structured_Byte_String and EM_Resource_Class classes. References to the EM_Instance_Vector class are made through the resource name portion of the Resource Variable name (the Resource Variable name without the final component).

EM_Structured_Byte_String

The Structured Byte String variable descriptions are stored in this class and are referenced from the EM_Resource_Variables class.

The relations between the Event Management SDR classes are shown in Figure 22.

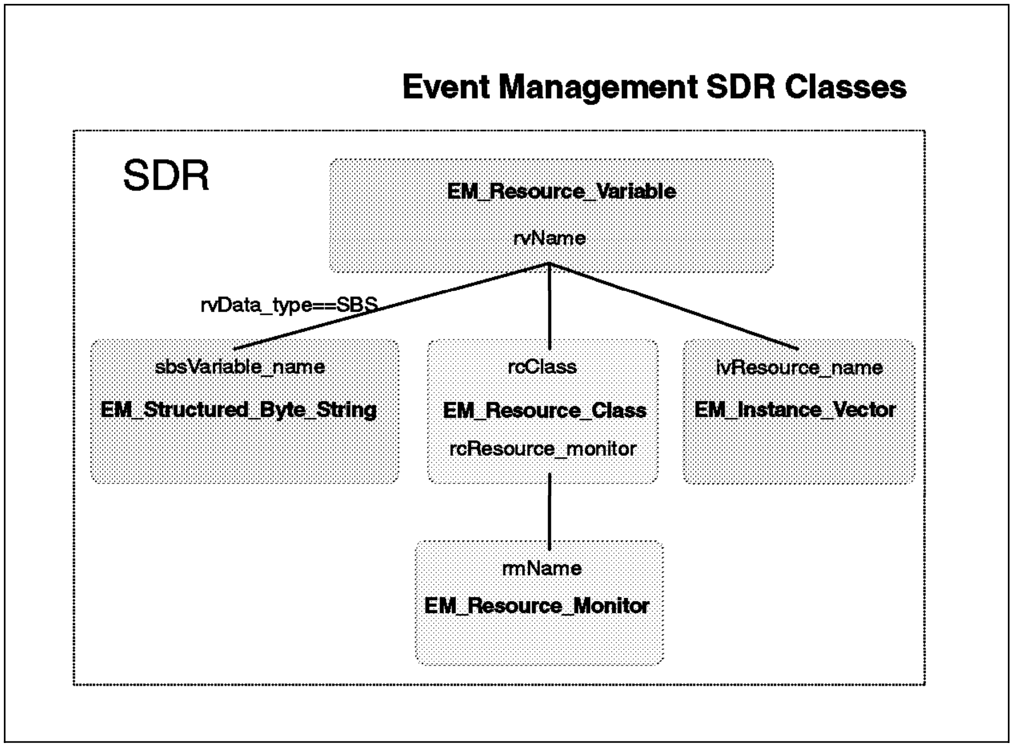


Figure 22. Event Management SDR Classes

As already mentioned, the EM_Resource_Variable contains information about all defined Resource Variables. These are grouped into resource classes with the same basic characteristics. Every resource class has a dedicated Resource Monitor and every Resource Variable consists of a name and an Instance Vector. More details for each field of a Structured Byte String variable can be found in the EM_Structured_Byte_String class.

2.5 Resource Variable Definition

Resource Variable Definition is the process by which Resource Variables are made known to Event Management. The definition of a Resource Variable includes the following information:

- Resource Variable name
- Resource Variable Instance Vector
- Resource Variable description
- Default predicate (optional)
- Name of the Resource Monitor that supplies the Resource Variable

The Resource Variable description should describe the semantics of the variable. This information is necessary to understand how and why to specify predicates.

In order to define a Resource Variable, you need more information about all the attributes of the SDR class EM_Resource_Monitor. Following is an overview of the SDR attributes of the EM_Resource_Monitor and their meanings:

rmName This is the name of the Resource Monitor definition. It must be unique in the database, and it uniquely identifies the SDR object in the EM_Resource_Monitor class. It is referenced from the EM_Resource_Class SDR class.

rmPath This specifies the executable path name of the Resource Monitor if it is a daemon that is startable by the Event Management daemon. If this is not possible (for example, if the Resource Monitor is incorporated into the application), then the entry is a null string.

rmArguments This is an optional string of arguments that are passed to the executable path when starting the Resource Monitor daemon.

rmMessage_file This specifies the name of the message catalog for the Resource Monitor. This catalog contains descriptions for all of this monitor′s Resource Variables.

rmMessage_set This specifies the message set within the message catalog.

rmConnect_type This indicates whether the Event Management daemon connects to the Resource Monitor (server) or whether the Resource Monitor connects to the Event Management daemon (client).

rmPTX_prefix This name is used to read and write the variable in the PTX/6000 shared memory (SPMI). It is only required for variables of type Counter and Quantity. It is a variable name known to PTX/6000.

rmPTX_description This is an ID pointing to a textual description for PTX/6000.

rmPTX_asnno An integer that is the ASN.1 number equal to the SNMP Assigned Enterprise Number for the vendor that supplies this Resource Monitor. For IBM-supplied Resource Monitors, this value is 2. See also *AIX Performance Toolbox/6000 User′s Guide*.

The example in Figure 23 on page 49 creates the entries for the EM_Resource_Monitor class.

```
# SDRCreateObjects EM_Resource_Monitor \
                   ′rmName=IBM.PSSP.SampleDaeMon′ \
                   ′rmPath=′/usr/lpp/ssp/samples/haem/rmapi/rmapi_smpdae′ \
                   ′rmMessage_file=rmapi_smp.cat′ \
                   ′rmMessage_set=1′ \
                   ′rmConnect_type=server′ \
                   ′rmPTX_prefix=IBM/PSSP.SampleDaeMon′ \
                   ′rmPTX_description=1,2′ \
                   ′rmPTX_asnno=2′
```

Figure 23. Create EM_Resource_Monitor Entries

Let us take a closer look at each item for the EM_Resource_Monitor entries.

1. rmName

   This is the name of the Resource Monitor definition.

2. rmPath

   This is the name of our sample Resource Monitor daemon. It is located in the PSSP 2.2 production path /usr/lpp/ssp/samples/haem/rmapi.

3. rmMessage_file

   This is the name of the message file. It should be located in the NLS directory for the language set by your environment. All description IDs point to the messages in this file.

4. rmMessage_set

   The message catalogs support multiple message sets. Each set can be dedicated to a special subsystem. Within each message set, the message IDs are unique. Only one message set is supported in the example in Figure 23.

5. rmConnect_type

   The sample Resource Monitor daemon is of type server, so the Event Management subsystem can start this daemon as soon as a client registers for the appropriate Resource Variables.

6. rmPTX_prefix

   The last line of Figure 13 on page 38 is: `/sp21n01/DDS/IBM/PSSP.SampleDaeMon/StaticVars/static_var1`

   The middle part, `IBM/PSSP.SampleDaeMon`, is exactly what is appended to the DDS variable by the rvPTX_prefix. The last part, `StaticVars/static_var1`, is defined in the EM_Resource_Variable Class.

7. rmPTX_description

   See Figure 13 on page 38 for the messages on the second and third line starting with `/sp21n01/DDS`.

8. rvPTX_asnno

   This is the ASN.1 number assigned to IBM.

Note: Resource Variables may be defined at any time without disrupting the operation of Event Management. But in order to make it known to the Event Management subsystem, you need to stop and restart the Event Management daemons. See the next section for more information.

2.6 Pseudocode for a Resource Monitor

Having created the SDR entries for the Resource Monitor, let us have a look at how to code a daemon-based Resource Monitor. This section only presents a program flow of such a daemon. For details, see Chapter 4, “Application Program Interfaces (APIs)” on page 97, or *Event Management Programming Guide and Reference*, SC23-3996.

A daemon-based Resource Monitor performs the following functions:

- Initializing
- Creating a server session
- Registering Resource Variables and their instances
- Starting a session
- Getting control messages from the Event Management subsystem
- Processing the control messages
- Ending a session
- Terminating

Figure 24 shows a pseudocode outline of a Resource Monitor that uses RMAPI subroutines.

```
Call ha_rr_init

Call ha_rr_makserv
     Add the file descriptor (fdc) to the select mask.

Call ha_rr_reg_var with all known variable instances.

Loop on select
     If fdc is ready
            Call ha_rr_start_session
                 Add the file descriptor (fdN) to the select mask.

     If fdN is ready
            Call ha_rr_get_ctrlmsg with fdN

            Process the control message

            If the session is disconnected
                   Call ha_rr_end_session

end Loop

Call ha_rr_terminate.
```

Figure 24. Pseudocode of a Daemon-Based Resource Monitor

See also the source code examples. A Makefile for the examples can be found in Appendix H, "How to Get the Examples in This Book" on page 257.

2.7 Event Management Configuration

Event Management requires information about which Resource Variables are defined and which predicates should be applied to them. The Resource Monitors will supply these variables. All this information for the Event Management subsystem is placed in the Event Management Configuration Database (EMCDB), which includes the following:

- Resource Variable definition
- Resource Class definition
- Resource Monitor definition

The EMCDB is a binary database. Therefore, if you add new Resource Variables and new Resource Monitor definitions to the SDR, the next step is to compile this modified SDR into a form readable for the Event Management daemon. This is done by using the haemcfg command, which compiles the EMCDB into a binary form placed in a file. This file is created in the /spdata/sys1/ha/cfg directory, which is used as a staging directory. The haemcfg command also updates the SDR Syspar class with the new version of the configuration. During the compilation, the last compiled version of the EMCDB in the staging directory is renamed by appending its version string to its name (the version string is a time stamp).

When the Event Management daemon starts up, it determines the correct version of the EMCDB to use by checking the SDR Syspar class and then matching the version string found there against the version string used by the other running Event Management daemon. The version string used by the already running Event Management daemons takes precedence. The correct copy of the EMCDB is then copied from the staging area on the Control Workstation to /etc/ha/cfg. The haemrcpcdb command is used for this transfer. For more details about the Event Management configuration, see the *RS/6000 SP: Administration Guide*, GC23-3897.

So far we have created the SDR entries for the sample Resource Monitor daemon. Now we need to compile the EMCDB. Before doing this, let us look at the SP configuration we used for this test in order to understand the following screen output.

The SP was divided into two partitions; Partition 1 was named sp21en0 and Partition 2 was named sp21en1. The system was equipped as follows:

- Partition1 = sp21en0
  - High Node = sp21n01
  - Thin Node = sp21n05
  - Thin Node = sp21n06
  - Thin Node = sp21n07
  - Thin Node = sp21n08
- Partition2 = sp21en1

- Wide Node = sp21n09
- Wide Node = sp21n11
- Wide Node = sp21n13
- Wide Node = sp21n15

Event Management configuration requires several steps. From the Control Workstation, store the resource definitions as a set of objects in the SDR. There are several ways to do this.

The preferred way to load your data for a new Resource Monitor into the SDR is to create a load list file and run the haemloadcfg command. For details on how to do this, see the haemloadcfg command in the *RS/6000 SP: Command and Technical Reference*, GC23-3900.

Another solution for loading data into the SDR is to use commands such as the SDRCreateObjects command. You can do this most easily by creating a shell script with the appropriate commands. The steps to configure a new EMCDB are listed in Figure 25.

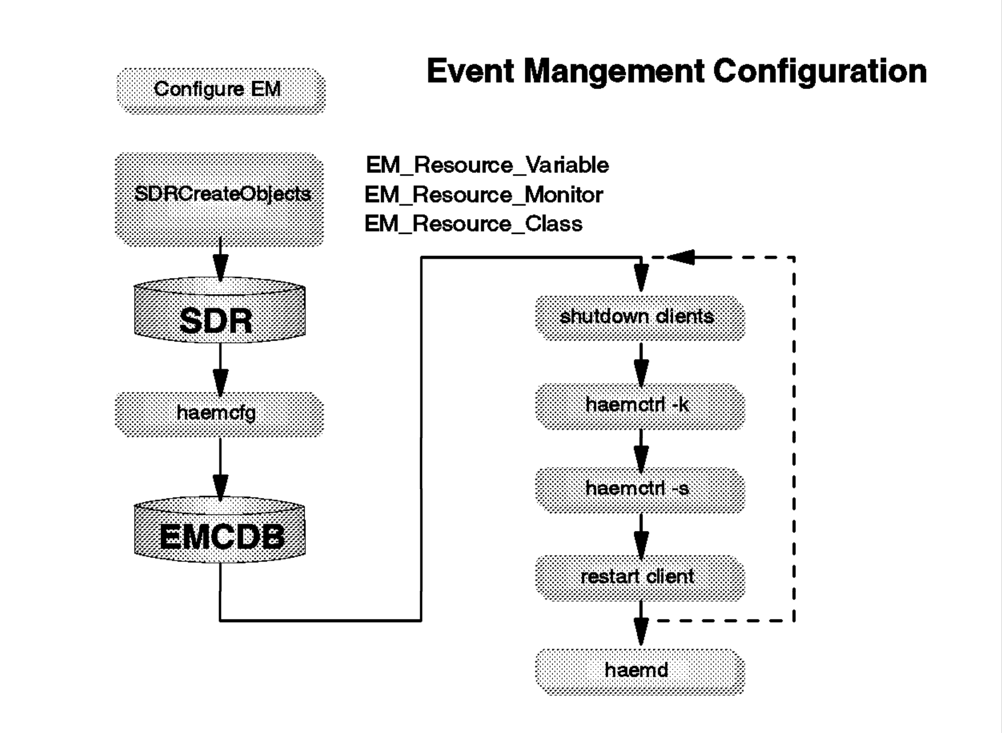


Figure 25. Event Management Configuration Steps

The configuration consists of the following steps:

1. Create the SDR objects that uniquely identify the entries for the EM_Resource_Variable, EM_Resource_Class, and EM_Resource_Monitor SDR classes.

   See 2.4.5, "Resource Variable Names" on page 38, 2.4.6, "Classes of Resource Variables" on page 42, and 2.5, "Resource Variable Definition" on page 48.

2. Compile the SDR data to the Event Management Configuration Database (EMCDB) by using the haemcfg command, as shown in Figure 26 on page 53.

```
# export SP_NAME=sp21en0
# haemcfg

haemcfg: Reading Event Management data for partition: sp21en0.
haemcfg: Created EMCDB file: /spdata/sys1/ha/cfg/em.sp21en0.cdb \
                             Version: 843059753,172068608,0.
#
```

Figure 26. Compile the SDR Data to EMCDB

Now look at the appropriate subdirectories:

```
# cd /spdata/sys1/ha/cfg
# ls -l

total 528
-rw-r--r-- 1 root system 89284 Sep  9 15:29 em.sp21en0.cdb
-rw-r--r-- 1 root system 89284 Sep  9 10:01 em.sp21en0.cdb.842281279,950688768,0
-rw-r--r-- 1 root system 89284 Sep  9 15:30 em.sp21en1.cdb

# cd /etc/ha/cfg
# ls -l

total 368
-rw-r--r-- 1 root system 89284 Sep  9 15:29 em.sp21en0.cdb
-rw-r--r-- 1 root system    22 Sep 13 10:28 em.sp21en0.cdb_vers
-rw-r--r-- 1 root system 89284 Sep  9 15:30 em.sp21en1.cdb
-rw-r--r-- 1 root system    21 Sep 13 10:28 em.sp21en1.cdb_vers
#
```

Figure 27. Event Management Database-Related Subdirectories

The screen in Figure 27 lists all the files before starting the compilation. The screen in Figure 28 on page 54 shows the situation after the compilation.

Note that each time you update the EMCDB for a system partition, haemcfg modifies the version number. The version number allows each instance of the Event Management subsystem and each Resource Monitor in a system partition to verify that they have access to the correct version of configuration information. The haemcfg command maintains the version number in both the EMCDB and the SDR.

Each database may be updated at any time without interfering with the operation of any Event Management subsystem, client, or Resource Monitor. However, once a database has been updated in a system partition, the Event Management subsystem in that partition must be restarted to refresh its configuration information from the newly updated database.

```
# cd /spdata/sys1/ha/cfg
# ls -l

total 720
-rw-r--r-- 1 root system 95332 Sep 18 10:15 em.sp21en0.cdb
-rw-r--r-- 1 root system 89284 Sep  9 10:01 em.sp21en0.cdb.842281279,950688768,0
-rw-r--r-- 1 root system 89284 Sep  9 15:29 em.sp21en0.cdb.842300962,458441216,0
-rw-r--r-- 1 root system 89284 Sep  9 15:30 em.sp21en1.cdb

# cd /etc/ha/cfg
# ls -l

total 384
-rw-r--r-- 1 root system 95332 Sep 18 10:15 em.sp21en0.cdb
-rw-r--r-- 1 root system    22 Sep 18 10:30 em.sp21en0.cdb_vers
-rw-r--r-- 1 root system 89284 Sep  9 15:30 em.sp21en1.cdb
-rw-r--r-- 1 root system    21 Sep 13 10:28 em.sp21en1.cdb_vers
#
```

Figure 28. Event Management Database Subdirectories with New EMCDB

3. Stop all clients of the Event Management subsystem, such as the Problem Management subsystem and Perspectives.

4. Stop all Event Management daemons:

```
# haemctrl -k haem.sp21en0

0513-044 The stop of the haem.sp21en0 Subsystem was completed successfully.

# lssrc -a | grep haem

 haem.sp21en1     haem             24820   active
 haem.sp21en0     haem                     inoperative

# cat /tmp/wcoll | dsh -w - /usr/lpp/ssp/bin/haemctrl -k
sp21n01: 0513-044 The stop of the haem Subsystem was completed successfully.
sp21n05: 0513-044 The stop of the haem Subsystem was completed successfully.
sp21n06: 0513-044 The stop of the haem Subsystem was completed successfully.
sp21n07: 0513-044 The stop of the haem Subsystem was completed successfully.
sp21n08: 0513-044 The stop of the haem Subsystem was completed successfully.
#
```

Figure 29. Shutdown of the Event Management Daemons

5. Restart the Event Management daemons:

```
# haemctrl -s

0513-059 The haem.sp21en0 Subsystem has been started. Subsystem PID is 14580.

# cat /tmp/wcoll | dsh -w - /usr/lpp/ssp/bin/haemctrl -s

sp21n01: 0513-059 The haem Subsystem has been started. Subsystem PID is 13066.
sp21n05: 0513-059 The haem Subsystem has been started. Subsystem PID is 6280.
sp21n06: 0513-059 The haem Subsystem has been started. Subsystem PID is 14944.
sp21n07: 0513-059 The haem Subsystem has been started. Subsystem PID is 15338.
sp21n08: 0513-059 The haem Subsystem has been started. Subsystem PID is 13886.
#
```

Figure 30. Restart of the Event Management Daemons

6. Check all Event Management daemons with commands like `lssrc` or `ps -ef`.

```
# lssrc -l -s haem.sp21en0 | awk '{print NR $0}'

 1 Subsystem         Group            PID     Status
 2  haem.sp21en1     haem             24820   active
 3
 4 Trace flags set:  None
 5
 6 Configuration Data Base version: 842301020,69089024,0(SDR)
 7
 8 Daemon started on 09/09/96 at 15:37:41.670352384
 9    running 0 days, 0 hours, 6 minutes and 58 seconds
10 Daemon connected to group services: TRUE
11 Daemon has joined peer group:       TRUE
12 Daemon communications enabled :     TRUE
13 Peer count:                         5
14
15 Logical Connection Information
16  Type   LCID   FD  Node/PID  Start Time
17 peer       0            6
18 peer       1            7
19 peer       2            5
20 local     10   21    41158   Mon Sep 09 18:22:23 1996
21 local     12   18    26378   Mon Sep 09 15:39:19 1996
22 local     13   22    41158   Mon Sep 09 18:22:39 1996
23 local     26   14    15794   Mon Sep 09 17:19:10 1996
24 local     27   15    56226   Mon Sep 09 17:37:12 1996
25
26 Resource Monitor Information
27   Resource Monitor Name      Type     FD    PID
28 IBM.PSSP.harmld            server     19   62196
29 IBM.PSSP.harmpd            server     20   29430
30 IBM.PSSP.hmrmd             server     13   45896
31 IBM.PSSP.pmanrmd           client     17      -1
32 Membership                 internal   -1      -1
33 Response                   internal   -1      -1
34 aixos                      internal   -1      -1
35 IBM.PSSP.SampleDaeMon      server     16   36174
36
37 Highest file descriptor in use is 22
38
39 Peer Daemon Status
40    0 S S     1 I A     2 - A     3 - A     4 - A     5 I A
41    6 I A     7 I A     8 I A
.
```

Figure 31. List of the lssrc Command

The partial screen output shown in Figure 31 gives you the following information:

- Line 6

  Compare the version number of the Configuration Database to the version number from Figure 28 on page 54. In the staging area /spdata/sys1/ha/cfg, the old and new versions of the EMCDB are still there. The Event Management daemon for Partition sp21en0 is using the newest version.

- Line 13

  `Peer count: 5` shows that there are 5 peer connections to other Event Management daemons. That is exactly the number of nodes in Partition sp21en0. The two last lines of the screen output also give you this information.

- Lines 20 - 24

  These lines show you that there are five external clients locally connected to the Event Management daemon. Peer connections are listed for those Event Management daemons that have sent requests to this Event Management daemon.

- Lines 28 - 35

  These lines give more details about which external and internal monitors are up and running.

- Lines 40 - 41

  The Peer Daemon Status gives you the following information:

  - The first field gives you the node number [**1** I A].

  - The second field gives you the connection state [1 **I** A]. I means *In the group*. It indicates that the peer on the specified node is a peer group member.

    O means *Out of the group*. It indicates that the peer is no longer a peer group member (but was at one time).

    - indicates that no peer on that node has ever joined the group.

    S in the second and third position indicates that this is the node on which the daemon is running.

  - The third field gives you group information [1 I **A**]

    A means *Accepting join requests*, which indicates

    the node is accepting join requests from the group.

    R means *Rejecting join requests*, which indicates the node is rejecting join requests from the group.

    This status information should be seen only for a short period (15 seconds) during a node recovery.

7. Restart all the clients of the Event Management subsystem.

8. Repeat steps 3 to 7 for all partitions.

For details, see the *RS/6000 SP: Administration Guide*, GC23-3897.

Note: The data in the EMCDB file, not the SDR, is the data that is used by the Event Management subsystem.

2.8 Event Generation

The Event Management daemon observes each Resource Variable at some periodic rate, which is configurable, or as the variable changes. At each observation, a predicate is applied. A predicate is a relational expression between the value of a Resource Variable and some other value. This other value may be a constant or a value of the variable from the prior observation. An event is generated if the Boolean value of the predicate is *true*.

Note: More than one predicate may be applied to a Resource Variable at the same observation.

In order to see the reporting rate of a Resource Variable, use the command at the top of Figure 32 on page 58.

```
# SDRGetObjects EM_Resource_Class

rcClass             Resource_monitor Observation_interval Reporting_int.
IBM.PSSP.CSS        IBM.PSSP.harmld        5                 5
IBM.PSSP.HARMLD     IBM.PSSP.harmld       30                30
IBM.PSSP.LL         IBM.PSSP.harmld      250               250
.
IBM.PSSP.VSD        IBM.PSSP.harmld       10                10
.
IBM.PSSP.aixos.CPU  aixos                 15                 0
IBM.PSSP.aixos.Disk aixos                 30                 0
.
```

Figure 32. Observation Interval Example

Changing the Observation or Reporting Intervals requires the following steps:

1. Change the appropriate entries in the EM_Resource_Class.
2. Compile the EMCDB as described.
3. Stop the Event Management subsystem clients.
4. Stop the Event Management subsystem.
5. Restart the Event Management subsystem.
6. Restart the Event Management subsystem clients.
7. Repeat steps 3 to 6 for every partition.

This is the same procedure as described in 2.7, “Event Management Configuration” on page 51. Currently, it is the only available solution with PSSP 2.2.

2.9 Event Registration and Notification

Event registration is the process that an application or subsystem uses to declare its desire to be notified of changes in the state of resources. The application or subsystem provides the following information:

- The name of the Resource Variable from which the event is generated
- The Instance Vector of the Resource Variable from which the event is generated

The variable Instance Vector may be wildcarded. Wildcarding is used to receive events generated from more than one variable instance.

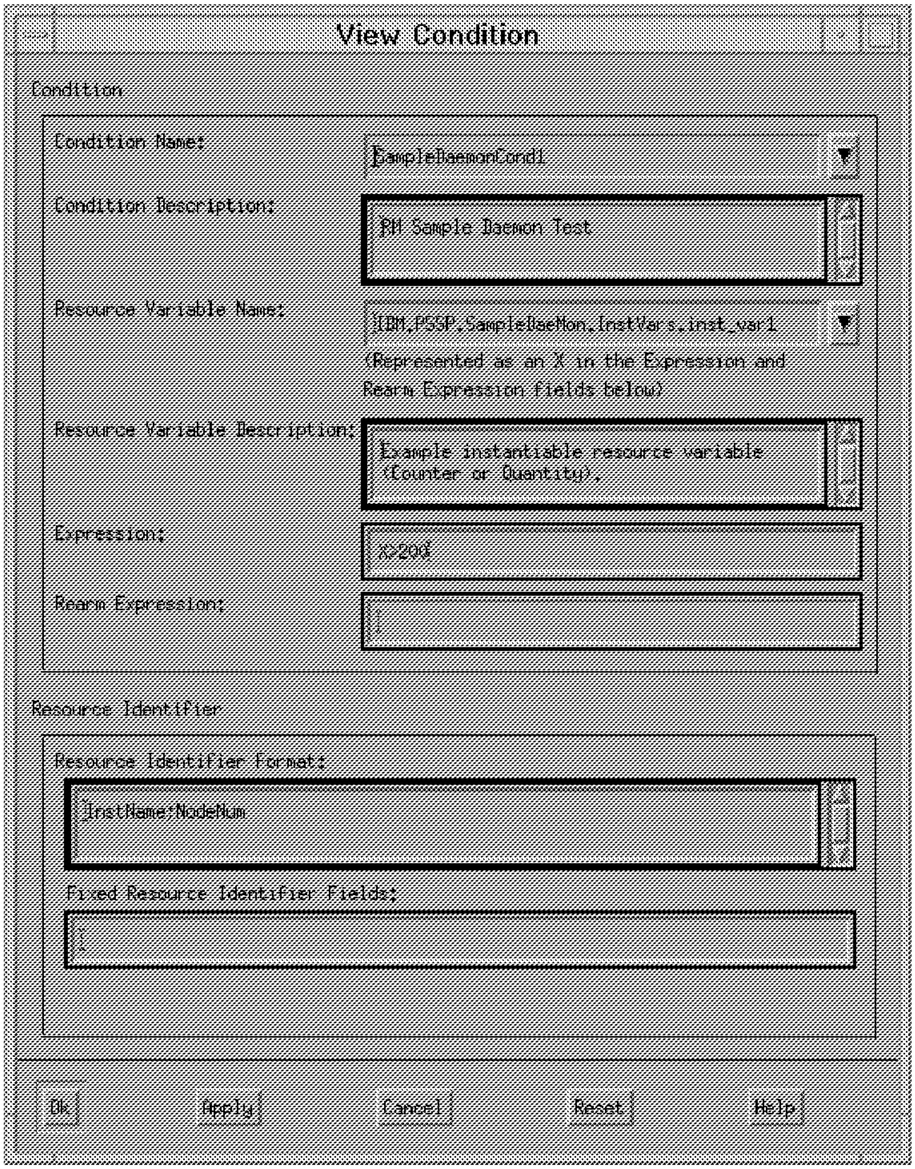
Each Resource Variable is optionally defined with a single default predicate for event generation. However, the application or subsystem may provide predicates to be applied against the variable in addition to, or instead of, the default predicate. These additional predicates are used to generate events for the calling application or subsystem.

As an event is generated when applying a predicate to a resource variable, and the result is true, it is sent to the application or subsystem which registered for that variable with that predicate.

Note: Applications or subsystems that registered for a variable without specifying a predicate receive the event generated by the application of the default predicate. Included with the event information is the current value of the Resource Variable specified in the predicate.

After adding the appropriate entries to the SDR class, compiling the EMCDB, and restarting the Event Management daemons, you should be able to register for your new Resource Variables. Perspectives is an easy-to-use tool to test the new Resource Monitor.

Remember that our Resource Monitor is generating random numbers from 0 to 500. Define an event within Perspectives, and you will be notified if the Resource Variable exceeds the value of 200. You may also want to send a message about this fact to all the users (by using the `wall` command).



Condition Name: SampleDaemonCond1

Condition Description: RM Sample Daemon Test

Resource Variable Name: IBM.PSSP.SampleDaeMon.InstVars.inst_var1

(Represented as an X in the Expression and Rearm Expression fields below)

Resource Variable Description: Example instantiable resource variable (Counter or Quantity).

Expression: X>200

Rearm Expression:

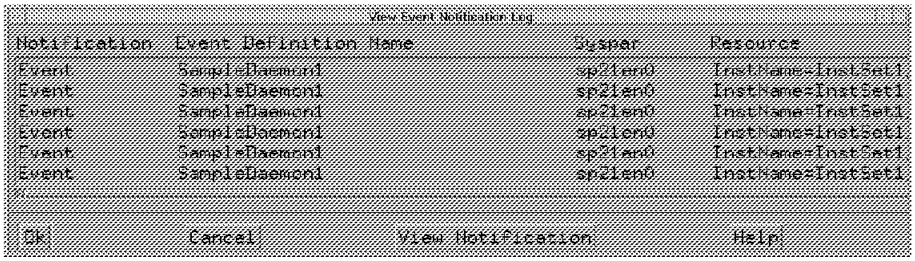
Resource Identifier Format: InstName;NodeNum

Fixed Resource Identifier Fields:

Figure 33. Event Perspective View Condition Dialog Box

The screen capture of the Event Perspective View Condition dialog box shows some of the definitions of our event. For details about Perspectives see Chapter 5, “SP Perspectives GUI” on page 137.

The notification output from Perspectives is shown in Figure 34.



Notification	Event Definition Name	Syspar	Resource
Event	SampleDaemon1	sp21en0	InstName=InstSet1
Event	SampleDaemon1	sp21en0	InstName=InstSet1
Event	SampleDaemon1	sp21en0	InstName=InstSet1
Event	SampleDaemon1	sp21en0	InstName=InstSet1
Event	SampleDaemon1	sp21en0	InstName=InstSet1
Event	SampleDaemon1	sp21en0	InstName=InstSet1

Figure 34. Event Notification Log

As you can see, the Resource Monitor is up and running and producing random numbers. You should also be able to monitor this Resource Variable by using PTX/6000. After starting PTX and defining a new console, you get the screen shown in Figure 35 on page 61.

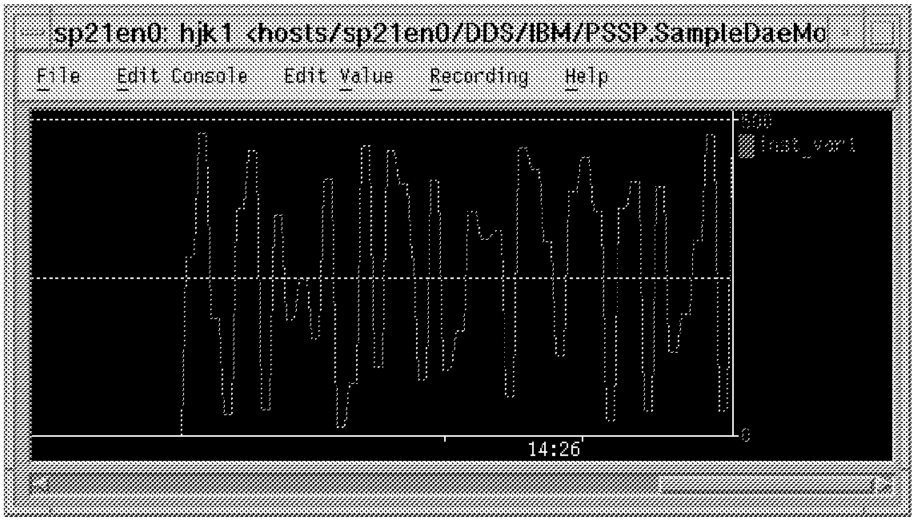


Figure 35. Monitoring Resource Variables with PTX

This gives an overview of monitoring the newly created Resource Variable.

2.10 Predicates

Predicates have already been mentioned, but not defined in detail. A predicate is a relational expression containing a Resource Variable in one or more modified forms. The evaluation of a predicate is Boolean; it occurs each time a Resource Variable is observed.

A predicate is of the following form:

predicate:: *expression rel_op* constant
expression rel_op expression
predicate log_op predicate
unary_op predicate
unary_op expression

expression:: *var_name*
var_name arith_op constant
var_name arith_op expression

var_name:: *X*
X@var_name_mod

var_name_mod:: One of:
P,R,PR, *sbs_sn*, P*sbs_sn*

unary_op:: !

rel_op:: One of:

= = ! = < > < = > =

log_op:: One of:

&& ||

arith_op:: One of:

* / % + -

sbs_sn:: A structured field serial number

Operators have the same meaning and the same precedence as in the C language. Parentheses may be used for grouping, as in C. Constants are integer or floating point, also as in C. The variable name modifier P indicates that the value of the variable from the *previous* observation is used. The variable name modifier R indicates that the raw value of the variable is used; this is only useful with a resource variable of type Counter.

When both of these modifiers are present, the raw value of the variable from its previous observation is used. The variable name modifier sbs_sn is a structured field serial number. This modifier is only used with an SBS Resource Variable and is used to select which structured field value is used in the evaluation of the expression. When used with P, the selected structured field is taken from the value of the variable from the previous observation.

Operands that are modified to select structure fields of type string or byte may only be used with the relational operators; both must be of the same type. When the operands are character strings, the implied comparison and its result are equivalent to the C library function strcmp(). When the operands are byte strings, the implied comparison and its result are equivalent to the C library function memcmp().

Predicates may only contain the name of a single Resource Variable. The variable name may be repeated in the predicate and may be one of its modified forms.

In the following example, we want to monitor the file system space of the /tmp file system. As soon as it is more than 80% used, we want an event to be generated. (This could be used, for example, to send mail to the system administrator or pop up a new red flashing window.) For this event, you need to define a predicate.

The Resource Variable name (var_name) is represented by an uppercase X, followed by a modifier:

```
var_name::  X
            X@var_name_modifier
```

The variable name modifier is one of the following:

X@P This means the previous value of the Resource Variable.

X@R This means the raw value.

X@PR This means the previous raw value.

X@sbs_sn The sbs_sn has to be substituted by the Structured Byte String field serial number. The predicate expresses the field value in the Structured Byte String identified by the serial number.

X@Pssb_sn The sbs_sn has to be substituted by the Structured Byte String field serial number. The predicate expresses the field value in the Structured Byte String identified by the serial number returned from the previous observation.

This is the predicate you are looking for:

```
X>80                 The Resource Variable is greater than 80
```

Another example for a valid predicate is the following:

```
(X@0!=X@1)&&(X@2<20)  If the first field (sbs_sn=0) of a Structured Byte
                      String is not equal to the second field (sbs_sn=1)
                      and at the same time the third field (sbs_sn=2) of
                      the structured byte field is less the 20, then
                      the expression becomes true and an event is
                      generated.
```

Note: A predicate may not include more than one Resource Variable.

For more information, refer to Chapter 4 in the *RS/6000 SP: Event Management Programming Guide and Reference*, SC23-3996.

2.11 PSSP 2.2 Resource Monitors

In PSSP 2.2, Resource Monitors have already been implemented. As mentioned before, there are two types:

- Server
- Client

Both types of monitors are either implemented as daemons (this means separated processes) or incorporated in other software components, for example directly into AIX (look at EM_Resource_Class IBM.PSSP.aixos).

To get an overview of which Resource Monitors are already there, use:

```
# SDRGetObjects EM_Resource_Monitor rmName rmPath rmArguments rmConnect_type
```

Table 1. EM_Resource_Monitor

rmName	rmPath	rmArguments	rmConnect_type
IBM.PSSP.harmld	/usr/lpp/ssp/bin/haemRM/harmld	-f	server
IBM.PSSP.harmpd	/usr/lpp/ssp/bin/haemRM/harmpd		server
IBM.PSSP.hmrmd	/usr/lpp/ssp/bin/haemRM/hmrmd	IBM.PSSP.hmrmd	server
IBM.PSSP.pmanrmd	/usr/lpp/ssp/bin/pmand		client
Membership			internal
Response			internal
aixos			internal

From Table 1 you can see that there are seven Resource Monitors in PSSP 2.2.

2.11.1 External Resource Monitors

Four of the seven Resource Monitors are *external* monitors (see also Figure 36 on page 65). They perform the following tasks:

IBM.PSSP.harmld

This monitor supplies Resource Variables from the CSS, VSD, LoadLeveler, Processor_online information and internal variables of the harmld daemon. The data is of type Counter or Quantity, and is transferred to the Event Management daemon through the SPMI shared memory interface. Therefore, the data is also furnished to the performance monitoring subsystem. This is a server-type daemon.

IBM.PSSP.harmpd

This monitor supplies the Resource Variables that represent the number of processes executing a particular program. These variables are used to determine whether or not a particular system daemon is running. All variables are sent directly to the Event Management daemon and are of Structured Byte String (SBS) format. This is a server-type daemon.

IBM.PSSP.hmrmd

This monitor supplies the Resource Variables that represent the hardware state of the RS/6000 SP. The resource information is obtained from the PSSP hardware monitoring subsystem (hardmon). All the variables are of type State and are transferred to the Event Management daemon directly as messages. This is a server-type daemon.

IBM.PSSP.pmanrmd

This monitor supplies the Resource Variables provided by the Problem Management subsystem. The variables are of type State and are transferred to the Event Management daemon directly as messages. This is a client-type daemon.

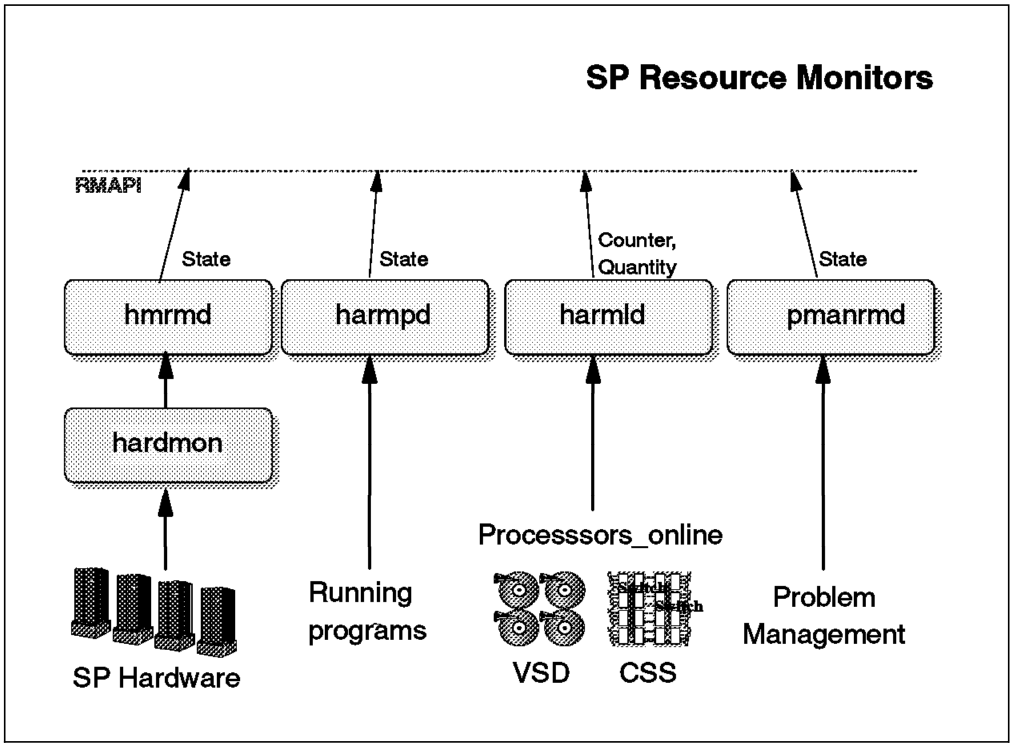


Figure 36. External Resource Monitors

2.11.2 Internal Resource Monitors

The three *internal* Resource Monitors embedded in the Event Management subsystem are shown in Figure 37 on page 66. They perform the following tasks:

Membership

This monitor supplies the Resource Variables that represent host membership and adapter membership states. This information is obtained directly from the Group Services subsystem by subscribing to the system groups hostMembership, enMembership and cssMembership. The variabes are of type State.

The Resource Monitor supplies data to two Resource Variables:

- IBM.PSSP.Membership.Node.state
- IBM.PSSP.Membership.LANAdapter.state

Response

This monitor supplies the Resource Variables that represent the information in the SDR classes host_response and switch_response. These Resource Variables are provided for compatibility with prior releases of PSSP. They are:

- IBM.PSSP.Response.Host.state
- IBM.PSSP.Response.Switch.state

The IBM.PSSP.Response.Host.state Resource Variable represents the *host response* information that is obtained from the Host Response daemon (hrd). For nodes running PSSP 2.2, hrd obtains information about the state of the nodes from the Event Management daemon. The hrd uses the client Event Management API (EMAPI) receiving events from the IBM.PSSP.Membership.LANAdapter.state, with Instance Vector (NodeNum=*, AdapterType=en, AdapterNum=0).

For nodes running at a lower level than PSSP 2.2 (level 2.1 or level 1.2), the hrd obtains information about the state of the nodes from the PSSP Heartbeat daemon (hbd).

The information about the state of the nodes is then stored in the IBM.PSSP.Response.Host.state Resource Variable. At the same time, the hrd updates the SDR host_response class.

The IBM.PSSP.Response.Switch.state is taken directly from the SDR switch_response class.

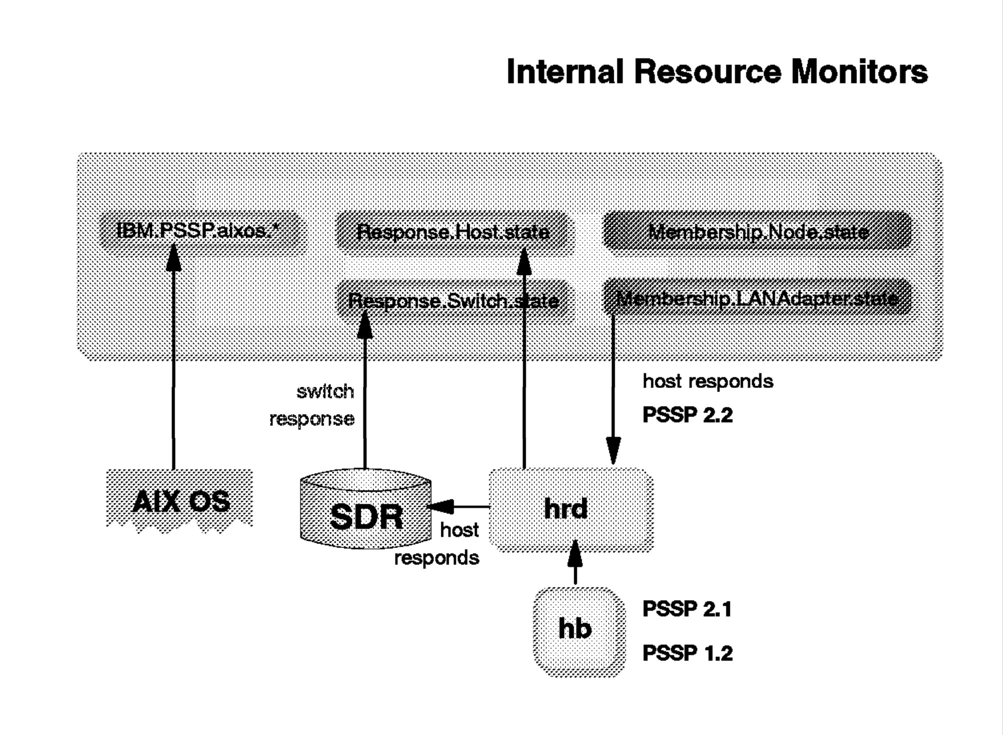


Figure 37. Internal Resource Monitors

aixos

This monitor supplies the Resource Variables that represent AIX Operating System resources like CPU (idle, kern, user, and wait), disks, file systems, LAN, memory, paging space, and processes information.

The aixos Resource Monitor supplies the Resource Variables IBM.PSSP.aixos.*.

2.12 Support for System Partitioning

Whenever a reference is made to *system* in this redbook, it should be interpreted as an SP system partition. If the SP consists of only one partition, then the term *system* refers to the entire SP.

Note: The Control Workstation (CW) is a member of each partition, so that clients on the CW or connecting to the CW can receive events from the partition.

Event Manager daemons only communicate with other Event Manager daemons in the same partition.

Note: On the CW, there is one Event Manager daemon per system partition.

While there may be multiple resource monitors of the same name executing on the Control Workstation (one per Event Manager daemon), such resource monitors may only supply State variables. Only one instance of a resource monitor supplying Counter or Quantity variables may execute on the Control Workstation.

Clients executing on SP nodes may establish connections with the Event Management subsystem in a partition other than their current partition. This is done by connecting to the Event Management daemon on the Control Workstation that is in the other partition.

2.13 Extensibility

The Event Management subsystem is extensible for the following reasons:

- Resource Variable names follow an open naming convention. Names for new Resource Variables can be selected without conflicting with existing variable names.
- Resource Variables and Resource Monitors are defined to the Event Management daemon. This means the knowledge of Resource Variables and Resource Monitors is not hard-coded in Event Management.
- The interfaces necessary to define Resource Variables and Resource Monitors are published in the *Event Management Programming Guide and Reference* documentation.
- The API to send Resource Variables from Resource Monitors to the Event Management daemon is also published in the *Event Management Programming Guide and Reference* documentation.

All this was demonstrated by the sample Resource Monitor generating random numbers.

2.14 Resource Monitor API

Resource Monitors send Resource Variables to the Event Management daemon through the Resource Monitor Application Programming Interface (RMAPI). The RMAPI uses one of two mechanisms to actually transfer the data to the daemon. If the Resource Variable is of type Counter or Quantity, the data is placed in the shared memory. If the data is of type State, then the data is sent as a message using the communication path established between the Event Manager daemon and the Resource Monitor. Familiarity with the RMAPI should enable you to write customized Resource Monitors, or to incorporate the functionality into an application. More information about the RMAPI can be found in the *Event Management Programming Guide and Reference* documentation.

2.15 Summary of Functional Flow

At the most elemental level, Event Management gathers the Resource Variables, applies predicates to those Resource Variables and, if the predicates are true, generates events. These events are then delivered to the clients that registered interest in the events. A client is any program that uses the Event Management Application Programming Interface (EMAPI) to obtain events. Events are generated by the Event Management daemon from Resource Variables supplied by Resource Monitors.

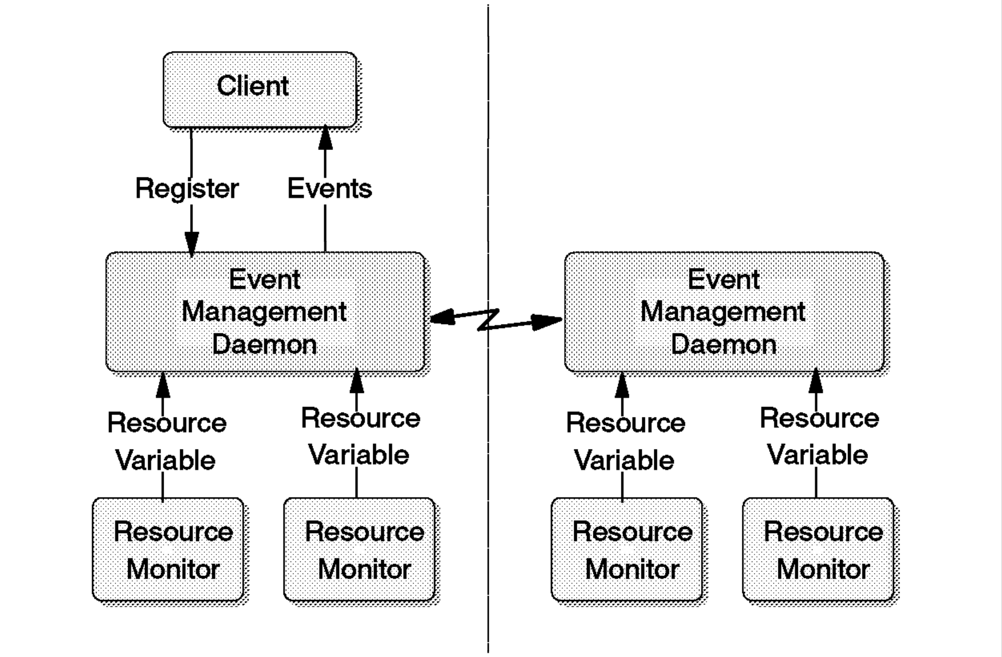


Figure 38. Event Management Information Flow

An Event Management daemon and one or more Resource Monitors are located on any SP node that contains Resource Variables from which events of interest may be generated. An Event Management daemon is also located on any SP node where an Event Management client program is expected to run.

A client registers interest in an event by passing the name and the Instance Vector of the Resource Variable from which the event is generated and by passing a predicate to be applied to the Resource Variable. If no predicate is provided by the client, a default predicate configured with the Resource Variable is used. The client may name more than one Resource Variable and optional associated predicates in a single registration. Multiple Resource Variables can also be specified by wildcarding the Instance Vector. If each Instance Vector is listed, then a different predicate may be specified for each variable. If an Instance Vector is wildcarded, then the associated predicate applies to each variable instance found as result of the wildcard.

If the Resource Variables specified are located on other nodes, then Event Management assumes the role of a client and sends a registration request to Event Management on the appropriate nodes. In this case, Event Management is a proxy on behalf of the client. A registration request from either a client or a proxy client (for example an Event Management daemon), is processed identically by the receiving Event Management daemon.

When the Event Management daemon receives the registration, it ensures that the Resource Monitors that supply the named Resource Variables are running, assuming the Resource Monitors are not implemented as commands. If they are not running, and if they are of a type that can be started by Event Management, then Event Management starts them.

A Resource Monitor *sends* a Resource Variable by one of two methods. If the Resource Variable is of type Counter or Quantity, then the resource variable is passed to the Event Management through shared memory. If the Resource Variable is of type State, then it is passed to Event Management as a message.

Once an event is generated, it is sent to the client that registered for it, including proxy clients. An Event Management daemon acting as a proxy forwards events it receives from other daemons to the appropriate clients.

Chapter 3. Problem Management Subsystem

The Problem Management subsystem (pman) is an application of the Event Management subsystem designed to provide system administrators with configurable access to the Event Management client and the Resource Monitor function, without the necessity of writing C programs that use the Event Management APIs. It also provides some generic recovery scripts for handling events, and a set of “built-in” actions to take on commonly monitored events.

In this chapter, the following topics are described:

Section 3.1, “Introduction to the Problem Management Subsystem” describes what you can do with the Problem Management subsystem.

Section 3.2, “Create Your Own Monitor Using pmand” on page 72, and section 3.3, “Create Your Own Monitor Using pmand and pmanrmd” on page 77, describe how to create your own monitors, step by step, along with three scenarios.

Section 3.4, “Problem Determination” on page 83 provides you with some hints and tips on how to debug your monitor.

Section 3.5, “Hints and Tips for Problem Management Subsystem Commands” on page 90 provides you with some hints and tips on how to use Problem Management subsystem commands.

Section 3.6, “Short Examples” on page 93 provides you with several short examples.

Appendix D, “Essence of the Event Management and Problem Management Subsystems” on page 235 provides you with a condensed summary of the Event Management and Problem Management subsystems. It contains enough information to create your own monitor.

3.1 Introduction to the Problem Management Subsystem

There are three ways to monitor your system if you use POWERparallel System Support Programs Version 2.2 High Availability Infrastructure:

1. Commands
2. A GUI
3. Programs

If you are a command-oriented administrator, this chapter is for you. If you are a GUI-oriented administrator, use Perspectives in Chapter 5, “SP Perspectives GUI” on page 137. If you are a programmer, use the APIs in Chapter 4, “Application Program Interfaces (APIs)” on page 97.

3.1.1 What Can You Monitor?

To monitor your system means that you monitor Resource Variables. PSSP 2.2 provides you with many Resource Variables. These can be categorized into two groups:

1. All the Resource Variables that the Event Management subsystem provides. (There are more than 300 of these available.)
2. Up to 16 user-configurable Resource Variables. (These are provided by the Problem Management subsystem.)

3.1.2 What Can You Do When System Resources Are Changed?

The Problem Management subsystem provides you with three possible actions to perform when your system resources are changed:

1. Run a command.
2. Issue an SNMP trap.
3. Write to the AIX Error Log or BSD syslog facilities.

3.1.3 What Kind of Monitoring Tools Can You Use?

The Problem Management subsystem provides you with two monitoring daemons:

1. The Problem Management daemon (pmand)
2. The Problem Management Resource Monitor daemon (pmanrmd)

Use their commands to create your own monitor.

If you monitor Resource Variables provided by the Event Management subsystem, you need to have only the Problem Management daemon running. If you monitor Resource Variables provided by the Problem Management subsystem, consisting of up to 16 user-configurable Resource Variables, you need to have the Problem Management Resource Monitor daemon running also.

3.2 Create Your Own Monitor Using pmand

This section explains how to create your own monitor using the Problem Management daemon.

The Event Management subsystem provides you with more than 300 sophisticated and useful Resource Variables. The Problem Management daemon monitors these Resource Variables and performs an action according to their changes.

In the first part of this section, we provide you with a sample scenario to monitor your system, and then help you create your own monitor, step by step, using this scenario.

Once you become familiar with this example, it will be easy to create other monitors.

3.2.1 Scenario 1

Following is the sample scenario used in this section:

Scenario 1

You want to monitor CPU usage. When the CPU idle time on Node 5 becomes less than 10%, you want to send an alert message to all users on Node 5. In this case, you use the `wall` command with the message `Operator: Save CPU time`.

3.2.2 Get a Rough Idea

Before you start with the Problem Management subsystem, you have to know what you want to monitor. Because more than 360 variables are provided by the Event Management subsystem, you have to choose the one(s) that best fit your monitoring requirements.

To help with this task, take a look at the Event Management Resource Class first. This class gives you a rough idea as to which Resource Class the Resource Variable you are interested in belongs.

Use the following command to get Event Management Resource Class data:

```
# SDRGetObjects EM_Resource_Class
```

Refer to Table 8 on page 237. There are 17 Resource Class classes available.

All the Resource Class classes provided by the Event Management subsystem are listed in Table 8 on page 237. Resource Class IBM.PSSP.aixos.CPU is in this table. It includes CPU-related Resource Variables, and these are the ones you want. You will also see that Resource Class IBM.PSSP.aixos.CPU is monitored by Resource Monitor aixos, is observed every 15 seconds, and is reported at intervals fixed by the design of aixos.

3.2.3 Find Resource Variables You Might Want

If you are interested in *all* Resource Variables that are provided by the Event Management subsystem, you may use the following command (remember, there are more than 300 Resource Variables):

```
# SDRGetObjects EM_Resource_Variable | more
```

To get Resource Variables that belong to Resource Class IBM.PSSP.aixos.CPU, use the following command:

```
# SDRGetObjects EM_Resource_Variable rvClass==IBM.PSSP.aixos.CPU
```

You will see the following information:

```
# SDRGetObjects EM_Resource_Variable rvClass==IBM.PSSP.aixos.CPU
rvName rvDescription rvValue_type rvData_type rvInitial_value
rvClass rvPTX_name rvPTX_description rvPTX_min rvPTX_max rvPredicate
rvEvent_description rvLocator rvDynamic_instance rvIndex_vector
IBM.PSSP.aixos.CPU.glidle           1 Quantity     float        0
IBM.PSSP.aixos.CPU CPU/glidle    ″″           ″″           ″″           ″″
 ″″           NodeNum                 0 ″″
IBM.PSSP.aixos.CPU.glkern           2 Quantity     float        0
IBM.PSSP.aixos.CPU CPU/glkern    ″″           ″″           ″″           ″″
 ″″           NodeNum                 0 ″″
IBM.PSSP.aixos.CPU.gluser           3 Quantity     float        0
IBM.PSSP.aixos.CPU CPU/gluser    ″″           ″″           ″″           ″″
 ″″           NodeNum                 0 ″″
IBM.PSSP.aixos.CPU.glwait           4 Quantity     float        0
IBM.PSSP.aixos.CPU CPU/glwait    ″″           ″″           ″″           ″″
 ″″           NodeNum                 0 ″″
IBM.PSSP.aixos.cpu.idle             5 Quantity     float        0
IBM.PSSP.aixos.CPU CPU/$CPU/idle ″″           ″″           ″″           ″″
 ″″           NodeNum                 0 ″″
IBM.PSSP.aixos.cpu.kern             6 Quantity     float        0
IBM.PSSP.aixos.CPU CPU/$CPU/kern ″″           ″″           ″″           ″″
 ″″           NodeNum                 0 ″″
IBM.PSSP.aixos.cpu.user             7 Quantity     float        0
IBM.PSSP.aixos.CPU CPU/$CPU/user ″″           ″″           ″″           ″″
 ″″           NodeNum                 0 ″″
IBM.PSSP.aixos.cpu.wait             8 Quantity     float        0
IBM.PSSP.aixos.CPU CPU/$CPU/wait ″″           ″″           ″″           ″″
 ″″           NodeNum                 0 ″″
#
```

From this we see that the following Resource Variables belong to Resource Class IBM.PSSP.aixos.CPU:

- IBM.PSSP.aixos.CPU.glidle
- IBM.PSSP.aixos.CPU.glkern
- IBM.PSSP.aixos.CPU.gluser
- IBM.PSSP.aixos.CPU.glwait
- IBM.PSSP.aixos.cpu.idle
- IBM.PSSP.aixos.cpu.kern
- IBM.PSSP.aixos.cpu.user
- IBM.PSSP.aixos.cpu.wait

We also know that their value type is Quantity, their data type is float, and their initial value is 0.

Descriptions of these Resource Variables are available in a message catalog. Note that `rvDescription` (in this case, 1 through 8) is a message number in this catalog.

3.2.4 Find the Resource Variable You Want

To find out which message catalog your Resource Monitor uses, you have to look up the Event Management Resource Monitor class. This class provides you with the name of a message catalog and the *set number* in the message catalog that your Resource Monitor uses.

Use the following command to get you this information:

```
# SDRGetObjects EM_Resource_Monitor
```

Refer to Table 5 on page 236 and Table 6 on page 236. Two message catalogs are available for seven Resource Monitor classes.

The Resource Monitor aixos uses message catalog harm_des.cat and set number 6 in this catalog, whose full path name is /usr/lib/nls/msg/$LANG/harm_des.cat. You already know the message numbers from the previous section; they are 1 through 8.

To display the message catalog, use the following command:

```
# dspcat harm_des.cat 6 | head -8
```

You will see the following information:

```
# dspcat harm_des.cat 6 | head -8
System-wide time CPU is idle (percent)
System-wide time executing in kernel mode (percent)
System-wide time executing in user mode (percent)
System-wide time waiting for IO (percent)
Time CPU is idle (percent)
Time CPU executing in kernel mode (percent)
Time CPU executing in user mode (percent)
Time CPU waiting for IO (percent)
#
```

There are two kinds of idle Resource Variables: one is for system-wide use, and the other is for the use of a specific CPU. Pick up the idle Resource Variable for system-wide use, IBM.PSSP.aixos.CPU.glidle.

3.2.5 Find the Instance Vector You Want

The previous section mentioned that monitoring your system is equivalent to monitoring a Resource Variable. It also means using the pmandef command. When you use pmandef, you have to specify the -e option. In the -e option, you need a Resource Variable, an Instance Vector, and a Predicate. The Instance Vector you have to specify is dependent on the Resource Variable. What kind of Instance Vector can you use for Resource Class IBM.PSSP.aixos.CPU?

To answer this question, look up the Event Management Instance Vector class, using the following command:

```
# SDRGetObjects EM_Instance_Vector
```

Refer to Table 9 on page 239. Only one Instance Vector, NodeNum, is provided for Resource Class IBM.PSSP.aixos.CPU. You can see that it uses message number 42. If you do not know what Instance Vector NodeNum is, use the following command:

```
# dspcat harm_des.cat 6 42
```

You will see the following information:

```
# dspcat harm_des.cat 6 42
The number of the node for which the information applies.
#
```

If your Resource Variable uses SBS type data, you have to get familiar with the Event Management Structured Byte String class. Refer to section 3.3, “Create Your Own Monitor Using pmand and pmanrmd” on page 77 for information on this topic.

3.2.6 Use the pmandef Command

You now have all the information needed to create your own monitor. Try to use the pmandef command.

Following is the shell script file scenario_1.sub used to subscribe this monitor:

```
pmandef -s CPU_Idle_Monitor \
        -e ′IBM.PSSP.aixos.CPU.glidle:NodeNum=5:X<10′ \
        -c ″wall Operator: Save CPU time″ \
        -n 5
```

Following is the shell script file scenario_1.unsub used to unsubscribe this monitor:

```
pmandef -u CPU_Idle_Monitor
```

Figure 39 shows a flow of how to build the pmandef command to create your own monitor.

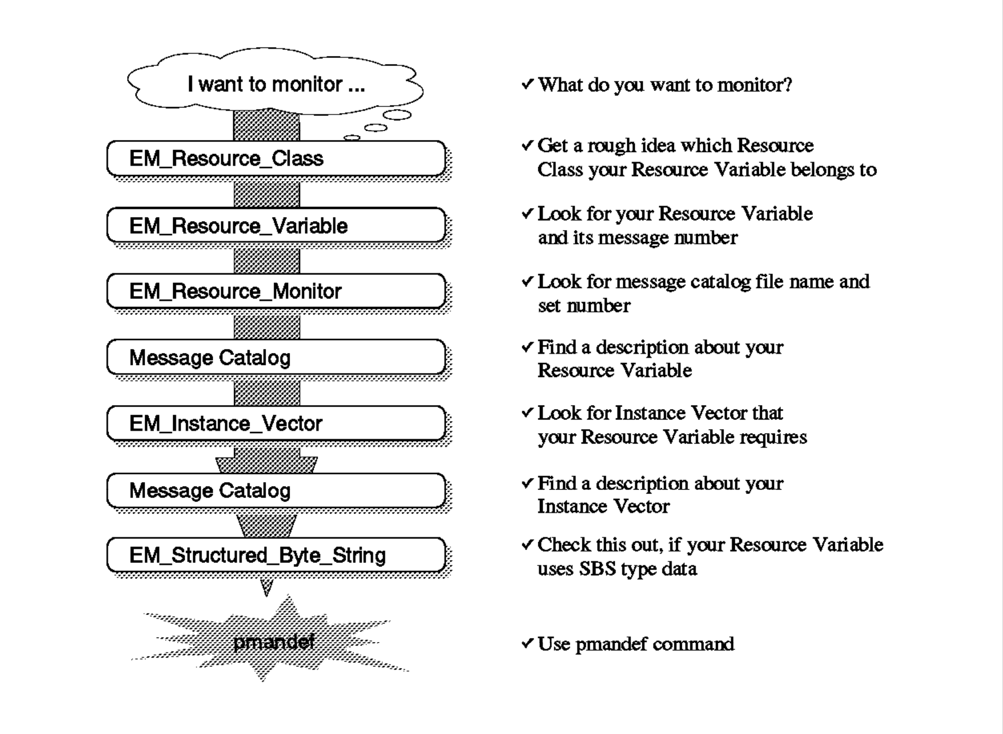

I want to monitor ...
✓ What do you want to monitor?
EM_Resource_Class
✓ Get a rough idea which Resource Class your Resource Variable belongs to
EM_Resource_Variable
✓ Look for your Resource Variable and its message number
EM_Resource_Monitor
✓ Look for message catalog file name and set number
Message Catalog
✓ Find a description about your Resource Variable
EM_Instance_Vector
✓ Look for Instance Vector that your Resource Variable requires
Message Catalog
✓ Find a description about your Instance Vector
EM_Structured_Byte_String
✓ Check this out, if your Resource Variable uses SBS type data
pmandef
✓ Use pmandef command

Figure 39. Flow of Using the pmand Command

3.2.7 Try Your Own Monitor

To check your monitor, issue the pmandef command, then execute a CPU-eater process like the following program on Node 5:

```
main()
{
   for(;;);
}
```

You will get the message Operator: Save CPU time every 15 seconds on Node 5. After receiving the message, kill this process.

```
#

Broadcast message from root@sp21n05 (tty) at 10:42:36 ...

Operator: Save CPU time


#
```

If you want to create your own monitors with your own Resource Variables, the next section will be helpful to you.

3.3 Create Your Own Monitor Using pmand and pmanrmd

This section explains how to create your own monitor using the Problem Management daemon and the Problem Management Resource Monitor daemon.

The Event Management subsystem provides you with more than 300 sophisticated and useful Resource Variables, but at times you may want to use a unique Resource Variable that you defined. For this purpose, the Problem Management Resource Monitor daemon provides you with up to 16 user-configurable Resource Variables, IBM.PSSP.pm.User_state1 through IBM.PSSP.pm.User_state16. The Problem Management daemon monitors these variables and takes action according to their changes.

There are two ways to use user-configurable Resource Variables:

1. Use the Problem Management Resource Monitor daemon configuration file.

   In this case, the Problem Management Resource Monitor daemon automatically changes these Resource Variables at certain intervals. You can define these intervals in the Problem Management Resource Monitor daemon configuration file.

2. Do not use the Problem Management Resource Monitor daemon configuration file.

   In this case, you change these Resource Variables with the pmanrminput command.

This section provides sample scenarios 2 and 3 to monitor your system, and then helps you create your own monitors, step by step, using these scenarios.

3.3.1 Scenario 2

Scenario 2 uses the Problem Management Resource Monitor daemon configuration file.

Scenario 2

In this scenario, the su command is not allowed to be used to become a root user account on Node 5. You want to check the file /var/adm/sulog every minute for this purpose. If someone uses the su command to become a root user account, you want to send SNMP trap 10001 locally.

3.3.2 Create a pmanrmd Configuration File

First you have to create a Problem Management Resource Monitor daemon configuration file. In this file, you have to define one of the user-configurable Resource Variables. This scenario uses IBM.PSSP.pm.User_state9.

There is a sample Problem Management Resource Monitor daemon configuration file named pmanrmd.conf in directory /spdata/sys1/pman. Figure 40 on page 79 shows the content of this file.

```
*************************************************************************
*                                                                       *
*    Licensed Materials - Property of IBM                               *
*                                                                       *
*    5765-529 PSSP                                                      *
*                                                                       *
*    (C) Copyright IBM Corporation 1996  All Rights Reserved.           *
*                                                                       *
*    US Government Users Restricted Rights - Use, duplication or        *
*    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.  *
*                                                                       *
*************************************************************************

* "@(#)70   1.7   src/ssp/pman/pmanrmd.conf, probmgmt, ssp_rloc, rloct7d4
7/21/96 10:45:57"

*  Problem Management resource monitor configuration file.

*
* This will run a user script resource monitor every 1 minutes and
* place the stdout of the monitor into event management as a resource
* variable when it runs.  It will run on the CWS only.
*
*TargetType=NODE_LIST
*Target=CWS
*Rvar=IBM.PSSP.pm.User_state1
*SampInt=60
*Command=/u/joe/checker

*
* This will provide a resource monitor that will run every 10 minutes
* and will provide the most recently changed file in the
* etc file system as a string state variable named
* IBM.PSSP.pm.User_state2.
* It runs on all the nodes in the partition.
*
*TargetType=NODE_GROUP
*Target=ALLPMAN
*Rvar=IBM.PSSP.pm.User_state2
*SampInt=600
*Command="/bin/ls -tl /etc | /bin/head -2 | /bin/grep -v total"
```

Figure 40. File pmanrmd.conf

You need to specify the following items in the Problem Management Resource Monitor daemon configuration file:

- TargetType
- Target
- Rvar
- Command
- SampInt

The Problem Management Resource Monitor daemon configuration file for this monitor, scenario_2.conf, is as follows:

```
TargetType=NODE_RANGE
Target=5
Rvar=IBM.PSSP.pm.User_state9
Command=″grep -e -root /var/adm/sulog | tail -1″
SampInt=60
```

Before you use Resource Variable IBM.PSSP.pm.User_state9, you have to load this configuration to the pmanrmdConfig SDR. To do this, use the following command on the Control Workstation:

```
# pmanrmdloadSDR scenario_2.conf
```

Even though your Resource Variable is in the pmanrmdConfig SDR, the Problem Management Resource Monitor daemon does not know this Resource Variable dynamically. To let it recognize the variable, refresh the Problem Management subsystem with the following command on the Control Workstation:

```
# dsh -a stopsrc -s pmanrm; stopsrc -s pmanrm.partition_name
```

Then issue this command:

```
# dsh -a startsrc -s pmanrm; startsrc -s pmanrm.partition_name
```

3.3.3 Find the Instance Vector You Want

Again, you have to use the pmandef command to monitor your system. This command requires the -e option. In the -e option, you have to specify an Instance Vector.

To find the Instance Vector required by IBM.PSSP.pm.User_state9, use the following command:

```
# SDRGetObjects  EM_Instance_Vector | grep IBM.PSSP.pm
IBM.PSSP.pm  NodeNum                3
#
```

There is one Instance Vector, NodeNum, required for IBM.PSSP.pm.User_state9. To see its description, use the following command:

```
# dspcat harm_des.cat 3 3
The number of the node on which the resource resides.
#
```

Using Table 8 on page 237, you can find that Resource Class IBM.PSSP.pm is monitored by Resource Monitor IBM.PSSP.pmanrmd.

Then, using Table 6 on page 236, you can see that Resource Monitor IBM.PSSP.pmanrmd uses message catalog harm_des.cat and set number 3.

3.3.4 Find the Type of Your Resource Variable

Next, you must specify the predicate. To do this, you have to know the type of the Resource Variable, which you can find with the following command::

```
# SDRGetObjects EM_Resource_Variable | grep IBM.PSSP.pm.User_state9
IBM.PSSP.pm.User_state9            2 State        SBS          ""
IBM.PSSP.pm  ""           ""           ""           ""           ""
""           NodeNum                 0 ""
#
```

The fourth member, rvData_type, is SBS. This means that Resource Variable IBM.PSSP.pm.User_state9 is Structured Byte String-type data.

3.3.5 Find the Structured Byte String You Want

When you use a Resource Variable and its data type is SBS, you have to refer to the Event Management Structured Byte String class. To find the SBS of Resource Variable IBM.PSSP.pm.User_state9, use the following command:

```
# SDRGetObjects EM_Structured_Byte_String | grep IBM.PSSP.pm.User_state9
IBM.PSSP.pm.User_state9 STRING       cstring                0 ""
#
```

Resource Variable IBM.PSSP.pm.User_state9 uses only one cstring type. When you specify a predicate, you have to use X@0 instead of X, and you have to compare X@0 with the character string.

3.3.6 Use the pmandef Command

To monitor Resource Variable IBM.PSSP.pm.User_state9, you have to use the pmandef command. Following is the shell script file scenario_2.sub used to subscribe this monitor:

```
pmandef -s sulog_changed \
        -e 'IBM.PSSP.pm.User_state9:NodeNum=5:X@0!=X@P0' \
        -t 10001 \
        -h local
```

Following is the shell script file scenario_2.unsub used to unsubscribe this monitor:

```
pmandef -u sulog_changed
```

3.3.7 Try Your Own Monitor

Execute the pmandef command. Use the su command to become a root user account. You will have SNMP trap 10001 within a minute.

3.3.8 Scenario 3

Scenario 3 does not use the Problem Management Resource Monitor daemon configuration file; instead, it uses the pmanrminput command.

Scenario 3

A program uses a lot of resources. When this program is executed, it has to notify the Problem Management subsystem when it is started and ended.

The administrator prepares a reaction to this notification, as follows:

- When the program is started, the Problem Management subsystem sends the message This node is going to be busy.
- When the program is ended, the Problem Management subsystem sends the message This node is not busy now via the wall command.

This program runs on Node 5.

3.3.9 Use the pmandef Command

This scenario uses Resource Variable IBM.PSSP.pm.User_state8. To monitor it, you must create a protocol. This scenario uses the following protocol:

- When the program is started, it changes Resource Variable IBM.PSSP.pm.User_state8 to START.
- When the program is ended, it changes Resource Variable IBM.PSSP.pm.User_state8 to END.

Following is the shell script file scenario_3.sub used to subscribe this monitor:

```
pmandef -s simple_protocol \
        -e ′IBM.PSSP.pm.User_state8:NodeNum=5:X@0==″START″′ \
        -r ′X@0==″END″′ \
        -c ″wall This node is going to be busy″ \
        -C ″wall This node is not busy now″ \
        -h local
```

Following is the shell script file scenario_3.unsub used to unsubscribe this monitor:

```
pmandef -u simple_protocol
```

3.3.10 Try Your Own Monitor

Execute the pmandef command, then execute the shell script file scenario_3.appl:

```
pmanrminput -s pman -a″IBM.PSSP.pm.User_state8+START+″
#
# USE A LOT OF RESOURCES HERE!
sleep 10
#
pmanrminput -s pman -a″IBM.PSSP.pm.User_state8+END+″
```

You will receive the message This node is going to be busy when the program is started, and This node is not busy now when the program is ended.

3.4 Problem Determination

You may encounter some difficulties while creating your own monitor. This section provides hints and tips to help you solve some problems.

3.4.1 Are You Authorized?

Following is a list of Problem Management subsystem commands:

```
-r-xr-x---   1 bin      bin        7760 Sep 28 00:18 pmanctrl
-r-xr-xr-x   1 bin      bin       41099 Sep 28 00:18 pmandef
-r-xr-xr-x   1 bin      bin        8399 Sep 28 00:17 pmanquery
-r-xr-x---   1 bin      bin       15545 Sep 28 00:17 pmanrmdloadSDR
-r-x------   1 root     system     4528 Sep 28 00:25 pmanrminput
```

This shows that even though you are a general user belonging to the staff group, you can use the pmandef and pmanquery commands. But there is another hurdle to overcome, and that is Kerberos.

The pmandef command is based on Sysctl, which uses Kerberos for user authentication. All users of pmandef must have valid Kerberos credentials.

You may see the following message when you use pmandef:

```
sysctl: 2501-122 svcconnect: Insufficient Authorization.
pmandef: You are not authorized to use Problem Management.
```

To solve this problem, check the Access Control List (ACL) file for the Problem Management subsystem, where the file sysctl.pman.acl in directory /etc exists for this purpose. If there is no user ID that will use pmandef, you must add one. Figure 41 on page 84 shows the default content of this file:

```
#acl#

# These are the kerberos principals for the users that can configure
# Problem Management on this node.  They must be of the form as indicated
# in the commented out records below.  The pound sign (#) is the comment
# character, and the underscore (_) is part of the ″_PRINCIPAL″ keyword,
# so do not delete the underscore.

#_PRINCIPAL root.admin@PPD.POK.IBM.COM
#_PRINCIPAL joeuser@PPD.POK.IBM.COM
```

Figure 41. File sysctl.pman.acl

As the comment in the default content indicates, this file is effective only on this node.

The user′s Kerberos principal must be listed *both* in this file on the local node (in order to store the subscription in the SDR), and on the nodes that are affected by the new subscription. In this way, the affected Problem Management subsystem daemons will be notified of the new subscription.

If the user′s Kerberos principal is listed only in this file on the local node, the subscription will be stored in the SDR, but the Problem Management subsystem daemons will not act on the new subscription until the next time they are restarted.

If the requested action is to write an entry in the AIX error log and BSD syslog or to generate an SNMP trap, then the Kerberos principal that owns the subscription must have been listed in the root user′s $HOME/.klogin file.

If the requested action is to execute a command, then the Kerberos principal must be listed in the $HOME/.klogin file of the user, which will be used to run the command.

3.4.2 Is the Problem Management Subsystem Active?

There are three states for Problem Management subsystem daemons:

1. The active state
2. The inoperative state
3. `The pman Group is not on file` state

To check the current status of the Control Workstation, use the following command on the Control Workstation:

```
# lssrc -g pman
```

To check the current status of nodes, use the following command on the Control Workstation:

```
# dsh -a lssrc -g pman
```

The lssrc command reports only Problem Management subsystem daemons on the local node. Therefore, you have to use this command with the dsh command, which gives you a short list showing Problem Management daemon status and Problem Management Resource Monitor daemon status. If some Problem Management subsystem daemons are inactive, you have to activate them.

Following is an example showing two active nodes, sp21n01 and sp21n05:

```
# lssrc -g pman
Subsystem         Group            PID     Status
 pman.sp21en0     pman             64610   active
 pmanrm.sp21en0   pman             45156   active
# dsh -a lssrc -g pman
sp21n01: Subsystem         Group            PID     Status
sp21n01:  pman             pman             11300   active
sp21n01:  pmanrm           pman             15520   active
sp21n05: Subsystem         Group            PID     Status
sp21n05:  pman             pman             17620   active
sp21n05:  pmanrm           pman             18656   active
```

If the Problem Management subsystem daemons on node sp21n01 are in the inoperative state, you will see the following message:

```
# lssrc -g pman
Subsystem         Group            PID     Status
 pman.sp21en0     pman             64610   active
 pmanrm.sp21en0   pman             45156   active
# dsh -a lssrc -g pman
sp21n01: Subsystem         Group            PID     Status
sp21n01:  pman             pman                     inoperative
sp21n01:  pmanrm           pman                     inoperative
sp21n05: Subsystem         Group            PID     Status
sp21n05:  pman             pman             17620   active
sp21n05:  pmanrm           pman             18656   active
#
```

To solve this problem, you have to start the Problem Management subsystem daemons on node sp21n01 by using one of the following commands on the Control Workstation:

```
# dsh -w sp21n01 startsrc -g pman
sp21n01: 0513-059 The pman Subsystem has been started.  Subsystem PID is 17074.
sp21n01: 0513-059 The pmanrm Subsystem has been started.  Subsystem PID is 17588.
#
```

or

```
# dsh -w sp21n01 pmanctrl -s
sp21n01: 0513-059 The pman Subsystem has been started.  Subsystem PID is 13492.
sp21n01: 0513-059 The pmanrm Subsystem has been started.  Subsystem PID is 19144.
#
```

Of course, you can execute these commands on Node sp21n01 without the dsh command.

Figure 42 on page 86 shows the relationship between the three states for Problem Management subsystem daemons, and options for the pmanctrl command:

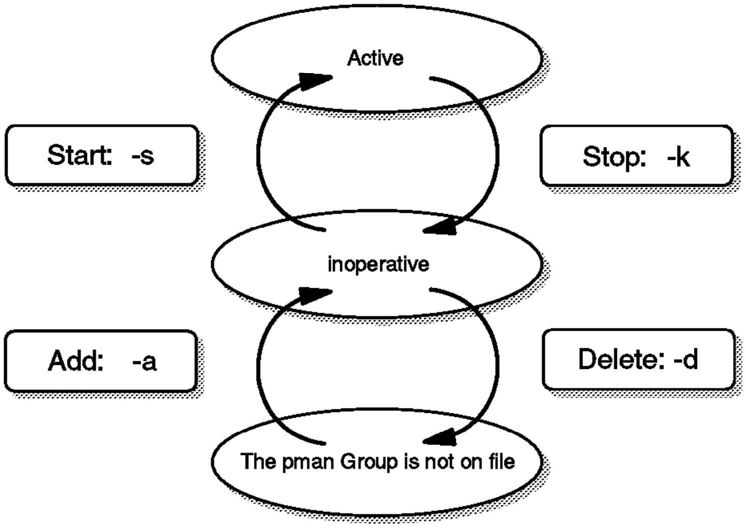


Figure 42. Status of Problem Management Subsystem Daemons and pmanctrl Command Options

3.4.3 Is Your Event Subscribed?

You have to subscribe your Problem Management subscription to the pmandConfig SDR.

To check the content of the pmandConfig SDR, use the following command:

```
# pmanquery -n all -k all
```

This command provides all handles and all the Kerberos principal′s subscriptions. If you are interested in your particular handle, CPU_Idle_Monitor (used in Scenario 1), use the following command:

```
# pmanquery -n CPU_Idle_Monitor
pmActivated:pmHandle:pmRvar:pmIvec:pmPred:pmCommand:pmCommandTimeout:pmTrapid:pm
PPSlog:pmText:pmRearmPred:pmRearmCommand:pmRearmCommandTimeout:pmRearmTrapid:pmR
earmPPSlog:pmRearmText:pmUsername:pmPrincipal:pmHost:pmTargetType:pmTarget:pmUse
rLabel
1:CPU_Idle_Monitor:IBM.PSSP.aixos.CPU.glidle:NodeNum=5:X<10:wall Operator: Save
CPU time:0:-1:0: : : :0:-1:0: :root:
root.admin@MSC.ITSO.IBM.COM:sp21en0.msc.itso.ibm.com:NODE_RANGE:5:

#
```

If you cannot find your Problem Management subscription, you have to subscribe your Problem Management subscription to the pmandConfig SDR.

To solve this problem, use pmandef with the -s option.

3.4.4 Is Your Event Ready to Be Acted On?

Unfortunately, pmandef allows you to subscribe meaningless data into the pmandConfig SDR. So, even though you find your Problem Management subscription in the pmandConfig SDR, if there is something wrong (a syntax error, for example), the Problem Management subsystem will not accept your Problem Management subscription.

When you use pmandef, you might see the following error message:

```
pmansubscr: subscribe for my_subscription failed.
       Reason: HA_EM_RSP_EVECSYNTAX (syntax error in vector)
       Please execute ″pmandef -u my_subscription″
```

Note the third line, Please execute ″pmandef -u my_subscription″. This message is received because, even though your subscription has a syntax error, pmandef allows you to subscribe it into the pmandConfig SDR. You have to unsubscribe it as soon as possible.

To check whether the Problem Management subsystem accepts your Problem Management subscription, use the following command:

```
# pmandef -q all
```

This command gives you all events that are currently subscribed and ready to be acted on. If you are interested in your particular handle, CPU_Idle_Monitor used in Scenario 1, use the following command:

```
# pmandef -q CPU_Idle_Monitor
sp21n05: event CPU_Idle_Monitor status:
sp21n05: event subscribed and ready to be acted on
#
```

If your Problem Management subscription is not subscribed and ready to be acted on, you have to unsubscribe it first. Then subscribe your corrected Problem Management subscription.

To do this, use pmandef with the -u option to unsubscribe, and then use it with the -s option to subscribe.

Following is an example in which your Problem Management subscription′s handle is CPU_Idle_Monitor:

```
# pmandef -u CPU_Idle_Monitor
sp21n05: event CPU_Idle_Monitor unsubscribed
# pmandef -s CPU_Idle_Monitor \
         -e ’IBM.PSSP.aixos.CPU.glidle:NodeNum=5:X<10’ \
         -c ″wall Operator: Save CPU time″ \
         -n 5
#
```

3.4.5 Is Your Event Active and Correct?

When you subscribe your Problem Management subscription, it will be activated normally.

To check your Problem Management subscription, use the following command:

```
# lssrc -ls pman
```

This command provides you with a current status (active or inactive), of Problem Management subscriptions on the local node. If you are interested in the status of Node sp21n05, use the following command on the Control Workstation:

```
# dsh -w sp21n05 lssrc -ls pman
sp21n05: Subsystem         Group            PID     Status
sp21n05:  pman             pman             4502    active
sp21n05:
sp21n05:  pmand started at: Wed Sep 25 09:51:31 1996
sp21n05:  pmand last refreshed at:
sp21n05:  Tracing is off
sp21n05:  =========================================================
sp21n05:  Events for which registrations are as yet unacknowledged:
sp21n05:  =========================================================
sp21n05:  =========================================================
sp21n05:  Events for which actions are currently being taken:
sp21n05:  =========================================================
sp21n05:  =========================================================
sp21n05:  Events currently ready to be acted on by this daemon:
sp21n05:  =========================================================
sp21n05:  ------------------ CPU_Idle_Monitor ----------------
sp21n05:  Currently INACTIVE
sp21n05:  Client root root.admin@MSC.ITSO.IBM.COM at sp21en0.msc.itso.ibm.
sp21n05:  com
sp21n05:  Resource Variable: IBM.PSSP.aixos.CPU.glidle
sp21n05:  Instance: NodeNum=5
sp21n05:  Predicate: X<10
sp21n05:  Command to run: wall Operator: Save CPU time
sp21n05:  Has run 0 times
#
```

In this example, the message indicates that CPU_Idle_Monitor is `Currently INACTIVE`.

To activate your Problem Management subscription, CPU_Idle_Monitor, use the following command:

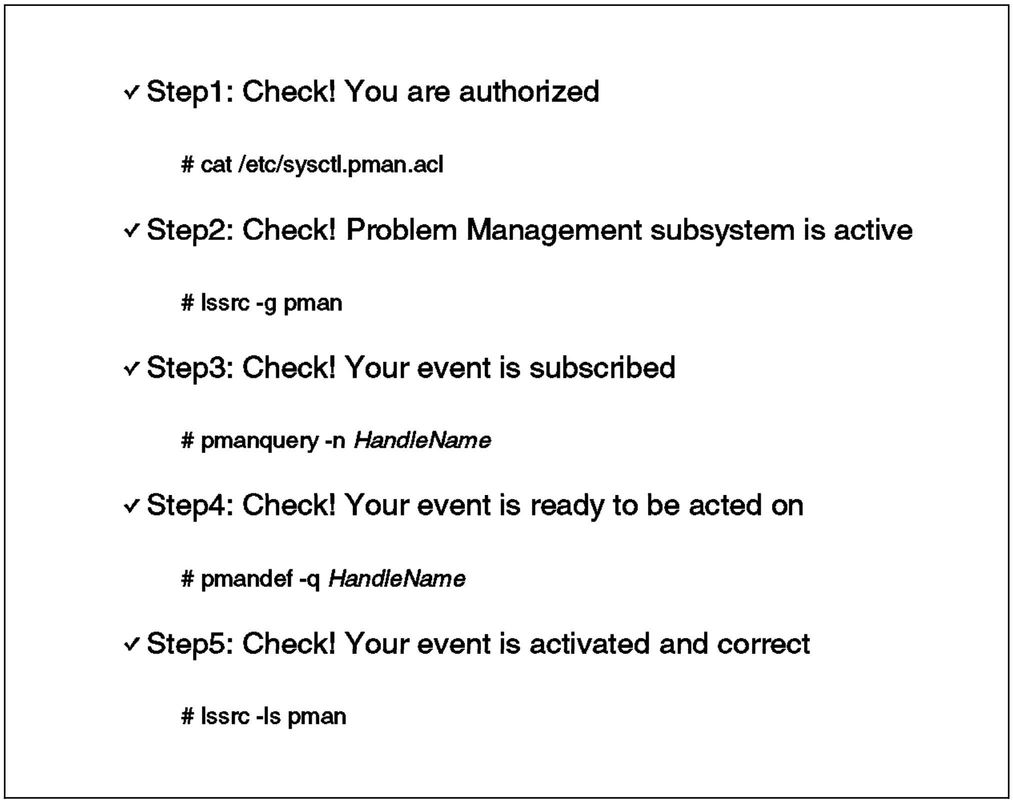
```
# dsh -w sp21n05 pmandef -a CPU_Idle_Monitor
sp21n05: sp21n05: event CPU_Idle_Monitor activated
#
```

You will see CPU_Idle_Monitor become Currently ACTIVE, as follows:

```
# dsh -w sp21n05 lssrc -ls pman
sp21n05: Subsystem         Group            PID     Status
sp21n05:  pman             pman             4502    active
sp21n05:
sp21n05:  pmand started at: Wed Sep 25 09:51:31 1996
sp21n05:  pmand last refreshed at:
sp21n05:  Tracing is off
sp21n05:  =========================================================
sp21n05:  Events for which registrations are as yet unacknowledged:
sp21n05:  =========================================================
sp21n05:  =========================================================
sp21n05:  Events for which actions are currently being taken:
sp21n05:  =========================================================
sp21n05:  =========================================================
sp21n05:  Events currently ready to be acted on by this daemon:
sp21n05:  =========================================================
sp21n05:  ------------------ CPU_Idle_Monitor ----------------
sp21n05:  Currently ACTIVE
sp21n05:  Client root root.admin@MSC.ITSO.IBM.COM at sp21en0.msc.itso.ibm.
sp21n05:  com
sp21n05:  Resource Variable: IBM.PSSP.aixos.CPU.glidle
sp21n05:  Instance: NodeNum=5
sp21n05:  Predicate: X<10
sp21n05:  Command to run: wall Operator: Save CPU time
sp21n05:  Has run 0 times
#
```

This message also shows you how the Problem Management subsystem treats Problem Management subscription, and its statistical data.

Figure 43 on page 90 shows a flow of problem determination. If something is wrong with your monitor, check the following items, step by step, to find the problem:



✓ **Step1: Check! You are authorized**

```
# cat /etc/sysctl.pman.acl
```

✓ **Step2: Check! Problem Management subsystem is active**

```
# lssrc -g pman
```

✓ **Step3: Check! Your event is subscribed**

```
# pmanquery -n HandleName
```

✓ **Step4: Check! Your event is ready to be acted on**

```
# pmandef -q HandleName
```

✓ **Step5: Check! Your event is activated and correct**

```
# lssrc -ls pman
```

Figure 43. Flow of Problem Determination

3.5 Hints and Tips for Problem Management Subsystem Commands

When you operate Problem Management subsystem daemons, you have to know whether you are operating a daemon, SDR, or both. Operating a daemon is effective for the current setting, but once the daemon dies, the current setting will be gone.

Operating SDR is effective the next time the daemon is started or refreshed. Operating SDR does not change the current setting.

You should be aware, also, that commands for Problem Management subsystem can be separated into three groups:

1. Node-wide commands
2. Partition-wide commands
3. Partition-wide commands executable only on the Control Workstation

The `pmanctrl` and `pmanrminput` commands are node-wide commands. If you want to execute on another node, you must use the `dsh` command with these commands.

On the other hand, `pmandef` and `pmanquery` are partition-wide commands. You can use these commands on any node as long as it is in the same partition.

The `pmanrmdloadSDR` command is also a partition-wide command, but it is executable only on the Control Workstation.

The following sections offer information about the Problem Management daemon and the Problem Management Resource Monitor daemon.

3.5.1 Commands for The Problem Management Daemon

The Problem Management daemon refers to the pmandConfig SDR when it is started or refreshed. The Problem Management subsystem provides you with the pmanctrl command for this purpose. But this command works for both the Problem Management daemon and the Problem Management Resource Monitor daemon at the same time. If you want to operate only the Problem Management daemon, you have to use the commands provided by System Resource Control (SRC).

You can start the Problem Management daemon with the commands:

```
• startsrc -s pman
• pmanctrl -s
```

You can refresh the Problem Management daemon with the command:

```
refresh -s pman
```

The pmandef command works for both pmand and the pmandConfig SDR when you use it to subscribe, unsubscribe, activate, or deactivate.

Be careful when you use pmandef to query. It reports only the current pmand status. If you want to know the content in the pmandConfig SDR, you have to use the pmanquery command instead.

Figure 44 on page 92 shows the relationship among commands for the Problem Management daemon, pmand, and the pmandConfig SDR.

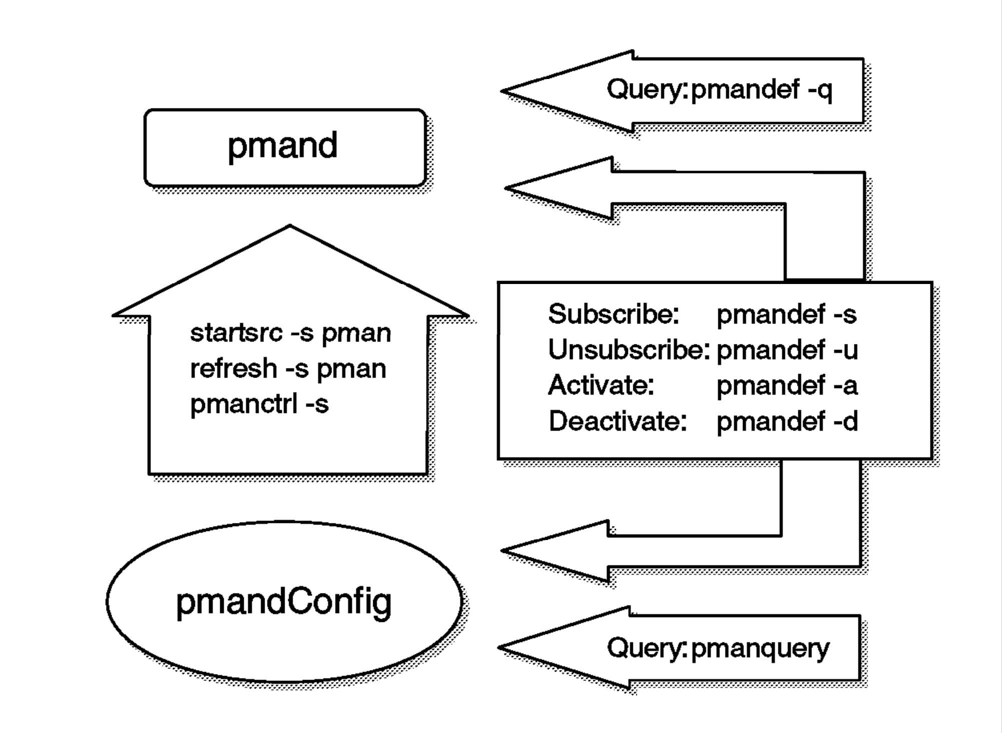


Figure 44. Commands for the Problem Management Daemon

3.5.2 Commands for the Problem Management Resource Monitor Daemon

The Problem Management Resource Monitor daemon refers to the pmanrmdConfig SDR when it is started or refreshed. The Problem Management subsystem provides the `pmanctrl` command for this purpose. But this command works for both the Problem Management daemon and the Problem Management Resource Monitor daemon at the same time. If you want to operate only the Problem Management Resource Monitor daemon, you have to use commands provided by System Resource Control (SRC).

You can start the Problem Management Resource Monitor daemon with the following commands:

- `startsrc -s pmanrm`
- `pmanctrl -s`

There is no refresh command for the Problem Management Resource Monitor daemon.

To create a new object in the pmanrmdConfig SDR, the Problem Management subsystem provides the `pmanrmdloadSDR` command. If you want to perform an action other than create, you can use regular SDR commands, such as SDRGetObjects or SDRDeleteObjects.

Be careful when you use the `pmanrmdloadSDR` command, because it works on the pmanrmdConfig SDR only. If you want to use this object, you have to restart the Problem Management Resource Monitor daemon.

The pmanrminput command changes the status of the Resource Variable immediately.

Figure 45 shows the relationship among commands for the Problem Management Resource Monitor daemon, pmanrmd, and the pmanrmdConfig SDR.

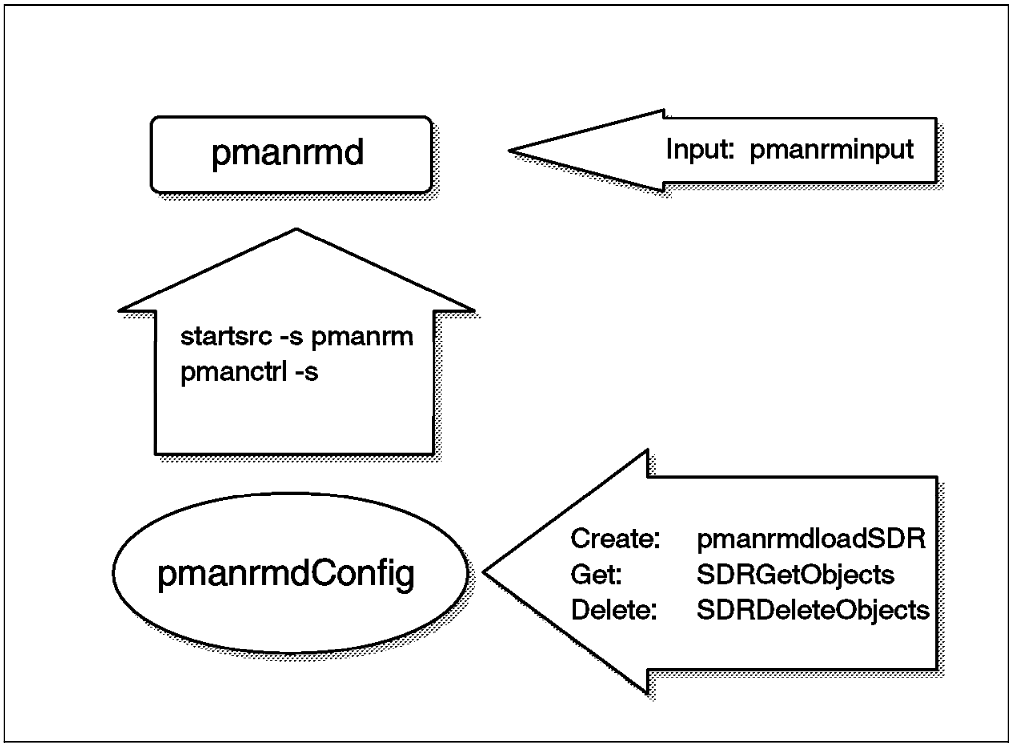


Figure 45. Commands for the Problem Management Resource Monitor Daemon

3.6 Short Examples

This section offers several simple examples of monitors.

3.6.1 The File System Monitor

Whenever the file system associated with the hd3 logical volume and the rootvg volume group on Nodes 1 through 8 becomes more than 90% full, the text /tmp file system is almost full is written to the AIX error log and BSD syslog facilities on the node where the file system filled up.

```
# pmandef -s Tmp_Filesystem_Monitor \
          -e ′IBM.PSSP.aixos.FS%totused: \
              NodeNum=1-8;VG=rootvg;LV=hd3: \
              X>90′ \
          -l ″/tmp file system is almost full″ \
          -h local
```

3.6.2 The Process Monitor

Running this example on Node 1 causes the command echo program has stopped >/tmp/root_proc.out to run on Node 1 whenever the number of processes named root_proc and owned by user root on Node 2 becomes 0. When this number increases back to 1, the command echo program has restarted >/tmp/root_proc.out runs on Node 1.

```
# pmandef -s Process_Monitor \
          -e ′IBM.PSSP.Prog.pcount: \
             NodeNum=2;ProgName=root_proc;UserName=root: \
             X@0==0′ \
          -r ″X@0>0″ \
          -c ″echo program has stopped >/tmp/root_proc.out″ \
          -C ″echo program has restarted >/tmp/root_proc.out″
```

3.6.3 /etc File Changed Monitor

If you use the following Problem Management Resource Monitor daemon configuration file, the Problem Management Resource Monitor daemon will run the command /bin/ls -tl /etc | /bin/head -2 | /bin/grep -v total every 10 minutes on Node 5, and it will send the output of that command as the value of the Resource Variable IBM.PSSP.pm.User_state2 to Event Management.

```
TargetType=NODE_RANGE
Target=5
Rvar=IBM.PSSP.pm.User_state2
SampInt=600
Command=″/bin/ls -tl /etc | /bin/head -2 | /bin/grep -v total″
```

The output of this command is the name and time stamp of the last file to be changed. You could then subscribe to changes in the /etc directory with the following command:

```
# pmandef -s Etc_File_Changed_Monitor \
          -e ′IBM.PSSP.pm.User_state2:NodeNum=5:X@0!=X@P0′ \
          -c ″echo File changed in /etc directory on node 5 \
            at \$(date) >> /tmp/etc_change_log″ \
          -n 1-5
```

Whenever the most recent output of the command /bin/ls -tl / etc | /bin/head -2 | /bin/grep -v total on Node 5 is different from the previous output, the event definition will be satisfied and the command echo File changed in /etc directory on node 5 at \$(date) >> /tmp/e tc _change_log will get executed on Nodes 1 through 5.

3.6.4 Server Monitor

Suppose you have a server application running on Node 10 and you sometimes want to tell users on Nodes 1 through 9 to stop using this application. Create a Problem Management subscription on Nodes 1 through 9 to run the wall command to inform the users about the state of the server application.

When the server application is ready for users, it uses the pmanrminput command to communicate its state to the Problem Management subsystem. The Problem

Management subsystem passes the string READY to Event Management as the value of the Resource Variable IBM.PSSP.pm.User_state3. This value satisfies the event definition, so the command wall Node 10 is ready runs on Nodes 1 through 9.

When the server application has a problem, it can use pmanrminput to communicate a state change. The Resource Variable NOT READY satisfies the rearm event definition, so the command wall Stop using node 10 runs on Nodes 1 through 9.

```
$ pmandef -s Server_Monitor \
          -e ’IBM.PSSP.pm.User_state3:NodeNum=10:X@0==”READY”’ \
          -r ’X@0!=”READY”’ \
          -c ”wall Node 10 is ready” \
          -C ”wall Stop using node 10” \
          -n 1-9

$ pmanrminput -s pman -a ”IBM.PSSP.pm.User_state3+READY+”

$ pmanrminput -s pman -a ”IBM.PSSP.pm.User_state3+NOT READY+”
```

Chapter 4. Application Program Interfaces (APIs)

The Event Management subsystem provides two APIs that allow you to create your own monitors. One API allows programs to be clients of the Event Management subsystem, while the other supports the creation of new Resource Monitors.

Customized Event Management clients can be created if the Problem Management subsystem and Perspectives programs that are supplied with POWERparallel System Support Programs Version 2.2 (PSSP 2.2) are not appropriate for your special circumstances. Customized Resource Monitors allow information from new applications and facilities to be collected and passed on to the Event Management subsystem.

4.1 Some Details Before We Start

Important

It is possible for Event Management clients to cause serious performance problems for any partition on an RS/6000 SP, not just for the partition that the client is monitoring. In particular, it is possible to cause very high network traffic on the RS/6000 SP systems′ networks. The CPU and virtual memory utilization on the Control Workstation can be severely affected by Event Management clients.

Carefully evaluate whether it is appropriate to develop and test EMAPI code on your systems.

Note: All of these examples and programs are written in ANSI C and compiled using the xlc compiler. This compiler is a separate product and must be purchased and installed on your system before you can compile the examples. Be careful in the use of _BSD when building distributed applications. The AIX API documentation is pretty vague on the usage of _BSD. Our examples are coded using interfaces that do not require _BSD.

The include files and libraries needed by these examples are installed on the Control Workstation when PSSP 2.2 is installed. A number of samples can be found in the /usr/lpp/ssp/samples/ directory.

The examples depend on the ssp.clients fileset, which must be installed on an AIX system so that the libraries can be loaded by the example programs. You should be aware that the construction of the programs must be changed if this dependency needs to be removed for some reason.

The complete source code is available in machine-readable form. See Appendix H, “How to Get the Examples in This Book” on page 257 for more details.

All of the code was compiled and tested. We reformatted the source code figures in this book and made other small changes to improve the appearance. It is possible that we introduced some error during this process. If you find anything that looks wrong, check the diskette or online versions.

The Event Management subsystem APIs are documented in *IBM Parallel System Support Programs for AIX, Event Management Programming Guide and Reference*, SC23-3996. An online version of this manual is installed in /usr/lpp/ssp/docs.

Review the contents of the files in the /usr/lpp/ssp/README directory for release-specific information.

Attention

Read this *before* you rewrite the whole system.

The Problem Management subsystem can be configured to use the information that is obtained from the Event Management subsystem to provide many customized interfaces. Though writing your own Event Management subsystem client is an interesting and worthwhile project, it would be a good idea to take some time to learn about the tools that are provided with PSSP 2.2, as these are very different from those available in PSSP 2.1 and PSSP 1.2.

Even if you find that the current tools do not provide the facilities you need, an understanding of how those clients work will be helpful in designing and creating your custom Event Management subsystem client.

4.2 Example EMAPI Programs

Perhaps the best way to understand the Event Manager APIs is to construct a program that uses them. PSSP 2.2 provides you with a few sample programs. They can be found in the /usr/lpp/ssp/samples directory.

Three Event Management subsystem clients are described in this section. They are:

lsemv Lists Event Management variables in various ways, obtaining the information from the Event Management subsystem rather than using the possibly stale information in the SDR.

getemv Gets the current values of the specified instances of an Event Management subsystem variable.

monemv Registers a user-specified Event Management event predicate, then sends messages to stdout when the events occur.

4.2.1 Utility Functions and Construction of EMAPI Clients

The construction process for EMAPI clients requires little special work. Note the following points:

- EMAPI clients must be linked with libha_em.a to resolve the EMAPI functions.
- EMAPIs are defined in ha_emapi.h, which includes other files. These files are installed in the normal locations in /usr when PSSP 2.2 is installed, so no special search flags are needed in your Makefile.

The Makefile that is used to build these sample programs is shown in Figure 46 on page 99.

```
PROGS=lsemv getemv monemv
DESTDIR=/var/local/SPmonitoring/bin
CC=xlc
CFLAGS=-g

all:    $(PROGS)

lsemv:  lsemv.o util.o Makefile
        $(CC) -o $@ lsemv.o util.o -lha_em

getemv: getemv.o util.o Makefile
        $(CC) -o $@ getemv.o util.o -lha_em

monemv: monemv.o util.o Makefile
        $(CC) -o $@ monemv.o util.o -lha_em


lsemv.o:        lsemv.c util.h  Makefile
        $(CC) $(CFLAGS) -c lsemv.c -o $@

getemv.o:       getemv.c util.h Makefile
        $(CC) $(CFLAGS) -c getemv.c -o $@

monemv.o:       monemv.c util.h Makefile
        $(CC) $(CFLAGS) -c monemv.c -o $@

util.o: util.c util.h Makefile
        $(CC) $(CFLAGS) -c util.c -o util.o



install:        all
        cp $(PROGS) $(DESTDIR)

clean:
        rm -f *.o $(PROGS)
```

Figure 46. The Makefile File for Constructing lsemv, getemv, and monemv

The util.c utility contains a number of common routines. The util.h header file, which is shown in Figure 47 on page 100, is included in all EMAPI sample programs.

These utility functions fall into three categories:

- Error message routines
- Wrapper functions for `printf()` that test the return codes
- Functions to manipulate SBS data items

Check the sources for these routines if you are interested in the details.

Error handling is very important in Event Management subsystem clients and many of the routines help with this task. Generally, functions that provide return codes are checked to make sure that all calls complete correctly. The error message routines are an exception to this, because if the return code is in an error function, things are probably already going badly and any meaningful recovery may not be possible. There are other instances where error codes are ignored. Examples include the movement of data to successfully allocated memory.

```
#ifndef ___UTIL_H
#define ___UTIL_H

struct sbs_elem_t {
  struct sbs_elem_t *next;  /* points to next element, NULL in end */
  unsigned short    length; /* as defined in SBS */
  unsigned char     sn;     /* serial number 0-255 */
  unsigned char     type;   /* enum ha_emField_Type */
  union {
    long    l; /* length is 4 */
    float   f; /* length is 4 (this is really a float, not a double */
    char   *c; /* \0 terminated string.  \0 is included in length */
    char   *b; /* string of bytes. no trailing \0 */
  } value;
};

extern const int EXIT_NOEXIT;
extern const int EXIT_NOERROR;
extern const int EXIT_USAGE;
extern const int EXIT_NOTFOUND;
extern const int EXIT_TIMEOUT;
extern const int EXIT_RESOURCE;
extern const int EXIT_UNEXPECTED;

extern void setThisProgramName(const char *name);

extern char * getThisProgramName();

extern void errorExit(const int exitCode, const char *format, ...);

extern void printEMMsg(const int exitCode, const char *msg,
                       const struct ha_em_err_blk *error);

void printEventItemError(const struct ha_em_rpb_event error);
void printCurrentItemError(const struct ha_em_rpb_qcur error);
void printQueryItemError(const struct ha_em_rpb_qerr error);
void printDefinedItemError(const struct ha_em_rpb_qdef error);
void printRegisterItemError(const struct ha_em_rpb_rerr error);

extern void safePrintf(const char *format, ...);

unsigned long writeSBSElements(FILE *file, const struct sbs_elem_t *data);
void freeSBSElements(struct sbs_elem_t *data);
char * elementsToSBS(const struct sbs_elem_t *sbs);
struct sbs_elem_t * SBSToElements(const char *sbs);
void safePrintSBS(char *sbs);

#endif /* ___UTIL_H */
```

Figure 47. Excerpts from util.h Showing Utility Functions

The util.h utility is shown in Figure 47; the following describes details of using it.

Using setThisProgramName(char *name) stores the program name, argv[0], for later use by the error printing functions.

errorExit(const int exitCode, const char *format, ...) is built on printf(const char *format, ...).

The parameter exitCode causes the program to terminate with an exit code after printing the formatted message (if its value is not EXIT_NOEXIT).

A variety of functions are provided to send formatted messages to stdout that describe errors returned by EMAPI routines.

safePrintf(const char * format, ...) is similar to regular printf(), except that the return code is tested, and if nothing is written to stdout, an error is generated.

Finally, a number of routines are provided to manipulate SBS data objects. These break an SBS object into a list of its elements and allow it to be written to a stream in a human-readable format.

To see how these utility functions are used, let us turn our attention to the first example.

4.2.2 The lsemv Program

We start with the simplest sample program, lsemv. This program queries the Event Management subsystem to discover the Resource Variables and Instance Vectors that are currently available. Two examples of this program (lsemv and lsemv -l) are shown in Figure 48.

```
pgc@sp2en0: lsemv
IBM.PSSP.aixos.Proc.swpque
IBM.PSSP.aixos.Proc.runque
⋮
IBM.PSSP.CSS.bcast_tx_ok
IBM.PSSP.CSS.bcast_rx_ok
pgc@sp2en0:
pgc@sp2en0: lsemv -l
IBM.PSSP.aixos.Proc.swpque              Q F IBM.PSSP.aixos.Proc  NodeNum=int
IBM.PSSP.aixos.Proc.runque              Q F IBM.PSSP.aixos.Proc  NodeNum=int
IBM.PSSP.aixos.pagsp.size               Q L IBM.PSSP.aixos.PagSp Name=cstring;N
odeNum=int

⋮
```

Figure 48. Two Examples of the lsemv Program

```
pgc@sp2en0: lsemv -v IBM.PSSP.aixos.Proc.swpque
======================
Resource Variable Name: ″IBM.PSSP.aixos.Proc.swpque″
Variable Value Type:    Quantity
Variable Data  Type:    float
Variable SBS Format:    ″″
Variable Initial Value: ″0.000000″
Variable Class:         ″IBM.PSSP.aixos.Proc″
Instance Vector:        ″NodeNum=int″
PTX Name:               ″″
Default Predicate:      ″″
Locator:                ″NodeNum″
--------------------
Variable Description
--------------------
Average count of processes waiting to be paged in
--------------------------
Instance Vector Desciption
--------------------------
The number of the node for which the information applies.
-----------------
Event Description
-----------------
pgc@sp2en0:
```

Figure 49. Another Example of lsemv

The flow of lsemv is:

1. Parse the command line.
2. Initialize the environment for EMAPI.
3. Open a session with the Event Management subsystem.
4. For each name on the command line:
   a. Construct and send the query request to Event Management.
   b. Receive and process the responses from Event Management.
5. Close the session with the Event Management subsystem.
6. Clean up the environment and exit.

Figure 50 on page 103 shows the main() function. The program name is stored so that it can be printed in error messages. The values of the command line flags are stored in global variables. The `parse()` function uses `getopt()` to retrieve parameters from the command line. If specified on the command line, an alarm is started to terminate the process if something stalls. The EMAPI documents encourage you to ignore SIGPIPE. (This might not be possible, depending on your application′s needs.) In the lsemv program, SIGPIPE is not needed, so it is ignored.

Next, a session is opened to the Event Management subsystem. processNames() processes each name on the command line. If there are no names on the command line, an empty line is created.

After all the names are processed, the Event Management subsystem session is closed and the program terminates with the appropriate return code.

```
int
main(int argc, char *argv[])
{
int sessionFD;
int nextName;
struct ha_em_err_blk error;

  setThisProgramName(argv[0]);  /* keep this for use in error messages */
  nextName = parse(argc, argv);
  if (helpFlag) {
    usage();
    exit(EXIT_USAGE);
  }

  if (timeoutFlag > 0) {
    if (signal(SIGALRM, timeoutHandler) == ( void(*)(int) ) -1) {
      errorExit(EXIT_UNEXPECTED, ″problem installing SIGPIPE handler\n″);
    }
    alarm(timeoutFlag);
  }

  if (signal(SIGPIPE, SIG_IGN) == ( void(*)(int) ) -1) {
    errorExit(EXIT_UNEXPECTED, ″problem ignoring SIGPIPE\n″);
  }

  sessionFD = ha_em_start_session(spNameFlag, &error);
  if (sessionFD == -1) {
    printEMMsg(EXIT_UNEXPECTED, ″ha_em_start_session″, &error);
  }

  if (argc == nextName) {
  char *emptyString = ″″;
    processNames(&sessionFD, 1, &emptyString);  /* no names on command l
  } else {
    processNames(&sessionFD, argc - nextName, &argv[nextName]);
  }

  if ( ha_em_end_session(sessionFD, &error) == -1) {
    printEMMsg(EXIT_UNEXPECTED, ″ha_em_end_session″, &error);
  }

  exit( numberNotFound ? EXIT_NOTFOUND : EXIT_NOERROR );
  return 0;
}
```

Figure 50. The main() Function for lsemv

```
void
processNames(int *sessionFD, int count, char *list[])
{
int loop;
size_t commandSize;
struct ha_em_cmd_blk *commandP;
struct ha_em_err_blk error;
ha_em_qid_t qId;

  for (loop = 0; loop < count; loop++) {
    commandSize = sizeof(struct ha_em_cmd_blk)+sizeof(struct ha_em_rb_que
    if ( (commandP = malloc(commandSize)) == NULL) {
      errorExit(EXIT_RESOURCE, ″Can not allocate emCommand struct.\n″);
    }

    commandP->em_cmd_num_elem = 1;
    commandP->em_cmd =          HA_EM_CMD_QUERY;
    commandP->em_subcmd =       HA_EM_SCMD_QDEF;
    commandP->em_qcb =          NULL;
    commandP->em_qcb_arg =      NULL;

    commandP->em_res_blk.em_rb_query[0].em_class =   ″″;
    commandP->em_res_blk.em_rb_query[0].em_name =    list[loop];
    commandP->em_res_blk.em_rb_query[0].em_ivector = ″*″;

    if ( ha_em_send_command(*sessionFD, commandP, &error) == -1 ) {
      printEMMsg(EXIT_UNEXPECTED, ″ha_em_send_command″, &error);
    }

    qId = commandP->em_qid;  /* save this so we can match up the response
    free(commandP);

    processResponses(sessionFD, qId);
  }  /* for (loop ... ) through names on command line */
}
```

Figure 51. The processNames() Function for lsemv

Figure 51 shows processNames(), which constructs a command block that requests a query of the defined Resource Variables. That command block is sent to the Event Management subsystem using the `ha_em_send_command()` command. After the command block has been successfully sent, processResponses() is called to sort through all the responses that the Event Management subsystem sends to satisfy the query.

This program wildcards the class, because we are looking for Resource Variables by name. We also wildcard the Instance Vector. Notice that the class is wildcarded by using an empty string, while the Instance Vector requires a string containing an asterisk (*) character. The name parameter uses an empty string (like the one that is used if no name is provided on the command line) as a wildcard.

In this program, the command block is allocated and then freed each time through the loop. We could have declared a command block as an automatic variable in the function. There is no requirement that the command block be dynamically allocated.

The command block is sent to the Event Manager via the `ha_em_send_command()` function. Few errors are reported by this function. Most of the interesting errors are discovered when the responses are processed. Still, you must check for errors at this point.

If the send function completes successfully, the EMAPI will generate a unique query identifier for this request. You should store this value so you can make sure that the responses you receive are for the appropriate query. In this simple program, this should not be a problem. In a program that issues multiple queries, the query identifiers are used to sort out the information as it is received.

Creating the command block and sending it to the Event Manager is relatively easy. Processing the information that is received in response to the query is the tough part. Let us see what happens in processResponses(), which is shown in Figure 52 on page 106.

```
void
processResponses(int *sessionFD, const ha_em_qid_t qId)
{
int receiveRC;
struct ha_em_rsp_blk *responseP;
struct ha_em_err_blk error;

  for ( ; ; ) {
    receiveRC = ha_em_receive_response(*sessionFD, &responseP, &error);
    switch ( receiveRC ) {
      case -1:
        /* the only error that can be recovered is HA_EM_ECONNLOST */
        if ( error.em_errno == HA_EM_ECONNLOST) {
        int newFD;
          newFD = ha_em_restart_session(*sessionFD, &error);
          if (newFD == -1) {
            printEMMsg(EXIT_UNEXPECTED,
                       ″ha_em_restart_session failed\n″, &error);
          }
          *sessionFD = newFD;
        }
        break;

      case 0:
        break;

      default:
        if (responseP->em_qid == qId) {
          processOneResponse(responseP);
        } else {
          errorExit(EXIT_UNEXPECTED, ″got wrong qid″);
        }
    }

    if (receiveRC != 0) {
      if (responseP->em_qend) {
        free(responseP);  /* must free responseP in both cases */
        return;
      } else {
        free(responseP);  /* must free responseP in both cases */
      }
    }
  }  /* for ( ; ; ) */
  errorExit(EXIT_UNEXPECTED, ″error in program logic!!\n″);
}
```

Figure 52. The processResponses() Function for lsemv

The function processResponses() continues to call ha_em_receive_response() until a response is received with the query end indicator, em_qend, set. The for (; ;) structure will loop forever. The only ways out of this loop are the return near the end of the loop, or exiting if a nonrecoverable error is detected.

The most difficult task is making sure that all possible error-reporting variables are tested correctly. Certain error indicators are only valid when other error indicators are in certain states.

The first thing to test is the return value from ha_em_receive_response(). If this is 0, there is nothing for your process to do except to call ha_em_receive_response() again. None of the other error indicators should be tested, as they do not have valid information.

If the return value was -1, the error block of type struct ha_em_err_blk that was passed by reference will contain information about the problem that occurred. The only problem that is recoverable is the loss of the session connection, reported as HA_EM_ECONNLOST. If this error occurs, the application can try to restart the session using ha_em_restart_session(). Remember that if the session is successfully restarted, it will get a new file descriptor, so you must be able to update this value. That is why sessionFD is passed by reference.

Once the connection to the Event Management subsystem has been lost, it may not be possible to reestablish the connection immediately. The call may return an HA_EM_ECONNREFUSED error in this case. lsemv tries only once to restart the session, then gives up. However, other Event Management subsystem clients might be more persistent in attempts to restart the failed session. The interval selected to restart the connection depends on the needs of the client application, but it is probably not useful for intervals less than 5 seconds long.

If the return code was not -1 or 0, then ha_em_receive_response() has allocated a response block and returned its address to your application. The response block must be freed by your application when you are done processing the data it contains. The response block is created so that all of the data areas that are referenced in the response block will be correctly freed if your application simply calls free() with the address of the response block. As usual, do not free the memory till you are certain your process will no longer need the data.

Note: The contents of the error block passed to ha_em_receive_response() will only be set to valid values when ha_em_receive_response() returns -1.

While the response block has a number of fields, it is critical to check three particular fields. The first is the response block level error number. Many errors are reported at this level. The query identifier should be checked to make certain this information has been sent in response to your application′s query. Also, the query end indicator is set in the last response that contains information for the query. This is how lsemv gets out of the loop under normal circumstances.

If the ha_em_receive_response() return value is greater than 0 and this is data for the correct query, the response block is passed to processOneResponse(), shown in Figure 53 on page 108, to extract the desired information. The response block is freed by processResponses().

```
void
processOneResponse(const struct ha_em_rsp_blk *responseP)
{
  switch ( responseP->em_cmd ) {
    case HA_EM_CMD_QUERY:
      switch ( responseP->em_subcmd ) {
        case HA_EM_SCMD_QDEF:
          processDefinedList(responseP);
          break;

        default:
          /* something has gone wrong */
          errorExit(EXIT_UNEXPECTED, ″value of em_subcmd = %d\n″,
                      responseP->em_subcmd);
          /* NOTREACHED */
          break;
      }  /* switch (responseP->em_subcmd ) */
      break;

    case HA_EM_CMD_QERR:
      processQerrList(responseP);
      break;

    case HA_EM_CMD_REG:
    case HA_EM_CMD_REG2:
    case HA_EM_CMD_UNREG:
    case HA_EM_CMD_RERR:
    case HA_EM_CMD_R2ERR:
    default:
      /* something has gone wrong */
      errorExit(EXIT_UNEXPECTED, ″value of em_cmd = %d\n″,
                  responseP->em_cmd);
      /* NOTREACHED */
      break;
  }
}
```

Figure 53. The processOneResponse() Function for lsemv

The function processOneResponse() uses two fields in the response block, em_cmd and em_subcmd, to determine what type of data is in the block. Different functions are called to do the actual work of processing the information, depending on the type of response.

The lsemv client expects only two types of response blocks, in particular, HA_EM_CMD_QUERY and HA_EM_CMD_QERR. No other type of information should be returned. If the response block contains error information (HA_EM_CMD_QERR), processQerrList() processes the information. Responses that contain the data that lists the Resource Variables (HA_EM_CMD_QUERY), are processed by processDefinedList(). Let us look at processQerrList() in Figure 54 on page 109 first.

```
void
processQerrList(const struct ha_em_rsp_blk *responseP)
{
int loop;

  for (loop = 0; loop < responseP->em_rsp_num_resp; loop++ ) {
    printQueryItemError(responseP->em_resp_blk.em_rpb_qerr[loop]);
    numberNotFound++;
  }
}
```

Figure 54. The processQerrList() Function for lsemv

In processQerrList() we see that a single response block can hold many items. em_rsp_num_resp provides the number of items that are contained in the response block. Early versions of this function would look for a particular type of error that is to be expected in the normal operation of the lsemv command. That error is reported when a Resource Variable that was listed on the command line is not found in the Event Management subsystem.

The current version calls the utility function printQueryItemError(). This function sends an error message to stderr. This collects the code for decoding the errors in a single place.

A global variable is incremented so that we can provide the correct exit code when the application ends. This is similar to the way the ls command handles names that are not found.

Finally we reach the function that prints the list of event manager Resource Variables that are defined in the Event Management subsystem.

processDefinedList() looks more complex than processQerrList(), but it is quite simple. processDefinedList() loops through the items. After it checks the per item error code, it formats the information as requested on the command line.

processDefinedList() is shown in Figure 55 on page 110. Using #define D as shown in the figure can save a lot of typing because the names can be long.

We could pass a pointer to the array of em_rpb_qdef, but then we would have to pass an extra parameter with the value of em_rsp_num_resp. Good compilers should be able to handle either choice efficiently.

```
void
processDefinedList(const struct ha_em_rsp_blk *responseP)
{
int loop;

/* the following saves lots of typing */
#define D responseP->em_resp_blk.em_rpb_qdef[loop]

  for (loop = 0; loop < responseP->em_rsp_num_resp; loop++ ) {
    if (responseP->em_resp_blk.em_rpb_qdef[loop].em_errnum) {
      printDefinedItemError(D);
      numberNotFound++;
    } else {
      /* OK, no errors, figure what kind of output is desired and print i
      if ( verboseFlag) {
        /* verbose version */
        safePrintf(″======================\n″);
        safePrintf(″Resource Variable Name: \″%s\″\n″, D.em_name);
        safePrintf(″Variable Value Type:    %s\n″,
                   (D.em_value_type == ha_emVTcounter) ? ″Counter″ :
                   (D.em_value_type == ha_emVTquantity) ? ″Quantity″ :
                   (D.em_value_type == ha_emVTstate) ? ″State″ : ″UNKNOWN
        safePrintf(″Variable Data  Type:    %s\n″,
                   (D.em_data_type == ha_emDTlong) ? ″long″ :
                   (D.em_data_type == ha_emDTfloat) ? ″float″ :
                   (D.em_data_type == ha_emDTsbs) ? ″SBS″ : ″UNKNOWN″);
        safePrintf(″Variable SBS Format:    \″%s\″\n″, D.em_sbs_format);
        safePrintf(″Variable Initial Value: \″%s\″\n″, D.em_init_value);

⋮

⋮
      /* long version */
      } else if ( longFlag ) {
        safePrintf(″%-40s ″,                           D.em_name);
        safePrintf(″%1s ″, (D.em_value_type == ha_emVTcounter) ? ″C″ :
                           (D.em_value_type == ha_emVTquantity) ? ″Q″ :
                           (D.em_value_type == ha_emVTstate) ? ″S″ : ″U″)
        safePrintf(″%1s ″, (D.em_data_type == ha_emDTlong) ? ″L″ :
                           (D.em_data_type == ha_emDTfloat) ? ″F″ :
                           (D.em_data_type == ha_emDTsbs) ? ″S″ : ″U″);
        safePrintf(″%-20s ″,                            D.em_class);
        safePrintf(″%s\n″,                              D.em_ivector);
      /* normal version */
      } else {
        safePrintf(″%s\n″,                              D.em_name);
      }
    }
  }
}
/* clean up */
#undef D
```

Figure 55. The processDefinedList() Function for lsemv

The other functions in lsemv offer no insight into the use of the EMAPI and are not discussed. The complete set of source code is provided. See Appendix H, “How to Get the Examples in This Book” on page 257 for details.

4.2.3 The getemv Program

The getemv program queries the Event Management subsystem for the current values of Resource Variables. Some examples of its use are shown in Figure 56 and Figure 57.

```
pgc@sp2en0: getemv IBM
========================================
Resource Variable: ″IBM.PSSP.Membership.LANAdapter.state″
Instance Vector:   ″NodeNum=16;AdapterType=en;AdapterNum=0″
Location:          0
Data Type:         LONG
Value:             1

========================================
Resource Variable: ″IBM.PSSP.Membership.LANAdapter.state″
Instance Vector:   ″NodeNum=15;AdapterType=en;AdapterNum=0″
Location:          0
Data Type:         LONG
Value:             1

========================================
Resource Variable: ″IBM.PSSP.Membership.LANAdapter.state″
Instance Vector:   ″NodeNum=14;AdapterType=en;AdapterNum=0″
Location:          0
Data Type:         LONG
Value:             1

========================================

⋮
pgc@sp2en0:
```

Figure 56. Using getemv to List All Instances of Names That Start with IBM

```
pgc@sp2en0: getemv -l IBM.PSSP.SP_
HW.Node.keyModeSwitch NodeNum=1-8
IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=8 0 LONG 0
IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=7 0 LONG 0
IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=6 0 LONG 0
IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=5 0 LONG 0
IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=4 0 LONG 0
IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=3 0 LONG 0
IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=2 0 LONG 0
IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=1 0 LONG 0
pgc@sp2en0:
```

Figure 57. Using getemv to List Specific Instances of One Resource Variable

The structure of getemv is very similar to that of lsemv. Both programs query the Event Management subsystem for information, and then exit. This section discusses some of the differences between these programs.

The main() functions differ in the way the command lines are parsed. The lsemv function can take any number of names, while getemv requires a single Resource Variable name and allows an optional Instance Vector to be specified. The basic setup and cleanup tasks are the same.

The processName() function in Figure 58 on page 112 illustrates how different command blocks are created. Notice that in getemv, em_subcmd is set to HA_EM_SCMD_QCUR rather than HA_EM_SCMD_QDEF. The Instance Vector might not be a wildcard in getemv if the user specified a value on the command line.

```
void
processName(int *sessionFD, char * name, char *ivector)
{
size_t commandSize;
struct ha_em_cmd_blk *commandP;
struct ha_em_err_blk error;
ha_em_qid_t qId;

    commandSize = sizeof(struct ha_em_cmd_blk)+sizeof(struct ha_em_rb_que
    if ( (commandP = malloc(commandSize)) == NULL) {
      errorExit(EXIT_RESOURCE, ″Can not allocate emCommand struct.\n″);
    }

    commandP->em_cmd_num_elem = 1;
    commandP->em_cmd =          HA_EM_CMD_QUERY;
    commandP->em_subcmd =       HA_EM_SCMD_QCUR;
    commandP->em_qcb =          NULL;
    commandP->em_qcb_arg =      NULL;

    commandP->em_res_blk.em_rb_query[0].em_class =   ″″;
    commandP->em_res_blk.em_rb_query[0].em_name =    name;
    commandP->em_res_blk.em_rb_query[0].em_ivector = ivector;

    if ( ha_em_send_command(*sessionFD, commandP, &error) == -1 ) {
      printEMMsg(EXIT_UNEXPECTED, ″ha_em_send_command″, &error);
    }

    qId = commandP->em_qid;  /* save this so we can match up the response
    free(commandP);

    processResponses(sessionFD, qId);
}
```

Figure 58. The processName() Function for getemv

The processResponse() functions perform the same tasks in lsemv and getemv. However, the processOneResponse() function differs slightly.

In getemv, the responses of type HA_EM_SCMD_QCUR are the ones that contain the information that answers the user′s request.

The processOneResponse() function is shown in Figure 59 on page 113, where you can compare it with the version for lsemv in Figure 53 on page 108.

```
void
processOneResponse(const struct ha_em_rsp_blk *responseP)
{
  switch ( responseP->em_cmd ) {
    case HA_EM_CMD_QUERY:
      switch ( responseP->em_subcmd ) {
        case HA_EM_SCMD_QCUR:
          processValueList(responseP);
          break;

        default:
          /* something has gone wrong */
          errorExit(EXIT_UNEXPECTED, ″value of em_subcmd = %d\n″,
                      responseP->em_subcmd);
          /* NOTREACHED */
          break;
      }  /* switch (responseP->em_subcmd ) */
      break;

    case HA_EM_CMD_QERR:
      processQerrList(responseP);
      break;

    case HA_EM_CMD_REG:
    case HA_EM_CMD_REG2:
    case HA_EM_CMD_UNREG:
    case HA_EM_CMD_RERR:
    case HA_EM_CMD_R2ERR:
    default:
      /* something has gone wrong */
      errorExit(EXIT_UNEXPECTED, ″value of em_cmd = %d\n″,
                  responseP->em_cmd);
      /* NOTREACHED */
      break;
  }
}
```

Figure 59. The processOneResponse() Function for getemv

```
void
processValueList(const struct ha_em_rsp_blk *responseP)
{
#define D responseP->em_resp_blk.em_rpb_qcur[loop]

  for (loop = 0; loop < responseP->em_rsp_num_resp; loop++ ) {
    if (responseP->em_resp_blk.em_rpb_qcur[loop].em_errnum) {
      printCurrentItemError(D);
      numberNotFound++;
    } else {
      /* OK, no errors, figure what kind of output is desired and print i
      if (longFlag) {
        /* long version */
        safePrintf(″%s ″, D.em_name);
        safePrintf(″%s ″, D.em_ivector);
        safePrintf(″%d ″, D.em_location);
        switch (D.em_data_type) {
          case ha_emDTlong:
            safePrintf(″%s ″, ″LONG″);
            safePrintf(″%ld″, (long) D.em_val.em_vall);
            break;
          case ha_emDTfloat:
            safePrintf(″%s ″, ″FLOAT″);
            safePrintf(″%f″, (double) D.em_val.em_valf);
            break;
          case ha_emDTsbs:
            safePrintf(″%s ″, ″SBS″);
            safePrintSBS(D.em_val.em_valsbs);
            break;
          default:
            errorExit(EXIT_NOEXIT,″UNKNOWN DATA TYPE OF %d″,
                     (int) D.em_data_type);
        }
      } else {
        /* normal version */
        safePrintf(″=======================================\n″);
        safePrintf(″Resource Variable: \″%s\″\n″, D.em_name);
        safePrintf(″Instance Vector:   \″%s\″\n″, D.em_ivector);

⋮

⋮
          default:
            errorExit(EXIT_NOEXIT,″UNKNOWN DATA TYPE OF %d″,
                     (int) D.em_data_type);
        }
      }
      safePrintf(″\n″);
    }
  }
}
/* clean up */
#undef D
```

Figure 60. The processValueList() Function for getemv

The last important difference is shown in Figure 60. processValueList() is similar to processDefinedList(). In getemv there is less information to display, though there will probably be more items in the lists, since each Resource Variable can have many instances.

These first two programs are very similar. An interesting exercise would be to rewrite lsemv and getemv as a single program. This program would alter its behavior depending on the name by which it was invoked. Once the basic flow is understood, it is fairly easy to create an EMAPI client that extracts information from the Event Management subsystem.

4.2.4 The monemv Program

You could use getemv to periodically sample the values of interesting Resource Variables. However, if these change slowly, this approach would waste resources at the client and in the Event Management subsystem. It is more efficient to register the Resource Variables and the instances of the variables that are interesting to you, and have the Event Management subsystem tell you when an event has occurred. This is what monemv does.

To see an example of how monemv works, refer to Figure 61, which shows monemv waiting for the events of the key mode switch not being in the normal position on any node and the switch being returned to the normal position. Figure 62 on page 116 shows how the key mode switches were manipulated to generate the events that monemv received. The date command shows when the key mode switches were altered.

Notice that the quick transition from normal to service and back to normal that occurred near 13:53:00 was completely missed by the Event Management subsystem. The subsystem was never given the event by the Resource Monitor. Actually, the monitor never got the information from lower-level RS/6000 SP subsystems. The Resource Monitor for the RS/6000 SP hardware only samples the system through the hardmon process. By default, hardmon samples the hardware at 5-second intervals. It never noticed that the switch was set and then reset.

It is important to understand these types of limitations of the Event Management subsystem and the subsystems that are used to collect information. For more information on Resource Monitors, see Chapter 2, “Resource Monitors” on page 27.

```
pgc@sp2en0: monemv IBM.PSSP.SP_HW.Node.keyModeSwitch ’NodeNum=*’ ’X!=0’ ’X==0’
PRED1 19960925135223 IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=1 2
PRED1 19960925135238 IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=5 1
PRED2 19960925135248 IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=1 0
PRED2 19960925135253 IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=5 0
PRED1 19960925135648 IBM.PSSP.SP_HW.Node.keyModeSwitch NodeNum=1 2
```

Figure 61. Example of monemv Monitoring Key Mode Switches on Nodes

```
# date ; spmon -key service node1
Wed Sep 25 13:52:21 EDT 1996
# date ; spmon -key secure node5
Wed Sep 25 13:52:36 EDT 1996
# date ; spmon -key normal node1
Wed Sep 25 13:52:47 EDT 1996
# date ; spmon -key normal node5
Wed Sep 25 13:52:51 EDT 1996
# date ; spmon -key service node1
Wed Sep 25 13:53:00 EDT 1996
# date ; spmon -key normal node1
Wed Sep 25 13:53:02 EDT 1996
# date ; spmon -key service node1
Wed Sep 25 13:56:47 EDT 1996
#
```

Figure 62. Use of spmon to Alter Key Mode Switches on Nodes

The structure of monemv differs from the structure of lsemv and getemv. The most important difference is the way monemv never stops running once it is started. It only exits if it encounters an error or is killed by a signal.

The processResponses() function is different in monemv in two important ways. First, once the function is started, it is designed to loop forever waiting for responses from the Event Management subsystem. The `for ( ; ; ) { ... }` structure is commonly used in C programs for this purpose. The second difference is in the use of event identifiers.

In the query programs, each response was associated with a particular query identifier. The Event Management subsystem API does not treat the responses for events in the same way. The event identifiers are stored in the arrays of events that are returned in each response. This means we must check each item in the `em_resp_blk.em_rpb_event` array to see whether it is for the desired event.

This is an important difference between the way replies to queries and replies delivering events are processed. processResponses() is shown in Figure 63 on page 117.

```
void
processResponses(int *sessionFD, const ha_em_eid_t eId)
{
int receiveRC;
struct ha_em_rsp_blk *responseP;
struct ha_em_err_blk error;

  for ( ; ; ) {
    receiveRC = ha_em_receive_response(*sessionFD, &responseP, &error);
    switch ( receiveRC ) {
      case -1:
        /* the only error that can be recovered is HA_EM_ECONNLOST */
        if ( error.em_errno == HA_EM_ECONNLOST) {
        int newFD;
          newFD = ha_em_restart_session(*sessionFD, &error);
          if (newFD == -1) {
            printEMMsg(EXIT_UNEXPECTED,
                       ″ha_em_restart_session failed\n″, &error);
          }
          *sessionFD = newFD;
        }
        break;

      case 0:
        /* book says this is OK, just try again */
        break;

      default:
        processOneResponse(responseP, eId);
        free(responseP);  /* must free responseP object */
    }

  }  /* for ( ; ; ) */
  errorExit(EXIT_UNEXPECTED, ″error in program logic!!\n″);
}
```

Figure 63. The processResponses() Function for monemv

The individual responses are classified into normal and error types by processOneResponse(), just as in the query programs. Normal values cause formatted messages to be sent to stdout, while error items generate messages to stderr.

```
void
processEventList(const struct ha_em_rsp_blk *responseP)
{
int loop;
char timeBuffer[15];  /* to hold ″YYYYMMDDHHMMSS″ */
time_t thisTime;

#define D responseP->em_resp_blk.em_rpb_event[loop]

  for (loop = 0; loop < responseP->em_rsp_num_resp; loop++ ) {
    if (D.em_errnum) {
      printEventItemError(D);
    } else {
      if (D.em_event_flags == HA_EM_EVENT_RE_ARM) {
        safePrintf(″PRED2 ″);
      } else {
        safePrintf(″PRED1 ″);
      }
      thisTime = D.em_timestamp.tv_sec;  /* get in right kind of containe
      if (strftime(&timeBuffer[0], sizeof timeBuffer, ″%Y%m%d%H%M%S″,
          localtime(&thisTime) ) ) {
        safePrintf(″%s ″, timeBuffer);
      } else {
        safePrintf(″00000000000000 ″);
      }
      safePrintf(″%s %s ″, D.em_name, D.em_ivector);
      switch(D.em_data_type) {
        case ha_emDTlong:
          safePrintf(″%ld ″, D.em_val.em_vall);
          break;
        case ha_emDTfloat:
          safePrintf(″%f ″, D.em_val.em_valf);
          break;
        case ha_emDTsbs:
          safePrintSBS(D.em_val.em_valsbs);
          break;
        default:
          errorExit(EXIT_UNEXPECTED, ″datatype of %d\n″,
                    (int) D.em_data_type);
      }
      if (fflush(stdout) == EOF) {
        errorExit(EXIT_RESOURCE,″can′t write to stdout.\n″);
      }
    }  /* if (D.em_error) */
  }  /* for (loop ... */
}
/* clean up */
#undef D
```

Figure 64. The processEventList() Function for monemv

processEventList() must determine whether an event item was caused by a regular or by a re-arm predicate. One line of output is sent to stdout, providing information about the event that occurred, and the stdout stream is flushed to force the event out of any buffers. This is important because another process could be waiting to receive the event from monemv. Without the `fflush()`, an event would not be visible until a whole buffer full of event messages was present.

4.2.5 EMAPI Summary

The lsemv, getemv, and monemv clients are probably the simplest general purpose Event Management subsystem clients possible. You will notice that many of the details in the EMAPI documentation found in *IBM Parallel System Support Programs for AIX, Event Management Programming Guide and Reference*, SC23-3996, are not needed for these programs.

Though these examples are simple, they allow most of the possible queries for information and enable all possible events to be received. (Queries of defined Resource Variables by class are not supported, but are easy to add.) The simple control flows show you how little code is required to get useful and reliable results from the EMAPI.

Applications can be designed using basically two programming models: threaded and non-threaded. The use of one model over the other depends on the application, because the EMAPI supports both of these two programming models.

4.3 Sample RMAPI Program

Resource Monitors are constrained to execute on the Control Workstation or on the nodes of an RS/6000 SP. When a Resource Monitor wishes to monitor a supplied service from the client′s point of view, it must be built in two parts. One part runs locally on the RS/6000 SP, while the other runs at the client.

This sample Resource Monitor demonstrates this arrangement. The program discussed in this section, httprtA, is one half of an HTTP Response Time Monitor. The second part, httprtB, is described in C.2.1, “httprtB, an HTTP Response Time Server” on page 226.

The construction of Resource Monitors is pretty simple, but their configuration and operation is more difficult than that of Event Monitors. The Resource Monitors must be described to the Event Management subsystem, then that subsystem must be restarted for the changes to take effect. Also, a Resource Monitor must run with root authority.

The program is described first, followed by the configuration and operational details.

4.3.1 The httprtA Program

The monitor in this example is a command-based Resource Monitor. Daemon-based and integrated Resource Monitors are also available, but their structures are often more complicated. The command-based Resource Monitor requires the least amount of program overhead. In the other models, the program does what the Resource Monitor subsystem tells it to do. The command-based model allows the program to control what is happening.

This comes at the expense of the types of values that can be collected. All Resource Monitors can supply State values. The other models can also supply Counter and Quantity types of values. Any of the data types (long, float, SBS) can be used by any model, so this is probably not a problem for a Resource Monitor that does not produce a lot of data. Data-intensive Resource Monitors might wish to use other value types since they are transmitted by a different mechanism. If you are writing your own Resource Monitor, you will probably want to try both approaches to see which is better suited to your needs.

Command-based Resource Monitor processes are just another daemon on a system. These must be started with a cron entry, a line in /etc/inittab, or some similar mechanism. The structure of httprtA shows that the process can be started in a variety of ways. Selected parts of the main() function are shown in Figure 66 on page 122. Note that httprtA takes a number of arguments that are needed to specify which HTTP object, as described by a URL, to monitor, and how to monitor it.

```
main(int argc, char *argv[], char *envp[])
{
#define programNameArg argv[0]
#define intervalArg    argv[1] /* how often to recheck response time */
#define timeoutArg     argv[2] /* how long to give remote server to check */
#define remoteHostArg  argv[3] /* host httprtB is running on */
#define urlArg         argv[4] /* URL to check */
#define nameArg        argv[5] /* name since IV elem limited to 31 char */
⋮
  remoteServerHost = gethostbyname(remoteHostArg);
  if ( remoteServerHost == NULL) {
    WriteLog(″can not resolve observing host name ′%s′, exit()ing\n″,
             remoteHostArg);
    exit(1);
  }

  if ( strlen(nameArg) > 31 ) {
    nameArg[31] = ′\0′;
    WriteLog(″Truncting name to 31 characters.  Now = ′%s′\n″, nameArg);
  }

  thisSignal.sa_handler = (void (*)() ) AlarmHandler;
  sigemptyset(&thisSignal.sa_mask);
  sigaddset(&thisSignal.sa_mask, SIGALRM);
  thisSignal.sa_flags = 0;
  if (sigaction(SIGALRM, &thisSignal, 0) == -1) {
    WriteLog(″problem installing SIGALRM handler\n″);
    exit(2);
  }

  thisSignal.sa_handler = (void (*)() ) QuitHandler;
  sigemptyset(&thisSignal.sa_mask);
  sigaddset(&thisSignal.sa_mask, SIGQUIT);
  thisSignal.sa_flags = 0;
  if (sigaction(SIGQUIT, &thisSignal, 0) == -1) {
    WriteLog(″problem installing SIGQUIT handler\n″);
    exit(2);
  }

  WriteLog(″Monitoring %s via %s every %d sec with %d sec timeout.\n″,
           urlArg, remoteHostArg, interval, timeout);
  DoIt(interval, timeout, remoteServerHost, remoteHostArg, urlArg, nameArg);

  WriteLog(″exit()ing\n″);
  exit(0);
}
```

Figure 65. The main() Function for httprtA

The task of parsing the arguments is not shown, but nothing it does is surprising. The command format is rigid, because it is assumed that the command will be started by some automated process rather than by an interactive shell.

The nameArg argument is needed because we cannot use the URL as an Instance Vector. Instance Vector element values are limited to 32 characters (including the null terminator), and a URL could easily exceed that limit. The remoteHostArg (truncated to 31 characters if needed) and name functions are used to create the

Instance Vector. This allows many monitors to be active, or the same URL to be monitored from a number of remote locations.

The gethostbyname() function is used to get a list of network addresses for the remote system. It returns a structure that is used later in the program when a connection is made to the remote httprtB process.

If everything looks good, signal handlers are installed for SIGALRM and SIGQUIT. A message is then written to the log and the DoIt() function is called to connect to the RMAPI and send values.

DoIt() is a rather large function; it is easier to describe in parts. It could be broken up into more parts if your taste in programming style requires smaller pieces. The reason for the decision to go with a larger function is the way calls to ha_rr_...() should be paired. For example, once ha_rr_start_session() completes successfully, ha_rr_end_session() should be called before the process ends.

This leads to a nesting of control flow. While the program is running, the success of the routines is checked. If something has gone wrong, an error is logged, and the function that is needed to pair up with previously successful function calls is executed. Note that the error case is always the first case in the logic flow of this function.

Eventually, if all goes well, the RMAPI attachment is complete and the function ConnectAndGo() is called to connect to the remote httprtB process and start sending information to the RMAPI.

Let us look at DoIt() in parts, starting with Figure 66. This function has a lot of parameters. This is a trade-off. The other option would be to use global variables.

There are also a number of automatic variables in this function. These are registered with the RMAPI as the buffers for passing information to the RMAPI. These buffers are used in other parts of the program to send the data.

```
void
DoIt(int interval, int timeout, struct hostent *remoteHost,
     char *remoteHostName, char *url)
{
int rc;
int rMSocket;
struct ha_rr_val      rMVals[HTTPRT_NUM_VARS];  /* for holding the values */
struct ha_rr_variable rMVars[HTTPRT_NUM_VARS];  /* for reg/add/delete */
long values[HTTPRT_NUM_VARS];  /* location of buffers to send info */
char instanceVector[MAX_URL_LEN + 256];  /* should be big enough */
struct ha_em_err_blk error;

  rc = ha_rr_init(HTTPRT_RM_NAME, &error);
  if ( rc == HA_RR_FAIL ) {
    writeEMMsg(logFile, &error, ”ha_rr_init() failed with\n”);
    return;  /* after this, the only way out is the end of the function */
  }
  /* if we get this far, we must call ha_rr_terminate() */
```

Figure 66. The Top of the DoIt() Function for httprtA

The return statement in the error handling code after ha_rr_init() is the only one in the function. After ha_rr_init() has successfully completed, ha_rr_terminate() is called on the only path out of the function.

Once the RMAPI is initialized, a session is created and the variables are registered with the RMAPI. When the variables have been added, the program is ready to start sending information to the RMAPI.

```
  rMSocket = ha_rr_start_session(HA_RR_NOTIFY_SELECT, &error);
  if ( rMSocket < 0 ) {
    writeEMMsg(logFile, &error, ″ha_rr_start_session() failed with\n″);
  } else {
    /* if we got this far, we must call ha_rr_end_session() */

    sprintf(instanceVector, HTTPRT_IV_TEMPLATE, remoteHostName, url);
    WriteLog(″Instance Vector = ′%s′\n″, instanceVector);
    rMVars[0].rr_var_name = HTTPRT_VAR1_NAME;
    rMVars[0].rr_var_ivector = instanceVector;
    rMVars[0].rr_varu.rr_var_inst_id = HTTPRT_VAR1_ID_NO;
    rMVars[1].rr_var_name = HTTPRT_VAR2_NAME;
    rMVars[1].rr_var_ivector = instanceVector;
    rMVars[1].rr_varu.rr_var_inst_id = HTTPRT_VAR2_ID_NO;
    rc = ha_rr_reg_var(rMVars, HTTPRT_NUM_VARS, &error);
    if ( rc != HTTPRT_NUM_VARS ) {
      if ( rc == HA_RR_FAIL ) {
        writeEMMsg(logFile, &error, ″ha_rr_reg_var() failed with\n″);
      } else {
        WriteLog(″problem in ha_rr_reg_var()\n″);
        WriteLog(″var1 errno = %d, var2 errno = %d\n″,
                 rMVars[0].rr_var_errno, rMVars[1].rr_var_errno);
      }
    } else {
  /* if we get this far there is no additional to clean up so keep going */

      rMVars[0].rr_varu.rr_var_hndl = &(rMVals[0].rr_var_hndl);
      rMVars[0].rr_value = &values[0];
      rMVals[0].rr_value = &values[0];
      rMVars[1].rr_varu.rr_var_hndl = &(rMVals[1].rr_var_hndl);
      rMVars[1].rr_value = &values[1];
      rMVals[1].rr_value = &values[1];
      values[0] = 0;  /* initial value */
      values[1] = HTTPRT_ERROR_UNKNOWN;  /* initial value */
      rc = ha_rr_add_var(rMSocket, rMVars, HTTPRT_NUM_VARS, 1, &error);
      if ( rc != HTTPRT_NUM_VARS ) {
        if ( rc == HA_RR_FAIL ) {
          writeEMMsg(logFile, &error, ″ha_rr_add_var() failed with\n″);
        } else {
          WriteLog(″problem in ha_rr_add_var()\n″);
          WriteLog(″var1 errno = %d, var2 errno = %d\n″,
                   rMVars[0].rr_var_errno, rMVars[1].rr_var_errno);
        }
      } else {
        /* if we get this far we must call ha_rr_del_var() */
⋮
```

Figure 67. The Middle of the DoIt() Function for httprtA

Once the session is initialized, an ha_rr_variable structure is filled in with the information needed to register the variables. The strings used to create the Instance Vectors were checked for length restrictions in main(), so they are used here as provided. In a more general program or function, they should be checked before they are used.

The ha_rr_reg_var() function does not have to be paired with any other RMAPI function.

After ha_rr_add_var() has successfully completed, the interface to RMAPI is ready to receive values for the variables. Notice that the initial values for the variables are set by ha_rr_add_var(). It is probably worth taking some time to carefully determine these initial values, since there could be an Event Management client waiting to take action on that value.

```
        ConnectAndGo(interval, timeout, rMVals, remoteHost, url);

        rc = ha_rr_del_var(rMSocket, rMVars, HTTPRT_NUM_VARS, &error);
        if ( rc == HA_RR_FAIL ) {
          writeEMMsg(logFile, &error, "ha_rr_del_var() failed with\n");
        }
      }
      /* no need to call anything to pair with ha_rr_reg_var() */
      /* but if there is such a need, it would go here */
    }
    rc = ha_rr_end_session(rMSocket, &error);
    if ( rc == HA_RR_FAIL ) {
      writeEMMsg(logFile, &error, "ha_rr_end_session() failed with\n");
    }
  }
  rc = ha_rr_terminate(&error);
  if ( rc == HA_RR_FAIL ) {
    writeEMMsg(logFile, &error, "ha_rr_terminate() failed with\n");
  }
}
```

Figure 68. The Bottom of the DoIt() Function for httprtA

The ConnectAndGo() function connects to the remote part of the response time monitor and starts sending delay and status information to the RMAPI.

If ConnectAndGo() ever ends, all of the ha_rr_...() function pairs are called as required by the RMAPI specifications.

```
void
ConnectAndGo(int interval, int timeout, struct ha_rr_val *rMVals,
             struct hostent *remoteHost, char *url)
{
int remoteSocket;
struct sockaddr_in remoteSockAddr;
in_addr_t **hostAddresses;
int loop;    /* used to loop through things */
int matched; /* used to indicate a host address worked */
int rc;

  remoteSocket = socket(AF_INET, SOCK_STREAM, 0);
  if ( remoteSocket < 0 ) {
    WriteLog(″can not create socket for remote host\n″);
  } else {
    /* loop through list of network addresses for remote host */
    hostAddresses = (in_addr_t **) remoteHost->h_addr_list;
    for (loop = matched = 0 ; hostAddresses[loop] != 0 ; loop++ ) {
      memset((char *) &remoteSockAddr, 0, sizeof(remoteSockAddr) );
      remoteSockAddr.sin_family = AF_INET;
      remoteSockAddr.sin_len    = sizeof(remoteSockAddr);
      remoteSockAddr.sin_port   = htons(HTTPRT_PORT_NUMBER);
      memmove((char *) &remoteSockAddr.sin_addr.s_addr,
              (char *) hostAddresses[loop],
              sizeof (in_addr_t) );
      WriteLog(″trying to connect() to address 0x%08x\n″,
               (long) *hostAddresses[loop]);
      rc = connect(remoteSocket,
                   (struct sockaddr *) &remoteSockAddr,
                   sizeof(remoteSockAddr) );
      if ( rc == 0 ) {  /* connect()ed OK? */
        matched = 1;
        break;  /* found one, get out */
      } else {
        WriteLog(″failed to connect() with errno = %d\n″, errno);
      }
    }

    if ( matched ) {
      WriteLog(″connected to remote host at address 0x%08x\n″,
               (long) remoteSockAddr.sin_addr.s_addr);
      /* get in a loop to send back changes in response-time states */
      rc = Monitor(interval, timeout, remoteSocket, rMVals, url);
      WriteLog(″Monitor() returned with result of %d\n″, rc);
      shutdown(remoteSocket, 2);
      close(remoteSocket);
    } else {
      WriteLog(″could not connect() to remote httprtB process\n″);
    }
  }
}
```

Figure 69. The ConnectAndGo() Function for httprtA

The ConnectAndGo() function shown in Figure 69 first looks for a network connection to the remote part, the httprtB part, of the response time monitor. This is done by cycling through the list of addresses provided by the gethostbyname() call in main(). If a good address cannot be found, the error is logged.

If a connection can be made, the Monitor() function is called to get delay and status information and pass the values on to the RMAPI. Monitor() is not expected to end. If it does, the socket is flushed and closed with the shutdown()and close() functions.

Monitor() returns information about what actions might be appropriate. It is possible that a new socket should be created and the monitoring process should continue. In this example, the socket is closed and the function ends.

```
#define resultSIGQUIT (0)
#define resultError   (-1)
#define resultRetry   (1)
int
Monitor(int interval, int timeout, int remoteSocket,
        struct ha_rr_val * rMVals, char *url)
{
char request[MAX_URL_LEN + 100];              /* should be big enough */
char result[] = ″-2147483647 -2147483647   ″; /* big enough plus a bit */
time_t lastTime, thisTime;
int delay;
int status;
int rc;
struct ha_em_err_blk error;

  sprintf(request, ″%d %s\n″, timeout, url);/* only needs to be done once */

  for ( ; ; ) {
    alarm(timeout + 10);  /* Don′t wait forever */
    lastTime = time(0);   /* remember time so we know when to run again */

    send(remoteSocket, request, strlen(request), 0);  /* send the request */
    rc = GetALine(remoteSocket, result, sizeof(result) );  /* get results */
    alarm(0);  /* turn off alarm */
```

Figure 70. The Top of the Monitor() Function for httprtA

The Monitor() function shown in Figure 70 is a loop that only exits if some problem arises. An alarm is set so we do not wait forever if something happens to the remote httprtB process or the network. We also record the time when this sample is taken so we can later calculate when the next sample should be taken. A request is sent to the remote server and the result is received. The alarm is then turned off.

The bottom part of Monitor(), shown in Figure 71 on page 127, shows what is done with the result. The following three results can be returned by Monitor(), depending on the reason for ending:

0 SIGQUIT was received. End the program.

1 An error occurred that might be temporary. It might make sense to make a new connection to the remote server and try again.

-1 An error occurred that is not normal or expected. Retries probably will not help.

Most of Monitor() looks for these conditions and ends if one of them exists. In most cases, a value is sent to the RMAPI so that interested Event Management clients can find out what happened.

If no trouble was found, the information from httprtB is passed on to the RMAPI. After the information is sent, the time this loop has taken is computed so we know how long to sleep() before taking the next sample.

```
    if ( rc == 0 ) { /* EOF */
      WriteLog(″EOF on client socket\n″);
      SendRMData(rMVals, 0, HTTPRT_ERROR_UNKNOWN);
      return resultRetry;
    }
    if ( rc < 0 ) { /* some problem getting results */
      WriteLog(″problem on client socket\n″);
      SendRMData(rMVals, 0, HTTPRT_ERROR_UNKNOWN);
      return resultError;
    }

    rc = sscanf(result, ″%d %d\n″, &delay, &status);
    if ( rc != 2 ) {
      WriteLog(″problem parsing reply\n″);
      SendRMData(rMVals, 0, HTTPRT_ERROR_UNKNOWN);
      return resultError;
    }

    if ( timeoutFlag ) {
      WriteLog(″Timed out!\n″);
      SendRMData(rMVals, 0, HTTPRT_ERROR_HTTPRTB_CONNECT);
      return resultRetry;
    }

    if ( quitFlag ) {
      SendRMData(rMVals, 0, HTTPRT_ERROR_UNKNOWN);
      WriteLog(″Got SIGQUIT\n″);
      return resultSIGQUIT;
    }

    SendRMData(rMVals, delay, status);

    /* figure out how long to pause */
    thisTime = time(0);
    sleep( interval - (thisTime - lastTime) );/* ignore time already used */
  }
}
#undef resultSIGQUIT
#undef resultError
#undef resultRetry
```

Figure 71. The Bottom of the Monitor() Function for httprtA

The response time information is actually sent to the RMAPI with the SendRMData() function shown in Figure 72 on page 128. There is not much to do. The values are copied to the buffers that were registered in DoIt(), and ha_rr_send_val() is then used to send the information to the RMAPI.

```
void
SendRMData(struct ha_rr_val * rMVals, int delay, int status)
{
int rc;
struct ha_em_err_blk error;

  *((long *) (rMVals[0].rr_value)) = delay;
  *((long *) (rMVals[1].rr_value)) = status;

  /* 0 means this a a new value, now a refresh */
  rc = ha_rr_send_val(rMVals, HTTPRT_NUM_VARS, 0, &error);
  if ( rc == HA_RR_FAIL ) {
    writeEMMsg(logFile, &error, ″problem calling ha_rr_send_val()\n″);
  }
}
```

Figure 72. The SendRMData() Function for httprtA

Remember that this program must operate with root authority in order to use the RMAPI successfully. Also, the SDR and HAEM must be configured to accept connections from this Resource Monitor before httprtA can be used. This will be discussed in the next section.

There are a number of other small routines, macros and comments in the program code that is available. See Appendix H, “How to Get the Examples in This Book” on page 257 for more information.

4.3.2 Resource Monitor Configuration

The coding of a Resource Monitor program is only the start of what must be done to use a Resource Monitor on your system. A number of auxiliary files must be updated and the Event Management subsystem must be restarted to register your Resource Monitor before it can be used.

The items that must be updated include:

Message Catalogs

The use of message catalogs allows a program to be used in a variety of locations because the text for messages is stored away from the program. The messages are referenced by ID rather than being coded directly into the code. The Event Management subsystem supports (and requires) the use of this mechanism. Descriptions of Resource Monitors, Resource Variables and Instance Vectors are stored in message catalogs.

SDR The SDR is used to store a wide variety of RS/6000 SP configurations. Information about the Resource Monitors is stored in the SDR, but the data is not used directly from the SDR.

EMCDB The Resource Monitor information in the SDR is translated to create the Event Management Configuration Data Base, EMCDB, with the haemcfg command. This file is read by the Event Management subsystem when it is started.

haem Processes

The haemctrl command is used to stop and start the Event Management subsystem to pick up the new configuration information.

Event Management clients

Resource Monitors are only useful if an Event Management client is monitoring the data the Resource Monitors are producing. If you use Perspectives or Problem Management, you may want to change your configurations to monitor data from your Resource Monitor.

These steps are described in the publication *Event Management Programming Guide and Reference*, SC23-3996. Only an overview is presented here, with details about the configuration of httprtA.

The message catalogs allow the same program to be used in a variety of locations. The message catalog for httprtA is shown in Figure 73 on page 130.

Note that both comment lines and special directives to the message catalog, such as `quote` and `set`, start with a “$” character. Comments, however, have a space character after the “$” to avoid confusing the compiler.

If you want to define a message that contains more than one line of text, you must explicitly add “\n.” The example shows multiline messages.

Message catalogs are compiled with the `runcat` command. `runcat httprtA httprtA.cat` is run on the file httprt.msg, creating the file httprt.cat. This file is then copied to the C, en_US, and En_US directories of /usr/lib/nls/msg. It is possible to make links rather than copy the same file to multiple locations.

The Makefile file for the Event Management subsystem examples in this book has a target called `install.cat`, which performs these operations for you.

```
$ $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$
$ httprt.msg -- message for httprt Event Monitor
$
$  (C) Copyright International Business Machines Corporation 1997.
$  All rights reserved
$
$  Written by Paul G. Crumley, IBM Research
$  pgc@watson.ibm.com
$
$ You must run the command ″runcat httprt httprt.msg″ on the control
$ workstation then copy httprt.cat to the NLS directory for your
$ language.
$
$ $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$
$quote ″
$set 1
$
1 ″WWW HTTP Response Time Monitor″
2 ″httprt Resource Monitor″
3 ″Delay time (in seconds) of HTTP responses at remote measuring point″
4 ″Status (<0 monitor status, >0 HTTP status) of responses″
5 ″Place Holder″
6 ″The value of the Instance Variable named ′Observer′ indicates the \n\
remote location of the httprtB process that will fetch the HTTP \n\
objects and report on what happens.″
7 ″The value of the Instance Variable named ′Name′ is assigned on the \n\
httprtA command line and is used because Instance Vectors are limited \n\
to 31 characters so the URL can not be used directly.″
8 ″The value of the Instance Variable named ′NodeNum′ is needed so the \n\
EM client can find the RM.  This should not be needed but the current \n\
code requires an rvLocator.  This is set to the SP node number \n\
automatically by the RMAPI code.″
$
$ $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$                          End of httprt.msg file
$ $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Figure 73. The Message Catalog File for httprtA

Once the message catalogs are in place, you can update the SDR with information about your Resource Monitor. This can be done in two ways. The first is to construct a file in the format that the `haemloadcfg` command expects and then use that command to update the SDR. This is convenient for distributing a stable Resource Monitor, but since the `haemloadcfg` function will not overwrite existing objects in some cases, you have to remove any altered objects from the SDR before running `haemloadcfg`.

The second approach is to create a script to execute `SDRCreateObjects` for the needed configuration information. It is probably a good idea to create a similar script to remove the configuration information from the SDR. The sample Resource Monitor uses this approach. Figure 74 on page 131 shows part of the script that installs SDR configuration information.

```
#!/bin/ksh
#########################################################################
#
#  httprt.add.sdr -- load needed config info into SDR
#
#
#
#########################################################################

/usr/lpp/ssp/bin/SDRCreateObjects \
        EM_Resource_Monitor \
        rmName=WWW.HTTP.ResponseTimeMon \
        rmMessage_file=httprt.cat \
        rmMessage_set=1 \
        rmConnect_type=client \
        rmPTX_prefix=dummy \
        rmPTX_description=5 \
        rmPTX_asnno=0

/usr/lpp/ssp/bin/SDRCreateObjects \
        EM_Resource_Class \
        rcClass=WWW.HTTP.ResponseTimeClass \
        rcResource_monitor=WWW.HTTP.ResponseTimeMon \
        rcObservation_interval=0 \
        rcReporting_interval=0

/usr/lpp/ssp/bin/SDRCreateObjects \
        EM_Resource_Variable \
        rvName=WWW.HTTP.ResponseTimeMon.delay \
        rvDescription=3 \
        rvValue_type=State \
        rvData_type=long \
        rvClass=WWW.HTTP.ResponseTimeClass \
        rvDynamic_instance=0 \
        rvLocator=NodeNum \
        rvInitial_value=0
```

Figure 74. Part of the SDR Configuration Script for httprtA

After the SDR is updated, you must rebuild the EMCDB and use the haemctrl command to stop and start the Event Management subsystem daemons so they will start using the new information.

The install.sdr target in the sample Makefile file removes the old SDR entries, adds the new ones, rebuilds the EMCDB, and then prints a message to remind you to restart the Event Management subsystem processes.

After all the configuration information is refreshed, use the lsemv command from 4.2.2, "The lsemv Program" on page 101 to make sure that the current Event Management subsystem knows about the new Resource Monitor and its variables.

You can verify that the configuration process was successful by using lsemv as shown in Figure 75 on page 132. In this example we see the current information for the WWW.HTTP.ResponseTimeMon.delay variable. The lsemv command gets the information by querying the active Event Management subsystem, rather than by retrieving it from the SDR. You can be more confident, therefore, that the configuration information is properly installed and ready for use.

```
pgc@sp21en0 </u/pgc/itso>:./lsemv -v WWW.HTTP.ResponseTimeMon.delay
======================
Resource Variable Name: ″WWW.HTTP.ResponseTimeMon.delay″
Variable Value Type:    State
Variable Data  Type:    long
Variable SBS Format:    ″″
Variable Initial Value: ″0″
Variable Class:         ″WWW.HTTP.ResponseTimeClass″
Instance Vector:        ″Observer=cstring;Name=cstring;NodeNum=int″
PTX Name:               ″″
Default Predicate:      ″″
Locator:                ″NodeNum″
--------------------
Variable Description
--------------------
Delay time (in seconds) of HTTP responses at remote measuring point
---------------------------
Instance Vector Description
---------------------------
The value of the Instance Variable named ′Observer′ indicates the
remote location of the httprtB
process that will fetch the HTTP
objects and report on what happens.
The value of the Instance Variable named ′Name′ is assigned on the
httprtA command line and is used
because Instance Vectors are limited
to 31 characters so the URL can not be used directly.
The value of the Instance Variable named ′NodeNum′ is needed so the
EM client can find the RM.  This should not be needed, but the current
code requires an rvLocator.  This is set to the SP node number
automatically by the RMAPI code.
-----------------
Event Description
-----------------

pgc@sp21en0 </u/pgc/itso>:
```

Figure 75. Output for WWW.HTTP.ResponseTimeMon.delay from the lsemv Command

You can start the httprtB process on some remote (or local) machine, then start the httprtA process to monitor a URL and report on its performance. The getemv and monemv commands could be used to see whether the Resource Monitor is working before you reconfigure other Event Management clients.

4.3.3 RMAPI Summary

The httprtA monitor is a simple example of a Resource Monitor. The only complication is the connection to the remote response time server. As in the case of the EMAPI examples, we see that only a small amount of code is needed to collect information for the Event Management subsystem.

The httprtA monitor is a command-based Resource Monitor. The other types of Resource Monitors use similar data structures, but the flow of control is different. Daemon-based Resource Monitors are started by the Event Management subsystem when needed. Daemon-based and integrated Resource Monitors receive control directions that tell them when to take various actions.

The Resource Monitor programs must exercise a bit more care than Event Management programs because the Resource Monitor programs must run with root authority. As with Event Management programs, care must be taken to use Event Management subsystem resources.

As noted in 4.2.5, “EMAPI Summary” on page 119, not all RMAPI features are needed to create a useful Resource Monitor. With little effort, you should be able to adapt existing servers or create new Resource Monitors to collect important information about your system.

4.4 Corrections and Clarifications

These examples were developed on early versions of the Event Management subsystem and tested on the released software. Some details did not make it into the early versions of documentation, and early code had some limitations. Many of the items listed here might be fixed by newer versions of the Event Management subsystem and the associated documentation.

This section contains information that was received from the developers to assist in creating and debugging these Event Management examples, and lists corrections to information in the publications.

- The `ha_em_receive_response()` function must be called as soon as it has information that is available to receive. If an EMAPI client defers calling `ha_em_receive_response()`, the Event Management subsystem must hold the information for the client in Event Management subsystem buffers. This can cause performance problems for the entire RS/6000 SP system.
- The system limit on open sessions is about 230 per partition. The EMAPI documentation indicates that EMAPI clients can open multiple sessions to make it easier to process information. Do not take the use of multiple sessions to an extreme.
- SBS data structures do not preserve alignment of various data types, such as long and float. You might wish to use byte-oriented moves to get the data to aligned areas before you process the information.
- Connections between the Event Management subsystem and remote clients, for example clients that execute on RS/6000 workstations or clients executing on SP nodes that are connecting to an Event Management daemon on the Control Workstation, are as reliable as the underlying TCP/IP protocol being used. That is to say, neither side of the connection can immediately detect the loss of the other side due to network problems or failure of the other node/workstation, unless an attempt is made to send a message. If one side of the connection is only waiting for messages, such failures are detected only as TCP “keep alive” messages are sent. The “keep alive” protocol typically sends such messages every two hours. However, failure of a client process or the Event Management daemon is detected by the other side of a remote connection immediately.
- Structure Byte Strings may not be more that 1024 bytes in length.

  If a PTX name is defined for a Resource Variable, the total length of the PTX name, as it is formed by the Event Management Subsystem, must conform to the following limits (refer to the Event Management Configuration Data Reference chapter in the *RS/6000 SP: Event Management Programming Guide and Reference*, SC23-3996, for PTX name formation):

  - The length of the name, consisting of all components but the last, must not exceed 63 bytes.
  - The length of the last component of the name must not exceed 31 bytes.

  The length of any possible instance vector element value that is substituted into the PTX name must be used in determining this length.

- Remote clients cannot establish a connection with the Event Management subsystem if the SDR is not operational. This is also true for any Event Management clients executing on the Control Workstation.
- Some information that is returned through the EMAPI and RMAPI is read from the SDR. Although there is a mechanism that enforces consistency between the EMCDB and the running Event Management daemons, there is no mechanism to guarantee that message catalogs contain correct data.

  If it is necessary to change the message numbers or set numbers in message catalogs used by the Event Management subsystem, then before making such changes the entire Event Management subsystem should be shut down. After the subsystem is shut down, the changes can be made to the catalogs, the catalogs distributed throughout the SP and the CWS, the SDR updated, and the haemcfg command run, and then the Event Management subsystem can be restarted. Adding messages to the end of existing message sets, adding new sets, or adding new catalogs can be done at any time before updating the SDR and executing the haemcfg command. In these “add” cases, the Event Management subsystem can be shut down and restarted any time after executing the haemcfg command.
- If the length fields of a Structured Byte String are not internally consistent, then reference to such an SBS from within a predicate will have undefined results upon evaluation of the predicate. Other errors in the predicate, for example bad field types, are detected during evaluation and result in the generation of an event with an error.
- The SDR commands given as examples of how to load Resource Monitor information in Chapter 2 of *Event Management Programming Guide and Reference*, SC23-3996, are not complete. For example, the haemcfg command will complain if you do not include the rmPTX_prefix, rmPTX_description and rmPTX_asnno on EM_Resource_Monitor objects. rvDynamic_instance and rvInitial_value are needed for EM_Resource_Variable objects.

Part 3. Tools

Chapter 5. SP Perspectives GUI

Scalable POWERparallel (SP) Perspectives provides a highly interactive common graphical user interface (GUI) through which many system management and system monitoring tasks, as well as SP-specific user interactions, can be performed. System objects such as frames, nodes, switches, events, node groups, and so on, are represented by icons. You can simply click the icon with a mouse and select an action to perform on that object from the menu bar or tool bar.

This chapter briefly introduces the Perspectives GUI and then focuses on the Event Perspective. Specific scenarios of practical use are presented, followed by a step-by-step description of the way the Event Perspective may be used to solve the problem of interest. In addition, a variety of figures and tables are presented, as well as some tips and hints that are helpful in working with the Event Management subsystem and the Event Perspective. Note that the Event Management and the Event Perspective are completely new concepts in the PSSP 2.2 software product.

5.1 Introduction

The Perspectives GUI is initially intended for the SP platform, but eventually it will be implemented using a system management framework that will allow integration with other platforms, making the SP just one of the many machines being administered and monitored on a network.

5.1.1 What Tasks Can Perspectives Perform?

The present release of Perspectives provides functions for the following categories of system management tasks. It is also possible to make your own favorite applications accessible from the Perspectives GUI.

- Monitoring and controlling hardware
- Managing event definitions and events
- Managing IBM virtual shared disks (VSDs) and data striping devices (HSDs)
- Generating and saving system partition configurations
- Setting up performance monitoring
- Accessing SP-related SMIT panels:
  - smit config_data on the CW
  - smit SP_verify on the CW
  - smit cluster_mgmt
  - smit splogmgmt
  - smit spusers on the CW
- Accessing the following Visual System Management (VSM) panels:
  - vsm device
  - vsm install
  - vsm maintain
  - vsm print
  - vsm storage
  - vsm user

5.1.2 How Is Perspectives Invoked?

This section describes how to invoke the Perspectives GUI.

Executing the following command brings up the Launch Pad of SP Perspectives (see Figure 76):

```
# /usr/lpp/ssp/bin/perspectives &
```

The Launch Pad is a small, customable window from which you can launch (start) executables associated with monitoring, controlling, and managing your IBM RS/6000 SP. System administrators can also customize the Launch Pad, so their own applications appear in addition to the one provided by default. For more details on how to customize the Launch Pad, refer to *RS/6000 SP: Administration Guide*, GC23-3897.

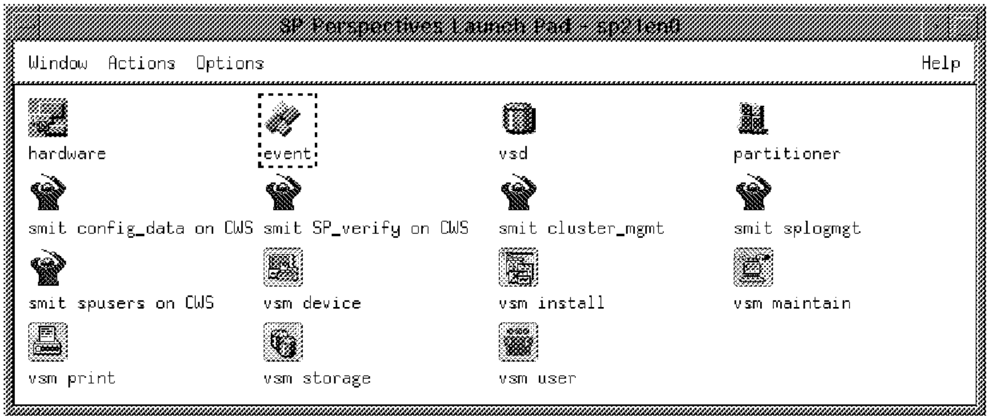


Figure 76. The Perspectives GUI Launch Pad

The following icons on the Launch Pad represent Perspectives applications: hardware, event, vsd, and partitioner. In order to invoke any application from the Launch Pad, double-click its icon. You can also invoke each Perspective without using the Launch Pad by executing its corresponding command. For example, the command for Event Perspective is spevent and that of Hardware Perspective is sphardware, assuming that the directory /usr/lpp/ssp/bin is included in your PATH. You can get the details of each Perspective by going to the Options pull-down menu of the Launch Pad and selecting the **Options→Details View** options.

Inspection shows that the file /usr/lpp/ssp/bin/perspectives is a symbolic link to /usr/lpp/ssp/bin/.startup_script with permission 555:

```
# cd /usr/lpp/ssp/bin
# ls -l pers* spev* sphar*
lrwxrwxrwx   1 bin      bin        32 Sep 12 10:34 perspectives →
/usr/lpp/ssp/bin/.startup_script
lrwxrwxrwx   1 bin      bin        32 Sep 12 10:34 spevent →/usr/
lpp/ssp/bin/.startup_script
lrwxrwxrwx   1 bin      bin        32 Sep 12 10:34 sphardware →/u
sr/lpp/ssp/bin/.startup_script
# ls -l .sta*
-r-xr-xr-x   1 bin      bin      4570 Sep 10 07:08 .startup_script
#
```

Therefore, any user may start the Launch Pad or start any individual Perspective such as the Event Perspective. Many actions, however, require root authorization, or Kerberos authorization, or both (see also 5.2.3.4, “Summary of Authorizations Needed” on page 156).

The .startup_script script invokes the Perspectives binaries. This script determines the language to be used and sets up the appropriate environment variables. It then executes the binary called by the user.

5.2 Using the Event Perspective

The Event Perspective allows you to define and manage *event definitions* within a system partition. An event definition allows you to specify under what condition the event occurs and what actions to take in response.

What is an event? An event occurs when the expression within the condition evaluates as true as the result of a change in the state of a resource within the system. A resource is an entity in the system that is observed. Examples of resources include hardware entities such as processors, disk drives, memory, and adapters, as well as software entities such as database applications, processes, and file systems. Each resource in the system has one or more attributes that define the state of the resource. The number of attributes and the semantics of each attribute are defined by the resource.

Note

Terms such as event definition, expression, condition, and others are discussed further in 5.2.3.1, “Some Terminology Considerations” on page 152 in this chapter.

Specifically, with the Event Perspective you can:

- Create an event definition
- Register or unregister an event definition
- View or modify an existing event definition
- Delete an event definition
- Create a new condition

In the following sections, we present examples using different scenarios and provide details of how you can use the Event Perspective to address the problems in each scenario.

5.2.1 Example 1: Monitoring Memory Usage

Important

Go through Examples 1 and 2 before you use the Event Perspective to create an event definition for the first time.

To create an event definition, you need a condition as one of the ingredients. Example 1 uses a predefined condition, while Example 2 creates the needed condition along the way.

These examples give step-by-step descriptions of how to create event definitions. At every step, all the different options and their implications are discussed, and helpful tips and hints are given.

Suppose you want the system to automatically take some actions and notify you whenever the real memory on any of the nodes in the sp21en0 partition is over 85% utilized. You need to create an event definition for this scenario, hence you invoke the Event Perspective by using the command:

```
# spevent &
```

The Event Perspective main window appears (Figure 77).

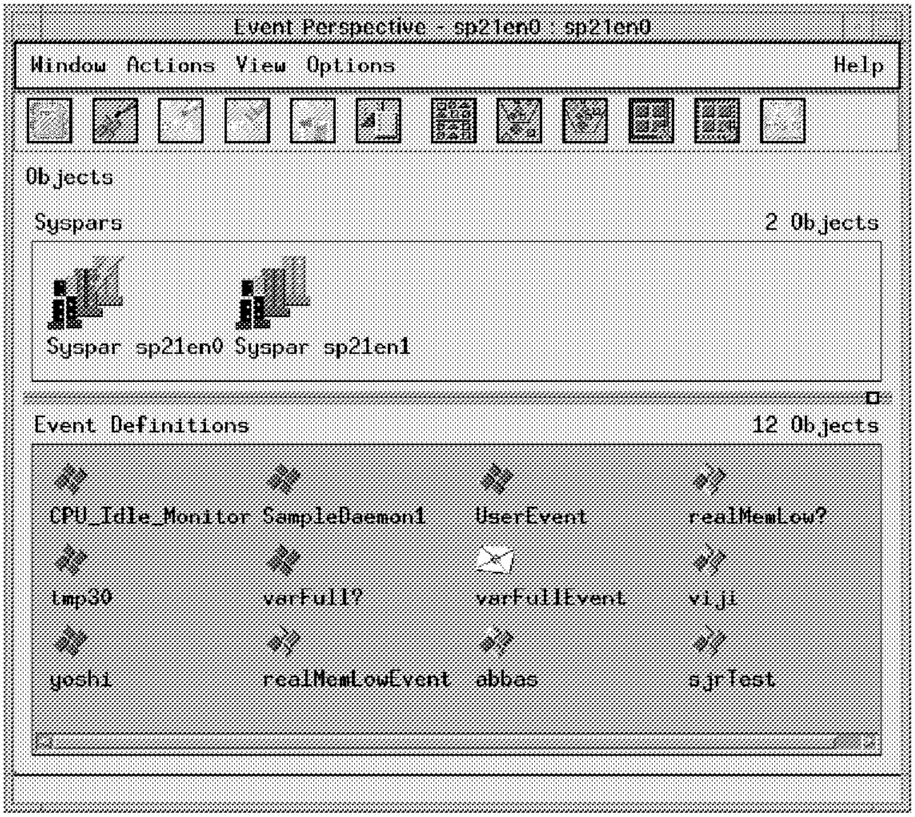


Figure 77. The Event Perspective Primary Window

Note that the title bar in Figure 77 displays the name of the application (Event Perspective), followed by a hyphen, followed by the name of the Control

Workstation (sp21en0) running the application, followed by a colon, followed by the name of the system partition (sp21en0).

The Event Perspective creates event definitions within a system partition and within a Kerberos principal. These event definitions are available only to users using that Kerberos principal; other Kerberos principals cannot subscribe to, unsubscribe to, modify, or delete them.

Before you create an event definition, you first need to make sure that the system partition to which you want it to belong is the currently active partition. This partition is identified by a yellow lightning bolt over its icon (see the Syspar sp21en0 icon in Figure 77 on page 140). If the desired system partition is not the currently active one, then:

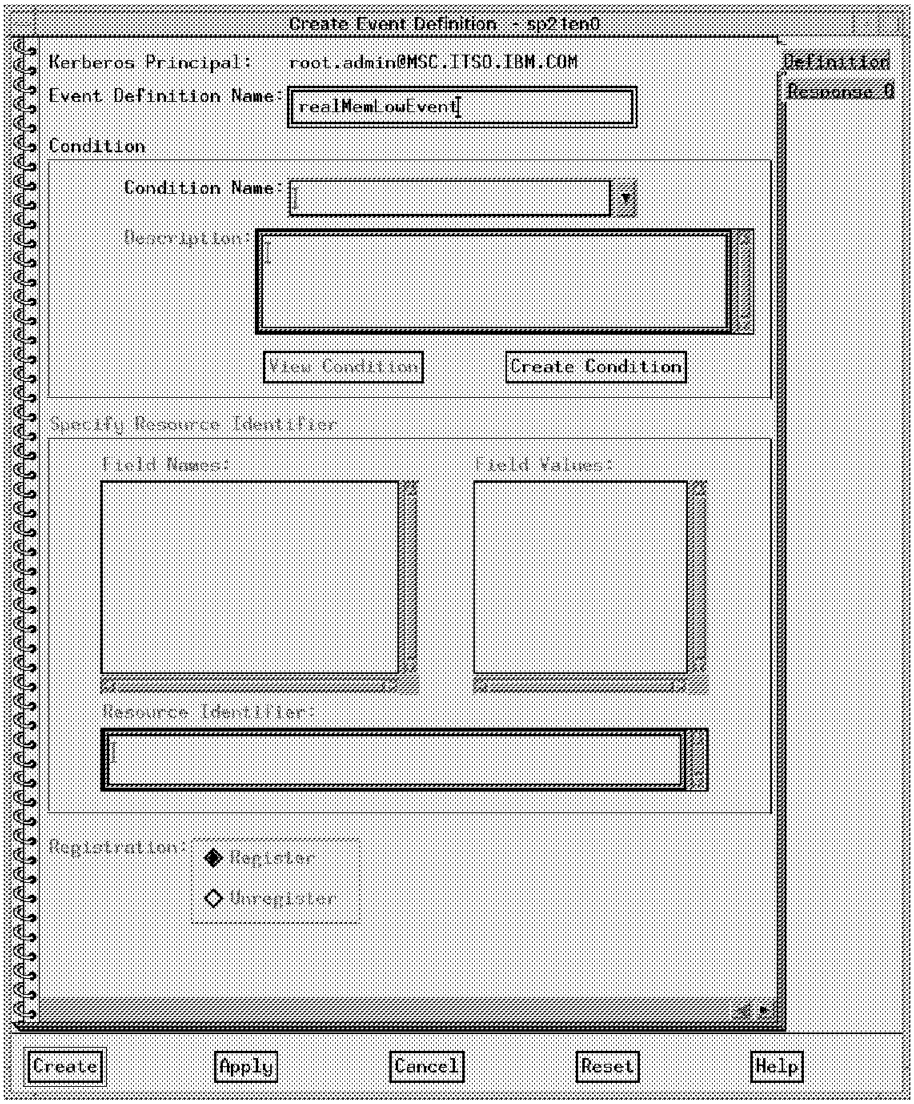
1. Click the system partition′s icon on the Syspars pane. It appears highlighted to show that you have selected it.
2. Go to the Actions pull-down menu and select the **Syspars**→ **Set Current Partition** options. The name of the new system partition now appears at the top of the Event Perspective window and the associated event definitions (if there are any) appear in the Event Definitions pane.

Now that the desired partition is the currently active one, you can proceed to create the event definition in this partition by doing the following:

Step 1 Open the Event Definition Notebook.

Click the **Event Definitions** pane once so the pane receives focus. Go to the Actions pull-down menu and choose the **Event Definitions**→**Create** options, or click the **Create Event Definition** icon on the tool bar instead.

The Create Event Definition notebook opens at the first page, which is the Definition page. You must perform the following tasks in the order shown.



Kerberos Principal: root.admin@MSC.ITSO.IBM.COM

Event Definition Name: realMemLowEvent

Registration: ◆ Register ◇ Unregister

Figure 78. The Definition Page of the Create Event Definition Notebook

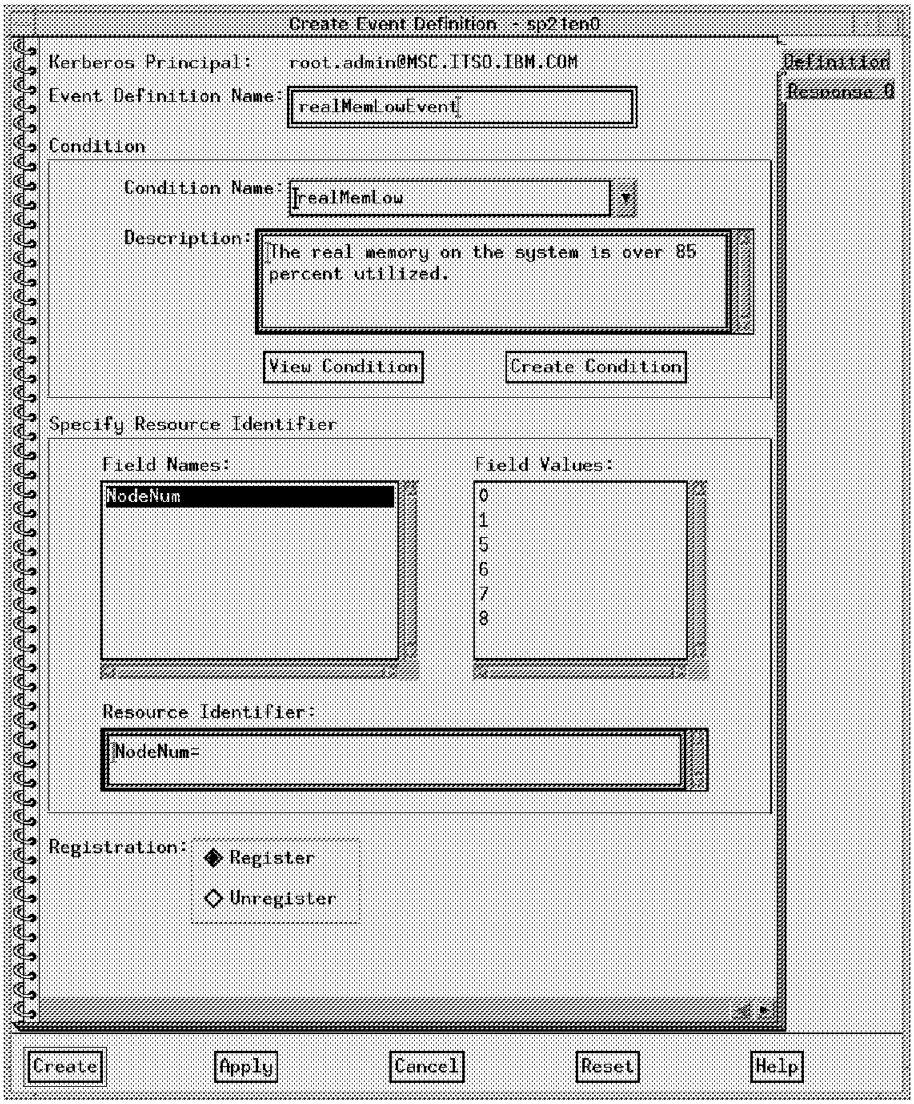
Step 2 Enter an Event Definition Name.

Enter a name of your choice in the Event Definition Name box. For example, choose realMemLowEvent (Figure 78).

The name you specify cannot include a period (.) or colon (:) and it must be unique within the system partition in which it is created. Notice also that the Kerberos Principal that is connected with this event definition is displayed at the top of the Definition page. In this case, it is listed as root.admin@MSC.ITSO.IBM.COM.

Step 3 Select a predefined condition.

Select an appropriate predefined condition using the selection box provided. Note that the conditions are listed in alphabetical order, with the uppercase letters listed before the lowercase letters. In the example, choose the condition realMemLow (see Figure 79 on page 143) based on its description, The real memory on the system is over 85 percent utilized, which is displayed in the Description box of the Definition page (see Table 4 on page 170).



Create Event Definition - sp21en0

Kerberos Principal: root.admin@MSC.ITSO.IBM.COM

Event Definition Name: realMemLowEvent

Condition

Condition Name: realMemLow

Description: The real memory on the system is over 85 percent utilized.

Specify Resource Identifier

Field Names: NodeNum

Field Values: 0 1 5 6 7 8

Resource Identifier: NodeNum=

Registration: ◆ Register ◇ Unregister

Figure 79. The Definition Page after Selecting a Predefined Condition

These predefined conditions are saved in the SDR. From outside the Event Perspective, you can display them by executing the following command:

```
# SDRGetObjects EM_Condition
```

See Table 4 on page 170 for the predefined conditions that are shipped with the PSSP 2.2 software product.

As soon as you choose the predefined condition, the remaining fields of this dialog box will automatically be updated with the information that was originally defined for the condition.

Note

At this stage, you may view the details of the condition you select by clicking **View Condition**.

In Example 1, you are choosing a predefined condition without changing it. If you decide, however, to make some changes to the condition, then you need to click **Cancel** on the View Condition window. This brings you back to the Definition page. Click **Create Condition**. Make all your changes in this dialog box. Then click **OK** to save the information, close the dialog box, and return to the Definition page of the Create Event Definition notebook.

Be aware that, in this situation, you are effectively creating a new condition. Therefore, make sure you give your new condition a different name than the predefined condition. Otherwise, you get an error message like `The provided Condition Name` *Your_Condition_Name* `already exists in the database. Duplicate is not allowed.` The Condition Name is a required field.

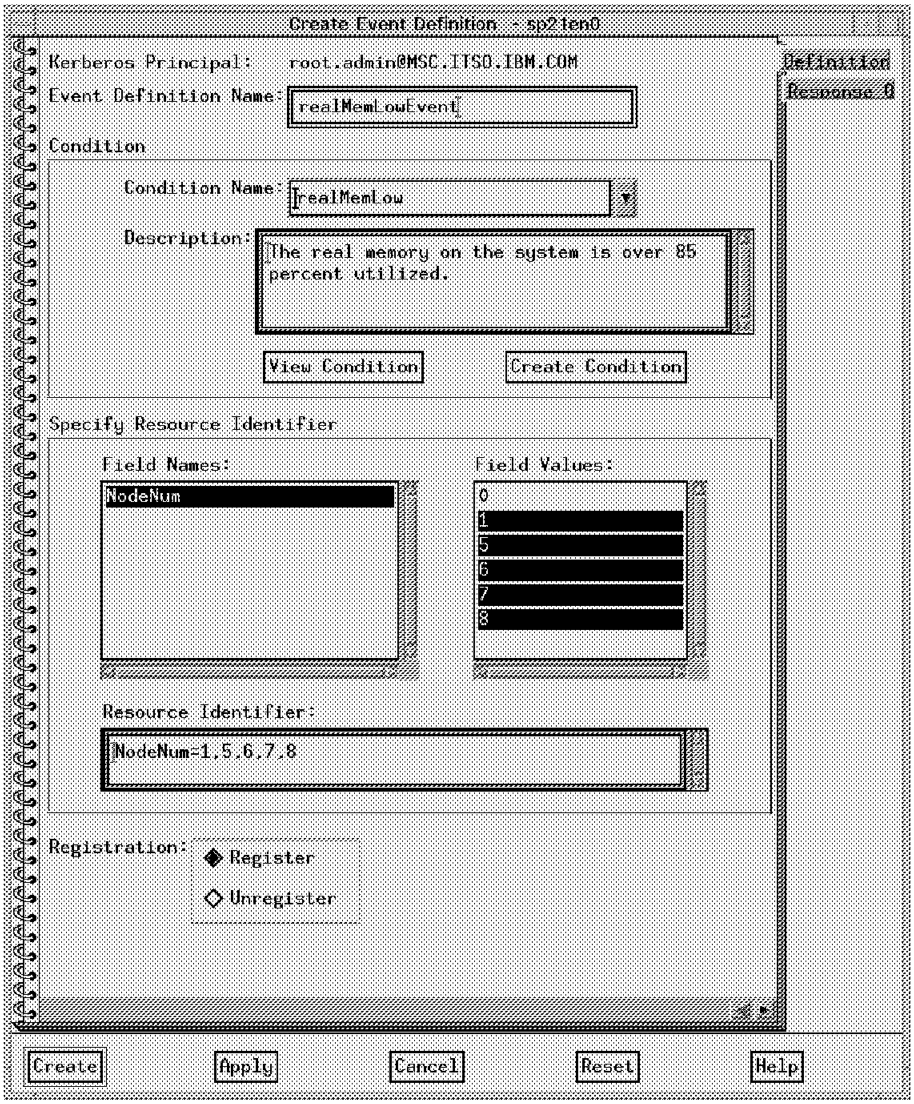
Step 4 Select resources.

Now you need to uniquely identify the resources for your event definition. In our scenario, you need to decide the real memory of which nodes you would like to monitor. To select the specific resource, select a field name from the Field Names box by clicking it, and then select one or more values in the Field Values box by clicking those values. Note that for selecting more than one item from the Field Values box simultaneously, you need to hold the Ctrl button down while clicking the desired items.

Repeat these steps for each field name that appears in the Field Names box. The result of your selection is displayed in the view-only Resource Identifier box. In the present scenario, select field values 1, 5, 6, 7, and 8 for NodeNum, the only field name available in your `realMemLowEvent` event definition. See Figure 80 on page 145.

Step 5 Select Register/Unregister.

Next you need to specify whether you want to subscribe to the event. You control whether or not you register for the event definition by using the Register and Unregister buttons of the Definition page of the Create Event Definition notebook. By registering an event definition, you indicate that you want that event to become active. This means that the expression that is included as part of the event definition is applied to the instances of the Resource Variable, which is also part of the event definition. If the expression is found to be true, an event is generated and the system notifies you or takes actions based on the way the event is defined (see Step 6). In our scenario, choose the **Register** option, which is the default, since we are planning to test our event definition creation later. At this stage, your Create Event Definition page should look like Figure 80 on page 145.



Create Event Definition - sp21en0

Kerberos Principal: root.admin@MSC.ITSO.IBM.COM

Event Definition Name: realMemLowEvent

Condition

Condition Name: realMemLow

Description: The real memory on the system is over 85 percent utilized.

Specify Resource Identifier

Field Names: NodeNum

Field Values: 0, 1, 5, 6, 7, 8

Resource Identifier: NodeNum=1,5,6,7,8

Registration: ◆ Register ◇ Unregister

Figure 80. The Completed Definition Page in Example 1

Now you need to specify the type of response you want for your event definition.

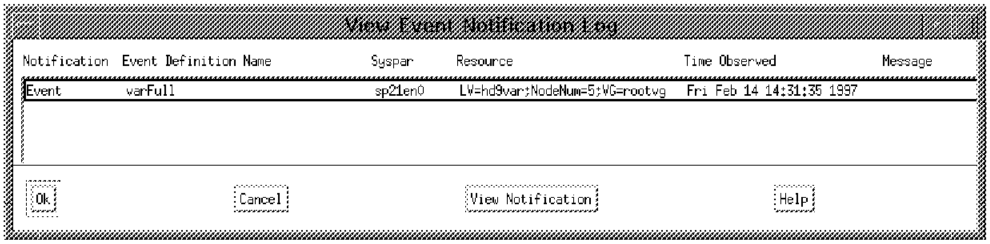
Step 6 Select Response Options.

Click the **Response Options** tab along the right-hand edge of the Create Event Definition notebook. The Response Options page (Figure 82 on page 148) appears. Select the type of response you would like for the event definition by clicking the appropriate response buttons. You can select one or more of the following responses for each event definition you create:

1. Get Notified During the Event Perspective Session (default)
2. Take Actions When the Event Occurs
3. Take Actions When the Rearm-Event Occurs

When an event occurs, the system responds by placing an entry in the Event Notification Log (Figure 81 on page 146) and, if you choose, it carries out commands that you supply.

If you select only the **Get Notified During the Event Perspective Session (default)** option, then there is no need to enter any more information on the Response Options page. In addition, when the event occurs and an entry appears in the Event Notification Log, the event definition′s icon, in the Event Definitions pane, changes to look like an envelope (see the **varFullEvent** icon in the Event Definition pane of Figure 77 on page 140). The pop-up View Event Notification Log window (Figure 81) also appears and shows you what event has occurred.



Notification	Event Definition Name	Syspar	Resource	Time Observed	Message
Event	varFull	sp21en0	LV=hd9var;NodeNum=5;VG=rootvg	Fri Feb 14 14:31:35 1997	

Figure 81. The View Event Notification Log Window

The act of notification in this option occurs during the Event Perspective session only. In other words, the notification stops when the user exits the Event Perspective session but starts again when the user runs the Event Perspective. Note that this type of event is not stored in the SDR, but in the Home directory of the user as `$HOME/.$USER:Events`. See 5.2.3.2, “Where Is My Event Definition Stored?” on page 154 for more details.

If you choose **Take Actions When the Event Occurs**, then you will have several options with this response. You may specify one, two, or all three of the following types of actions:

- Command
- SNMP Traps
- PSSP Log

The Command option lets you supply one or more system commands you would normally issue from the command line. The system automatically executes the commands when the event occurs.

The SNMP Traps option allows you to specify that you want an SNMP trap to be generated when an error log entry is made. You need to specify the Trap ID with this option.

If you select the PSSP Log option, an entry will be added to the PSSP log when an event occurs. You must, however, specify the text to be placed in the PSSP log. Similar considerations are appropriate if you choose the option **Take Actions When the Rearm-Event Occurs**.

In our scenario, select all three options. Choose the command, for example, to be:

```
wall The real memory is now over 85% utilized!
```

and that of the rearm-command to be:

```
wall The real memory is now less than 85% utilized!
```

Note

An event occurs when the expression evaluates to true. A rearm-event occurs when the rearm expression evaluates to true.

A command is the response action to an event and a rearm-command is the response action to a rearm-event.

A rearm expression is an expression used to generate an event that alternates with an original event expression in the following way: the event expression is used until it is true, then the rearm expression is used until it is true, then the event expression is used, and so on.

If you do not use a rearm expression, then the expression will evaluate true after every `Observation_interval` seconds (Figure 84 on page 150) before it evaluates false. For instance, in Example 1 the command:

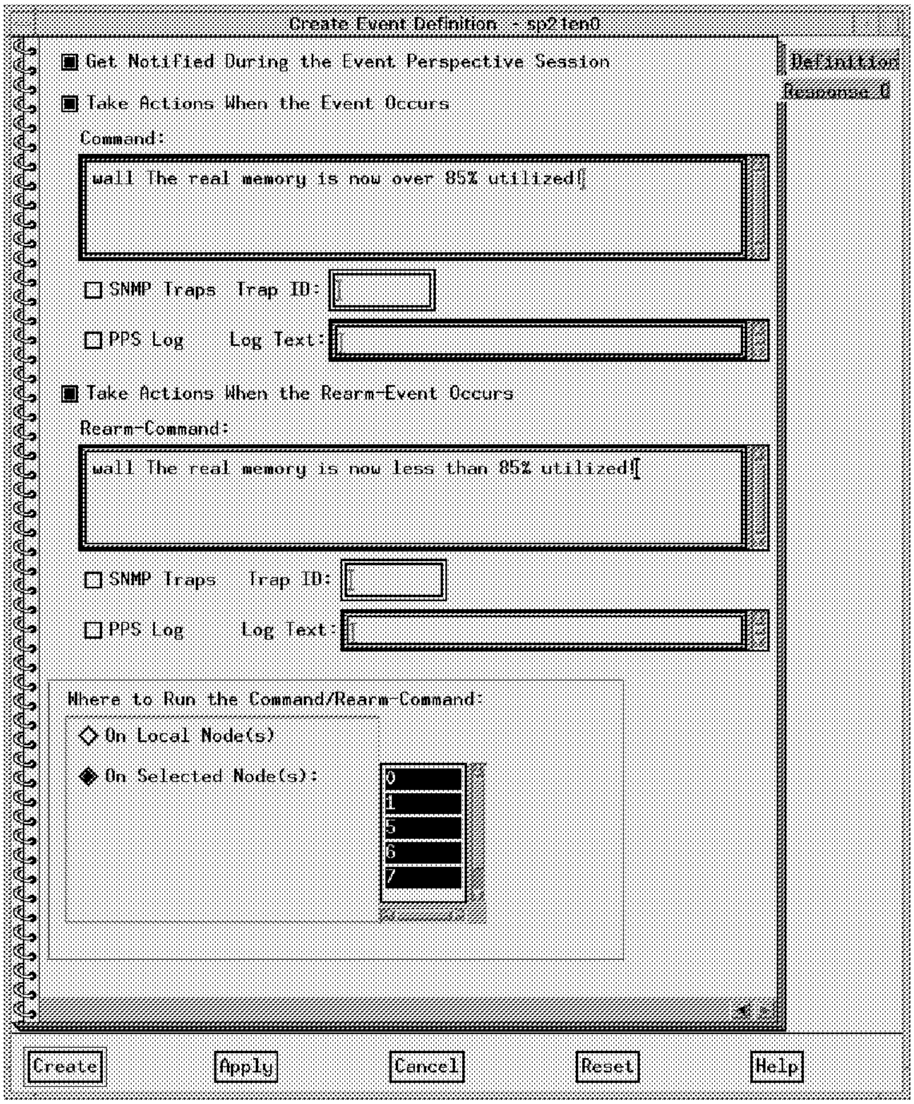
```
wall The real memory is now over 85% utilized
```

would run after every 60 seconds until the expression evaluates false, thereby indicating that real memory is not over 85% utilized any more.

The rearm expression is commonly the inverse of the event expression (Example 1). It can also be used with the event expression to define an upper and lower boundary for a condition of interest (Example 2).

Step 7 Select target nodes.

The last step in creating an event definition is to specify where the command or rearm-command should be run. You can choose to run it On Local Node(s) or On Selected Node(s). A local node is the node where the event occurs and was specified with a field name as part of the resource identifier for the event. In our example, click **On Selected Node(s)** and choose all the nodes (nodes 1, 5, 6, 7, and 8) and the Control Workstation (node 0) (see Figure 82 on page 148).



■ Get Notified During the Event Perspective Session

■ Take Actions When the Event Occurs

Command:

```
wall The real memory is now over 85% utilized!
```

□ SNMP Traps Trap ID:

□ PPS Log Log Text:

■ Take Actions When the Rearm-Event Occurs

Rearm-Command:

```
wall The real memory is now less than 85% utilized!
```

□ SNMP Traps Trap ID:

□ PPS Log Log Text:

Where to Run the Command/Rearm-Command:

◇ On Local Node(s)

◆ On Selected Node(s): 0, 1, 5, 6, 7

Figure 82. The Response Options Page

Step 8 Save and close.

To save the information, click **OK** or **Create** on any of the Create Event Definition notebook pages and close the notebook.

5.2.2 Example 2: Monitoring File System Size

Suppose you would like to be notified automatically whenever a file system (for example, /var) becomes more than a percentage (such as 90%) full on the Control Workstation and Node 5 of the sp21en0 partition.

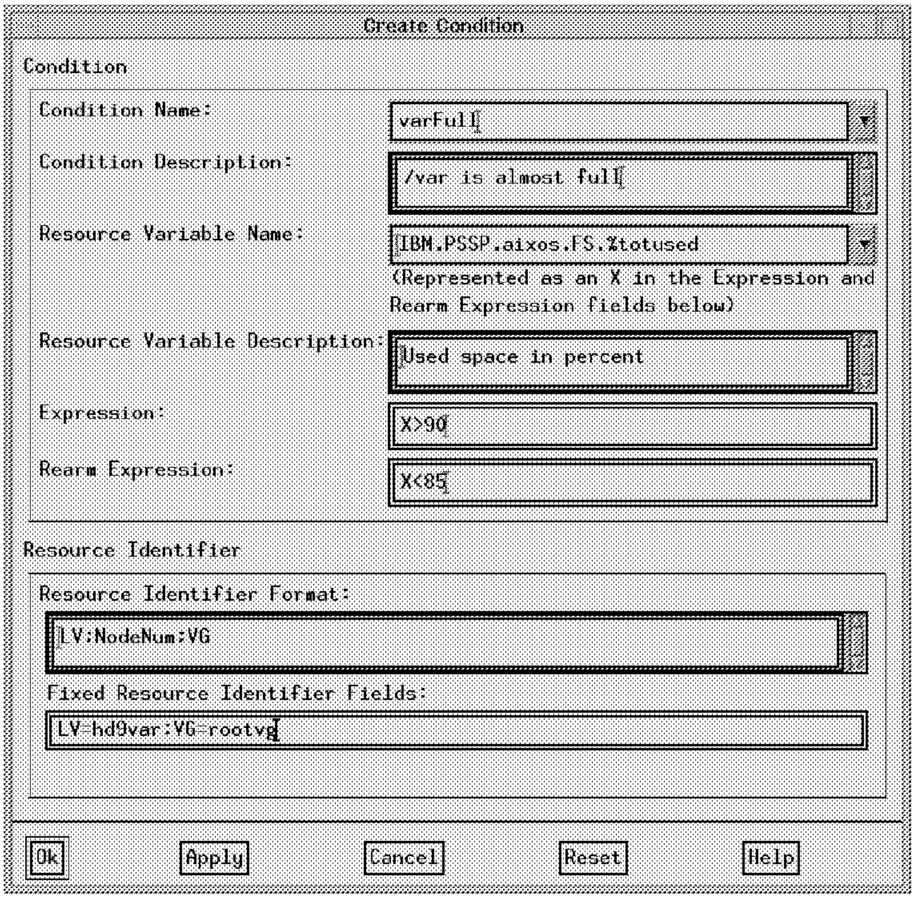
To do this, make sure that sp21en0 is the current partition. Follow the procedure described in 5.2.1, “Example 1: Monitoring Memory Usage” on page 140, except now you need to create a new condition as well (Step 3).

Step 1 Click the **Create Event Definition** icon.

Step 2 Make up a unique name (for example `varFullEvent`) and enter it in the Event Definition Name box. Unlike Example 1, there is no predefined condition that can help in this scenario. You have to create a new condition.

Step 3 Create a new condition.

1. Click the **Create Condition** button, which brings you to its corresponding dialog box (see Figure 83). You need to supply information in the fields of this box, as follows:



Create Condition

Condition

Condition Name: varFull

Condition Description: /var is almost full

Resource Variable Name: IBM.PSSP.aixos.FS.%totused

(Represented as an X in the Expression and Rearm Expression fields below)

Resource Variable Description: Used space in percent

Expression: X>90

Rearm Expression: X<85

Resource Identifier

Resource Identifier Format: LV:NodeNum:VG

Fixed Resource Identifier Fields: LV=hd9var:VG=rootvg

Figure 83. The Create Condition Dialog Box

2. Enter a name (for example, `varFull`) for the new condition in the Condition Name box, and a short description of the condition (for example, `/var is almost full`) in the Condition Description field. Both of these are required fields.

   Note: The name that you enter may not be the name of a predefined condition.

3. Select a Resource Variable by using the selection box of the Resource Variable Name field. This is a required field. Presently, there are more than 360 Resource Variables shipped with the IBM PSSP 2.2 software product.

```
# SDRGetObjects EM_Resource_Variable | wc -l
     367
#
```

When you select a Resource Variable, its description automatically appears in the Resource Variable Description box. However, since there are so many Resource Variables, it would not be a pleasant task to go through all of them one by one, read their descriptions, and then decide which one is right for your situation. Fortunately, all Resource Variables presently available in PSSP 2.2 belong to 17 distinct classes.

This information can be displayed by executing the following command:

```
# SDRGetObjects EM_Resource_Class
```

The output of this command is shown in Figure 84.

```
# SDRGetObjects EM_Resource_Class
rcClass             Resource_monitor Observation_interval Reporting_int.
IBM.PSSP.CSS        IBM.PSSP.harmld        5                 5
IBM.PSSP.HARMLD     IBM.PSSP.harmld       30                30
IBM.PSSP.LL         IBM.PSSP.harmld      250               250
IBM.PSSP.Membership  Membership            0                 0
IBM.PSSP.PRCRS      IBM.PSSP.harmld        5                 5
IBM.PSSP.Prog       IBM.PSSP.harmpd        0                 0
IBM.PSSP.Response   Response               0                 0
IBM.PSSP.SP_HW      IBM.PSSP.hmrmd         0                 0
IBM.PSSP.VSD        IBM.PSSP.harmld       10                10
IBM.PSSP.aixos.CPU   aixos                15                 0
IBM.PSSP.aixos.Disk  aixos                30                 0
IBM.PSSP.aixos.FS    aixos                60                 0
IBM.PSSP.aixos.LAN   aixos                40                 0
IBM.PSSP.aixos.Mem   aixos                15                 0
IBM.PSSP.aixos.PagSp aixos                30                 0
IBM.PSSP.aixos.Proc  aixos                60                 0
IBM.PSSP.pm         IBM.PSSP.pmanrmd       0                 0
#
```

Figure 84. Resource Variables Classes

Next you need to select the right Resource Class. For our scenario, choose Resource Class IBM.PSSP.aixos.FS, because this class is the only one with the string FS in it, which stands for "file system."

Now find the names of all Resource Variables in the Resource Class IBM.PSSP.aixos.FS.

```
# SDRGetObjects EM_Resource_Variable rvClass==IBM.PSSP.aixos.FS rvName
rvName
IBM.PSSP.aixos.FS.%nodesused
IBM.PSSP.aixos.FS.%totused
IBM.PSSP.aixos.FS.VG.free
#
```

Figure 85. Displaying the Resource Variables of a Resource Class

As you see, there are only three Resource Variables in this class. A Resource Variable represents the attribute of a resource in the system. Each is represented by a unique resource name (for example, IBM.PSSP.aixos.FS), followed by a period (.), followed by a resource attribute (for example, %totused).

Next, click the selection box of the Resource Variable field. Scroll down and click each of the three Resource Variables, one at a time, and read each description in the Resource Variable Description box. Note that the Resource Variables are listed in alphabetical order, with the uppercase letters listed before the lowercase letters.

For our scenario, choose IBM.PSSP.aixos.FS.%totused based on its description. Recall that when you choose a predefined condition, the Resource Variable and its description appear automatically.

4. Enter the expression or predicate for the condition in the Expression box. Note that if you choose a predefined condition, however, its expression appears here automatically. In our scenario, the expression is X>90.

5. Enter a rearm expression in the Rearm Expression box if you wish. This is an optional field. A rearm expression is used to generate an event that alternates with the original event expression. In the present example, choose X<85. With this expression and rearm expression, X>90 is used until it is true, then X<85 is used until it is true, then the X>90 is used, and so on.

   Note that the view-only Resource Identifier Format box automatically shows you the resource identifiers or the Instance Vector that is associated with the Resource Variable you choose.

6. The Fixed Resource Identifier Fields box is an optional one. It allows you to set a fixed value for a specific resource that is monitored for this event. By entering a field name and a field value here, you can designate a specific resource against which the expression is applied.

   In our scenario, the Resource Variable IBM.PSSP.aixos.FS.%totused is represented by three name=value pairs: LV=x, NodeNum=x and VG=x. Since you are interested in the /var file system, you may decide to restrict the logical volumes and volume groups that could be monitored for this event to only hd9var and rootvg, respectively. Enter LV=hd9var;VG=rootvg in this field.

   As a result, when you specify the condition varFull, you are only able to choose the nodes whose /var could be monitored. If you need to have a different logical volume or volume group than hd9var and rootvg, then you have to create a new condition.

   If you do not enter anything in the Fixed Resource Identifier Fields, then you need to specify all the elements (here LV, NodeNum, and VG) of the resource identifier every time you use the condition (here varFull).

   Once you enter the information in the fields of the Create Condition dialog box (Figure 83 on page 149), click **OK** to save the information, close the dialog box, and return to the Definition page of the Create Event Definition notebook. Notice that at this stage most of the boxes in the Definition page are greyed (not active). In order to activate the remaining boxes, you need to select your condition name (here varFull) in the Condition Name box. Note that the description is automatically displayed in the Description box.

Step 4 Select **NodeNum** in the Field Names box, and **0** and **5** in the Field Values box.

Step 5 Choose the **Register** option, which is the default.

Step 6 Click the **Response Options** tab and select the **Get Notified During the Event Perspective Session** option.

Step 7 Select all nodes.

Step 8 Click **OK** and close.

5.2.3 More about Using the Event Perspective

The use of GUI interfaces such as Perspectives is becoming increasingly popular mainly because GUIs are able to hide most underlying details and complexities from the end user. A drawback associated with GUIs, however, is that debugging and troubleshooting are more difficult.

This section discusses how Event Perspective interacts with the other subsystems of PSSP 2.2. There is also a discussion about terms that are used for the same concepts in different subsystems.

In addition, a more accurate definition of the term *condition* is given. (Condition is a concept used in Event Perspective exclusively.) Finally, we discuss the kind of authorization needed in the Event Perspective, and offer a few other hints and tips.

5.2.3.1 Some Terminology Considerations

Event Perspective′s terminology usage is slightly different from that of the Event Management subsystem, in order to have more user-friendly terms in the GUI.

Following is a partial list of these terms (see also Table 3 on page 154):

- The terms *predicate* and *rearm predicate* in Event Management are called *expression* and *rearm-expression* in Event Perspective, respectively.
- An Instance Vector in Event Management is equivalent to the combination of Fixed Resource Identifier Fields and Resource Identifier in Event Perspective.
- The term *pmHandle* in the Problem Management subsystem is equivalent to the Event Definition Name in Event Perspective.
- The term *condition* in Event Perspective does not have direct equivalents in other PSSP 2.2 subsystems. Condition is a collection of seven components, that is, the attributes of the EM_Condition class in the SDR (see Table 2).

Table 2. Equivalent Terms between the SDR and the Event Perspective

EM_Condition Attribute	Event Perspective Field
name	Condition Name
description	Condition Description
variable	Resource Variable Name
predicate	Expression
rearm	Rearm Expression
specified	Fixed Resource Identifier Fields
unspecified	Resource Identifier

There are 16 IBM-defined typical conditions included in PSSP 2.2. These 16 objects of the EM_Condition class are displayed in Table 4 on page 170. This table is the output of the following command (assuming no other conditions have been added or removed after the initial installation of PSSP 2.2):

```
# SDRGetObjects EM_Condition
```

- The term event definition stands for a collection that consists of four components:

  - Event Definition Name, which should be unique within a system partition for a given Kerberos principal.
  - Condition Name, which names the condition to be observed.
  - Resource Identifier, which specifies on what resource the condition is to be observed.
  - Response Options, which specifies what response is to be taken when the event occurs.

  Use of the Response Options component is completely optional. One can create an event definition without specifying the type of response when the event occurs.

  The first three components, however, are required in order to create an event definition.

Table 3. Equivalent Attribute Names between Problem Management and Event Perspective

Problem Management	Attribute Values for Our Example Event varFullEvent	Event Perspective
pmActivated	1	Register/Unregister
pmHandle	varFullEvent	Event Definition Name
pmRvar	IBM.PSSP.aixos.FS.%totused	Resource Variable Name
pmIvec	NodeNum=local;LV=hd9var;VG=rootvg	Fixed Resource Identifier Fields plus Resource Identifier
pmPred	X>90	Expression
pmCommand	wall /var now is more than 90% full!	Command
pmCommandTimeout	30	How many seconds to wait before assuming that the command has hung, and then killing it
pmTrapid	-1	Command Trap ID
pmPPSlog	0	Command PSSP Log
pmText		Command Log Text
pmRearmPred	X<85	Rearm Expression
pmRearmCommand	wall /var now is less than 85% full!	Rearm-Command
pmRearmCommandTimeout	45	How many seconds to wait before assuming that the Rearm-Command has hung, and then killing it
pmRearmTrapid	-1	Rearm-Command Trap ID
pmRearmPPSlog	0	Rearm-Command PSSP Log
pmRearmText		Rearm-Command Log Text
pmUsername	root	The username of the subscriber client
pmPrincipal	root.admin@MSC.ITSO.IBM.COM	Kerberos Principal
pmHost	sp21en0.msc.itso.ibm.com	The host from which the subscription initiated
pmTargetType	NODE_RANGE	pmTarget field format
pmTarget	0,5	Hosts on which to run Command or Rearm-Command
pmUserLabel	varFull	Condition Name

5.2.3.2 Where Is My Event Definition Stored?

Where the event definition is stored depends entirely on the type of Response Options selected in the event definition you created.

As indicated in Figure 86 on page 155, when a registered event definition is created and the **Get Notified During the Event Perspective Session** option is selected, then the Event Definition is registered with the Event Management subsystem directly without engaging the Problem Management subsystem. This event definition is stored in file $HOME/.$USER:Events, which is the event notification profile of the user. Note that, unlike the SDR, $HOME/.$USER:Events is not shared with any other user and is writable only by the user $USER.

As long as the user is in an Event Perspective session, the Event Management notifies the user whenever the event occurs. This notification stops when the user ends the Event Perspective session. However, as soon as the user starts the session again, even from a different system partition, the event information is retrieved from $HOME/.$USER:Events and the user receives notification whenever the event occurs.

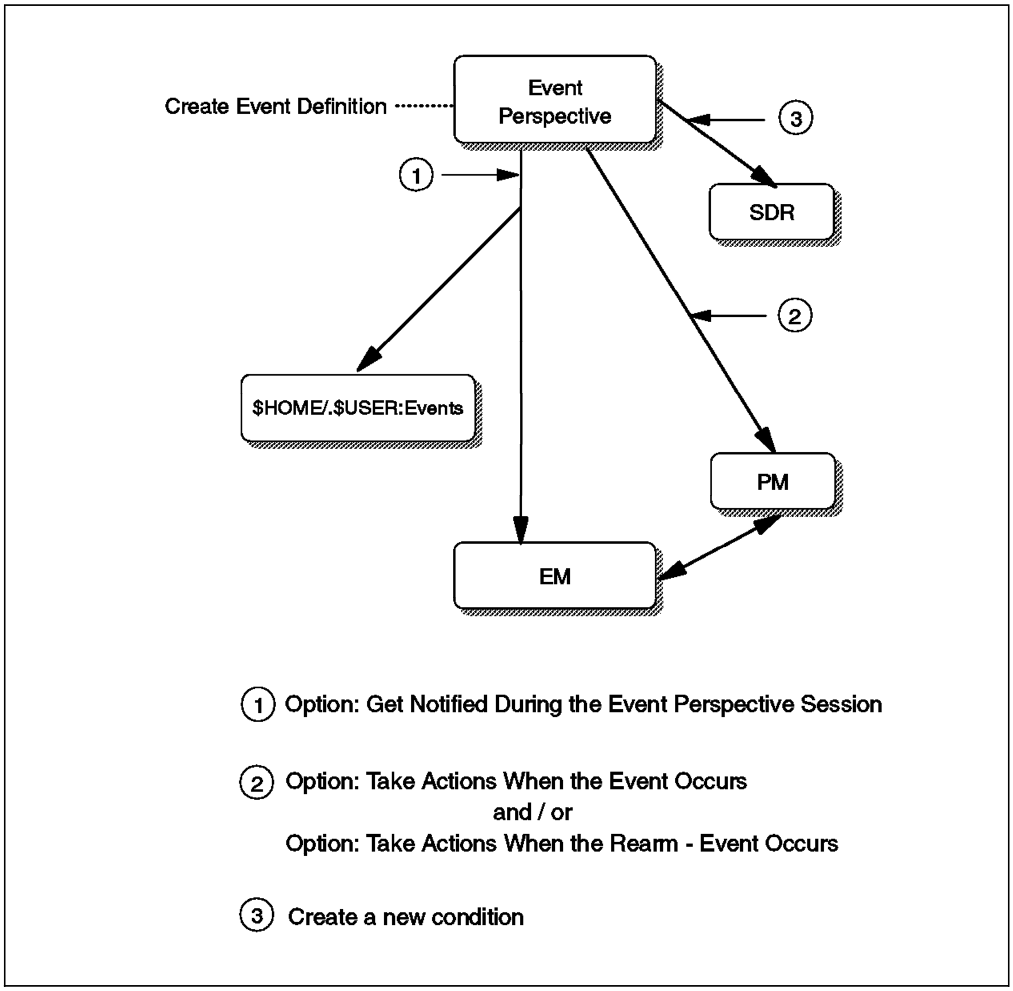

Create Event Definition
Event Perspective
3
1
SDR
2
$HOME/.$USER:Events
PM
EM
1 Option: Get Notified During the Event Perspective Session
2 Option: Take Actions When the Event Occurs
and / or
Option: Take Actions When the Rearm - Event Occurs
3 Create a new condition

Figure 86. Interrelation of Event Perspective with the Underlying Subsystems

When the event definition is registered and either or both options **Take Actions When the Event Occurs** and **Take Actions When the Rearm-Event Occurs** are selected, then the Event Definition registers with the Event Management through the Problem Management subsystem. In this case, when the event occurs, actions are taken whether the user is running Event Perspective or not.

If the condition of the event definition is a new one, then this condition is stored in the SDR as an object in the EM_Condition class. Note that you may create a new condition (similar to the Step 3 of Example 2) without creating any event definitions. Later you can use these conditions to create event definitions.

5.2.3.3 Can I Modify or Delete an Event Definition?

Using the Event Perspective, you can view, modify, and even delete an event definition but not the condition of the event definition. Conditions are stored in the SDR permanently until you execute the SDRDeleteObjects command, which we do not recommend. For example, to delete the condition varFull from the SDR, use the following command (see also 5.2.3.5, “SDR and Some Helpful SDR Commands”):

```
# SDRDeleteObjects EM_Condition name=varFull
```

Figure 87. Deleting a Condition from the SDR

You are usually better off not deleting any conditions from the SDR. These conditions may be used later to create new event definitions.

5.2.3.4 Summary of Authorizations Needed

In order to be able to create an event definition, you need to be a Kerberos Principal in the system partition of interest.

In addition, if in creating your event definition, you do not use a predefined condition and create a new condition, then you need to have the SDR write authority to update the SDR. This means that you need to be both running as root or as a user with uid=0, *and* running from the Control Workstation or from a node whose adapter is in the SDR Adapter class.

Finally, if you select response options of the type **Take Actions When the...** (Figure 86 on page 155), then in addition to the Kerberos, your principal name needs to be added to the Problem Management access control list (acl) file /etc/sysctl.pman.acl. This is to be done for every node that is affected when the event occurs and the response actions are taken.

5.2.3.5 SDR and Some Helpful SDR Commands

The System Data Repository (SDR) is an SP subsystem where your SP system configuration and some operational information are stored. The SDR itself is physically located on the Control Workstation, but it is made available to other network-connected nodes through a client/server interface featuring the SDR daemon sdrd.

Clearly, it is a good idea to understand the SDR and know how the data is organized in it. You should also be familiar with some of the SDR commands that are useful in creating and managing events. Read the following note carefully.

Important

The SDR commands are to be used by the IBM PSSP for AIX system management software. Use of these commands by a user can cause corruption of the system configuration data.

The exceptions are: SDRArchive, SDRGetObjects, SDRListClasses, SDRListFiles, SDRRetrieveFile, SDR_test, and SDRWhoHasLock.

Let us briefly review the logical model used in the SDR. It is composed of classes, objects, and attributes. Every class contains objects and every object is uniquely identified by its set of attribute values. For example, the class Switch in our system happens to contain only one object. The attributes of this object (our switch) are as follows (see also Figure 89 on page 158.):

```
switch_number = 1
frame_number = 1
slot_number = 17
switch_partition_number = 1
.
.
.
switch_name = SP_Switch
.
.
.
```

Figure 88 shows how you can display class names in the SDR.

```
# SDRListClasses
Adapter
Dont_care_pool_list
EM_Condition
.
.
.
Frame
.
.
.
Node
NodeGroup
Pool
SP
.
.
.
switch_responds
# SDRListClasses | wc -l
      35
# SDRListClasses | grep EM
EM_Condition
EM_Instance_Vector
EM_Resource_Class
EM_Resource_Monitor
EM_Resource_Variable
EM_Structured_Byte_String
#
```

Figure 88. Displaying the Class Names in the SDR

Most SDR interaction is performed using the SDR command line interface. In the next section, we see some of these commands in action.

There are 35 different classes in the SDR. Six of these pertain to the Event Management subsystem. One class, the EM_Condition class, is used by Event Perspective exclusively. The objects in the Switch class are shown in Figure 89 on page 158 as an example.

```
# SDRGetObjects Switch
switch_number frame_number slot_number  switch_partition_number switch_ty
pe  clock_input  switch_level switch_name  clock_source clock_change
           1            1           17            1         129 0
                1 SP_Switch              0 no
#
```

Figure 89. Displaying the Objects and Attributes of a Class

There are ten attributes for the class Switch. You can see that our system has only one switch (object or row of data) of type 129 with the name SP_Switch.

Table 4 on page 170 is the output of the following command:

```
# SDRGetObjects EM_Condition
```

This table lists all the conditions belonging to the EM_Condition class. A great deal of useful information about your system can be extracted from the SDR by using some simple SDR commands.

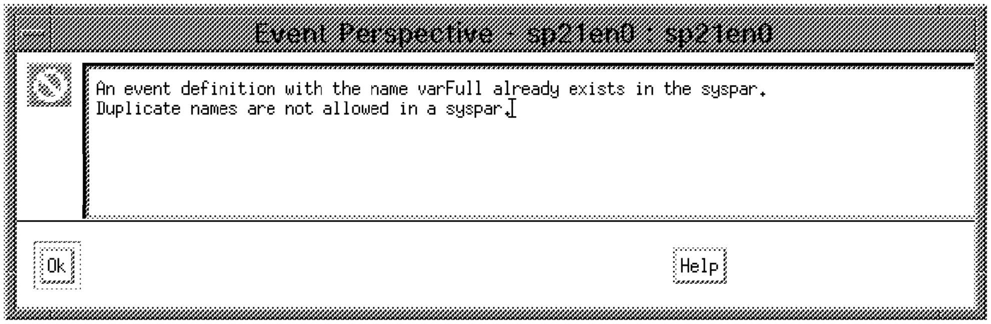
Remember to refer to the note labeled "Important" at the beginning of this section for information regarding some SDR commands. For further details on this subject, see *PSSP for AIX Administration Guide*, GC23-3897, and *PSSP for AIX Command and Technical Reference*, GC23-3900.

Finally, you may want to fill your ENV=$HOME/.kshrc file with your favorite aliases. Our .kshrc file looks like the following:

```
.
.
.
alias sdrcls=″SDRListClasses″
alias sdrc=″SDRGetObjects EM_Condition ″
alias sdriv=″SDRGetObjects EM_Instance_Vector ″
alias sdrrc=″SDRGetObjects EM_Resource_Class ″
alias sdrrm=″SDRGetObjects EM_Resource_Monitor ″
alias sdrrv=″SDRGetObjects EM_Resource_Variable ″
alias sdrsbs=″SDRGetObjects EM_Structured_Byte_String ″
.
.
.
```

5.2.3.6 Runtime Error Message Help Facility

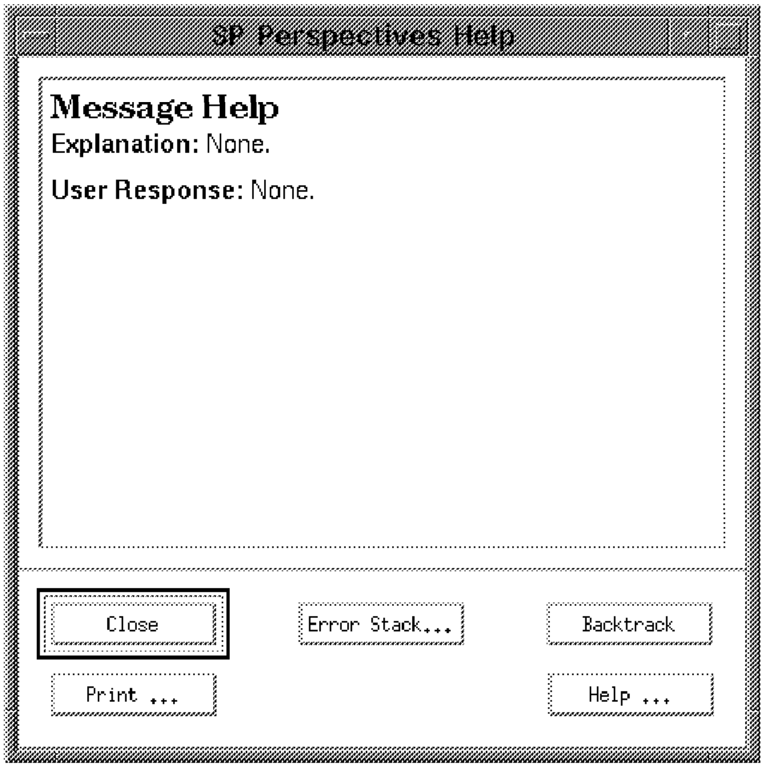
A very useful feature of SP Perspectives (including the Event Perspective) is the runtime error message help facility. Typically, when there is an error, the Event Perspective displays a pop-up window (Figure 90 on page 159), which gives a brief description of the nature of the error.



An event definition with the name varFull already exists in the syspar.
Duplicate names are not allowed in a syspar.

Figure 90. The Event Perspective Pop-Up Error Window

Now click **Help** on this pop-up window, and you are presented with a second window named SP Perspectives Help. This window displays a Message Help, if there is one (see Figure 91).



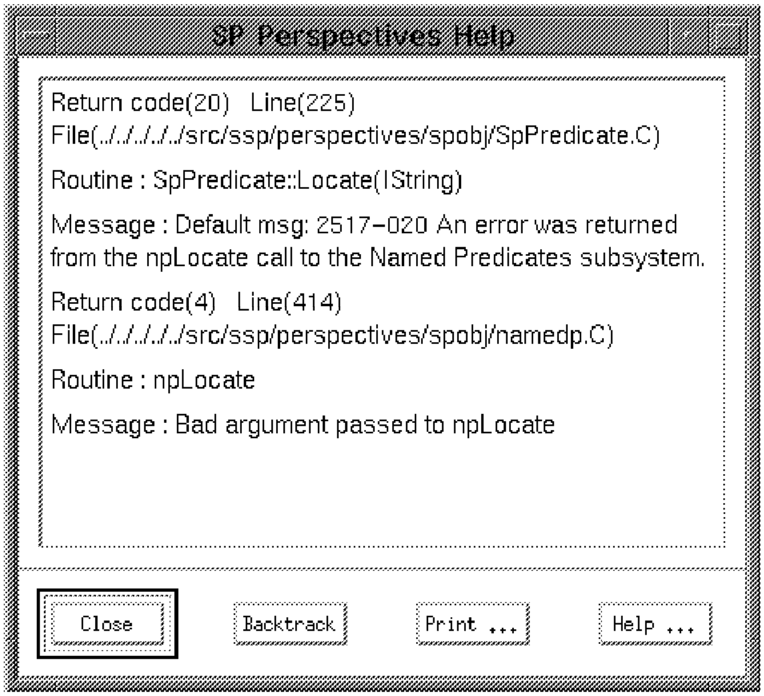
Message Help

Explanation: None.

User Response: None.

Figure 91. The SP Perspectives Help Window with Message Help

Finally, you can click **Error Stack...** on the SP Perspectives Help window, which gives you a third window with more detailed information about the error (see Figure 92 on page 160). If you are experiencing an error and need to call IBM support, it is helpful to have the error stack information handy.



```
Return code(20)   Line(225)
File(../../../../../src/ssp/perspectives/spobj/SpPredicate.C)

Routine : SpPredicate::Locate(IString)

Message : Default msg: 2517-020 An error was returned
from the npLocate call to the Named Predicates subsystem.

Return code(4)   Line(414)
File(../../../../../src/ssp/perspectives/spobj/namedp.C)

Routine : npLocate

Message : Bad argument passed to npLocate
```

Figure 92. The SP Perspectives Help Window with Detailed Error Information

5.2.4 Example 3: Monitoring CPU Usage

Suppose you want to execute the command `wall Do not waste CPU time!` and send an SNMP trap with ID=1006 when the CPU idle time is less than 10% on cpu5 of Node 1 of partition sp21en0.

To do this, you must first make sure that partition sp21en0 is the current partition, and then follow the procedure described in 5.2.2, "Example 2: Monitoring File System Size" on page 148.

Note

Details of the following steps are given in Examples 1 and 2.

Before you start, be sure the Event Definition pane has focus.

Step 1 Click the **Create Event Definition** icon.

Step 2 Make up a name which is unique within the system partition, for example, CPU_Idle_Monitor.

There is no predefined condition that can help you here (see Table 4 on page 170). Create a new condition.

Step 3 Create a new condition.

1. Click **Create Condition**.
2. Enter a name for the condition (for example, `CPU_Idle`) and a description (for example, `Nodes too busy`).
3. This step can be tricky unless you are either using a predefined condition or you already know which Resource Variable you need to use.

Since in our scenario you are creating a brand new condition and event definition, you have to do some searching, as described in Step 3 of 5.2.2, “Example 2: Monitoring File System Size” on page 148.

Looking at the output of the following command tells us that the class IBM.PSSP.aixos.CPU seems to be the right one.

```
# SDRGetObjects EM_Resource_Class
```

It is the only class that has the string CPU in it. (Instead of using this command, you can refer directly to Figure 84 on page 150 to determine this.)

Now you need to find the names of all the Resource Variables in this class.

```
# SDRGetObjects EM_Resource_Variable rvClass==IBM.PSSP.aixos.CPU rvName
rvName
IBM.PSSP.aixos.CPU.glidle
IBM.PSSP.aixos.CPU.glkern
IBM.PSSP.aixos.CPU.gluser
IBM.PSSP.aixos.CPU.glwait
IBM.PSSP.aixos.cpu.idle
IBM.PSSP.aixos.cpu.kern
IBM.PSSP.aixos.cpu.user
IBM.PSSP.aixos.cpu.wait
#
```

Click the selection box of the Resource Variable Name field. Scroll down and click the desired Resource Variables to read their descriptions, which are automatically displayed in the Resource Variable Description box. Based on the descriptions, choose IBM.PSSP.aixos.cpu.idle with the description of `Time CPU is idle (percent)`.

4. Enter the expression `X<10`.
5. A rearm-command is not requested by the scenario, but `X≥ 10` would be an appropriate one.

   Note: Look at the field value displayed for this Resource Variable in the view-only Resource Identifier Format box. The Instance Vector has two elements: CPU and NodeNum. The reason we have both CPU and NodeNum is that a node can have more than one CPU.

Step 4 Set NodeNum=1 by selecting **NodNum** in the Field Names box, and then **1** in the Field Values box. Similarly, set CPU=cpu5.

Step 5 If you need the event definition to become active as soon as you create it, then choose **Register**.

Step 6 Select **Take Actions When the Event Occurs**. Also click **SNMP Traps** and enter `1006` in the Trap ID window in the Command area.

Step 7 Click **On Selected Node(s)** and select Nodes **0, 1, 5, 6, 7**, and **8** so that you can observe the `wall` command run on every node in sp21en0, as well as on the Control Workstation.

Step 8 Click **Create**. You are now done.

5.2.5 The pcount and xpcount Resource Variables

The Resource Variables IBM.PSSP.Prog.pcount and IBM.PSSP.Prog.xpcount are used to monitor daemons (5.2.6, “Example 4: Monitoring a Daemon” on page 165) or any other process with a long lifetime. See Appendix E, “The IBM.PSSP.Prog.pcount Resource Variable” on page 245 and Appendix F, “The IBM.PSSP.Prog.xpcount Resource Variable” on page 249 for more information.

Both of these variables have an SBS data type. An SBS is a string that can have multiple fields and whose format is embedded in the string. The fields 0, 1, and 2 for these variables are:

X@0 CurPIDCount:

The number of processes of the type specified that are currently running

X@1 PrevPIDCount:

The number of processes of the type specified that were previously running

X@2 CurPIDList:

A list of the process IDs (PIDs) of the type specified that are currently running

For the pcount and xpcount Resource Variables, the processes to monitor are specified with the following field types:

```
NodeNum;ProgName;UserName
```

For example, there is a predefined condition named sdrDown (see Table 4 on page 170) that has the following values for the field types:

```
NodeNum=0;ProgName=sdrd;UserName=root
```

What is the difference between pcount and xpcount?

- The variable pcount represents all the processes (inherited or exec′d) running a specified executable file.
- The variable xpcount represents only the exec′d processes running a specified executable file.

Attention

About Inherited and Exec′d Processes

In AIX or any other UNIX system, the only way to make a new process is to use a fork() system call, which causes the existing process to duplicate into a parent-child pair.

The child process is an identical copy of the original parent process except for its process ID number (PID). In fact, the child process inherits a copy of its parent′s code, data, stack, open file descriptors, and signal table, and then continues to execute the same code as its parent. In this situation, the child process is an inherited process.

A typical child process, however, uses an exec() system call to replace its code with that of another executable file, thereby differentiating itself from its parent. This is called an exec′d process.

To find out whether a process is inherited or exec′d, you can use this command:

```
ps -e -o ″flag,pid,ppid,comm″
```

If the SEXECED process flag (0x200000) is set (see Figure 93 on page 164), then the process is exec′d. Otherwise, the process is inherited.

```
.
.
.
/*
 * NOTE: process flag values are or′d together for binary compatibility
 */

/*
 * process flags, p_flag
 *
 * This field can be updated under the process only. If the process
 * is single threaded, then the update can be made at base level. Otherwise
 * interrupts need to be disabled and the proc_int_lock held.
 */
#define SLOAD           0x00000001      /* user and uthread struct. pinned */
#define SNOSWAP         0x00000002      /* process can′t be swapped out    */
#define SFORKSTACK      0x00000004      /* special fork stack is allocated */
#define STRC            0x00000008      /* process being traced            */
#define SFIXPRI         0x00000100      /* fixed priority, ignoring p_cpu  */
#define SKPROC          0x00000200      /* Kernel processes                */
#define SSIGNOCHLD      0x00000400      /* do send SIGCHLD on child′s death*/
#define SSIGSET         0x00000800      /* process uses the SVID sigset int*/
#define SLKDONE         0x00002000      /* this process has ″done″ locks   */
#define STRACING        0x00004000      /* process is a debugger           */
#define SMPTRACE        0x00008000      /* multi-process debugging         */
#define SEXIT           0x00010000      /* process is exiting              */
#define SORPHANPGRP     0x00040000      /* orphaned process group          */
#define SNOCNTLPROC     0x00080000      /* session leader relinquished     */
                                        /* the controlling terminal        */
#define SPPNOCLDSTOP    0x00100000      /* Do not send parent process      */
                                        /* SIGCHLD when a child stops      */
#define SEXECED         0x00200000      /* process has exec′d              */
#define SJOBSESS        0x00400000      /* job control used in session     */
#define SJOBOFF         0x00800000      /* free from job control           */
#define SEXECING        0x01000000      /* process is execing              */
#define SPSEARLYALLOC   0x04000000      /* allocates paging space early    */
.
.
.
```

Figure 93. Process Flags: p_flags Field in the /usr/include/sys/proc.h File

For example, based on the output shown in Figure 94, the variable IBM.PSSP.Prog.pcount pertains to all 8 processes, but the variable IBM.PSSP.Prog.xpcount only pertains to the process with PID=13394. It is assumed that in this case, ProgName=nfsd with the appropriate NodeNum and UserName.

```
# ps -e -o ″flag,pid,ppid,comm″ | grep nfsd
   40001 12888 13394 nfsd
  240001 13394  3148 nfsd
   40001 14172 13394 nfsd
   40001 19806 13394 nfsd
   40001 20576 13394 nfsd
   40001 21858 13394 nfsd
   40001 22618 13394 nfsd
   40001 25686 13394 nfsd
#
```

Figure 94. Sample Output of the ps Command

The variables pcount and xpcount are State variables. This means that their values are reported to the Event Manager only when the state changes. Since the SBS in the value contains both the previous PID count and the current PID count, useful expressions (predicates) can be created by referencing these two fields of the SBS. For example, the expression:

```
    X@0<X@1
```

specifies that an event should be generated if the CurPIDCount is less than the PrevPIDCount. Thus, you will get notified if a process of the specified type dies. You also may choose to rearm this event with the expression:

```
    X@0>X@1
```

if you wish to be notified when a process is started to replace the one that died. As we discussed earlier, you can leave the rearm predicate null and only be notified when processes die.

Note that the expression X@0==X@1 will never happen. This is because pcount and xpcount, being State variables, will only be reported when the state changes. Thus, the CurPIDCount will never be the same as the PrevPIDCount.

These two variables, pcount and xpcount, are considered to be *dynamically instantiated*. This means that they refer to resources in the system (in this case processes) that are transient. Because of this, the PSSP 2.2 HA Infrastructure does not keep track of all of the instances that could be monitored at any given time. This tracking would require too much system resource. For the same reason, the Event Manager will not return the field values of all possible instances if one of the field values is wildcarded. Therefore, Perspectives cannot fill in the possible field values in the Event Definition window. This has an effect on conditions that you might create using pcount and xpcount. Conditions that use either of these two Resource Variables must fully specify *all* of the field types.

Note

Do not leave any of the field types unspecified. If you do, the Event Definition window will fail to produce a complete list of possible field values to choose from and you will reach a dead end when trying to create an event definition.

5.2.6 Example 4: Monitoring a Daemon

Suppose you want to be notified whenever the sendmail daemon on Node 5 dies.

After going through Examples 2 and 3, you realize that the key step in creating an event definition is Step 3C. To carry out this step, you need to know which of the more than 360 Resource Variables is the right one for your situation.

Note that if you use a predefined condition, as in Example 3, then you do not need to worry about Step 3C because the Resource Variable is automatically selected for you; it comes with the predefined condition. You can see, therefore, that storing a variety of conditions in the SDR is a smart thing to do because you can use these predefined conditions and create variety of event definitions to suit your situation. This is the reason why, by design, when you delete an event definition, the corresponding condition is not deleted from the SDR automatically.

A nice thing about monitoring a daemon or process that lasts long enough to be detected is that you need to decide between only two Resource Variables, namely, IBM.PSSP.Prog.pcount and IBM.PSSP.Prog.xpcount. The challenging part, however, is to have enough knowledge about the process to be able to answer questions such as: “Is the process going to fork (duplicate) later?” or “Am I interested in monitoring all the child and grandchild processes created?”

Important

Rule of Thumb for Monitoring a Process

In most cases, it is safer and perhaps most appropriate to use the IBM.PSSP.Prog.xpcount Resource Variable to monitor a process.

However, if the process is inherited by long-running processes which do not “exec,” then using IBM.PSSP.Prog.pcount might be appropriate. You have to know your process and its behavior before deciding which variable is the right one, xpcount or pcount.

In this example, the output of the ps command:

```
# ps -e -o ″flag,pid,ppid,comm″ | grep sendmail
  240401  4196  3148 sendmail
#
```

shows that the process running the sendmail program has the SEXECED process flag set (refer to Figure 93 on page 164). In other words, sendmail is exec′d. You also do not anticipate that the process will fork later. Therefore, you choose the xpcount rather than the pcount Resource Variable.

The sendmail daemon is a systems daemon and runs the same way on different nodes; it has the same process “flag” value of 240401. However, if you plan to monitor other processes, you should run the ps command on the node on which the desired process runs (here it is Node 5). In this way, you get the right flag value. Now you can proceed in a manner similar to Example 2.

Note

Details of the following steps are given in Examples 1 and 2.

Step 1 Click the **Create Event Definition** icon.

Step 2 Enter a unique name, such as sendmailDownEvent.

Step 3 Create a new condition and make up a name for it, such as sendmailDown. Make up a description, such as sendmail is down!, and press Enter. Select the **IBM.PSSP.Prog.xpcount** Resource Variable. Enter X@0==0 in the Expression box and NodeNum=5;ProgName=sendmail;UserName=root in the Fixed Resource Identifier Fields box.

Step 5 If you need the event definition to become active as soon as you create it, then choose **Register**.

Step 8 Click **Create**. You are now done.

5.2.7 Example 5: Monitoring the Health of Your System

You can use the monitor button on the tool bar of Event Perspective to monitor the health of your system. (This action only applies to the Syspars pane and therefore can be used only when the Syspars pane is the current pane.) Monitoring in this approach is different from using event definitions (as in Examples 1 to 4).

Using event definitions, a dialog will pop up and perhaps an action will be performed when one of the events occurs.

Using the monitor button on the tool bar of Event Perspective, the icons in the Syspars pane will be colored according to the aggregate of the states of all the conditions being monitored and will change whenever one of those states changes. To set up and monitor the health of your system, perform the following steps:

Step 1 Make the Syspars pane current by clicking it.

Step 2. Click the monitor icon in the tool bar, or select the **View→Monitor** options from the pull-down menu. The Set Monitoring for Syspars notebook appears with the following default conditions for system partitions already listed:

- Conditions to Monitor:
  - None
- Conditions of Contained Objects:

  hostResponds Status of communications between processor nodes and the Control Workstation

  keyNotNormal The position of the key mode switch on nodes

  nodeEnvProblem Status of the environment light on the front of nodes

  nodePowerDown Status of power to nodes

  nodePowerLED Status of power light on front of nodes

  nodeSerialLinkOpen Status of the serial one port

  nodeNotReachable Status of Group Services communication with nodes

  pageSpaceLow Paging space on nodes exceeds 85 percent

  realMemLow Real memory on the system exceeds 85 percent

  switchNotReachable Status of communications between switch adapters on nodes and IP

  switchResponds Status of switch activity on nodes

tmpFull	Status of space on the file system for LV=hd3 and VG=rootvg

The conditions just listed are all the conditions that apply to nodes. More exactly, they are the conditions that have an unspecified resource identifier of only NodeNum. You can add to this list of conditions using the Create Condition button in the Event Definition notebook as you did in Examples 2 and 3.

Step 3 Click a condition in the list box to select it for monitoring.

- To select multiple conditions that are listed consecutively, click the first desired condition, hold the left mouse button down, and move the cursor down the list over the desired conditions.
- To select multiple conditions that are not listed consecutively, hold the Ctrl key on your keyboard down while you click each desired condition.

Step 4 Click **Apply** to monitor the selected conditions and leave the notebook open, or click **OK** to monitor the selected conditions and close the notebook.

- To stop monitoring selected conditions, click **Stop Monitoring**.

The display of the objects in the Syspars pane reflects the status of the conditions being monitored. In a partitioned SP system, each condition selected is being monitored for each node in each system partition (syspar). The states of all of these are combined together to get a single state, which is then displayed in the icon of the object as follows:

Green This indicates the system is working properly; none of the monitored conditions for this syspar have triggered.

Red X This indicates there is a problem; one or more monitored conditions for this syspar object have triggered.

? This indicates the state of one or more monitored conditions for this syspar object is unknown; there may be a problem communicating with the event manager daemon, for example.

To see detailed information for monitored conditions of objects, double click the object to open its notebook and view the Monitored Conditions page.

If you iconify the Event Perspective window, you will also get a color cue indicating that one of the monitored conditions occurred (for example, it will turn red if a nodePowerDown occurs on any of the monitored nodes on the SP). You can then open the iconified window, and “drill down” on the node indicated with a Red X for more notebook type of information.

The Monitored Conditions page contains information about the attributes of the system partition that are being monitored. The page lists the following information:

Condition Name The object attribute or variable being monitored

Expression A mathematical statement that is used to compare an attribute′s value to some defined value

Resource An entity in the SP system that provides a set of services

Variable Value The current value of the object attribute

Condition Value Whether the condition has triggered, thereby changing the display of the object

As previously mentioned, Table 4 on page 170 lists all the conditions belonging to the EM_Condition class.

Table 4 (Page 1 of 2). IBM Default Predefined EM_Condition Class Objects

Name	Variable	Predicate	Rearm	Specified	Unspecified	Description
Note: • stands for IBM.PSSP						
Note: All *Names* and *Variables* are continuous strings						
frameControllerNotResponding	• .SP_HW.Frame.controllerResponds	X!=1	X==1	none	FrameNum	The frame controller is not responding.
framePowerOff	• .SP_HW.Frame.frPowerOff	X==0	X!=0	none	NodeNum	The power to the frame has been turned off.
hostResponds	• .Response.Host.state	X!=1	X==1	none	NodeNum	The node is not responding.
keyNotNormal	• .SP_HW.Node.keyModeSwitch	X!=0	X==0	none	NodeNum	Key mode switch on a node was switched out of the Normal position.
nodeEnvProblem	• .SP_HW.Node.envLED	X!=0	X==0	none	NodeNum	The environment indicator LED on the node is illuminated. A hardware problem was detected.
nodePowerDown	• .SP_HW.Node.nodePower	X==0	X!=0	none	NodeNum	The power to the node is off.
nodePowerLED	• .SP_HW.Node.powerLED	X!=1	X==1	none	NodeNum	Node power is off when powerLED is not 1.
nodeSerialLinkOpen	• .SP_HW.Node.serialLinkOpen	X==1	X!=1	none	NodeNum	The serial link to the node (TTY) is open.
nodeNotReachable	• .Membership.Node.state	X==0	X!=0	none	NodeNum	Group services has found no way to communicate with the node. The node is presumed to be down.

Table 4 (Page 2 of 2). IBM Default Predefined EM_Condition Class Objects

Name	Variable	Predicate	Rearm	Specified	Unspecified	Description
pageSpaceLow	• .aixos.PagSp.%totalused	X>85	X≤85	none	NodeNum	The paging space utilized on the node exceeds 85 percent.
sdrDown	• .Prog.pcount	X@0<X@1		NodeNum=0 ProgName=sdrd UserName=root	none	This condition will cause an event when an SDR daemon dies. Note that conditions using this Resource Variable must have all field values specified.
realMemLow	• .aixos.Mem.Real.%free	X<15	X≥15	none	NodeNum	The real memory on the system is over 85 percent utilized.
switchPowerLED	• .SP_HW.Switch.powerLED	X!=1	X==1	none	SwitchNum	Switch power is off when powerLED is not 1.
switchNotReachable	• .Membership.LANAdapter. state	X!=1	X==1	AdapterNum=0 AdapterType=css	NodeNum	The switch adapter on the node is not responding to IP, or the node is isolated.
switchResponds	• .Response.Switch.state	X!=1	X==1	none	NodeNum	The switch adapter on the node is not responding, or the node is isolated.
tmpFull	• .aixos.FS.%totused	X>90	X<80	LV=hd3 VG=rootvg	NodeNum	The file system for LV=hd3 and VG=rootvg is running out of space.

Chapter 6. Problem Management SNMP Subagent

As your SP systems increase in number, complexity, and geographic location, the ability to report SP problems and events to an existing TCP/IP-based network manager becomes not only desirable but necessary.

Typically the availability of these systems is vital, and therefore the ability to monitor and detect potential problems proactively is essential. This can be achieved by extending the new PSSP 2.2 High Availability Infrastructure by integrating the management of SP systems to an enterprise manager, either locally or remotely.

Figure 95 shows a hypothetical network of distributed SP systems within an existing enterprise where the main operations center is located in London. In this scenario, this center is responsible for the management of the total enterprise, including the remote SP systems.

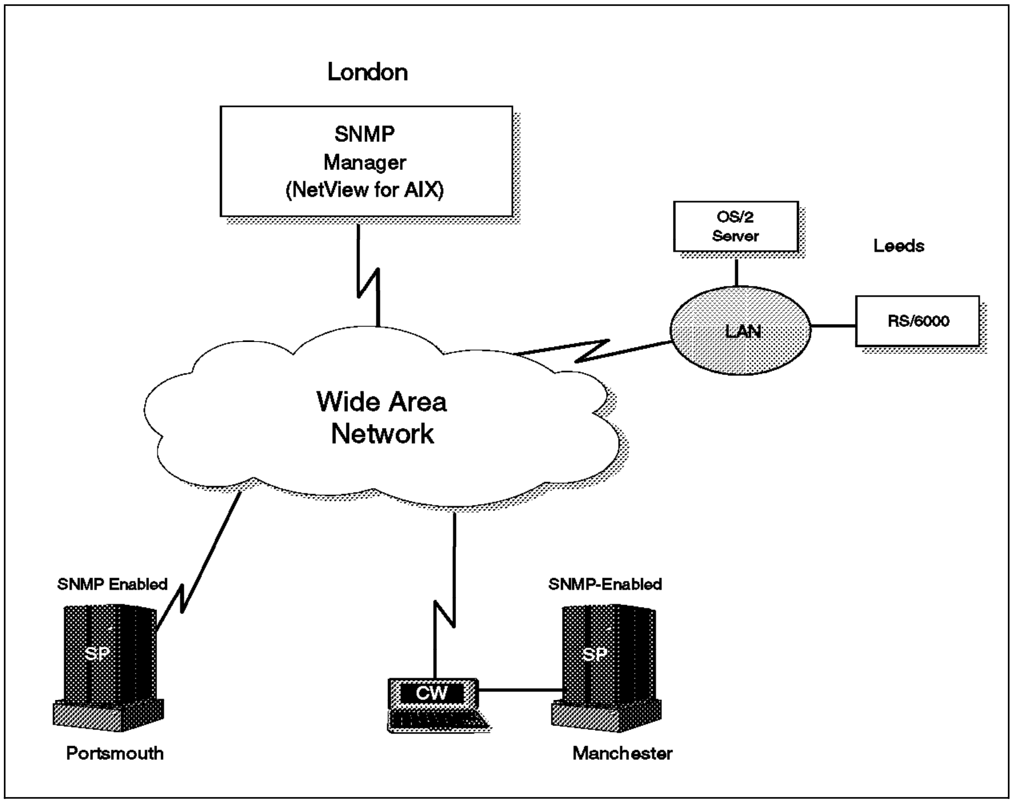


Figure 95. SP Distributed Management

This chapter describes the implementation specifics and the methodology used to integrate and report SP-related problems to an existing enterprise solution, specifically NetView for AIX or another Simple Network Management Protocol (SNMP) manager that understands SNMP.

The essence of the Problem Management subsystem and the Event Management subsystem have already been discussed in earlier chapters. This chapter does not cover those concepts again.

6.1 Problem Management SNMP Support

PSSP 2.2 includes a Problem Management subsystem that provides an infrastructure for recognizing and acting on problem events within the SP system. This infrastructure is based on an Event Management application that provides an Event Management client interface (EMAPI). This interface enables the Problem Management subsystem to access the Event Management function without the necessity of writing complex C programs that use the Event Management APIs.

The Problem Management subsystem lets you subscribe to Event Management events and specify actions for those events. One action you may wish to specify is to issue an SNMP trap as a response to an event.

The ability to issue an SNMP trap in response to an event allows you to report problem events occurring in your SP system to an existing SNMP network manager, such as NetView for AIX. You may want to both perform a local recovery action and send an SNMP trap to a remote network manager.

The Problem Management subsystem provides an SP SNMP proxy agent, sp_configd. This agent is often referred to as a subagent, and it runs on the Control Workstation and every SP processor node. It provides the following functions:

1. A Management Information Base (MIB).
2. Support for SNMP GET and GET NEXT, but not support for SNMP SET. The supported commands allow data in the MIB to be accessed by a network manager, such as NetView for AIX.
3. Creation and transmittal of SNMP traps to an installation-defined network manager application, when the following events occur on the node on which the SP proxy agent is running:
    - A “cold start” trap is issued when the proxy agent sp_configd is activated.
    - An enterprise-specific trap is issued when an entry with an Alert=true attribute is written to the AIX error log.
    - An enterprise-specific trap is issued when a user-specified event is detected within the Event Management services.

This chapter provides implementation specifics of incorporating the SP as part of a larger network management infrastructure. In such a system, the SP presents error event information in the form of SNMP traps to NetView for AIX, which then displays and logs the trap.

6.1.1 Understanding Network Management

The set of standards for Network Management in an IP network is commonly referred to as Simple Network Management Protocol (SNMP). This protocol describes only the means by which management data is transported through the network. See Appendix G, “SNMP-Related Request For Comments” on page 253 for more information about SNMP RFCs and how to get copies of them.

Equally important are the standards that define the structure of the data being transported, the Management Information Base (MIB).

In the following sections, we briefly explain these elements.

6.1.2 Understanding Simple Network Management Protocol (SNMP)

SNMP uses a client/server approach to management of resources, defining two roles: the *manager* and the *agent*.

- The manager (client) is where the network operators manage the overall network activity, using an application such as NetView for AIX.
- The agent (server) is responsible for reporting on and maintaining the data pertaining to a device, when the manager requests it to do so.

SNMP version 1, the version of SNMP supported by Problem Management, only defines a relationship between the manager and the agent, and there is no concept of a manager-to-manager connection.

6.1.3 Understanding the Management Information Base (MIB)

Every agent supports a Management Information Base (MIB), which is best described as a set of variables that represent the physical and logical resources of the managed systems or agents. The MIB is not a database, in the sense of a monolithic collection of data, but rather represents *live* information about the system resources. The values of these variables are maintained by different system functions such as the kernel, device drivers, and subsystems.

Figure 96 on page 176 shows an object identifier tree in a typical MIB structure.

Most SNMP agents available in the market place today support MIB-2 only. The standard MIB-2 contains 171 objects that relate to aspects of the IP network connectivity, and basic system information, such as system contact name, location, and interface information.

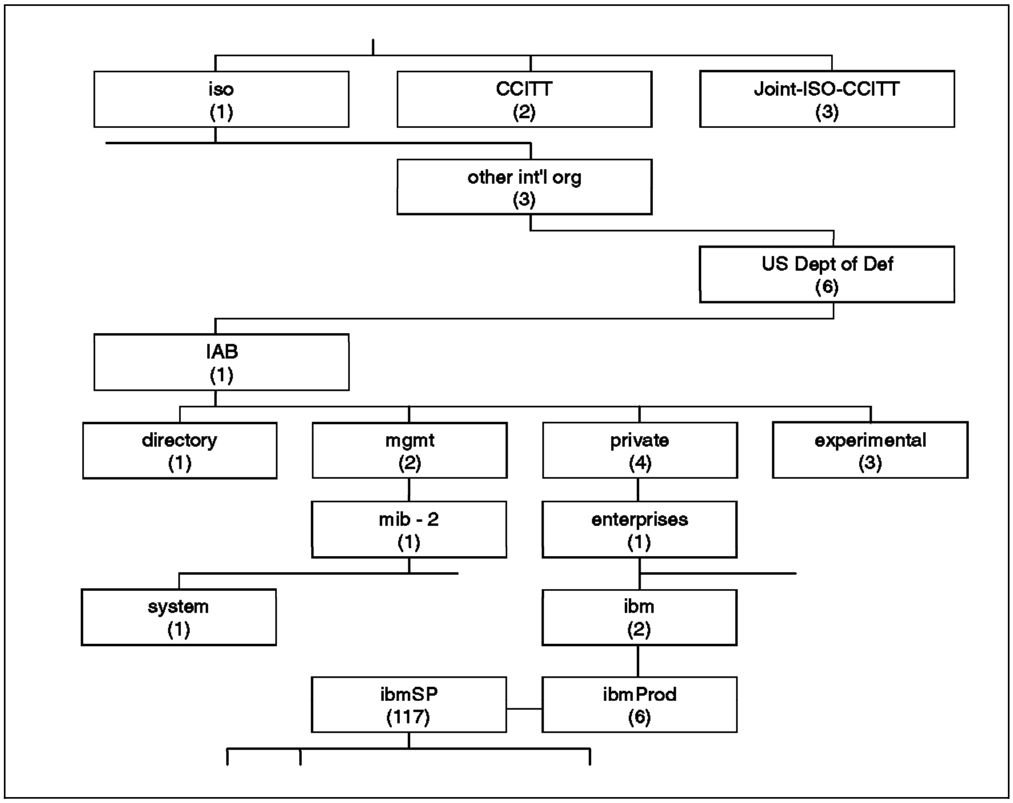


Figure 96. Object Identifier Tree - MIB Structure

In order to monitor SP-specific variables, such as Event Management variables, it is necessary to further extend the MIB with private extensions. These will be specific to the SP and are located off the private branch of the object tree.

6.1.4 Understanding the SNMP Multiplexor (SMUX) Protocol

By extending the default MIB tree with the MIB extensions, the manager must be able to access and modify these extended MIB variables, depending on the MIB access attributes. Hence the agent must also be extended to extract the information that the manager requests.

One method of achieving this is to install a modified SNMP agent, which is capable of understanding the extended MIB. Alternatively, a *subagent* can be implemented, using the standard SNMP agent interface that is provided. This interface allows additional code to register a MIB extension and handle requests for it. Such a subagent API is called the SNMP Multiplexor (SMUX) protocol, which is used by the sp_configd subagent to communicate with the SNMP agent running on the same node. The SP Problem Management subsystem communicates with the SNMP manager, such as NetView for AIX, that is using the SNMP protocol.

6.1.5 SP SNMP Multiplexor Agent

The sp_configd daemon is internally configured as an SMUX peer, or proxy agent, of the snmpd daemon on the Control Workstation and on each node of the SP.

This proxy agent is an example of a private MIB extension, written by IBM for the problem management of the SP. As with all objects found under the ibmProd branch of the tree, the SP subtree starts with an object ID of 1.3.6.1.4.1.2.6 and the name iso.org.dod.internet.private.enterprises.ibm.ibmProd.

The SP proxy agent MIB has an object ID of 1.3.6.1.4.1.2.6.117 and the name iso.org.dod.internet.private.enterprises.ibm.ibmProd.

The best method for determining the object ID of a particular variable is to use the MIB browser function of NetView for AIX. This is described in 6.2.9.2, “NetView for AIX MIB Browser” on page 196.

The SP MIB contains objects that relate to the SP configuration (ibmSPConfig), SP error log entries (ibmSPErrlogVars), and SP events (ibmSPEMVariable).

Figure 97 shows an IBM private SP MIB.

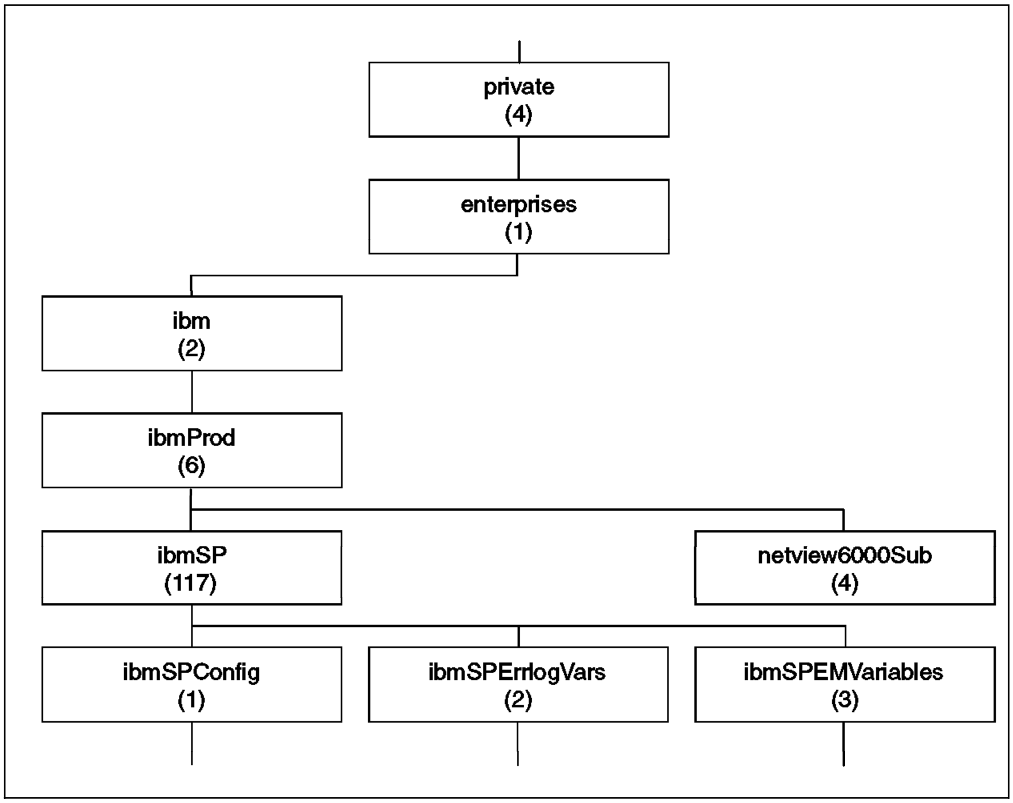


Figure 97. IBM Private SP MIB

6.1.6 How Problem Management Uses SNMP

In order for the SP to be part of such a network management system, the network managers must be notified when selected AIX error log entries are written and Event Management events occur. SP-specific configuration data is provided so that management applications can determine which nodes compose the SP system. The administrator must decide what error log and event management events will trigger manager notification.

One of the functions of the Problem Management subsystem is to interface to the SNMP manager. As clients subscribe to Event Management events, the subsystem is able to receive these events from the predefined resources, and subsequently pass these events to the sp_configd subagent, which then generates SNMP traps.

The sp_configd component of the Problem Management subsystem is able to alert or notify an SNMP manager when an *alertable* entry is posted in the AIX error log.

The relationships between the various parts of a distributed management system are illustrated in Figure 98.

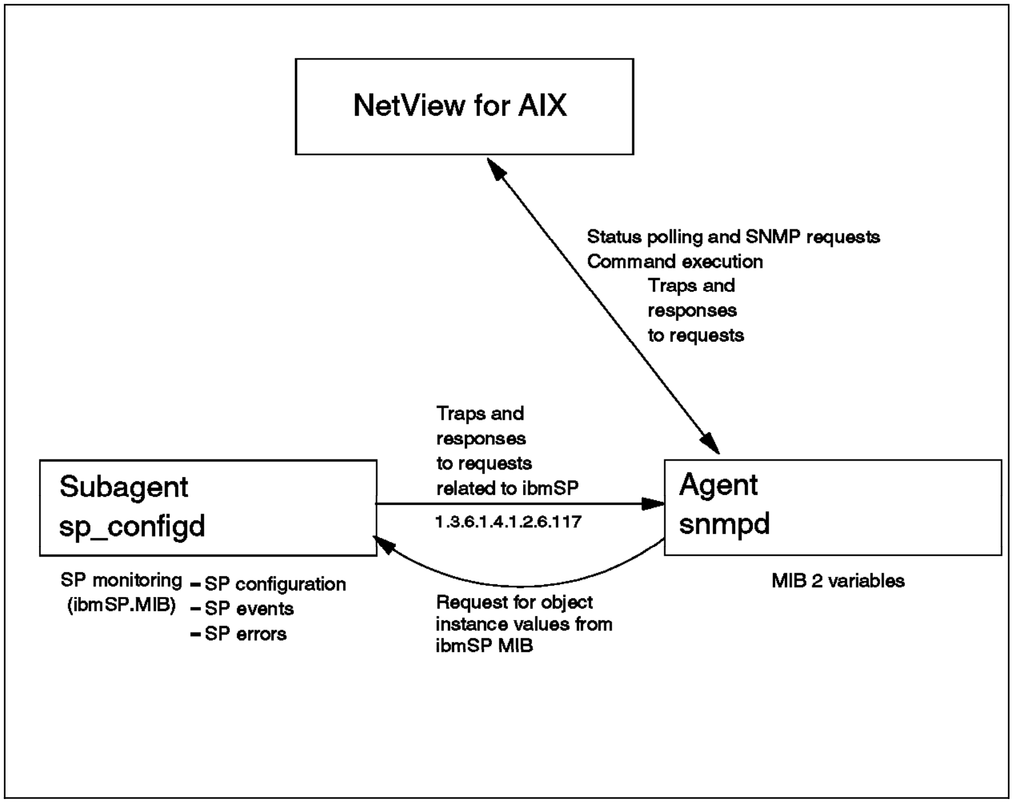


Figure 98. Relationship between NetView for AIX and SNMP Agents

6.2 Installation and Configuration

It is necessary to configure the SNMP manager, which in this case is NetView for AIX, to recognize SP trap and configuration information.

6.2.1 Configuring the Ports

This section describes the various SMUX and SNMP ports which are configured automatically when PSSP 2.2 is installed.

SMUX port: The /etc/services file contains port assignments for required services. The following entry must be present in the /etc/services file for the SMUX protocol:

```
smux            199/tcp                         # snmpd smux port
```

Note:

1. The SMUX port must be 199.

2. If the /etc/services file is being served from a server, this entry must be present in the server′s /etc/services file.
3. The /etc/services file is modified during installation and does not require any further customization.

SNMP Ports: During installation of AIX, the installation process adds the port definitions to the /etc/services file.

```
       snmp            161/tcp                          # snmp request
       snmp            161/udp                          # snmp request
       snmp-trap       162/tcp                          # snmp monitor
       snmp-trap       162/udp                          # snmp monitor
```

Port 162/UDP is normally defined and named snmp-trap. This is the port where NetView for AIX listens for incoming traps.

This file is modified automatically during installation, and does not require further customization.

6.2.2 Configuring SNMP Agents

This section describes how to configure SNMP agents and SMUX peers.

6.2.2.1 Community Name

It is necessary to configure SNMP on the agents, which are residing on the Control Workstation and on each of the nodes. Most SNMP agents and managers default to using the community name (or password) *public*. This community name provides read-only access to the MIB, and hence it is not possible to change any of the read/write MIB variables using the SNMP SET command.

```
   community       public
   community       private 127.0.0.1 255.255.255.255 readWrite
   community       system  127.0.0.1 255.255.255.255 readWrite 1.17.2
```

In most circumstances, this is sufficient and no changes need to be made to the SNMP configuration. However, if you want to restrict access to the ibmSP MIB, then it is necessary to restrict the public community name to MIB views under the management leaf, and to create a new community name below the enterprise leaf for the public community name. They must be explicitly listed in order to avoid providing access to the ibmSP leaf using the public community name.

The following customization of the /etc/snmpd.conf file must be performed on every SP node and the Control Workstation.

```
              Agent configuration (/etc/snmpd.conf)
              -------------------------------------


    logging         file=/usr/tmp/snmpd.log         enabled
    logging         size=0                          level=0

    community       public 1.17.1
(a) community       itso  9.12.1.71 255.255.255.255 readOnly 1.17.2
    community       private 127.0.0.1 255.255.255.255 readWrite
    community       system  127.0.0.1 255.255.255.255 readWrite 1.17.2

    view            1.17            mgmt view
    view            1.17.2          ibmSP

(b) trap            public          9.12.1.71       1.2.3   fe


    smux            1.3.6.1.4.1.2.3.1.2.1.2         gated_password

(c) smux    1.3.6.1.4.1.2.6.117     sp_configd_pw
    smux            1.3.6.1.4.1.2.3.1.2.1.3         xmservd_pw
```

Comments about the Agent Configuration File:

Point a. We used the community name, *itso*. This was configured both on the manager, in /usr/OV/conf/ovsnmp.conf (as described later), and in the agent, /etc/snmpd.conf. The community itso entry will allow the SNMP manager with IP address 9.12.1.71 read-only access to the ibmSP MIB on the nodes that have the configuration shown. Note, however, that ibmSP.MIB is a read-only MIB.

These entries are installation-specific and must be done by the user.

Point b. This trap destination statement must be in place on each of the SP nodes and the Control Workstation, which points to the IP address of the management station.

These entries are installation-specific and must be done by the user.

Point c. Specifies the SMUX association configuration for the sp_configd proxy agent.

The following entry must be present in the snmpd.conf file, and is done automatically during the installation procedure.

```
smux    1.3.6.1.4.1.2.6.117     sp_configd_pw   # sp_configd
```

6.2.2.2 SMUX Peer Configuration

The SMUX peer configuration is held in the /etc/snmpd.peers file on every node where the subagent will reside.

```
sp_configd      1.3.6.1.4.1.2.6.117     sp_configd_pw
```

sp_configd is the name of the process acting as an SMUX peer, and 1.3.6.1.4.1.2.6.117 is the unique object identifier of the SMUX peer. sp_configd_pw is the password that the snmpd daemon requires from the SMUX peer client to authenticate the SMUX association.

The /etc/snmpd.peers file is modified with these changes automatically during installation.

6.2.3 Configuring the SNMP Manager

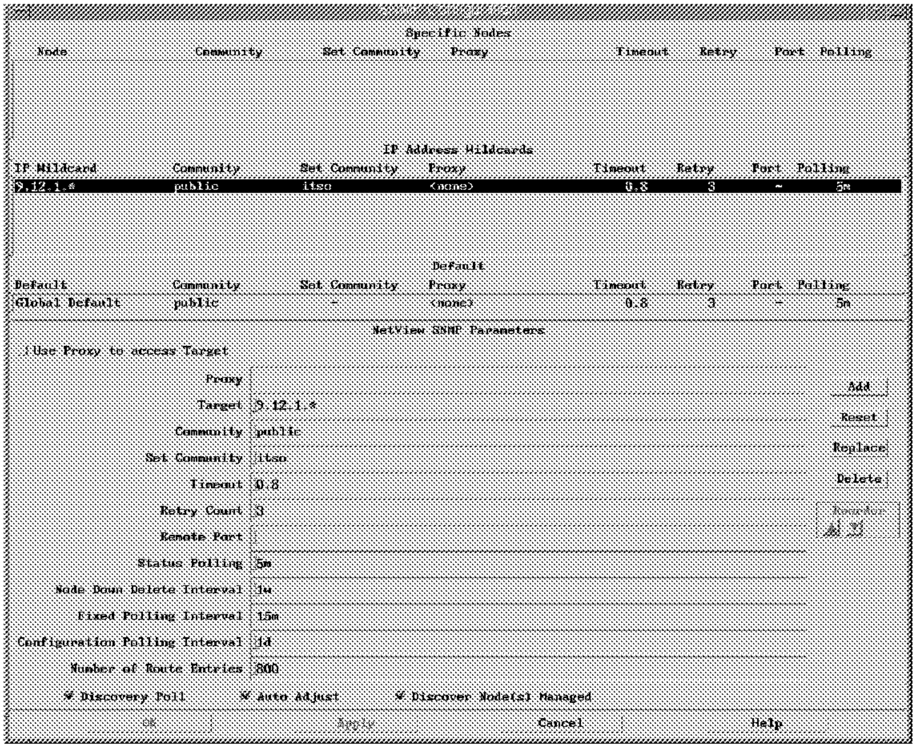
The SNMP manager must also be configured such that it can issue requests with the correct community name. The manager configuration file is called ovsnmp.conf and is located in the /usr/OV/conf directory on the NetView for AIX host.

```
          Manager Configuration (/usr/OV/conf/ovsnmp.conf)

9.12.1.*:public:*:8:3:300::itso:
*.*.*.*:public::8:3:300:::
```

The first entry in the file tells NetView to use the community name itso when issuing requests to nodes on Network 9.12.1. For communicating with all other nodes, the community name public will be used.

The manager can easily be configured by selecting the option **SNMP Configuration** from the Options field on the menu bar of NetView for AIX. This action updates the ovsnmp.conf file; it is the recommended way to update this file, and *should not* be edited in any other way.

Figure 99 on page 182 shows an SNMP configuration.



Specific Nodes

Node	Community	Set Community	Proxy	Timeout	Retry	Port	Polling

IP Address Wildcards

IP Wildcard	Community	Set Community	Proxy	Timeout	Retry	Port	Polling
9.12.1.*	public	itso	<none>	0.8	3	-	5m

Default

Default	Community	Set Community	Proxy	Timeout	Retry	Port	Polling
Global Default	public	-	<none>	0.8	3	-	5m

NetView SNMP Parameters

□ Use Proxy to access Target

Proxy:
Target: 9.12.1.*
Community: public
Set Community: itso
Timeout: 0.8
Retry Count: 3
Remote Port:
Status Polling: 5m
Node Down Delete Interval: 1w
Fixed Polling Interval: 15m
Configuration Polling Interval: 1d
Number of Route Entries: 800

☑ Discovery Poll ☑ Auto Adjust ☑ Discover Node(s) Managed

Figure 99. SNMP Configuration

If the configuration is incorrect, then numerous authentication failure traps appear on the NetView for AIX events console, and in trapd.log.

```
842997295  4  Tue Sep 17 16:54:55 1996 sp21en0.msc.itso.ibm.com A
       IBM Incorrect Community Name (authenticationFailure Trap)
```

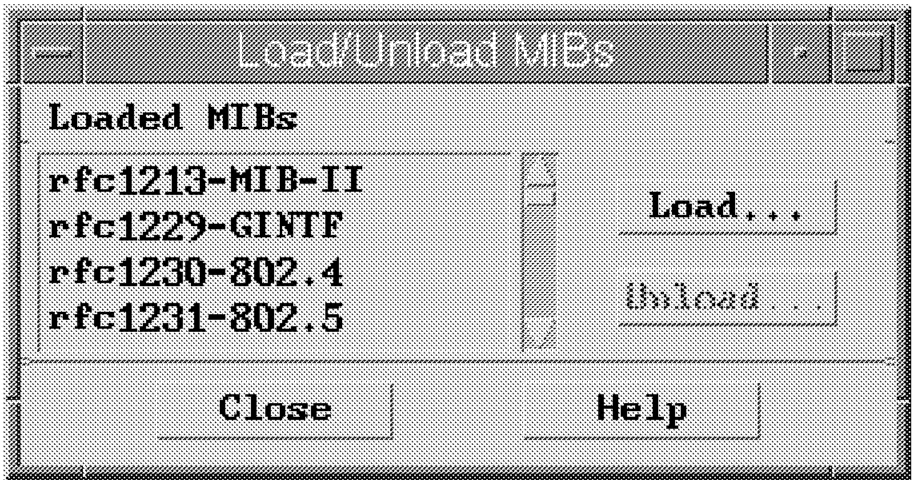
The community name is installation-specific, and all modifications must be done post-installation by the user.

6.2.4 Configuring the MIB

The SP MIB file /usr/lpp/ssp/config/snmp_proxy/ibmSPMIB.defs is appended to the /etc/mib.defs file. It is compiled from the ibmSPMIB.my file.

The /usr/lpp/ssp/config/snmp_proxy/ibmSPMIB.my file must be copied from the Control Workstation into the NetView manager directory as /usr/OV/snmp_mibs/ibmSP.MIB.

The MIB must be loaded into the SNMP manager, NetView for AIX (see Figure 100 on page 183), using **Options**→**Load/Unload**→**SNMP...** from the main NetView window.

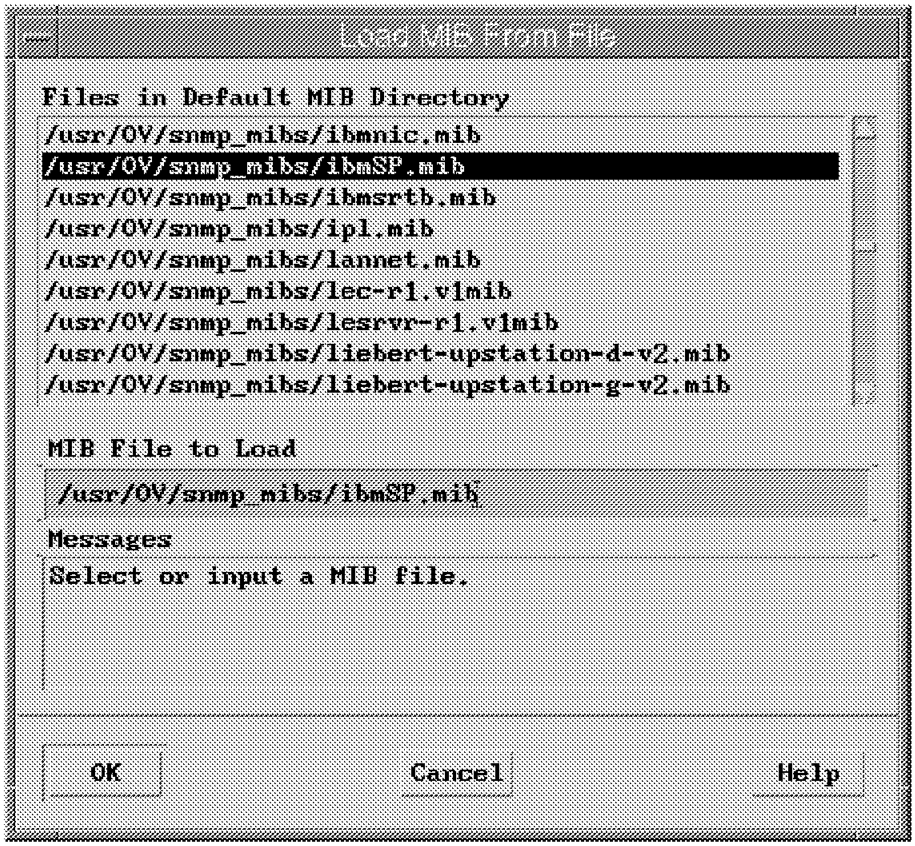


Loaded MIBs

rfc1213-MIB-II
rfc1229-GINTF
rfc1230-802.4
rfc1231-802.5

Figure 100. Loading a MIB

By selecting **Load**, the following panel is presented, where ibmSP.MIB is listed. Select this MIB file, and click **OK**, as shown in Figure 101.



Files in Default MIB Directory

```
/usr/OV/snmp_mibs/ibmnic.mib
/usr/OV/snmp_mibs/ibmSP.mib
/usr/OV/snmp_mibs/ibmsrtb.mib
/usr/OV/snmp_mibs/ipl.mib
/usr/OV/snmp_mibs/lannet.mib
/usr/OV/snmp_mibs/lec-r1.v1mib
/usr/OV/snmp_mibs/lesrvr-r1.v1mib
/usr/OV/snmp_mibs/liebert-upstation-d-v2.mib
/usr/OV/snmp_mibs/liebert-upstation-g-v2.mib
```

MIB File to Load: /usr/OV/snmp_mibs/ibmSP.mib

Messages: Select or input a MIB file.

Figure 101. Loading the MIB from File

6.2.5 Configure Subagent sp_configd

The sp_configd daemon is internally configured as an SMUX peer, or proxy agent, of the snmpd daemon on the Control Workstation and on each node of the SP. Figure 102 shows the relationship between snmpd and sp_configd.

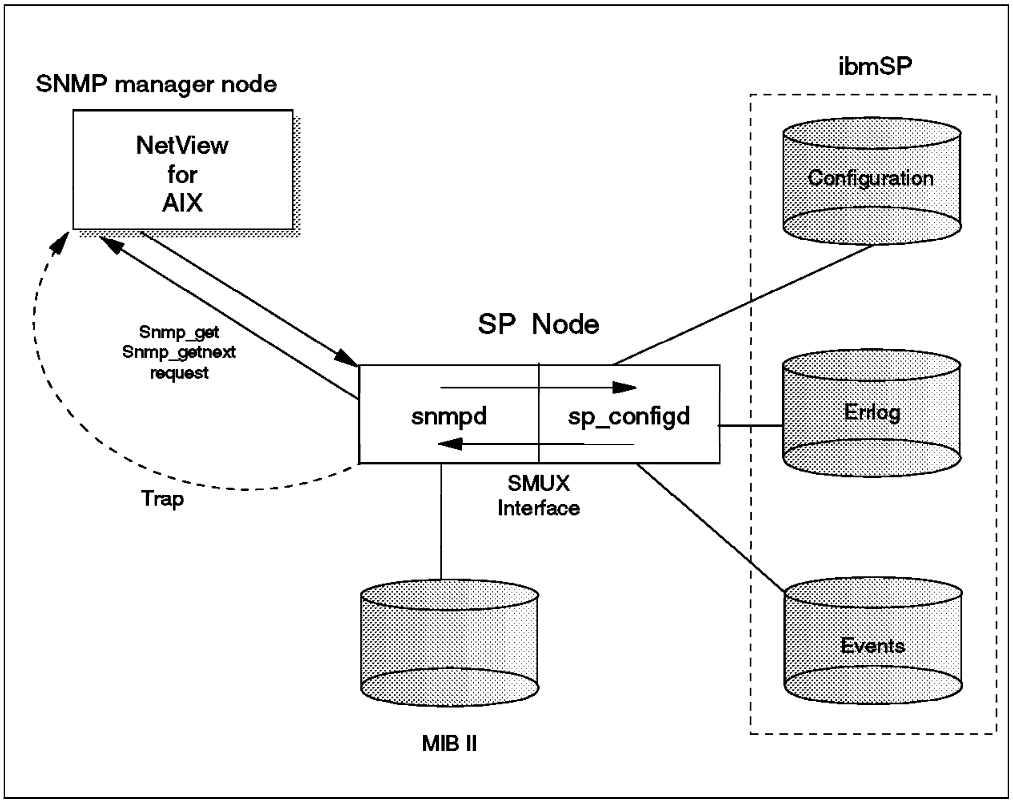


Figure 102. snmp and sp_configd

The sp_configd daemon provides the following functions:

- It receives requests from the network monitors for data from the ibmSP MIB (these requests are routed from the snmpd daemon to the sp_configd daemon over the SMUX interface). The results are returned by sp_configd via the SMUX interface and then sent back to the originating monitor by the snmpd agent.
- It sends trap notifications about events occurring on the SP to all hosts listed in the snmpd configuration file, /etc/snmpd.conf.

The snmpd should be active *before* the sp_configd daemon is started. The snmpd is controlled by the System Resource Controller (SRC) and is activated when the system is initialized, with the following command:

```
        startsrc -s snmpd
```

The AIX Object Data Manager (ODM) is updated with an src_entry for sp_configd, so that the daemon can be controlled using the following SRC commands:

- `startsrc`

  This starts a subsystem, a group of subsystems, or a subserver.

  The src_entry invokes the `startsrc` command with the following parameters:

```
startsrc
--------
-s sp_configd -a ″-t 600″ -p /usr/lpp/ssp/bin/spconfigd
-u 0 -i /dev/null -o /dev/null -e /dev/null -S -n 15 -F 15


per the ODM
-----------
subsysname = ″sp_configd″
synonym = ″″
cmdargs = ″-t600″
path = ″/usr/lpp/ssp/bin/sp_configd″
uid = 0
auditid = 0
standin = ″/dev/null″
standout = ″/dev/null″
standerr = ″/dev/null″
action = 2
multi = 0
contact = 2
svrkey = 0
svrmtype = 0
priority = 20
signorm = 15
sigforce = 15
display = 1
waittime = 20
grpname = ″″
```

The -t field determines the amount of time, in seconds, for which the data is to be cached by the daemon and refetched when a new request for the data is received. The default value is 60 seconds, but this can be changed.

An entry is added to the AIX inittab file to start the daemon by issuing the command:

```
        startsrc -s sp_configd
```

Issuing `startsrc` causes the sp_configd daemon to generate a coldStart trap. This should be displayed on the NetView events control desk as a trap.

```
Thu Sep 12 13:33:56 1996 sp21en0.msc.itso.ibm.com  A IBM Agent Up
with Possible Changes (coldStart Trap)
```

- `stopsrc`

  This command stops a subsystem such as sp_configd, a group of subsystems, or a subserver:

  ```
                 stopsrc -s sp_configd
  ```

- `lssrc`

  This command is used to acquire the status of the subsystem, group of subsystems, or subserver. The sp_configd daemon does not support the long status form of the `lssrc` command. However, snmpd does support the long status form of the `lssrc` command.

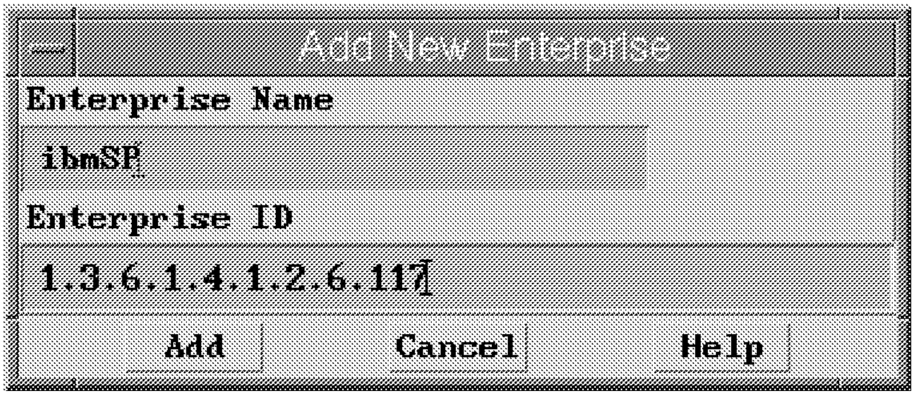
```
Logfile:         /usr/tmp/snmpd.log
Tracing:         ENABLED                ACTIVE
Debug level:     0
Max Packet Size: 32768
Query Timeout:   60
SMUX Timeout:    15

COMMUNITY:       itso
ADDRESS:         9.12.1.71
NETMASK:         255.255.255.255
PERMISSION:      readWrite
VIEW:            1.17.2

TRAP DESTINATION
COMMUNITY        ADDRESS
public           9.12.1.71

SMUX CLIENT:     1.3.6.1.4.1.2.6.117
PASSWORD:        sp_configd_pw
ADDRESS:         127.0.0.1
NETMASK:         255.255.255.255
```

The sp_configd daemon has several sessions with the Event Management subsystem. These sessions are used to maintain the SP Event Management variable instance data, and information from the last trap issued associated with that event.

6.2.6 Customizing Traps

NetView for AIX events can be configured by selecting **Options→Event Configuration→Trap Customization: SNMP...** from the menu bar. This opens the Event Configuration window with a list of enterprise names and IDs. The ibmSP enterprise is not listed and will have to be added.

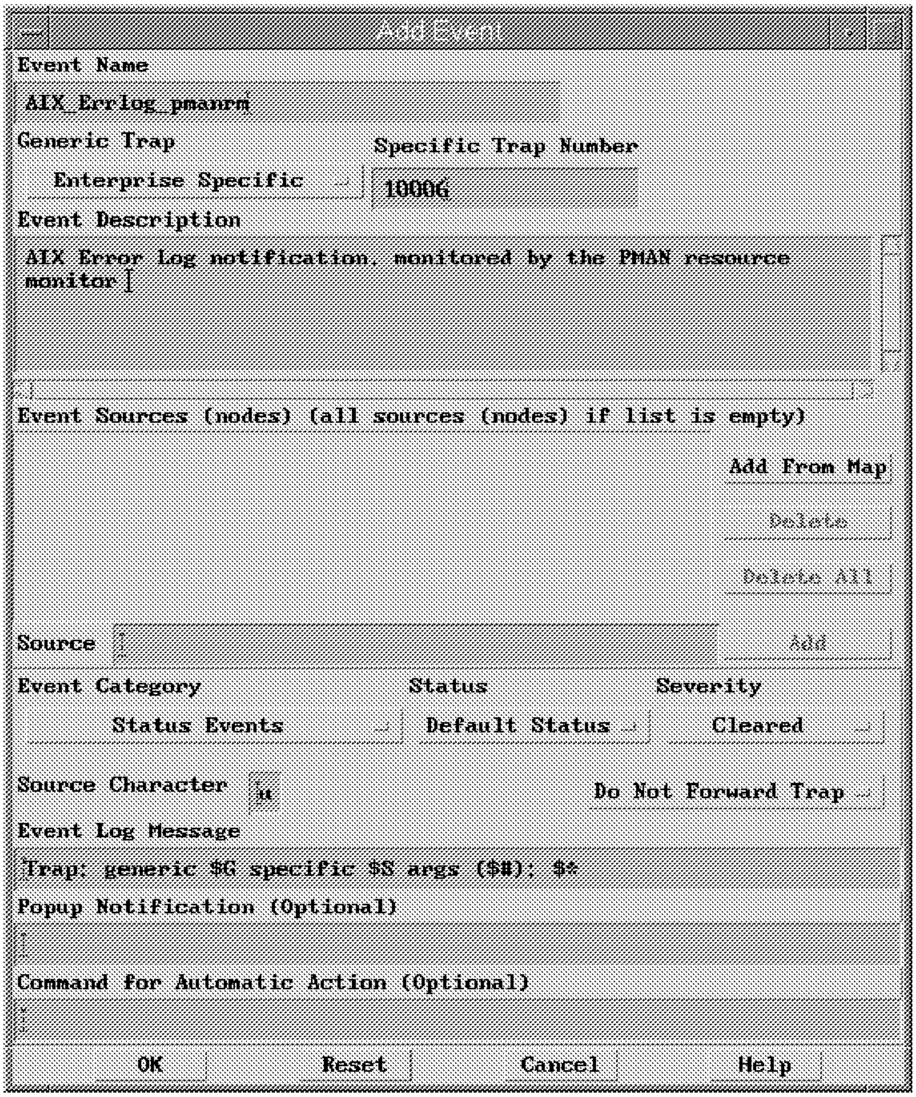
Step 1 Add an ibmSP (1.3.6.1.4.1.2.6.117) enterprise, as shown in Figure 103.



Enterprise Name
ibmSP
Enterprise ID
1.3.6.1.4.1.2.6.117

Figure 103. Add the ibmSP Enterprise

Step 2 In order to configure the Problem Management subsystem traps, select the ibmSP enterprise in the table and add all the appropriate traps, as shown in Figure 104 on page 187.



Add Event

Event Name: AIX_Errlog_pmanrm

Generic Trap: Enterprise Specific

Specific Trap Number: 10006

Event Description: AIX Error Log notification, monitored by the PMAN resource monitor

Event Sources (nodes) (all sources (nodes) if list is empty)

Source:

Event Category: Status Events

Status: Default Status

Severity: Cleared

Source Character: a

Do Not Forward Trap

Event Log Message: Trap: generic $G specific $S args ($#): $*

Popup Notification (Optional):

Command for Automatic Action (Optional):

Figure 104. Adding Predefined Traps

This will update the /usr/OV/conf/C/trapd.conf file.

This configuration file is used by the SNMP manager to activate the traps received.

```
Hardware_Errors_Arm {1.3.6.1.4.1.2.6.117} 6 10003 A 4 0 ″Status Events″
Trap: generic $G specific $S args ($#): $*
SDESC
Trap issued for SP hardware related events.
EDESC
```

If the traps sent from sp_configd are not configured for NetView for AIX, then the trapd daemon will receive traps that it will not be able to understand. Typically, a questioned trap is displayed as shown in the following screen. Therefore, it is important to configure these traps in the trapd.conf file.

```
    ? Trap found with no known format in trapd.conf(4)
```

In the case of the Event Management subsystem, the trap lists all of the objects and their instance values, which for an Event Management event are:

```
ibmSPEMEventID, ibmSPEMEventFlags, ibmSPEMEventTime,
ibmSPEMEventLocation, ibmSPEMEventPartitionAddress,
ibmSPEMEventVarsTableName, ibmSPEMEventVarsTableInstanceID,
ibmSPEMEventVarName, ibmSPEMEventVarValueInstanceVector,
ibmSPEMEventVarValuesTableInstanceID, ibmSPEMEventVarValue,
ibmSPEMEventPredicate.
```

An administrator familiar with Event Management events (the meaning of the objects is described in the ibmSP MIB) is able to understand the event that occurred. For many events, the administrator would want to automate a query to the SP trap originator to get the current value of the variable named in the trap to determine if the problem has been corrected by the local recovery code back on the SP. It is preferred, however, to ensure that all these traps are configured as in trapd.conf, to ensure operational consistency and operational management.

The traps should be configured such that:

- Helpful messages are displayed on the event card about the SP event.
- Windows pop up on the NetView console, or the Control Workstation, to gain the attention of the operators.
- Commands are automatically executed, without operator intervention.
- The category, status, and severity of the event will be reflected in the event display.

Step 3 Apply the Event Customization.

For more information about event customization, refer to *Examples of Using NetView for AIX V3*, GG24-4327.

With NetView for AIX V4, there is a further option for event configuration using event rulesets. These are briefly discussed in Appendix A, “Overview of NetView for AIX Ruleset Editor” on page 221, and more in-depth examples are provided in *Examples of Using NetView for AIX V4*, SG24-4515.

6.2.7 Topological View of SP

The configuration of the SP is provided with the SP proxy agent MIB. This MIB, ibmSP.MIB, contains SP-specifc SNMP configuration data, as shown in Figure 105 on page 189. This enables SNMP managers to view the SP as a network of SP processor nodes that will be attached to the Control Workstation. Figure 105 on page 189 illustrates the Control Workstation sp21cw0 attached to Network 9.12.1, which entails the SP nodes.

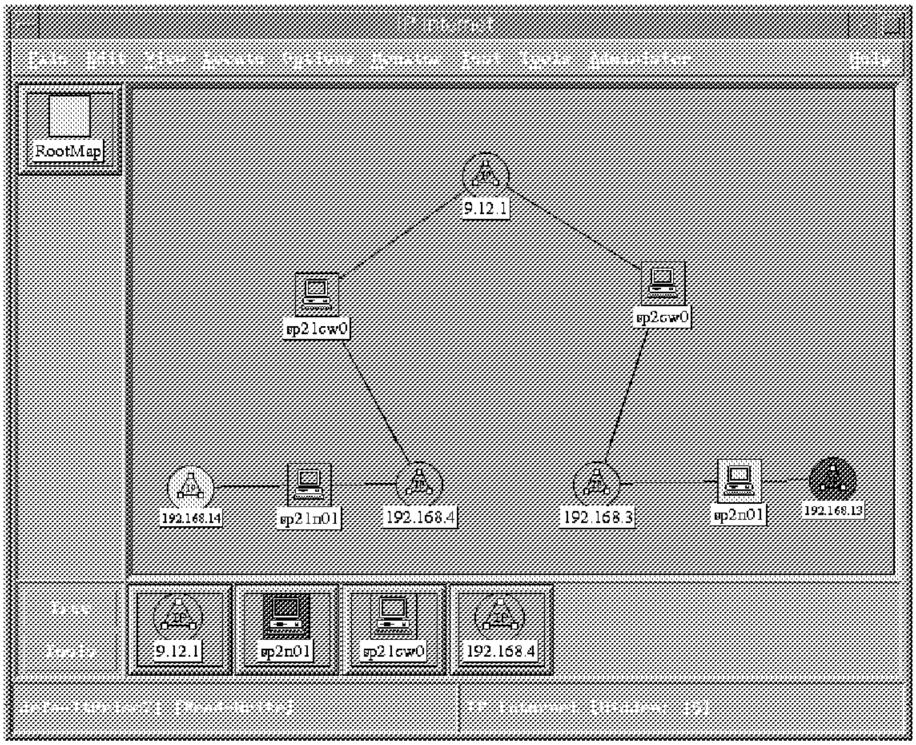


Figure 105. Topological Representation of the IBM SP

Topology is determined by the IPMAP application as usual. There is no interpretation of the ibmSP MIB to determine the topology. The figure shows a typical Netview submap that is determined by the Netview IPMAP application. The Netview topology depends entirely upon where the Netview application is installed. If Netview is installed on one of the nodes, the submap will show all the host nodes at the highest level. However, if the Netview application is on an external RS/6000, outside the SP environment, then the topology will differ, and in fact be as represented in the diagram.

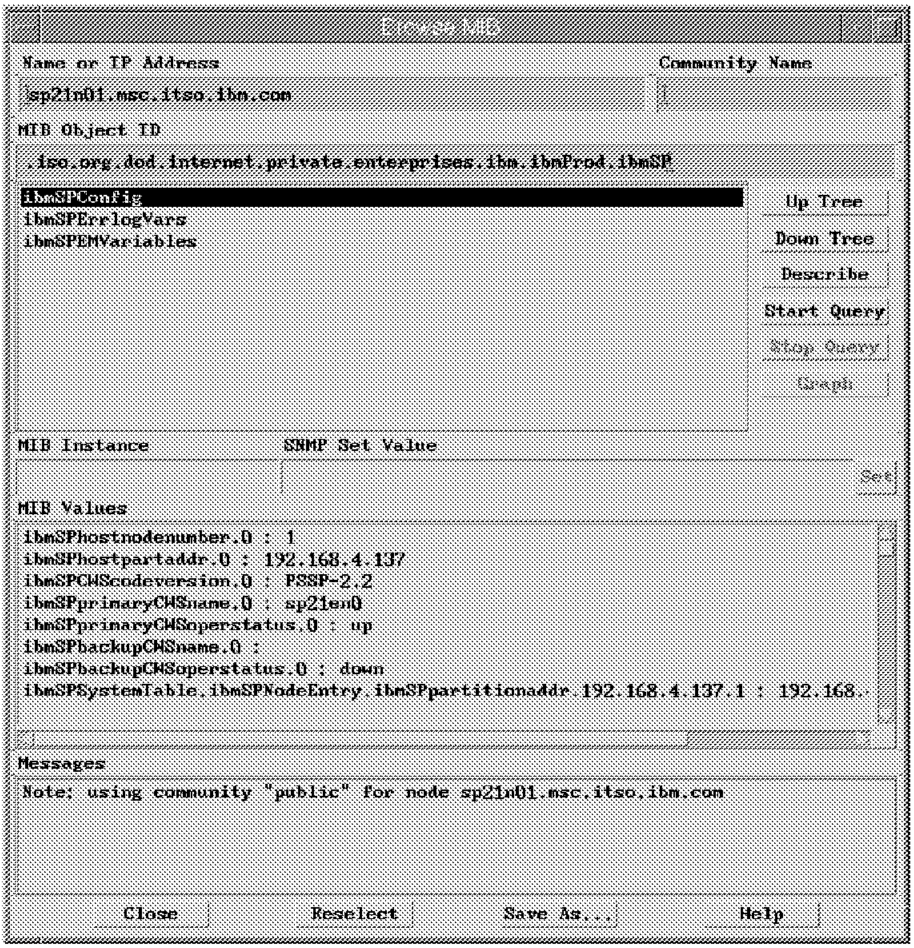
6.2.8 The ibmSP MIB

The ibmSP MIB is provided to define SP-specific information. It consists of three groups of objects containing pertinent SP information:

1. ibmSPConfig
2. ibmSPErrlogVars
3. ibmSPEMVariables

ibmSPConfig

The ibmSPConfig group of the ibmSP.MIB (see Figure 106 on page 190) defines objects containing SP system configuration information. The group is instantiated by every processor node in the SP as well as on the Control Workstation.



Name or IP Address: sp21n01.msc.itso.ibm.com

Community Name:

MIB Object ID: .iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmSP

```
ibmSPConfig
ibmSPErrlogVars
ibmSPEMVariables
```

MIB Instance:

SNMP Set Value:

MIB Values

```
ibmSPhostnodenumber.0 : 1
ibmSPhostpartaddr.0 : 192.168.4.137
ibmSPCWScodeversion.0 : PSSP-2.2
ibmSPprimaryCWSname.0 : sp21en0
ibmSPprimaryCWSoperstatus.0 : up
ibmSPbackupCWSname.0 :
ibmSPbackupCWSoperstatus.0 : down
ibmSPSystemTable.ibmSPNodeEntry.ibmSPpartitionaddr.192.168.4.137.1 : 192.168.
```

Messages

```
Note: using community "public" for node sp21n01.msc.itso.ibm.com
```

Figure 106. MIB Browser - ibmSPConfig

The ibmSPConfig group defines objects containing the following information:

- The queried subagent node number
- The IP address of the system partition
- The hostname of the primary Control Workstation
- The operational state of the primary Control Workstation
- The hostname of the backup Control Workstation, if it exists

  If your system does not have a backup Control Workstation, this field will be null.
- The operational state of the backup Control Workstation
- The version number of the IBM Parallel System Support Programs for AIX (IBM PSSP), running on the Control Workstation
- The table of SPNodeEntry

  Each row in the table is indexed by the IP address of a system partition combined with a node number, and contains the following information about the node:

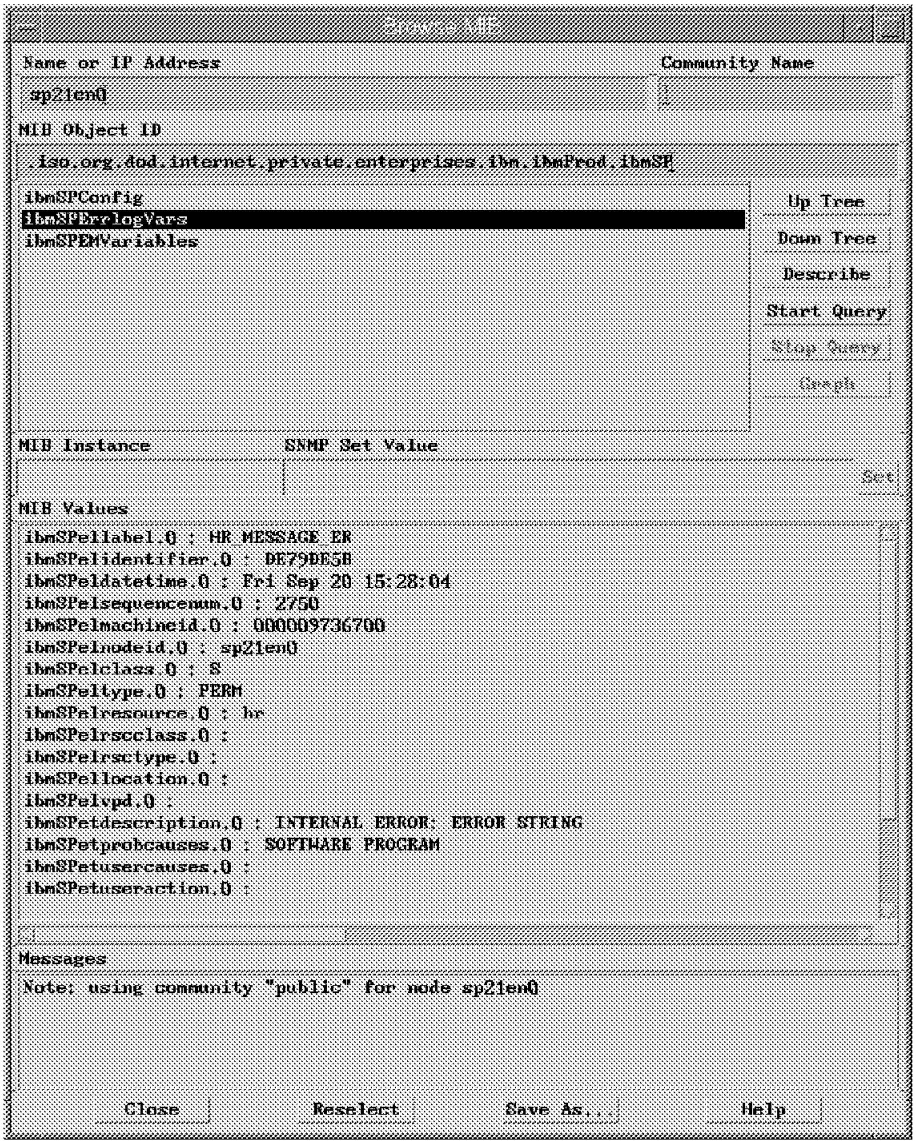
  – The IP address of the system partition in which it resides

  – The node number

- The number of the frame containing this node
- The lowest slot number in the frame occupied by the node
- A reliable hostname assigned to the node, which is the hostname associated with the SP Ethernet
- The initial hostname assigned to the node, which is the hostname that is assigned to the node during customization
- The name of the system partition in which the node resides
- The version number of the IBM Parallel System Support Programs for AIX (IBM PSSP) running on the node

ibmSPErrlogVars

This group of objects is instantiated on each SP processor node and on the Control Workstation. The group consists of a sequence of objects (see figref refid=errlog.) containing information about the last error log write that occurred on the reporting node.



Name or IP Address: sp21en0

Community Name:

MIB Object ID: .iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmSP

```
ibmSPConfig
ibmSPErrlogVars
ibmSPEMVariables
```

MIB Instance:

SNMP Set Value:

MIB Values

```
ibmSPellabel.0 : HR_MESSAGE_ER
ibmSPelidentifier.0 : DE79DE5B
ibmSPeldatetime.0 : Fri Sep 20 15:28:04
ibmSPelsequencenum.0 : 2750
ibmSPelmachineid.0 : 000009736700
ibmSPelnodeid.0 : sp21en0
ibmSPelclass.0 : S
ibmSPeltype.0 : PERM
ibmSPelresource.0 : hr
ibmSPelrscclass.0 :
ibmSPelrsctype.0 :
ibmSPellocation.0 :
ibmSPelvpd.0 :
ibmSPetdescription.0 : INTERNAL ERROR: ERROR STRING
ibmSPetprobcauses.0 : SOFTWARE PROGRAM
ibmSPetusercauses.0 :
ibmSPetuseraction.0 :
```

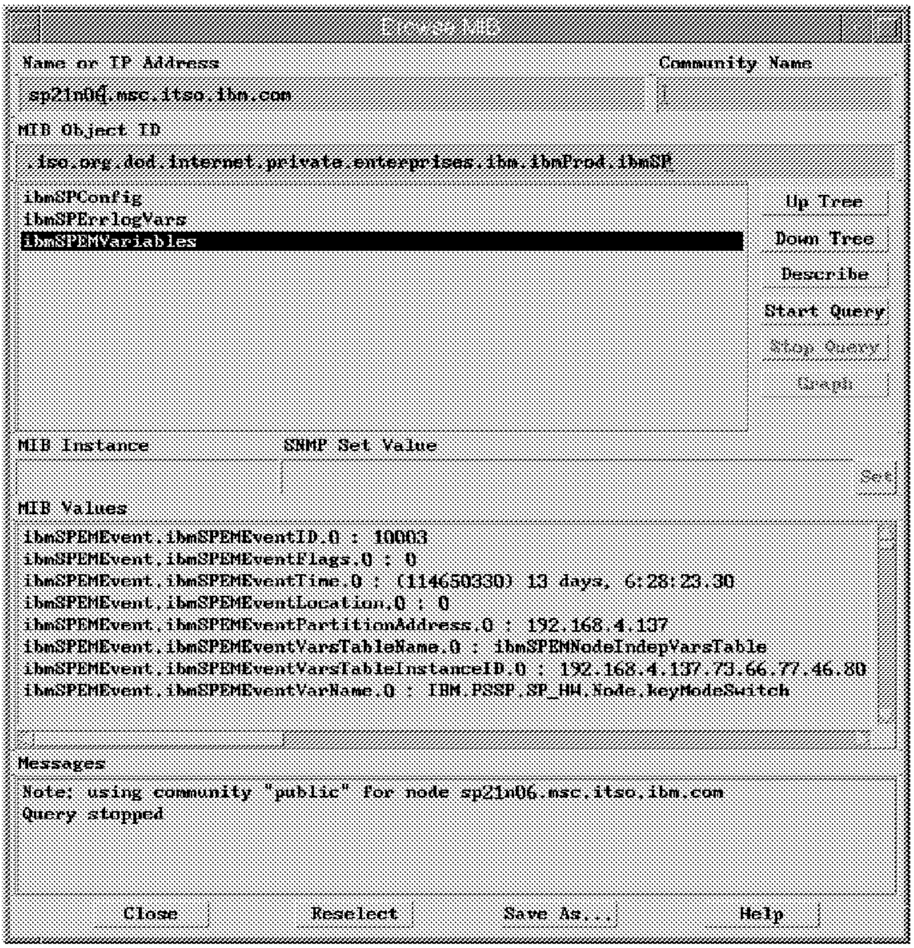
Messages

```
Note: using community "public" for node sp21en0
```

Figure 107. MIB Browser - ibmSPErrlogVars

ibmSPEMVariables

This group is instantiated on each SP processor node and on the Control Workstation. It consists of objects representing Event Management variables and contains up to three tables, as shown in Figure 108 on page 193.



Name or IP Address: sp21n06.msc.itso.ibm.com

Community Name:

MIB Object ID: .iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmSP

```
ibmSPConfig
ibmSPErrlogVars
ibmSPEMVariables
```

MIB Instance

SNMP Set Value

MIB Values

```
ibmSPEMEvent.ibmSPEMEventID.0 : 10003
ibmSPEMEvent.ibmSPEMEventFlags.0 : 0
ibmSPEMEvent.ibmSPEMEventTime.0 : (114650330) 13 days, 6:28:23.30
ibmSPEMEvent.ibmSPEMEventLocation.0 : 0
ibmSPEMEvent.ibmSPEMEventPartitionAddress.0 : 192.168.4.137
ibmSPEMEvent.ibmSPEMEventVarsTableName.0 : ibmSPEMNodeIndepVarsTable
ibmSPEMEvent.ibmSPEMEventVarsTableInstanceID.0 : 192.168.4.137.73.66.77.46.80
ibmSPEMEvent.ibmSPEMEventVarName.0 : IBM.PSSP.SP_HW.Node.keyModeSwitch
```

Messages

```
Note: using community "public" for node sp21n06.msc.itso.ibm.com
Query stopped
```

Figure 108. MIB Browser - ibmSPEMVariables

1. **ibmSPEMNodeDepVarsTable**

   This is a table of objects that allows access to the definitions of node-dependent Resource Variables, which are normally defined with an Event Manager locator attribute.

   The table only contains object instantiations for those variables that have a locator value equalling the node number of the node on which the SP proxy agent is running.

2. **ibmSPEMNodeIndepVarsTable**

   This is a table of objects that allows access to the definitions of node-independent Resource Variables, which are normally defined without an Event Manager locator attribute.

3. **ibmSPEMVarValuesTable**

   This is a table that allows access to the current values assigned to the Event Management Resource Variables.

For each of the Event Management Resource Variables in the ibmSPEMNodeDepVarsTable and ibmSPEMNodeIndepVarsTable tables, there is information about the object instantiations, such as:

- The name of the Resource Variable
- A description of the Resource Variable

- The variable type (Counter, Quantity or State)
- The data type (long, float, or structured byte string)
- The initial variable value
- An index into ibmSPEMVarValuesTable, which, when used in combination with an instantiation vector, provides access to an entry containing the current value of a variable instance
- The variable Resource Class
- The instantiation vector definition
- A description of each element in the instantiation vector
- The name used to read and write the variable in the PTX-shared memory (this may be null)
- The default predicate for event notification (this may be null)
- A description of the event
- The locator value specified for the variable
- The resource order group

The ibmSPEMVarValuesTable table provides object instantiations for the current values of variables from both ibmSPEMNodeDepVarsTable and ibmSPEMNodeIndepVarsTable.

The portion of the ibmSPEMVariable group instantiated by the sp_configd subagent depends on the type of SP node the subagent is running on.

The ibmSPEMNodeDepVarsTable and ibmSPEMVarValuesTable tables are instantiated on all the processor nodes and the Control Workstation.

The ibmSPEMNodeIndepVarsTable table is instantiated only on the Control Workstation.

6.2.9 Viewing the ibmSP MIB

The ibmSP MIB group objects can be viewed using the snmpinfo command, or, if available, the NetView for AIX MIB Browser.

6.2.9.1 The snmpinfo Command

The AIX snmpinfo command may be used in place of a manager client, such as NetView for AIX, to view the contents of the ibmSP MIB. This command is part of the BOS fileset:

```
bos.net.tcp.server 4.1.4.0
```

This command may be used in place of an snmpget query from an SNMP manager, such as NetView for AIX, to view the contents of the ibmSP MIB.

snmpinfo requires root authentication, and its usage can best be explained with some examples:

- To dump the contents of the ibmSP MIB located on Node sp21n01 in verbose mode, enter the command:

```
snmpinfo  -c itso -m dump -v -h sp21n01 ibmSP
---------------------------------------------

ibmSPhostnodenumber.0 = 1
ibmSPhostpartaddr.0 = 192.168.4.137
ibmSPCWScodeversion.0 = ″PSSP-2.2″
ibmSPprimaryCWSname.0 = ″sp21en0″
ibmSPprimaryCWSoperstatus.0 = 1
ibmSPbackupCWSname.0 = ″″
ibmSPbackupCWSoperstatus.0 = 2
ibmSPpartitionaddr.192.168.4.137.1 = 192.168.4.137
ibmSPpartitionaddr.192.168.4.137.5 = 192.168.4.137
```

- To dump the contents of the ibmSPConfig group without the verbose mode would result in the following:

```
snmpinfo  -c itso -m dump -h sp21n01 ibmSPConfig
------------------------------------------------

1.3.6.1.4.1.2.6.117.1.1.0 = 1
1.3.6.1.4.1.2.6.117.1.2.0 = 192.168.4.137
1.3.6.1.4.1.2.6.117.1.3.0 = ″PSSP-2.2″
1.3.6.1.4.1.2.6.117.1.4.0 = ″sp21en0″
1.3.6.1.4.1.2.6.117.1.5.0 = 1
1.3.6.1.4.1.2.6.117.1.6.0 = ″″
1.3.6.1.4.1.2.6.117.1.7.0 = 2
1.3.6.1.4.1.2.6.117.1.8.1.1.192.168.4.137.1 = 192.168.4.137
1.3.6.1.4.1.2.6.117.1.8.1.1.192.168.4.137.5 = 192.168.4.137
```

- To retrieve the values of a specific MIB variable, located on the local host, such as ibmSPhostnodenumber, enter:

```
snmpinfo  -c itso -m get  ibmSPhostnodenumber.0
-----------------------------------------------

1.3.6.1.4.1.2.6.117.1.1.0 = 0
```

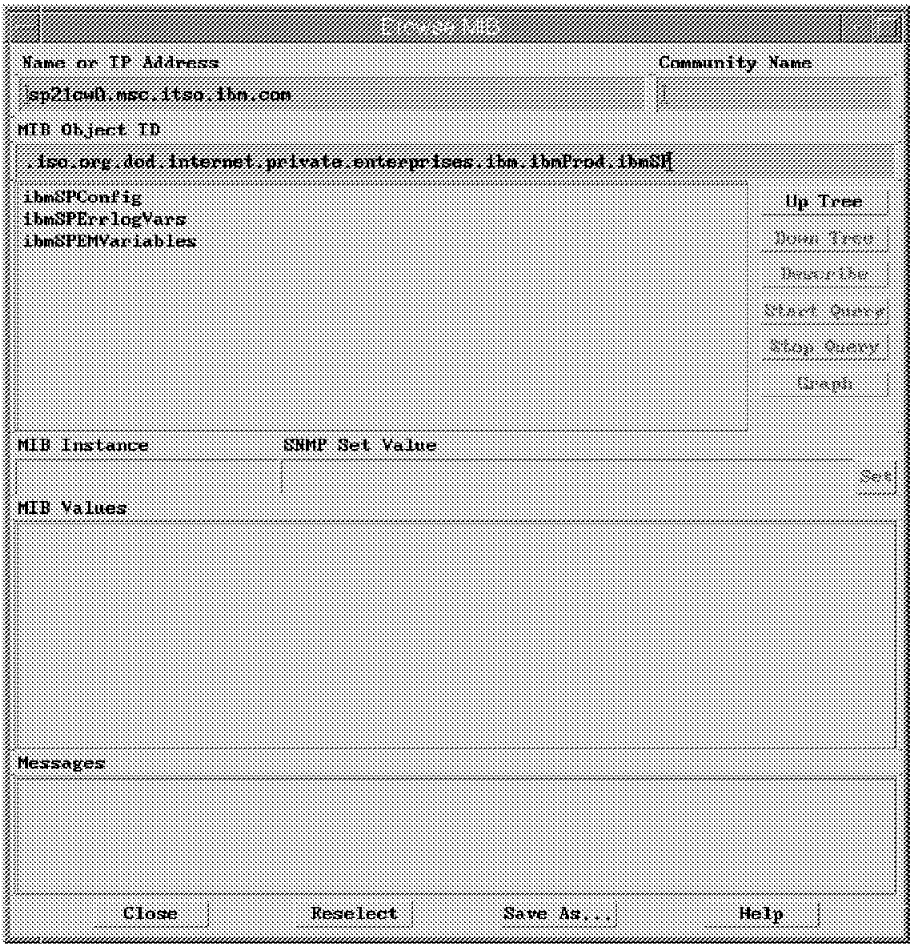
- To retrieve the value of the MIB variable following the ibmSPhostnodenumber variable (in other words, the variable at 1.3.6.1.4.1.2.6.117.1.2.0), enter:

```
snmpinfo  -c itso -m next ibmSPhostnodenumber.0
-----------------------------------------------

1.3.6.1.4.1.2.6.117.1.2.0 = 192.168.4.137
```

Only snmpinfo is available on the SP nodes and Control Workstation to query MIBs. The snmpget command will not usually be available, except on the NetView for AIX machine.

6.2.9.2 NetView for AIX MIB Browser

If NetView for AIX is accessible, it is better to browse the MIB using the NetView for AIX MIB Browser, as shown in Figure 109.



Name or IP Address: sp21cw0.msc.itso.ibm.com

Community Name:

MIB Object ID: .iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmSP

ibmSPConfig
ibmSPErrlogVars
ibmSPEMVariables

MIB Instance

SNMP Set Value

MIB Values

Messages

Figure 109. MIB Browser

The MIB Browser is used to query and set MIB values for both Internet-standard and enterprise-specific MIB objects, and also to graph MIB values, including specific instances.

The MIB Browser is used to display MIB values of a node by entering the name or IP address of the node, and the community name of the agent that is running on the selected node. Then, with the use of the Up and Down buttons, you can *walk* the MIB tree. Continue this sequence of walking the MIB tree until you reach the MIB object upon which the SNMP request is to be performed.

When selecting the MIB object, or specific instance of the object, a query can be performed. If the community name is public, or if the agent′s community name is configured in the SNMP Configuration dialog box, it is feasible to issue a Get request query without having to specify a community name. The application automatically displays the community name configured for the selected agent, for example public, or the configured community name, such as itso, as soon as you select an item from the MIB Object ID selection list. If the community name is not valid, then the Messages area will indicate the success or failure of the operation, as in the following example.

```
   Timed Out
   A timeout can result from an invalid community name
```

6.3 The AIX Error Log

In order for the RS/6000 SP to be part of an existing network management system, the network managers must be notified of the AIX error log or Event Management events that need to be centrally managed.

Therefore, there must be a way to create SNMP traps for SP AIX error log errors.

The RS/6000 SP uses both the BSD syslog and the AIX Error Logging facilities, as well as a number of function-specific log files, to record error events on each node.

Log management functions are built upon the Sysctl facility, which uses the SP authentication services. Generating parallel AIX error log and BSD syslog reports and performing general log viewing require that the user issue the `kinit` command to be identified to the SP authentication services.

All other log management commands additionally require that the user be defined as a principal in the /etc/logmgmt.acl file. All users defined in this file must also be placed in the authentication database as a principal.

It is worth noting that most log management represents administrative tasks that normally require root authority, and that a user defined in the logmgmt.acl file will execute commands as the root user.

```
#acl#
# This sample acl file for log management commands contains a
commented line for a principal
_PRINCIPAL root.admin@MSC.ITSO.IBM.COM
# The following principal is added to be able to trim SPdaemon.log
from cleanup.logs.ws
_PRINCIPAL rcmd.sp21en0
```

The file contains entries for the user root as principal root.admin, and the principal rcmd. This gives both principals authority to execute log management commands.

6.3.1 SNMP Subagent (sp_configd) Monitor of AIX Error Log

The snmp_trap_gen error notification method and the sp_configd daemon create SNMP traps when selected error types are recorded in the AIX error log.

This is similar to the NetView for AIX subagent, called trapgend, which also converts AIX error log events into SNMP traps. Therefore, only one of these methods should be used for monitoring the AIX error log.

```
errnotify:
        en_pid = 0
        en_name = ″snmp_trap_gen″
        en_persistenceflg = 1
        en_label = ″″
        en_crcid = 0
        en_class = ″″
        en_type = ″″
        en_alertflg = ″TRUE″
        en_resource = ″″
        en_rtype = ″″
        en_rclass = ″″
        en_symptom = ″″
        en_method = ″/usr/lpp/ssp/bin/snmp_trap_gen $1″
```

The method, /usr/lpp/ssp/bin/snmp_trap_gen, and the daemon, sp_configd, both run on the same processor node and Control Workstation composing the SP system. In order for a trap to be created, error log entries on each node and Control Workstation must contain an Alert=true attribute.

The following steps describe the flow of events for creating an SNMP trap for an AIX error log event:

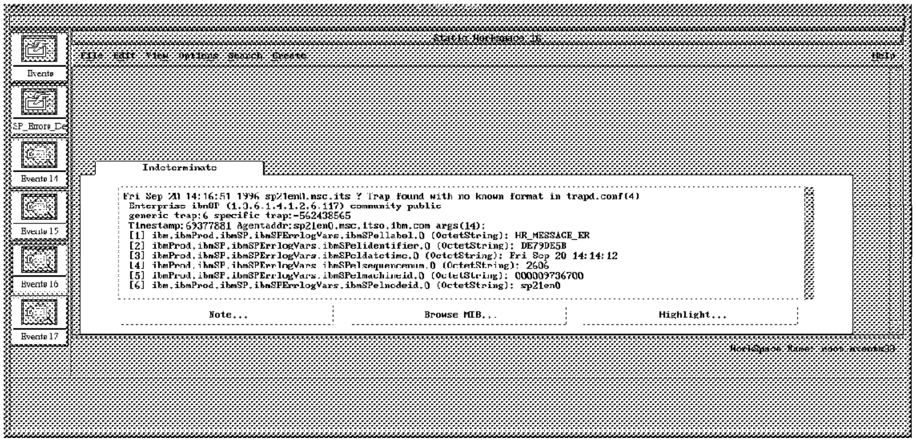
1. An application or subsystem writes to the AIX error log facility.
2. If the error log entry contains an `en_alertflg = TRUE` attribute, the snmp_trap_gen error notification method runs.
3. The snmp_trap_gen method uses the sequence number from the Error Log entry as input to the `errpt -a` command to obtain full information about the event.
4. The snmp_trap_gen process places the event information in a FIFO file, /var/rmp/errlog_entry.
5. The sp_configd daemon reads the FIFO file, parses the information into objects within the ibmSPErrlogVars group of the ibmSP MIB, and creates a trap from the objects whose instantiations contain non-null values.
6. The sp_configd sends the trap to the SNMP agent, snmpd, which in turn sends the trap to network managers specified in the /etc/snmpd.conf file existing on the node.

The following information is provided in SNMP traps from the AIX error log events generated by sp_configd:

- The enterprise field contains the object identifier of the sp_configd subagent.
- The specific-trap field contains the error ID from the error log entry. This value is the decimal notation of the AIX error ID.
- When the corresponding information exists in the error log entry, the variable bindings field contains the following object values paired with their IDs:
  - Error label
  - Error ID
  - Error log entry timestamp
  - Unique sequence number

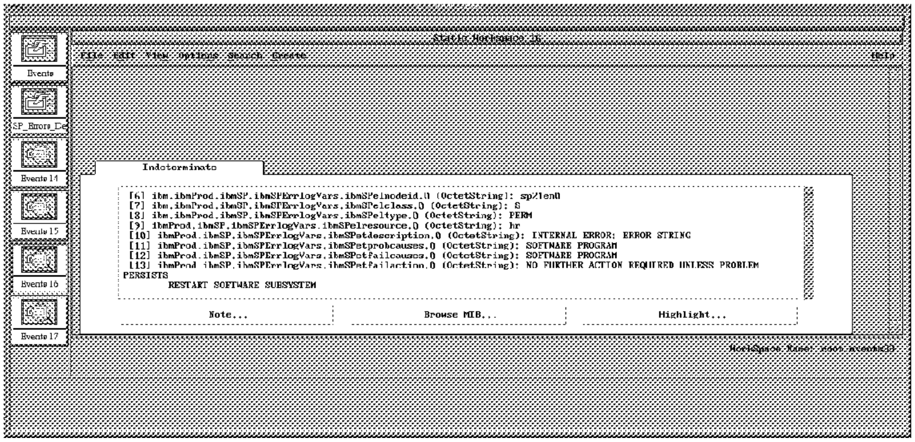
- Machine ID parameter
- Error class
- Error type
- Resource name
- Resource class
- Resource type
- Location code of a device
- Vital product data
- Error description
- Probable causes
- User causes
- User actions
- Install causes
- Install actions
- Failure causes
- Failure actions
- Detail data

This information will be displayed by the events application in the Control Desk, as shown in Figure 110 and Figure 111 on page 200.



```
Fri Sep 20 14:16:51 1996 sp21en0.msc.its ? Trap found with no known format in trapd.conf(4)
 Enterprise ibmSP (1.3.6.1.4.1.2.6.117) community public
 generic trap:6 specific trap:-562438565
 Timestamp:69377881 Agentaddr:sp21en0.msc.itso.ibm.com args(14):
 [1] ibm.ibmProd.ibmSP.ibmSPErrlogVars.ibmSPellabel.0 (OctetString): HR_MESSAGE_ER
 [2] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPelidentifier.0 (OctetString): DE79DE5B
 [3] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPeldatetime.0 (OctetString): Fri Sep 20 14:14:12
 [4] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPelsequencenum.0 (OctetString): 2606
 [5] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPelmachineid.0 (OctetString): 000009736700
 [6] ibm.ibmProd.ibmSP.ibmSPErrlogVars.ibmSPelnodeid.0 (OctetString): sp21en0
```

Figure 110. NetView Subagent Event (Page 1)



```
[6] ibm.ibmProd.ibmSP.ibmSPErrlogVars.ibmSPelnodeid.0 (OctetString): sp21en0
[7] ibm.ibmProd.ibmSP.ibmSPErrlogVars.ibmSPelclass.0 (OctetString): S
[8] ibm.ibmProd.ibmSP.ibmSPErrlogVars.ibmSPeltype.0 (OctetString): PERM
[9] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPelresource.0 (OctetString): hr
[10] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPetdescription.0 (OctetString): INTERNAL ERROR; ERROR STRING
[11] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPetprobcauses.0 (OctetString): SOFTWARE PROGRAM
[12] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPetfailcauses.0 (OctetString): SOFTWARE PROGRAM
[13] ibmProd.ibmSP.ibmSPErrlogVars.ibmSPetfailaction.0 (OctetString): NO FURTHER ACTION REQUIRED UNLESS PROBLEM
PERSISTS
        RESTART SOFTWARE SUBSYSTEM
```

Figure 111. NetView Subagent Event (Page 2)

If, however, the AIX error log entry is *not* set to alertable, then these errors will not be sent as SNMP traps. Therefore, the Error Record template for the entries must have a specific Alert field set to the value true. This can easily be achieved as root user:

1. To determine what SP errors are provided, run errpt -t.

```
28DB7CA6 HPS_FAULT9_ER       PERM H  Switch Adapter - Bus error
314A1C17 HPS_FAULT9_ER       PERM H  Switch Adapter - Bus error
```

2. Create a file that contains the two lines per error entry that you wish to be designated as alertable:

```
       =28DB7CA6:
       Alert=true

       =314A1C17:
       Alert=true
```

   Note that there must be a blank line between the different entry updates.

3. From the command line, run the command:

```
      /bin/errupdate file
```

4. To verify that the alert status has been changed to true (or 1), issue the following command to list all the alertable error templates:

```
      errpt -tF alert=1 | grep 28DB7CA6
```

5. If you wish to stop specific errors from alerting an SNMP manager, this can be done by reversing step 2, as follows:

```
      =28DB7CA6:
      Alert=false

      =314A1C17:
      Alert=false
```

6.3.2 Subsystem Logs

6.3.2.1 High Availability Subsystem Logs

The High Availability subsystem logs are located in /var/ha. For more information, refer to Chapter 1, “Overview” on page 3.

6.3.2.2 SNMP Agent and sp_configd Subagent Logs

The logs for the SNMP agent and subagent are held in the /var/tmp directory.

```
p---rw----   1 root     system         0 Sep 09 10:02 em_trap.192.168.4.
137
p---rw----   1 root     system         0 Sep 09 15:30 em_trap.192.168.4.
138
p---rw----   1 root     system         0 Sep 09 10:02 errlog_entry
-rw-r--r--   1 root     system      2557 Sep 10 14:43 snmpd.log
-rw-r--r--   1 root     system      2443 Sep 11 13:43 sp_configd.log
```

Pipes: The top three files are pipes that are used by sp_configd to hold the information that it needs to pass to the SNMP manager. These files are binary and cannot be viewed. When on the Control Workstation, there is one em_trap pipe per system partition; when on a processor node, there is only one em_trap pipe since a node cannot belong to more that one partition at the time. There is only one errlog_entry on a node or Control Workstation.

It is worth noting the date of usage for problem determination purposes. As described earlier, the pmand daemon writes into the em_trap pipe(s). The snmp_trap_gen method writes into the errorlog_entry pipe. The event response is supplied by the Event Management subsystem to this FIFO file or pipe. The sp_configd daemon accesses this data from the pipe and creates an SNMP trap. Hence, if this file is empty or untouched for a considerable period, then it is feasible that pman is not writing to this pipe.

If sp_configd is started with the trace option on, then the error log /var/tmp/sp_configd.log will provide more information when the subagent is initialized.

```
      startsrc -s sp_configd -a ″-T″
```

This log will typically contain information about the interaction between subagent initialization and registration.

```
sp_confi 10936 (root    )  sigterm signal received
sp_confi 10520 (root    )  sp_configd entered
sp_confi 10520 (root    )  time_in_cache value (-t) set to 600
sp_confi 42530 (root    )  child process opened errlog_entry FIFO for reading
sp_confi 42530 (root    )  child process opened
                           /var/tmp/em_trap.192.168.4.137
                           FIFO for reading
sp_confi 42530 (root    )  em_init: em connection with file
                           descriptor 8 opened for partition
                           192.168.4.137
sp_confi 42530 (root    )  em_init: em connection with file
                           descriptor 9 opened for partition
                           192.168.4.137
sp_confi 42530 (root    )  em_init: em monitor connection with file
                           descriptor 10 opened for partition
                           192.168.4.137
sp_confi 42530 (root    )  em_init: em monitor connection with file
                           descriptor 11 opened for partition
                           192.168.4.137
sp_confi 42530 (root    )  start_smux: SMUX open -
                           1.3.6.1.4.1.2.6.117 ”sp_configd”
sp_confi 42530 (root    )  start_smux: SMUX register -
                           readOnly 1.3.6.1.4.1.2.6.117.1 in=-

sp_confi 42530 (root    )  start_smux: SMUX register -
                           readOnly 1.3.6.1.4.1.2.6.117.2 in=-

sp_confi 42530 (root    )  start_smux: SMUX register -
                           readOnly 1.3.6.1.4.1.2.6.117.3 in=-

sp_confi 42530 (root    )  SMUX register: readOnly
                           1.3.6.1.4.1.2.6.117.1 out=0
sp_confi 42530 (root    )  SMUX register: readOnly
                           1.3.6.1.4.1.2.6.117.2 out=0
sp_confi 42530 (root    )  SMUX register: readOnly
                           1.3.6.1.4.1.2.6.117.3 out=0
sp_confi 42530 (root    )  Cold start trap issued
sp_confi 42530 (root    )  doit_em: registered resource
                           variable query response received for
                           partition 192.168.4.137 containing 366
                           variables
sp_confi 42530 (root    )  doit_em: issuing ha_em_send_command to
                           obtain the current nodes
                           of 159 variables from the EM
                           residing in partition 192.168.4.137
sp_confi 42530 (root    )  doit_em: issuing ha_em_send_command to
                           obtain the current node
                           values of 207 variables from the EM
                           residing in partition 192.168.4.137
sp_confi 42530 (root    )  doit_em: registered resource variable
                           query response received for partition
                           192.168.4.138 containing 366 variables
sp_confi 42530 (root    )  doit_em: issuing ha_em_send_command to obtain
                           the current node-independent
                           values of 207 variables from the
                           EM residing in partition 192.168.4.138
```

If the trace option **-t** is not set, then sp_configd will just log that it received a sigterm:

```
    9/11 14:00:33 sp_confi 10936 (root    )  sigterm signal received
```

6.3.2.3 SNMP Log

If there is an entry in the snmpd.log of the form shown below, then there is a problem between snmpd and sp_configd.

```
      lost peer (SMUX 127.0.0.1+4346+2)
```

It is possible to use the Problem Management subsystem to monitor logs for particular strings or keywords that are suspicious.

The Problem Management subsystem can be configured such that it can subscribe to the Event Management subsystem for a particular event. The event (in this case, to monitor a pman user_state Resource Variable) will contain a number. This number is determined by the Problem Management Resource Monitor that evaluates the expression. In this case, the keyword is `lost` in the /var/adm/snmpd.log file, and the expression is:

```
  /usr/bin/cat /var/tmp/snmpd.log | /usr/bin/grep lost | wc
```

So the file /var/tmp/snmpd.log, which is the key log file, may contain the string `lost`. Search for this string and count the number of occurrences of this string. This number will be used in the predicate.

The steps to implement this simple string monitoring are:

Step 1. Decide on an unused user_state Resource Variable supplied by the Problem Management subsystem, and create a pman resource monitor.

```
TargetType=NODE_RANGE
Target=7
Rvar=IBM.PSSP.pm.User_state9
Command=″/usr/bin/cat /var/tmp/snmpd.log | /usr/bin/grep lost | wc″
SampInt=10
```

The Resource Variable IBM.PSSP.pm.User_state9 contains a value. This variable is used in the pman subscription.

Step 2 To ensure that this Resource Variable is loaded correctly in the SDR, check the entry in the SDR:

```
  SDRGetObjects pmanrmdConfig
  ---------------------------

7 NODE_RANGE   7           IBM.PSSP.pm.User_state9 ″″/usr/bin/cat
/var/tmp/snmpd.log | /usr/bin/grep lost | wc″″          10
```

Step 3 The pman Resource Monitor must be stopped and restarted. Note that there is no refresh function of this subsystem.

```
   stopsrc -s pmanrm
   startsrc  -s pmanrm
```

Step 4 The pman client has to register for an event when some predicate is applied to the Resource Variable IBM.PSSP.pm.User_state9 by the Event Management subsystem.

This is a standard pman definition, as described earlier:

```
  pmandef -s private_directory_changed \
-e ′IBM.PSSP.pm.User_state9:NodeNum=7:X@0!=X@P0′ \
-c ″wall SNMPD lost connection - string lost found in /var/tmp/snmpd.log″ \
-n 7
```

The node must be the same as the target node specified in the pman Resource Monitor load. For this case, it is Node 7.

To monitor this user state on another node, use the following command:

```
pmandef -s private_directory_changed9 \
-e ′IBM.PSSP.pm.User_state9:NodeNum=9:X@0!=X@P0′ \
-c ″wall Say this is Node 9″ \
-n 9
```

If this user_state variable is not defined on Node 9, you will get the error message:

```
Warning: Node range ″9″ contains nonexistent nodes.
Problem occurred while adding subscription records to SDR.
```

Step 5 Test the scenario:

Add the string `lost` to the log file in question (/var/tmp/snmpd.log). The pman client will be notified of this change by the Event Management subsystem and a wall message will be issued on Node 7:

```
SNMPD lost connection - string lost found in /var/tmp/snmpd.log
```

The pman client will not reissue this message until there is another `lost` string added to this same file, /var/tmp/snmpd.log.

This can be a good and simple way to monitor logs for key strings. While the example is somewhat simplistic, it is quite possible to expand and tailor these definitions, so that pman can automatically issue some recovery scripts if this condition is consecutively met. The recovery scripts are entirely installation-dependent.

6.4 Monitoring SP Resources: The Mechanics

The mechanics for monitoring SP resources are almost identical for all resources, that is, for hardware, software, and applications.

Step 1 Decide which resource is of interest.

A resource is an entity in the system that is observed. Examples of resources include hardware entities such as processors, disk drives, memory, adapters, and software entities such as databases, processes, and file systems. An overview of resources follows. For further information, refer to Chapter 2, “Resource Monitors” on page 27.

- Hardware
  1. CPUs
  2. Memory
  3. Disk subsystems
  4. Adapters
- Software
  1. System software
     - AIX subsystems
     - SP subsystems
  2. Application software
     - Databases
     - OLTP
- Other System Resources
  1. File system space
  2. Network

Step 2 Choose a condition.

Choose the condition that you wish to be notified of. The condition comprises Resource Variable, predicate arm, and rearm.

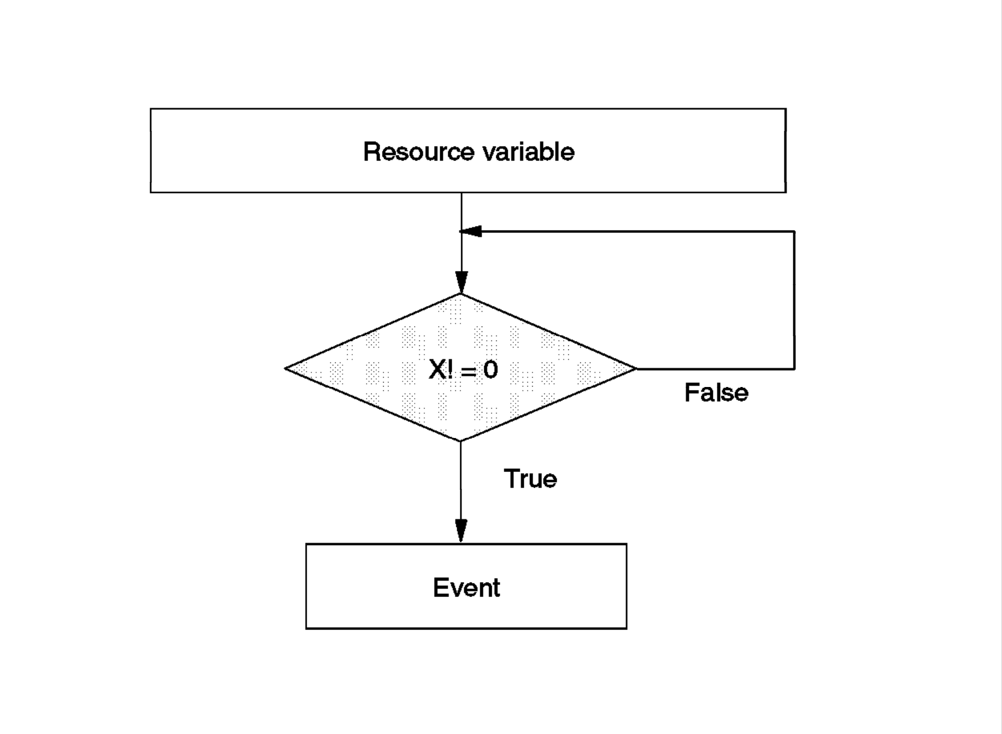


Figure 112. Condition

The Event Manager will apply this condition and generate the event if the condition is met.

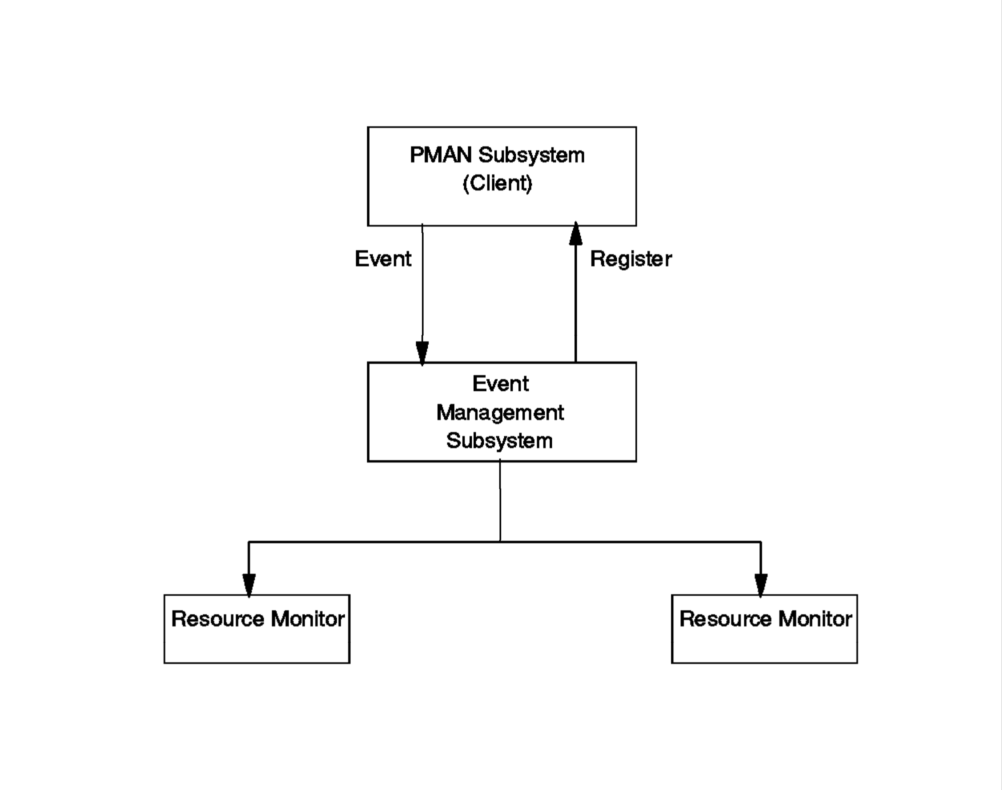


Figure 113. PMAN Subsystem

Step 3 Identify interested nodes.

Determine nodes that are interested in this event and need to subscribe to it.

The subscription is done using:

- Pmandef commands that are provided by the Problem Management subsystem for subscribing to Event Management events and associating actions for events. For more details, see Chapter 3, “Problem Management Subsystem” on page 71.

  Also, specify a trapID in the configuration information for an Event Management event.

- Event Perspectives, which are described in Chapter 5, “SP Perspectives GUI” on page 137.

Step 4 When the condition is true, Event Management will notify all subscribers of that event. The Event Management subsystem-supplied event response and the user-specified event configuration information are written into a FIFO file by the pmand command.

Step 5 The SNMP subagent sp_configd reads the data from the FIFO pipe and creates an SNMP trap from it.

The sp_configd daemon sends the trap to the SNMP managers specified in the /etc/snmpd.conf file on the node.

6.4.1 Basic Example - Monitoring Server Key Switch

Consider Node 1 to be a server that is rebooted every weekend. For a successful reboot, it is essential that the key position is in the *normal* state.

Our aim is to monitor the Hardware Resource IBM.PSSP.SP_HW.Node.keyModeSwitch, and if this key changes from normal to anything else, then an SNMP trap must be sent. This scheme is shown in Figure 114 on page 208.

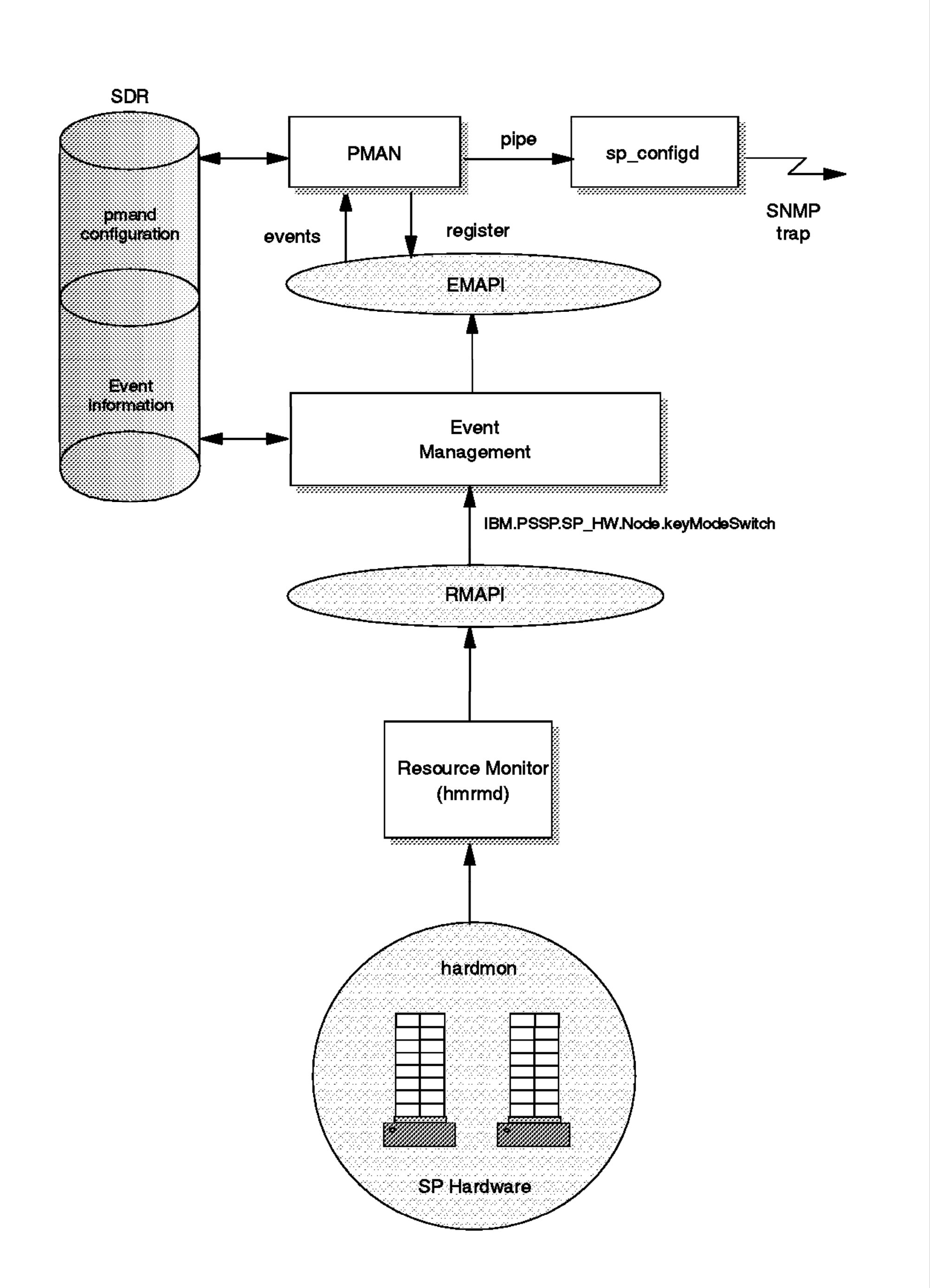


Figure 114. Registration for Event

Step 1 Identify the Resource Variable.

To ensure that the key is in the correct state, it is possible to monitor the resource variable IBM.PSSP.SP_HW.Node.keyModeSwitch, which is monitored by the Resource Monitor IBM.PSSP.hmrmd.

Step 2 Determine what condition exists.

The pman client must subscribe to the Event Management subsystem to be notified when there is a condition change. In this example, the condition change is a key change from `normal` to `service` or `secure`.

The Event Management subsystem holds the condition in the SDR:

```
       SDRgetObjects EM_Condition
       --------------------------
keyNotNormal IBM.PSSP.SP_HW.Node.keyModeSwitch X!=0         X==0
NodeNum      ″Key mode switch on a node was switched out of the Normal
position.″
```

The pman client will subscribe to this event, and when the event occurs, an SNMP trap is issued.

```
    pmandef -s key_mode_test
    -e ′IBM.PSSP.SP_HW.Node.keyModeSwitch:NodeNum=1:X!=0′
    -t 10003
    -n 1
```

Step 3 Determine what nodes subscribe to the event.

In this example, pman on Node 1 subscribes to the event, so that when the key is set to service or secure (X!=0), then pman will be notified by the Event Management subsystem, and can send a trap to NetView for AIX via sp_configd.

However, if pman on Node 1 dies, none of the subscribed events will be received. You will not be notified if the key setting is changed. Therefore, you cannot simply rely on pman on the local node to monitor a critical event, because that may affect the availability of this resource. It is better if an external pman client also monitors the event, thereby improving availability.

This can be overcome by requesting Problem Management subsystems on other nodes to additionally subscribe to the same event, so that if this condition is met on Node 1 (the key is set to service or secure, not to normal), then all subscribers on the other nodes will be informed. Thus the definition is:

```
    pmandef -s key_mode_test
    -e ′IBM.PSSP.SP_HW.Node.keyModeSwitch:NodeNum=1:X!=0′
    -t 10003
    -n 5,6,7
```

To ensure that pman accepted this subscription, run the following:

```
  lssrc -ls pman      (on Node 6 - node that subscribed for this event)


Subsystem         Group            PID     Status
 pman             pman             13144   active

 pmand started at: Tue Sep 10 14:52:28 1996
 pmand last refreshed at:
 Tracing is off
 =========================================================
 Events for which registrations are as yet unacknowledged:
 =========================================================
 =========================================================
 Events for which actions are currently being taken:
 =========================================================
 =========================================================
 Events currently ready to be acted on by this daemon:
 =========================================================
 ------------------ key_mode_test ----------------
 Currently ACTIVE
 Client root root.admin@MSC.ITSO.IBM.COM at sp21en0.msc.itso.ibm.
 com
 Resource Variable: IBM.PSSP.SP_HW.Node.keyModeSwitch
 Instance: NodeNum=1
 Predicate: X!=0
 SNMP Trapid: 10003
```

This pmand configuration is held in the SDR:

```
pmEventid             -1
pmNodenumber          6
pmTargetType          NODE_RANGE
pmTarget              5,6,7
pmRvar                IBM.PSSP.SP_HW.Node.keyModeSwitch
pmIvec                NodeNum==1
pmPred                X!=0
pmCommand             ” ”
pmCommandTimeout      0
pmHandle              key_mode_test
pmTrapid              10003
pmPPSlog              0
pmThrottle            ””
pmRearmPred           ” ”
pmRearmTrapid         -1
pmRearmPPSlog         0
pmRearmCommand        ” ”
pmRearmCommandTimeout 0
pmUsername,           root
pmPrincipal           root.admin@MSC.ITSO.IBM.COM
pmHost                sp21en0.msc.ibm.com
pmActivated           1
pmText                ” ”
pmRearmText           ” ”
pmUserLabel           ” ”
```

Nodes 5, 6, and 7 have subscribed to the same event that may occur on Node 1. If it does, then a trap is associated with it and forwarded to the SNMP manager.

Step 4 The event is issued.

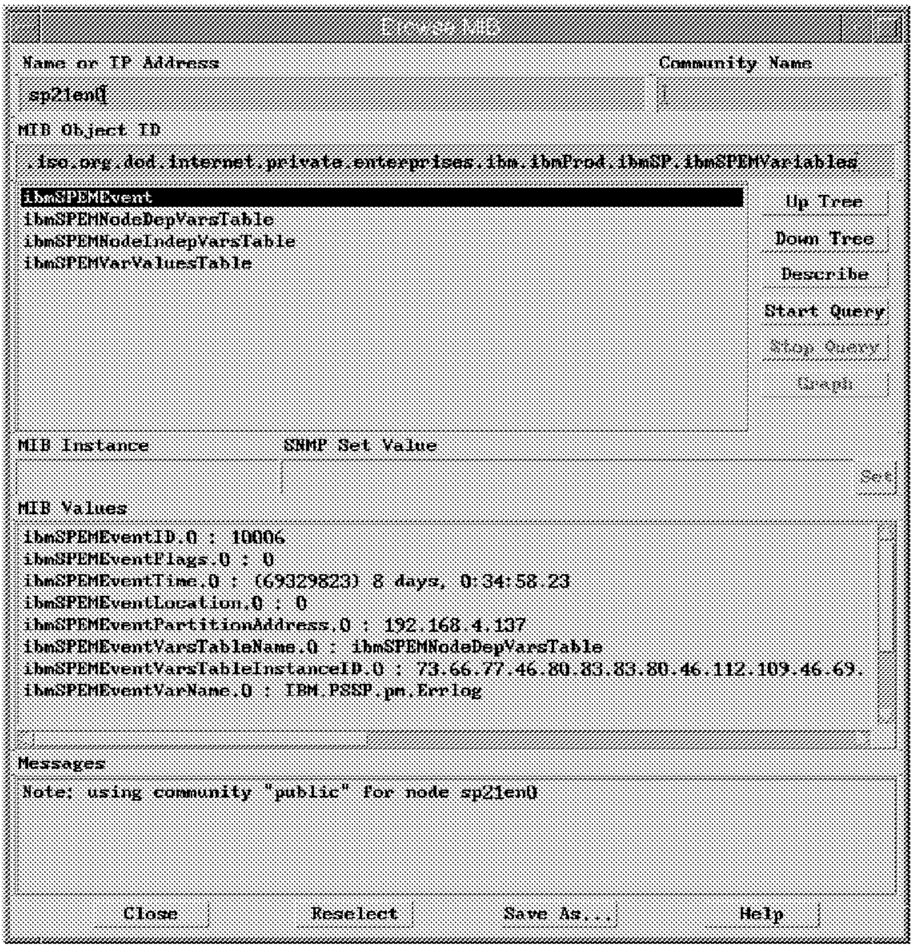
When the condition is true, Event Management notifies all subscribers, that is, Nodes 5, 6, and 7.

Step 5 The subagent forwards the event.

The subagent, sp_configd, on Nodes 5, 6 and 7, will receive this event information from the pipe in /var/tmp. The events are converted to

SNMP traps for forwarding to the SNMP manager, as defined in the snmpd configuration file.

Figure 115 shows the event that should also be displayed by the NetView for AIX MIB Browser:



Name or IP Address: sp21en0

Community Name:

MIB Object ID: .iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmSP.ibmSPEMVariables

```
ibmSPEMEvent
ibmSPEMNodeDepVarsTable
ibmSPEMNodeIndepVarsTable
ibmSPEMVarValuesTable
```

MIB Instance:

SNMP Set Value:

MIB Values

```
ibmSPEMEventID.0 : 10006
ibmSPEMEventFlags.0 : 0
ibmSPEMEventTime.0 : (69329823) 8 days, 0:34:58.23
ibmSPEMEventLocation.0 : 0
ibmSPEMEventPartitionAddress.0 : 192.168.4.137
ibmSPEMEventVarsTableName.0 : ibmSPEMNodeDepVarsTable
ibmSPEMEventVarsTableInstanceID.0 : 73.66.77.46.80.83.83.80.46.112.109.46.69.
ibmSPEMEventVarName.0 : IBM.PSSP.pm.Errlog
```

Messages

```
Note: using community "public" for node sp21en0
```

Figure 115. Event Description

Note: The Event Management Event Time MIB variable is the elapsed time between the activation of the SP proxy subagent and the occurrence of the event.

The output in Figure 115 does not correspond to the output from the snmpinfo command shown in the previous screen.

To determine if the Event Management subsystem raised the event, it is now appropriate to check the contents of the ibmSPEvent group object, using snmpinfo command on the Control Workstation.

```
snmpinfo -m dump -v -h sp21n06 ibmSPEMEvent
```

If the event was indeed triggered, then the ibmSPEVent MIB is updated with the information about this event on that node.

```
ibmSPEMEventID.0 = 10003
ibmSPEMEventFlags.0 = 0
ibmSPEMEventTime.0 = 8 days, 2 hours, 48 minutes, 53.32 seconds (70133332
ibmSPEMEventLocation.0 = 0
ibmSPEMEventPartitionAddress.0 = 192.168.4.137
ibmSPEMEventVarsTableName.0 = ″ibmSPEMNodeIndepVarsTable″
ibmSPEMEventVarsTableInstanceID.0 = ″192.168.4.137.73.66.77.46.80.83.83.8
ibmSPEMEventVarName.0 = ″IBM.PSSP.SP_HW.Node.keyModeSwitch″
ibmSPEMEventVarValueInstanceVector.0 = ″NodeNum=1″
ibmSPEMEventVarValuesTableInstanceID.0 = ″″
ibmSPEMEventVarValue.0 = ″2″
ibmSPEMEventPredicate.0 = ″X!=0″
```

Step 6 Configure the SNMP manager.

NetView for AIX will receive three traps, all concerning the same event. Node 1 has a problem with a resource, but only one action is required to amend the situation.

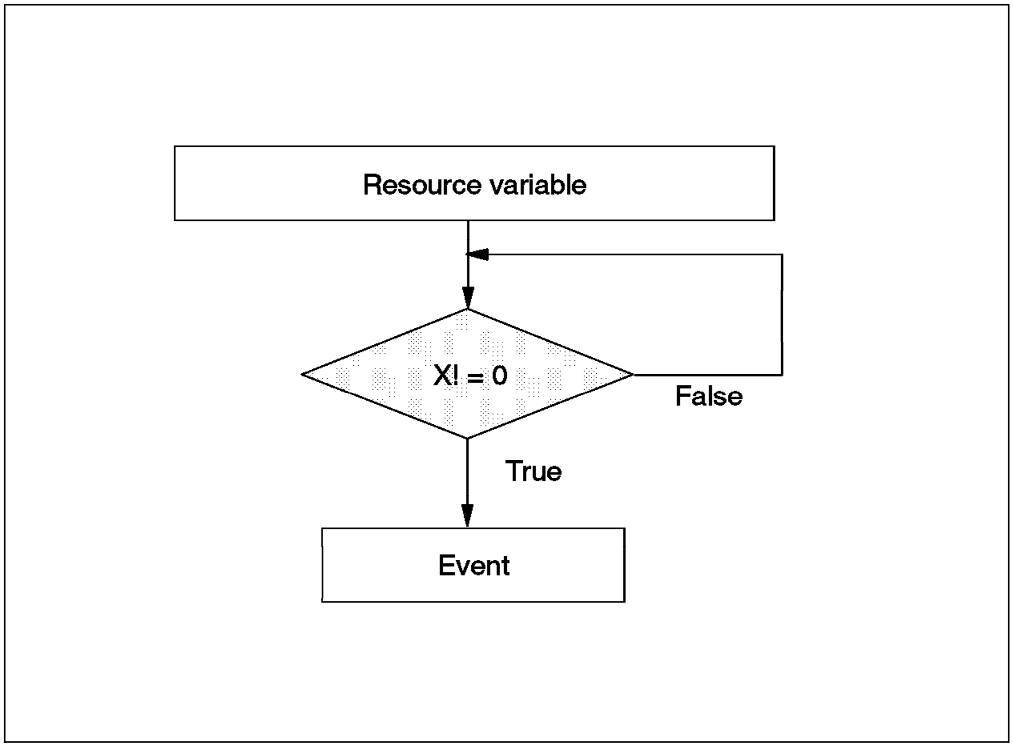


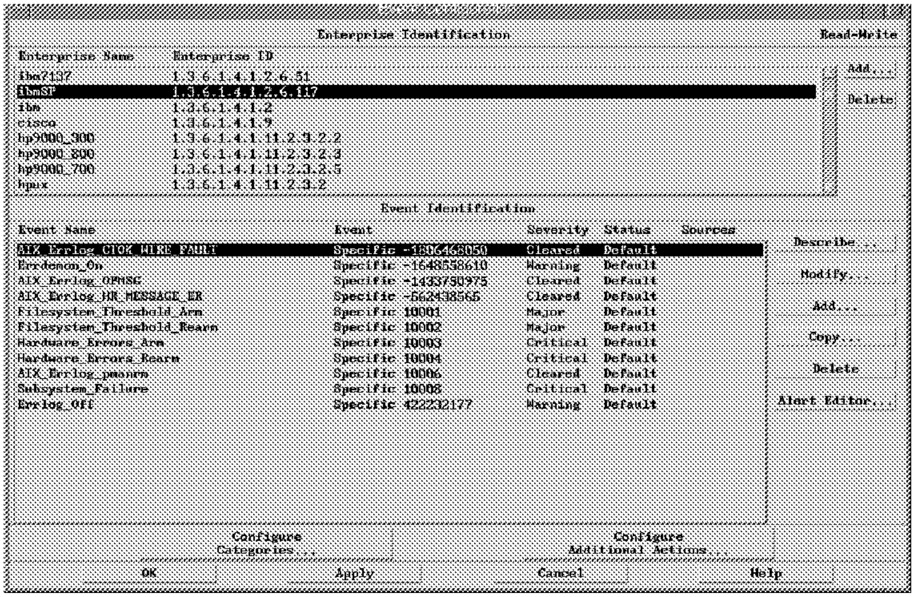
Figure 116. Ruleset Logic

This is feasible using the NetView for AIX ruleset editor (see Figure 116). Traps are captured by trapd and processed against the ruleset editor. In this case, the traps will follow the logic defined in the ruleset editor.

There will be one *forward* to the NetView events application, and one *action*. This action will be executed on the Control Workstation, since all the hardware commands must be issued from there.

Step 7 Define the trap.

The trap ID of 10003 needs to be configured, using the **Options→Event Configuration→Trap Customization** from the menu bar (see Figure 117 on page 213). This is described more specifically in 6.2.6, “Customizing Traps” on page 186.



Enterprise Identification

Read-Write

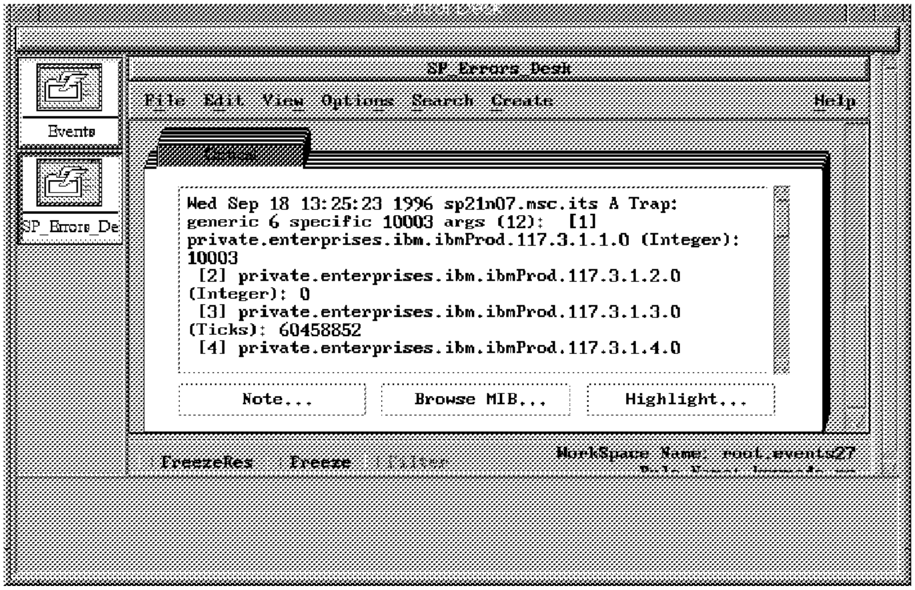
Enterprise Name	Enterprise ID
ibm7137	1.3.6.1.4.1.2.6.51
ibmSP	1.3.6.1.4.1.2.6.117
ibm	1.3.6.1.4.1.2
cisco	1.3.6.1.4.1.9
hp9000_300	1.3.6.1.4.1.11.2.3.2.2
hp9000_800	1.3.6.1.4.1.11.2.3.2.3
hp9000_700	1.3.6.1.4.1.11.2.3.2.5
hpux	1.3.6.1.4.1.11.2.3.2

Event Identification

Event Name	Event	Severity	Status	Sources
AIX_Errlog_CTOK_WIRE_FAULT	Specific -1806468050	Cleared	Default	
Errdemon_On	Specific -1648558610	Warning	Default	
AIX_Errlog_OPMSG	Specific -1433750975	Cleared	Default	
AIX_Errlog_HR_MESSAGE_ER	Specific -562438565	Cleared	Default	
Filesystem_Threshold_Arm	Specific 10001	Major	Default	
Filesystem_Threshold_Rearm	Specific 10002	Major	Default	
Hardware_Errors_Arm	Specific 10003	Critical	Default	
Hardware_Errors_Rearm	Specific 10004	Critical	Default	
AIX_Errlog_pmanrm	Specific 10006	Cleared	Default	
Subsystem_Failure	Specific 10008	Critical	Default	
Errlog_Off	Specific 422232177	Warning	Default	

Figure 117. Trap Configuration Window

This updates the /usr/OV/conf/C/trapd.conf file, which is used by the trapd daemon of the Control Desk (see Figure 118).



```
Wed Sep 18 13:25:23 1996 sp21n07.msc.its A Trap:
generic 6 specific 10003 args (12):  [1]
private.enterprises.ibm.ibmProd.117.3.1.1.0 (Integer):
10003
 [2] private.enterprises.ibm.ibmProd.117.3.1.2.0
(Integer): 0
 [3] private.enterprises.ibm.ibmProd.117.3.1.3.0
(Ticks): 60458852
 [4] private.enterprises.ibm.ibmProd.117.3.1.4.0
```

Figure 118. NetView Control Desk - Events Application

Step 8 Construct the ruleset.

The ruleset editor is a capability of NetView for AIX. Rulesets can be used in conjunction with IBM POWERparallel System Support Programs components to effectively monitor SP resources.

The ruleset editor is available by selecting **Tools→Ruleset Editor** from the NetView for AIX menu bar.

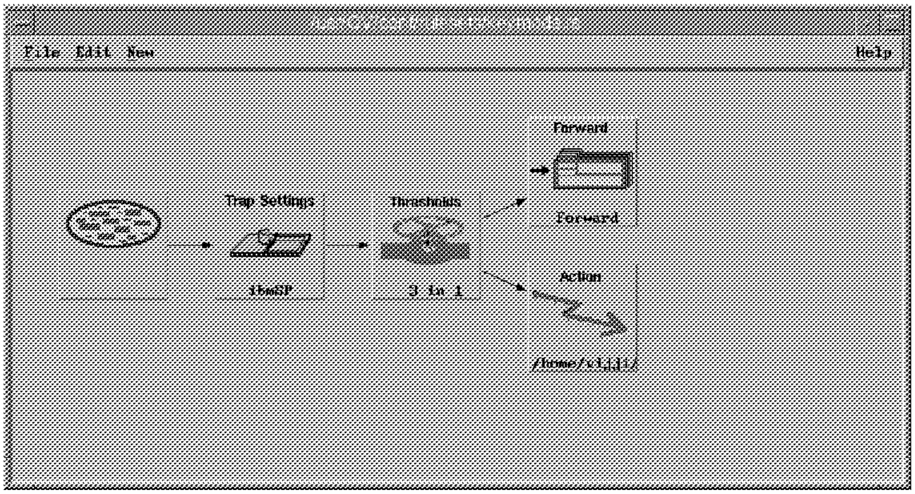
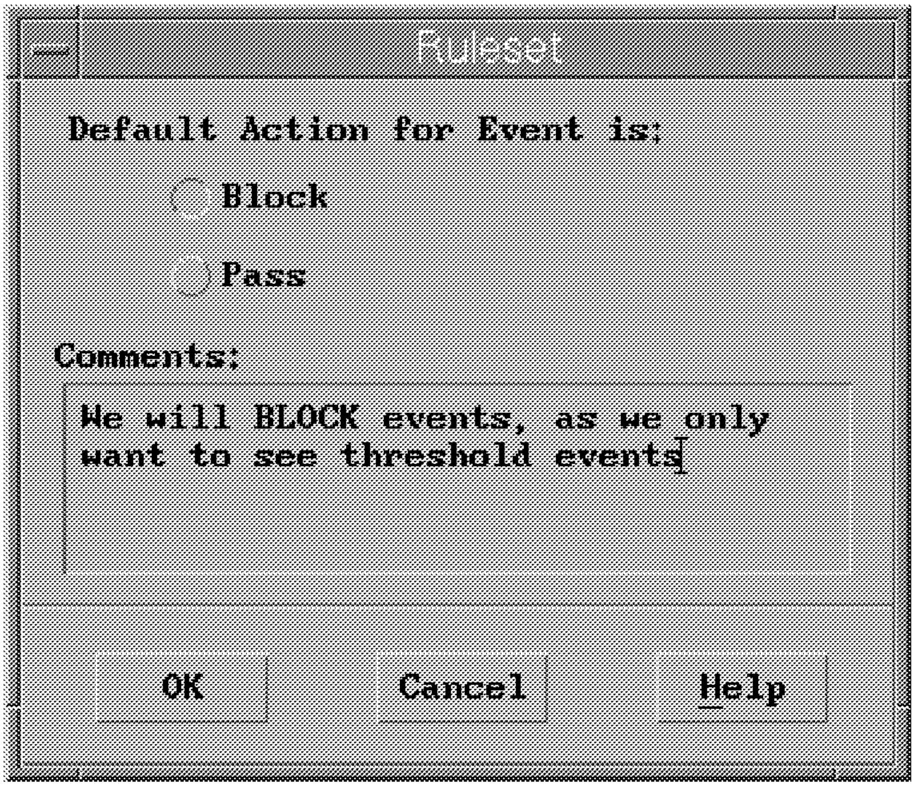


Figure 119. Ruleset Editor

1. The first node will set the behavior for the ruleset that we are constructing (see Figure 119). If you set it to pass, all events will be forwarded to the registered application. However, you want to block events, because you only want to see threshold events that pass the filter rule. Click **Block** (see Figure 120).



Default Action for Event is:

○ Block

○ Pass

Comments:

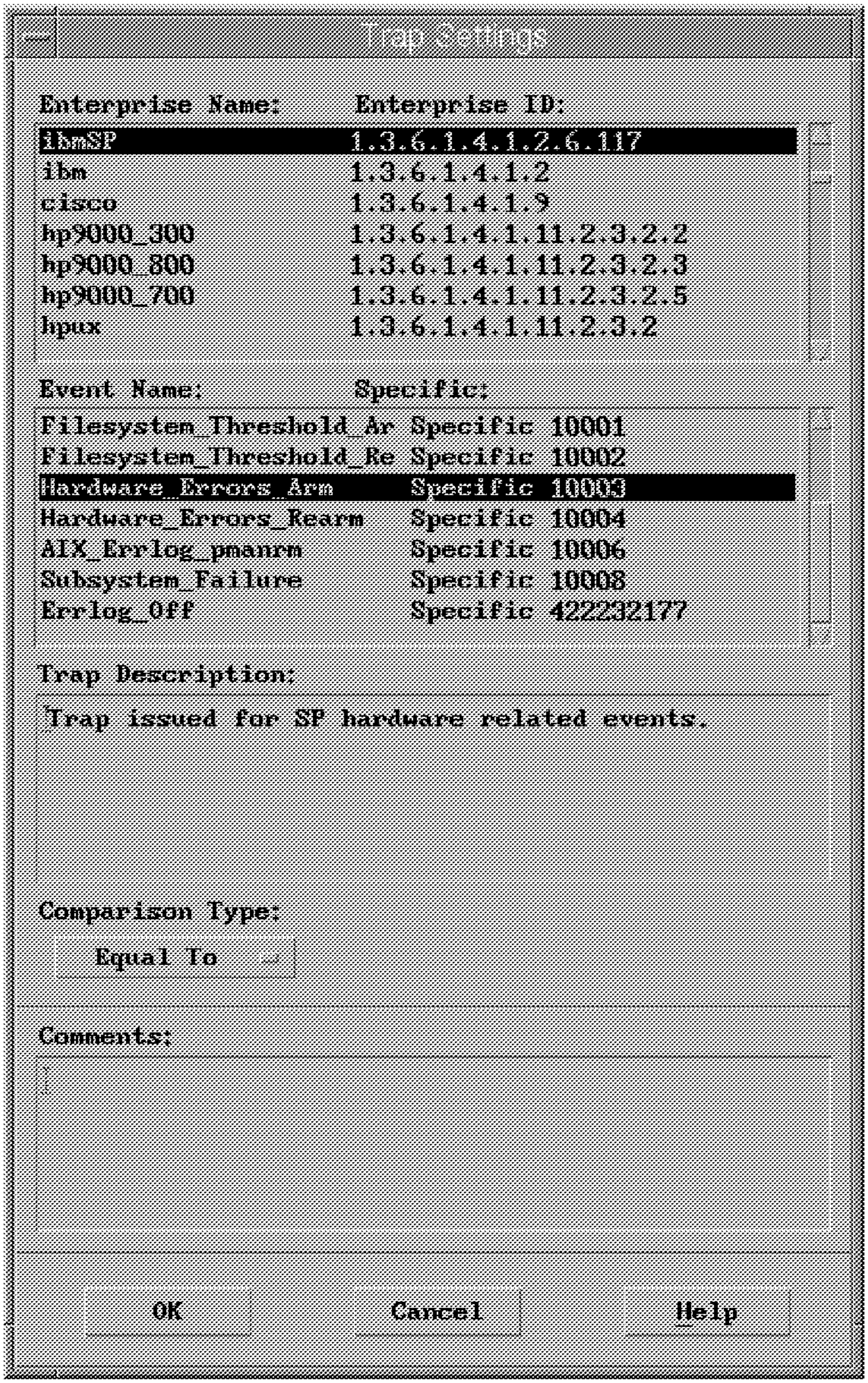
We will BLOCK events, as we only want to see threshold events

Figure 120. Block Event Behavior

2. The next step is to identify the hardware event. This is accomplished by dragging the trap-setting node into the workspace and selecting the following settings in the accompanying dialog box:

```
Enterprise Name     ibmSP    1.3.6.1.4.1.2.6.117
Event Name          Hardware_Errors_Arm
Specific            10003
```

Having selected these settings, as in Figure 121, the node should be attached to the Event Stream Node by selecting **Edit→Connect Two Nodes**.



Trap Settings

Enterprise Name:	Enterprise ID:
ibmSP	1.3.6.1.4.1.2.6.117
ibm	1.3.6.1.4.1.2
cisco	1.3.6.1.4.1.9
hp9000_300	1.3.6.1.4.1.11.2.3.2.2
hp9000_800	1.3.6.1.4.1.11.2.3.2.3
hp9000_700	1.3.6.1.4.1.11.2.3.2.5
hpux	1.3.6.1.4.1.11.2.3.2

Event Name:	Specific:
Filesystem_Threshold_Ar	Specific 10001
Filesystem_Threshold_Re	Specific 10002
Hardware_Errors_Arm	Specific 10003
Hardware_Errors_Rearm	Specific 10004
AIX_Errlog_pmanrm	Specific 10006
Subsystem_Failure	Specific 10008
Errlog_Off	Specific 422232177

Trap Description:

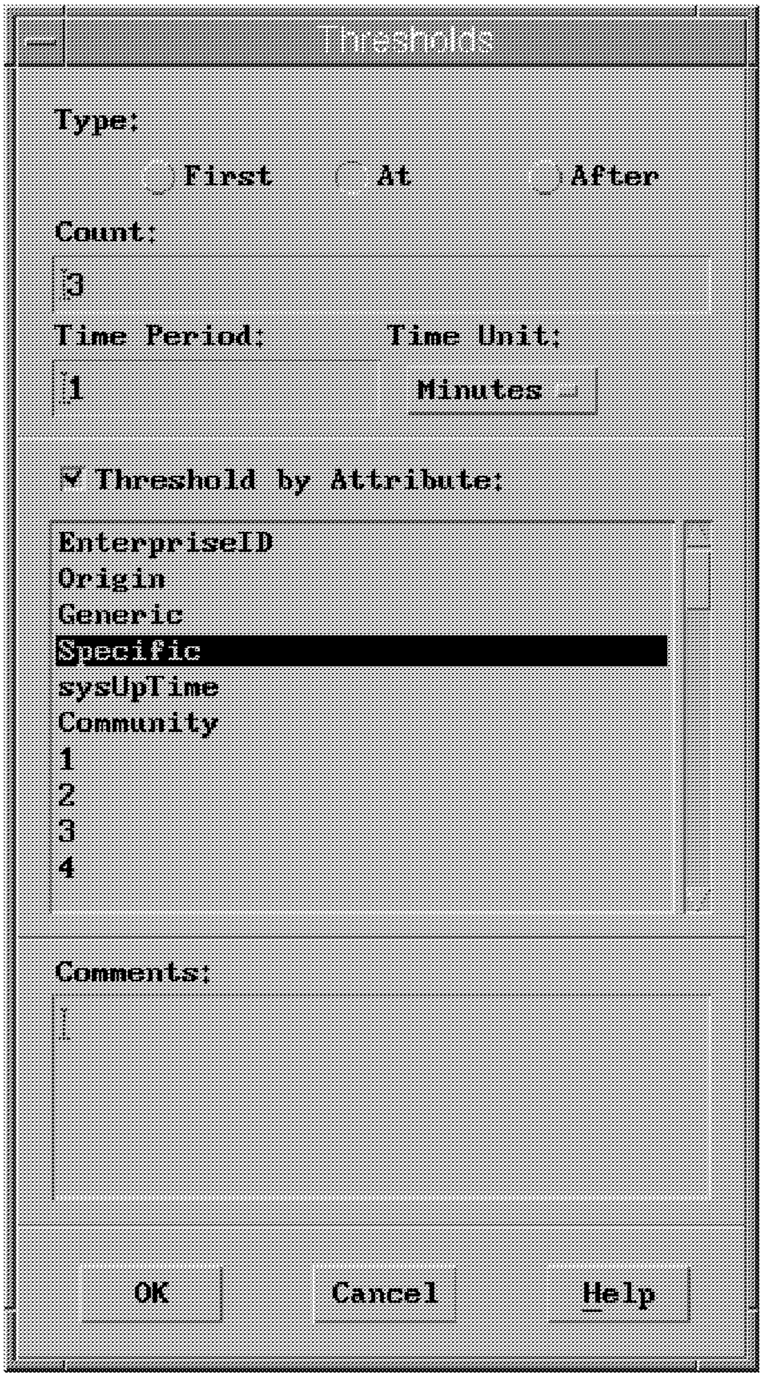
Trap issued for SP hardware related events.

Comparison Type: Equal To

Comments:

Figure 121. Trap Settings

3. The Threshold decision node was used to check for repeated occurrences of the same specific trap, 10003. Figure 122 on page 216 shows the Threshold dialog and the settings that were used in this example.



Type:

○ First ○ At ○ After

Count: 3

Time Period: 1

Time Unit: Minutes

☑ Threshold by Attribute:

- EnterpriseID
- Origin
- Generic
- Specific
- sysUpTime
- Community
- 1
- 2
- 3
- 4

Comments:

Figure 122. Thresholds Setup Window

Note that if the Type field is set to **At**, then when the threshold condition is reached, you only forward the nth trap to the next node, where n is the value specified for Count. Count is the number of traps required to reach the threshold condition. In this case, it is the number of events we expect to receive from the pman clients that have registered for this event. Only the third trap will be forwarded within the time period.

The time period filed is used to specify, along with Time Unit, the length of time in which the number of traps specified in Count must be received to reach the threshold condition.

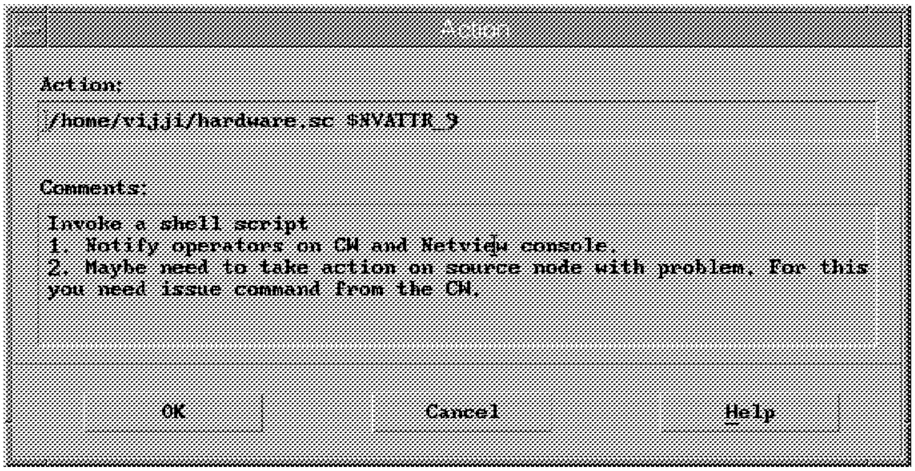
When the threshold has been met, no more traps will be forwarded until the time period has expired. At that point, the timer is reset. So it is important to set a realistic time period.

By default, the threshold will depend on the trap IDs of the traps passed into this node. To perform thresholding on another trap attribute, select this button and then choose an attribute from the list, such as Origin for the same IP address.

We are simply working on the trap ID, as we are checking for the same trap from different pman clients.

4. A Forward node is connected to the Thresholds node, which will cause a Single event to be displayed in the NetView Events Display.

5. It is also possible to attach an Action node to the Thresholds node as shown in Figure 123.



Action:

```
/home/vijji/hardware.sc $NVATTR_9
```

Comments:

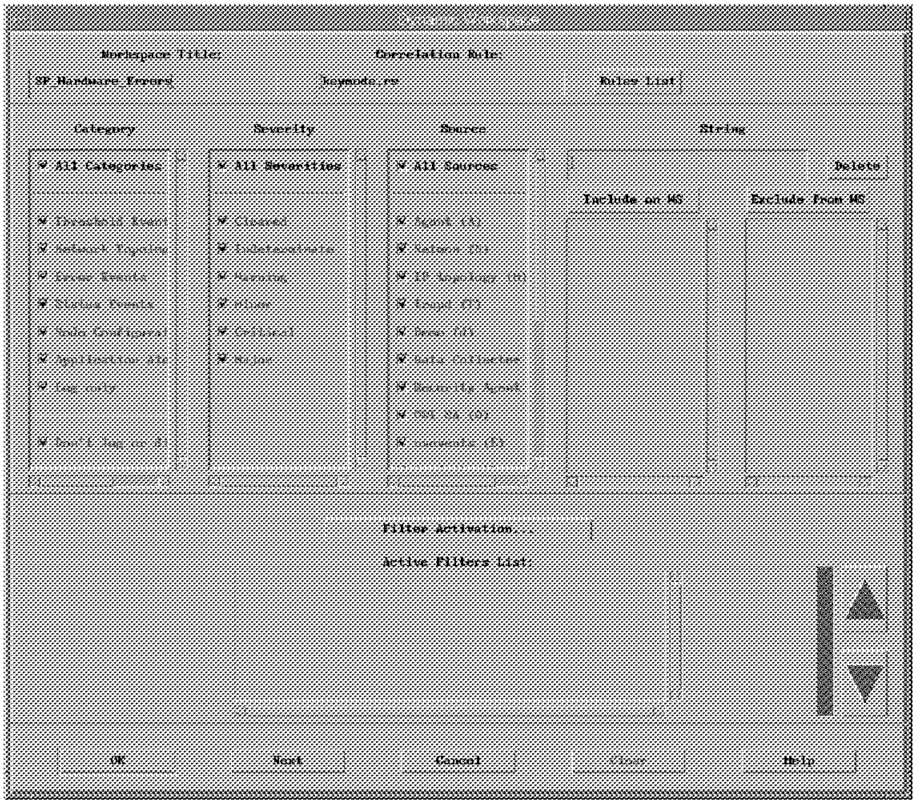
```
Invoke a shell script
1. Notify operators on CW and Netview console.
2. Maybe need to take action on source node with problem. For this
you need issue command from the CW.
```

Figure 123. Action Node

The command to be executed can use the environment variable $NVATTR_9. This passes the source node that is causing the problem to the shell script.

Step 9 Create a dynamic workspace.

From the Control Desk, create the dynamic workspace to show events that satisfy a specified correlation rule, as shown in Figure 124 on page 218.



Dynamic Workspace

Workspace Title: SP_Hardware_Errors

Correlation Rule: keymode.rs

Figure 124. Dynamic Workspace

The dynamic workspace is opened with all available events that match the update criteria you specify. New events are added to the workspace if they match the update criteria.

For more information on creating dynamic workspaces, refer to *NetView for AIX User's Guide for Beginners V4*, SC31-8158.

Now, we have to create this event and see whether this scenario works.

Step 10 Test the scenario.

- On the Control Workstation, change the setting of the key from `normal` to `service` for Node 1.

```
spmon -key service -t node01
```

- The Resource Variable has changed value from `normal` to `service`. Therefore, Event Management will create an event, and all subscribers, which in this case means pman clients on Nodes 5, 6 and 7, will be notified.
- On each of the subscribed nodes, the pmand daemon writes the event response supplied by the Event Manager daemon to a FIFO file in /var/tmp, such as em_trap.192.168.4.137. The sp_configd daemon reads the data from this pipe, and sends an SNMP trap to the SNMP manager defined in the snmpd.conf file.
- A trap from each SNMP subagent on Nodes 5, 6, and 7 will be sent to the SNMP manager on the SNMP-trap port 162.

In NetView for AIX, the trapd daemon receives the SNMP traps from the remote agents and processes the event based on records in the /usr/OV/conf/C/trapd.conf file, and displayed on the NetView for AIX events application.

Because the ruleset editor is applied, only one event is displayed instead of three.

When many traps are received containing notifications of Problem Management events, a likely action is to issue an SNMP GET request for the current value of the variable referenced within the trap from the trap originator (the MIB contains a description of how to do that). If the value is above the threshold specified in the trap, notify operators on the Control Workstation and NetView console.

Note: The hardware.sc function, which is invoked with the source node input parameter, is described in the comments part of the diagram. It notifies both SP and Netview (network) operators that there is a potential hardware problem, with a pop-up window or e-mail. The operators are able to take action from the Control Workstation, as Kerberos access may be required.

Appendix A. Overview of NetView for AIX Ruleset Editor

The ruleset editor is used to define rules for the processing of incoming traps. You can check traps against other traps, specific values, or object database values in order to determine what to do with them. By definition, the first node in every rule is either the event stream or the traps forwarded from the trapd daemon to the correlation daemon, nvcorrd.

Subsequent nodes are placed into a rule by dragging them from the template that is displayed below the ruleset drawing board, and then completing a dialog box to specify the values to be used in that node. Two types of nodes are available for rule construction:

- Decision nodes

  A decision node contains a test that determines whether processing of this trap should continue. Some attribute of the trap is tested against other traps, specific values, or something else.

  There are nine types of decision nodes.

- Action nodes

  An action node contains an action to be executed. Example: forwarding the trap information to applications, or setting database values.

  There are eleven types of action nodes.

In addition to these nodes, you can configure the default processing option for the rule by double-clicking the icon that represents the incoming event stream.

When a trap is received, it is processed through all the active rules. When a trap enters a decision node, the test made by that node determines whether the correlation process should stop processing that trap, or pass it on to the next node in the rule. If the value of the decision or comparison defined in that node is true, then processing continues to the next node in the rule. The only decision to be made is whether to stop or continue.

When a trap enters an action node, the action specified in that node is passed to the actionsvr daemon, along with the relevant parameters from the trap. After a trap is processed by an action node, it always continues to the next node. Rule processing continues based on the action′s return code.

If the default processing option for the rule is Block, then the trap is discarded. If the default processing option for the rule is Pass, and the trap has not been marked by any nodes, then the trap is forwarded to all interested applications.

A rule is activated if it is specified when a dynamic workspace is created, or if it is included in the ruleset automation file, /usr/OV/conf/C/ESE.automation, which is used by the actionsvr daemon during daemon initialization.

For more information, refer to *NetView for AIX Administrator′s Guide Version 4*, SC31-8168, or to *Examples Using NetView for AIX Version 4*, SG24-4515, which describes the process of correlating events and creating and editing rulesets.

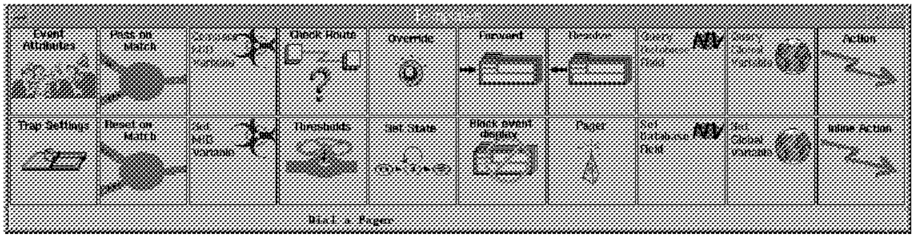


Figure 125. NetView for AIX Ruleset Editor

Appendix B. Makefile for the Resource Monitor Examples

This is the Makefile for the Resource Monitor examples:

```
#
# Sample Makefile for the Resource Monitor Examples
#
# some C-Compiler settings
#
CC=xlc
CLFAGS= -O
LFLAGS= -lha_rr
#
# some more definitions
#
RMAPI_SMP = rmapi_smp
RMAPI_MSG = $(RMAPI_SMP).msg
RMAPI_CAT = $(RMAPI_SMP).cat
#
#List of objects for the 3 examples
#
OBJ1=rmapi_smpcmd.o
OBJ2=rmapi_smpdae.o
OBJ3=rmapi_smpsig.o

all: rmapi_smpdae rmapi_smpsig rmapi_smpcmd

rmapi_smpcmd: $(OBJ1)
       $(CC) -o $@ $(LFLAGS) $(OBJ1)

rmapi_smpdae: $(OBJ2)
       $(CC) -o $@ $(LFLAGS) $(OBJ2)

rmapi_smpsig: $(OBJ3)
       $(CC) -o $@ $(LFLAGS) $(OBJ3)

install:
#
# load the SDR
#
       sh ./rmapi_smp.loadsdr pwd
#
# create the Message Catalog file
#
       runcat $(RMAPI_SMP) $(RMAPI_MSG)
#
# copy it to the appropriate directory
#
       @if [[ -d /lib/nls/msg/$(LANG) ]] ; then \
             cp $(RMAPI_CAT) /lib/nls/msg/$(LANG); \
             chmod a+r /lib/nls/msg/$(LANG)/$(RMAPI_CAT); \
       else \
             echo ″couldn′t find </lib/nls/msg/$(LANG)> directory″; \
             exit 1; \
       fi
       @echo ″″;
       @echo ″Stop and Restart your Event Management subsystem !!!″;
       @echo ″″;
#
deinst:
       sh ./rmapi_smp.unloadsdr
       @echo ″″;
       @echo ″Stop and Restart your Event Management subsystem !!!″;
       @echo ″″;
#
clean:
       rm -f *.o rmapi_smpcmd rmapi_smpsig rmapi_smpdae
```

Appendix C. User Response Time

When the resources you monitor appear normal but the user response time is too slow, it may not be clear where the cause of the problem is located. It could be in the operating system, the applications, the network, or in a combination of these and other places.

Perhaps you have neglected to monitor some critical resource. Perhaps there is some interaction that is timing- or load-dependent. Perhaps the system or application has a defect that causes the performance problem. Sometimes these and other problems occur in unfortunate combinations.

Although not all system problems can be avoided, there are ways to be aware of possible impending trouble.

C.1 Avoiding Surprises by Monitoring User Response Times

Determine what response times are required by the people or applications that use your system *before* performance problems occur on that system. Make sure there are quantitative goals. Methods of measuring the performance of the system must be available to determine if these goals are being met. If possible, use mechanisms that show the locations of problems.

Work with users to understand what they need, and know what the system can provide. (There is often a trade-off between what users want and what they can afford.) Take the time to ascertain their performance goals, so that, when you are in the middle of a performance crisis, you do not waste time arguing about what kind of performance is expected.

Once the performance requirements are defined, it should be possible to create Resource Monitors that test the performance of the system. These monitors allow a variety of SP and other system monitoring tools to be used to monitor performance and alert administrators and operators of possible problems. A warning that system performance is deteriorating may enable operators and support staff to react before performance falls below the required level.

You might be able to use the Problem Management subsystem to automate some of the tasks associated with maintaining the desired performance level. In cases where the performance problem is well understood and performance-improving tasks are easy to automate, the new tools provided with PSSP 2.2 may allow you to reduce your administrative workload.

Carefully consider what to do when performance falls below a desired threshold. The PSSP tools allow scripts to execute when certain conditions occur. You might be tempted to turn on additional monitors when performance gets bad. However, this approach often causes performance to degrade further. A better plan would be to use a number of different performance levels, each with an appropriate response.

If you set a threshold at a level that is less than desired, but better than required, you can then turn on additional monitoring even if you know it will degrade response time slightly. You might also try to add resources to improve the situation.

If your system reaches the point where your performance requirements are no longer being met, you can turn off the extra monitors. All resources can then be provided to the users. Monitors often consume a significant amount of resources, and can cause other problems with a system.

If you have decided to automate some tasks, you must be certain that the resources that were allocated, or the configuration changes that were made, are reclaimed or set to default values when the performance level improves. This can be difficult, since some applications or subsystems are not able to release resources without some type of restart. An example of this is swap space. Once disk space is allocated to the swap space, it cannot be reclaimed until the system is restarted.

By setting an event threshold at the expected level of performance, you can disable any extra monitors and reclaim resources when the system is again operating normally.

Remember that the tests needed to verify user or application response times consume resources. Tracking an HTTP server′s response time by occasionally fetching an HTML document probably consumes few resources. However, verifying the response times of an airline reservation system may require much more effort.

A sample Response Time Monitor is shown in the next section. It monitors the response time of an HTTP server and provides information to a Resource Monitor that passes the response times on to the Event Management subsystem.

C.2 Sample Response Time Monitor: HTTP Client

The current implementation of the Event Monitor forces the Resource Monitor to execute on a node or on the Control Workstation in an SP system. This constraint causes the Response Time Monitor to consist of at least two processes: one process running on a processor in the SP system, and the other running on the client system.

The program httprtB is a server that provides clients such as httprtA with information about the time it takes to fetch HTTP objects. The httprtB program runs on a workstation where the response time measurement is to be observed. The Resource Monitor httprtA is discussed in Chapter 4, “Application Program Interfaces (APIs)” on page 97.

For example, if your SP is a WWW server, you could have httprtB running on a few remote workstations and monitor the response times the users are experiencing.

C.2.1 httprtB, an HTTP Response Time Server

Let us take a look at how this server works. Appendix H, “How to Get the Examples in This Book” on page 257 tells you how to access the source code for this program.

This server is designed to run as a daemon on a workstation. It might seem a bit backwards, but httprtB is a server program located on a machine that is a client of your server. It waits for requests for connections from a client (your server), and when a valid request is made, httprtB forks, using fork(), a new process used to service that client.

The client then asks for an HTTP object to be retrieved, and the time and resulting status are returned to the client. The client can make multiple requests for HTTP object fetches. To terminate the session, the client simply closes its connection. This causes the child process that was forked to exit, using exit(), thereby freeing up any resources the child allocated.

The child process that is handling the client has a timeout, so if it does not hear from the client it closes the connection and frees up resources. This timeout is set on the command line when httprtB is started.

After opening a log file, using open(), signal handlers are installed for SIGCHLD and SIGQUIT. The command line is parsed to determine the client′s timeout value and which machine is allowed to make connections to this server.

```
listener = socket(AF_INET, SOCK_STREAM, 0);
if ( listener < 0 ) {
  WriteLog(″could not create listener socket, errno = %d\n″, errno);
  exit(2);
}

memset( (char *) &listenerSockAddr, 0, sizeof(listenerSockAddr) );
listenerSockAddr.sin_family = AF_INET;
listenerSockAddr.sin_len = sizeof(listenerSockAddr);
listenerSockAddr.sin_port = htons(HTTPRT_PORT_NUMBER);
listenerSockAddr.sin_addr.s_addr = htonl(INADDR_ANY);

if (bind(listener, (struct sockaddr *) &listenerSockAddr,
         sizeof(listenerSockAddr) ) < 0 ) {
  WriteLog(″could not bind() listener socket, errno = %d\n″, errno);
  exit(2);
}

listen(listener, 5);

while ( ! quitFlag ) {
int client;
struct sockaddr_in clientSocketAddr;
int clientSocketAddrLen = sizeof(clientSocketAddr);

  client = accept(listener, (struct sockaddr *) &clientSocketAddr,
                  &clientSocketAddrLen);
  if ( client < 0 ) {
    if ( errno != EINTR ) {
      WriteLog(″error from accept(), errno = %d\n″, errno);
    }
  } else {
    /* OK, got a potential client.  Are we allowed to talk */
    AcceptClient(client, clientSocketAddr, clientSocketAddrLen,
                      timeout, allowedHostP, listener);
  }
}
```

Figure 126. Interesting Part of httprtB′s main() Function

The interesting parts of main() are shown in Figure 126. First, using listen(), a socket is created for use in listening for client requests. This socket is registered at a particular IP port with the bind() routine. The httprtB server is willing to service requests on any port in case this machine has multiple network interfaces. The port number can be changed by editing the file httprt.h. After the socket is ready, the listen routine indicates that httprtB is ready to receive requests from clients.

The while() loop simply waits for a request, (by default, accept() blocks till a request is received), then passes that request to AcceptClient() for processing. Let us look at AcceptClient() to see what is done with the requests.

```
void
AcceptClient(int client,
             struct sockaddr_in clientSocketAddr,
             const int clientSocketAddrLen,
             int timeout,
             struct hostent *allowedHostP,
             int listener)
{
long **addrPP;
int i;
int matched;
int child;

  addrPP = (long **) allowedHostP->h_addr_list;
  for (i = matched = 0 ; addrPP[i] != 0 ; i++) {
    if ( memcmp (addrPP[i], &clientSocketAddr.sin_addr.s_addr,
                 sizeof(clientSocketAddr.sin_addr.s_addr)) == 0 ) {
      matched = 1;
      break;  /* no need to look further */
    }
  }

  if ( matched ) {
    child = fork();
    switch (child) {
      case 0:  /* child */
        close(listener);
        WriteLog(″starting new child\n″);
        BeChild(client, timeout);
        /*NOTREACHED*/
        break;
      case -1: /* error */
        close(client);  /* no need to leave this laying around */
        WriteLog(″fork() failed with errno = %d\n″, errno);
        break;
      default: /* parent */
        close(client);
        WriteLog(″accepted client 0x%08x\n″,
                 (int) clientSocketAddr.sin_addr.s_addr);
        break;
    }
  } else {
    close(client);  /* no need to keep this socket! */
    WriteLog(″rejected client 0x%08x\n″,
             (int) clientSocketAddr.sin_addr.s_addr);
  }
}
```

Figure 127. Function AcceptClient() in httprtB

First, the request is checked to see if it came from the expected client. This is done by looping through the list of addresses registered for the client host. This list was retrieved by the gethostbyname() routine in main(). Since a machine can have multiple interfaces, we should check all possibilities since we cannot always predict how a packet will be routed between two machines.

If the client is allowed to access this server, AcceptClient() will create a new process to service the client requests and return, using return, to main() to wait for more session requests. The fork() routine creates a new process that is almost an identical copy of the current function. The return code from fork() is used to determine whether the current process is the parent or the child. If −1 is returned, an error occurred.

If the process is the parent or an error occurred, the client socket is closed, using close(), to free up the resource. The function then returns after logging what happened. If the process is the child, the socket that used to listen is no longer needed, so it is closed. Then a message is sent to the log and the function BeChild() is called. BeChild() never returns control.

Note that the last few lines in AcceptClient() clean up the client socket and log rejected sessions.

```
void
BeChild(int client, int clientTimeout) {
int timeout;              /* timeout for a client request  to complete */
char url[MAX_URL_LEN];    /* url client wants */
char temp[MAX_URL_LEN];   /* used to read a line from client */
int delay;                /* time it took in seconds */
int status;               /* status from server or local status */
char results[32];         /* place to make results look nice */
struct sigaction thisSignal; /* used to register signal handlers */

  thisSignal.sa_handler = SIG_IGN;
  sigemptyset(&thisSignal.sa_mask);
  thisSignal.sa_flags = 0;
  if (sigaction(SIGCHLD, &thisSignal, 0) == -1) {
    WriteLog(″problem ignoring SIGCHLD\n″);
    exit(2);
  }

  thisSignal.sa_handler = (void (*)() ) TimeoutHandler;
  sigemptyset(&thisSignal.sa_mask);
  sigaddset(&thisSignal.sa_mask, SIGALRM);
  thisSignal.sa_flags = 0;
  if (sigaction(SIGALRM, &thisSignal, 0) == -1) {
    WriteLog(″problem installing SIGALARM handler\n″);
    exit(2);
  }

  timeoutFlag = 0;
  alarm(clientTimeout);
  while ( GetALine(client, temp, sizeof (temp)) > 0 ) {
    alarm(0);  /* OK this time */
    if (sscanf(temp, ″%d %s″,&timeout, url) == 2) {
      TimedFetch(timeout, url, &delay, &status);
    } else {
      delay = 0;
      status = HTTPRT_ERROR_REQUEST;
    }
    sprintf(results, ″%d %d\n″, delay, status);
    write(client, results, strlen(results));
    fsync(client);

    timeoutFlag = 0;
    alarm(clientTimeout);
  }
```

Figure 128. Processing Loop After Client is Accepted in httprtB

The first thing BeChild() does is set up new signal handlers for SIGALRM and SIGCHLD. The SIGALRM handler sets the global variable `volatile int timeoutFlag` to 1 to indicate that a timeout has occurred.

The function GetALine() is similar to fgets(). It gets a line from the client and returns -1 if there is an error or a timeout occurs. The call to GetALine() is surrounded by `alarm(clientTimeout)` and `alarm(0)` to enable and disable the timeout alarm.

After the client request is received, it is parsed to find out what the client wants. Requests are in the form “timeout URL.” If the request appears to be properly formed, it is passed on to TimedFetch() for processing.

The results from TimedFetch() are formatted and sent back to the client. The fsync() function is used to force the information back to the client in case any buffering delayed the transmission. The client timeout is then enabled and BeChild() waits for the next request.

When the while() loop ends, BeChild() figures out what happened, writes that information to the log, and exits the process. BeChild() never returns control.

The TimedFetch() function performs the bulk of the work. This routine and GetIt() would be kept if the server were converted to use a different communication mechanism, such as a DCE.

The first part (not shown) of TimedFetch divides the URL into httpServerName and httpPath. These are used as shown in Figure 129.

```
  /* httpServerName & httpPath were parsed from URL above */

  hostEntP = gethostbyname(httpServerName);
  if ( hostEntP == NULL ) {
    *statusP = HTTPRT_ERROR_BAD_HOST;
    return;
  }
  serverAddr.sin_family = hostEntP->h_addrtype;
  serverAddr.sin_port = httpPort;  /* alread in network byte order */
  memcpy(&serverAddr.sin_addr.s_addr,
         hostEntP->h_addr_list[0],
         hostEntP->h_length);

  GetIt(timeout, httpPath, (struct sockaddr *) &serverAddr,
        delayP, statusP);
  return;
}
```

Figure 129. Interesting Part of Function FetchIt()

httpServerName is used to get the network address of the HTTP server. This address and the path are given to GetIt() to do the work of requesting and retrieving the HTTP object.

```
void
GetIt(int timeout, char *pathP, struct sockaddr *serverP,
      int *delayP, int *statusP)
{
int serverSocket;
time_t startTime, endTime;
int rc;
int ch;
FILE *httpServer;
char junk[1024];
char httpBuffer[MAX_URL_LEN];

  if ( alarm(timeout) != 0 ) { /* someone else is using the alarm!! */
    WriteLog(″Program error!, trying to use active alarm\n″);
    exit(2);
  }

  startTime = time(NULL);

  serverSocket = socket(serverP->sa_family,  SOCK_STREAM, 0);
  if ( serverSocket < 0) {
    *statusP = HTTPRT_ERROR_SOCKET_CREATE;
    goto Leave;
  }

  /* connect to server */
  if (connect(serverSocket, serverP, sizeof(struct sockaddr)) < 0 ) {
    *statusP = HTTPRT_ERROR_HTTP_SERVER_CONNECT;
    goto Leave;
  }

  if (pathP[0] != ′\0′) {
    rc = sprintf(httpBuffer,
                 ″GET %s HTTP/1.0\nAccept: */*\nPragma: no-cache\n\n″,
                 pathP);
  } else {
    rc = sprintf(httpBuffer,
                 ″GET / HTTP/1.0\nAccept: */*\nPragma: no-cache\n\n″);
  }
```

Figure 130. First Half of GetIt() Preparing to Get HTTP Object

After saving the current time, GetIt() makes a connection to the HTTP server. This path is used to create the request to send to the HTTP server. An empty path is legal, so we must check for that case and add the leading "/" character.

If any problems are encountered, the status value is set and the function ends.

```
  if ( write(serverSocket, httpBuffer, strlen(httpBuffer)) !=
       strlen(httpBuffer) ) {
    *statusP = HTTPRT_ERROR_SENDING;
    goto Leave;
  }

  /* eat results */
  if ( read(serverSocket, httpBuffer, 12) != 12) {
    *statusP = HTTPRT_ERROR_RECEIVING;
    goto Leave;
  }
  rc = sscanf(httpBuffer,
              ″HTTP/1.0 %d ″,
              statusP);
  if (rc != 1) {
    *statusP = HTTPRT_ERROR_RECEIVING;
    goto Leave;
  }
  /* just eat characters */
  while ( (rc = read(serverSocket, junk, sizeof(junk)) ) > 0) {
    /* this space intentionally left blank */
  }

Leave:
  if ( serverSocket >= 0) {  /* if it was ever created */
    close(serverSocket);
  }
  endTime = time(NULL);
  alarm(0);   /* shut off alarm */
  *delayP = (endTime - startTime);
  return;
}
```

Figure 131. Bottom Half of GetIt() Where HTTP Object is Retrieved

The bottom half of GetIt(), shown in Figure 131, sends the request to the HTTP server and waits for the reply. The HTTP status code is parsed so it can be returned, then the HTTP object is consumed.

If the socket was successfully created, it is closed, the timeout is disabled, and the time required to handle the request is calculated.

The complete program is available. See Appendix H, “How to Get the Examples in This Book” on page 257 for details.

C.2.2 How to Improve httprtB

As presented in C.2, “Sample Response Time Monitor: HTTP Client” on page 226, httprtB is a simple and crude daemon. Possible improvements include:

- Use of the inetd routine
- Better and more flexible security and authentication
- Integration of httprtA and httprtB
- Use of a different communication mechanism, such as DCE

If you are planning to use this Response Time Monitor on many machines, you might want to consider using inetd to start the daemon only when it is needed. The inetd routine is designed to start network servers on demand, thereby reducing the amount of resources that are wasted running server programs that are never used. This change would require removing code in main() and changing the way I/O is handled. Servers compatible with inetd communicate over stdin and stdout.

The httprtB server was not created as an inetd program, so you can run it, as a normal user, on remote workstations to try things out. The configuration of inetd requires root authority.

The use of DCE would provide a nicer communication abstraction and could provide a better security model. In this example, DCE was not used to reduce the number of prerequisites. DCE provides functions that are designed for these types of distributed applications and services.

Appendix D. Essence of the Event Management and Problem Management Subsystems

This appendix offers information about the Event Management and Problem Management subsystems that is essential for creating your own monitor. The Problem Management subsystem is constructed by the Problem Management daemon (pmand) and the Problem Management Resource Monitor daemon (pmanrmd). These are client subsystems of the Event Management subsystem.

D.1 Event Management Subsystem SDR Class

When you use the Problem Management subsystem, you do not need to know what is going on between the Event Management subsystem and the Problem Management subsystem. But you do have to know what kind of information is provided by the Event Management subsystem. Therefore, you must be familiar with the Event Management subsystem SDR classes.

To see all Event Management subsystem SDR classes, use the following command:

```
# SDRListClasses | grep EM_
```

There are six Event Management subsystem SDR classes:

- EM_Resource_Monitor
- EM_Resource_Class
- EM_Resource_Variable
- EM_Instance_Vector
- EM_Structured_Byte_String
- EM_Condition

The following sections explain each class.

D.2 Event Management Resource Monitor Class

This SDR class contains information about Resource Monitors that may supply Resource Variables to the Event Management daemon.

You can find out about it by using the following command:

```
# SDRGetObjects EM_Resource_Monitor
```

D.2.1 What Can You Get?

The following is a default implementation:

Table 5. Event Management Resource Monitor Class (1 of 3)

rmName	rmPath	rmArguments
IBM.PSSP.harmld	/usr/lpp/ssp/bin/haemRM/harmld	-f
IBM.PSSP.harmpd	/usr/lpp/ssp/bin/haemRM/harmpd	
IBM.PSSP.hmrmd	/usr/lpp/ssp/bin/haemRM/hmrmd	IBM.PSSP.hmrmd
IBM.PSSP.pmanrmd	/usr/lpp/ssp/bin/pmand	
Membership		
Response		
aixos		

Table 6. Event Management Resource Monitor Class (2 of 3)

rmMessage_file	rmMessage_set	rmConnect_type
harm_des.cat	1	server
harm_des.cat	2	server
hmrmd_des.cat	1	server
harm_des.cat	3	client
harm_des.cat	4	internal
harm_des.cat	5	internal
harm_des.cat	6	internal

Notes:

1. rmMessage_file is a message catalog name.
2. rmMessage_set is a set number in the catalog specified by the rmMessage_file.

Table 7. Event Management Resource Monitor Class (3 of 3)

rmPTX_prefix	rmPTX_description	rmPTX_asnno
IBM/PSSP.harmld	1,2	2
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

This shows you that Resource Monitor *rmName* is started by command *rmPath* with argument *rmArguments*. This monitor uses message catalog /usr/lib/nls/msg/$LANG/*rmMessage_file* Its message set number is *rmMessage_set*. It is connected to the Event Management daemon as *rmConnect_type*.

D.2.2 Why Is It Useful?

This information is useful when you want to learn about a particular Resource Variable.

To find a message, follow these instructions:

1. From *rmName*, find the monitor name that monitors your Resource Variable.
2. The corresponding *rmMessage_file* is a message catalog name that you look for.
3. The corresponding *rmMessage_set* is a set number in the catalog specified by the *rmMessage_file*.

4. You can find the message number of your Resource Variable in EM_Resouce_Variable SDR class. Refer to Table 10 on page 240.

5. Use the dspcat command, as follows:

```
# dspcat  rmMessage_file rmMessage_set rvDescription
```

The catalog name is *rmMessage_file*, the set number is *rmMessage_set*, and the message number is *rvDescription* in EM_Resouce_Variable SDR class.

D.3 Event Management Resource Class Class

This class contains information about the Resource Class that is specified in the set of Resource Variable definitions.

You can learn about it by using the following command:

```
# SDRGetObjects EM_Resource_Class
```

D.3.1 What Can You Get?

The following is a default implementation:

Table 8. Event Management Resource Class Class

rcClass	rcResource_monitor	rcObservation_interval	rcReporting_interval
IBM.PSSP.CSS	IBM.PSSP.harmld	5	5
IBM.PSSP.HARMLD	IBM.PSSP.harmld	30	30
IBM.PSSP.LL	IBM.PSSP.harmld	250	250
IBM.PSSP.Membership	Membership	0	0
IBM.PSSP.PRCRS	IBM.PSSP.harmld	5	5
IBM.PSSP.Prog	IBM.PSSP.harmpd	0	0
IBM.PSSP.Response	Response	0	0
IBM.PSSP.SP_HW	IBM.PSSP.hmrmd	0	0
IBM.PSSP.VSD	IBM.PSSP.harmld	10	10
IBM.PSSP.aixos.CPU	aixos	15	0
IBM.PSSP.aixos.Disk	aixos	30	0
IBM.PSSP.aixos.FS	aixos	60	0
IBM.PSSP.aixos.LAN	aixos	40	0
IBM.PSSP.aixos.Mem	aixos	15	0
IBM.PSSP.aixos.PagSp	aixos	30	0
IBM.PSSP.aixos.Proc	aixos	60	0
IBM.PSSP.pm	IBM.PSSP.pmanrmd	0	0

This shows you that Resource Monitor *rcResource_monitor* supplies Resource Variable in Resource Class *rcClass.* This variable is monitored in *rcObservation_interval* seconds and is reported in *rcReporting_interval* seconds.

The value of *rcObservation_interval* must be greater than or equal to the value of *rcReporting_interval*.

The value of *rcReporting_interval* may be 0 if the design of the Resource Monitor fixes the interval between variable updates. It may also be 0 if the Resource Monitor is incorporated into a subsystem and the subsystem updates the variable as part of its normal execution.

D.3.2 Why Is It Useful?

This is the first file you have to look up when you begin to create your own monitor. The purpose of creating your own monitor is to monitor a particular Resource Variable. But to do this, you must choose one of 300 available variables. Selecting a Resource Class lowers the number of variables that you have to consider, making this task more manageable. Refer to Table 10 on page 240.

If you want to know the description of the variable you have chosen, you have to know which Resource Monitor monitors this Resource Variable. You can get this information from the SDR. The Resource Class *rcClass* is monitored by Resource Monitor *rcResource_monitor*.

Once you know which Resource Monitor monitors your Resource Variable, refer to Table 5 on page 236 and Table 6 on page 236.

D.4 Event Management Instance Vector Class

This class contains information about the Instance Vector class that is specified as Resource Class, and its Instance Vector.

You can learn about it by using the following command:

```
# SDRGetObjects EM_Instance_Vector
```

D.4.1 What Can You Get?

The following is the default implementation:

Table 9. Event Management Instance Vector Class

ivResource_name	ivElement_name	ivElement_description
IBM.PSSP.CSS	NodeNum	30
IBM.PSSP.HARMLD	NodeNum	30
IBM.PSSP.LL.SCHEDD	NodeNum	30
IBM.PSSP.LL.SCHEDD	SCHEDD	33
IBM.PSSP.LL.STARTD	STARTD	32
IBM.PSSP.LL.STARTD	NodeNum	30
IBM.PSSP.Membership.LANAdapter	NodeNum	3
IBM.PSSP.Membership.LANAdapter	AdapterType	4
IBM.PSSP.Membership.LANAdapter	AdapterNum	5
IBM.PSSP.Membership.Node	NodeNum	6
IBM.PSSP.PRCRS	NodeNum	30
IBM.PSSP.Prog	ProgName	3
IBM.PSSP.Prog	UserName	4
IBM.PSSP.Prog	NodeNum	5
IBM.PSSP.Response.Host	NodeNum	3
IBM.PSSP.Response.Switch	NodeNum	4
IBM.PSSP.SP_HW.Frame	FrameNum	248
IBM.PSSP.SP_HW.Node	NodeNum	249
IBM.PSSP.SP_HW.Switch	SwitchNum	250
IBM.PSSP.VSD	NodeNum	30
IBM.PSSP.VSD	VSD	31
IBM.PSSP.VSDdrv	NodeNum	30
IBM.PSSP.aixos.CPU	NodeNum	42
IBM.PSSP.aixos.Disk	NodeNum	45
IBM.PSSP.aixos.Disk	Name	46
IBM.PSSP.aixos.FS	NodeNum	47
IBM.PSSP.aixos.FS	VG	48
IBM.PSSP.aixos.FS	LV	49
IBM.PSSP.aixos.LAN	Adapter	53
IBM.PSSP.aixos.LAN	NodeNum	52
IBM.PSSP.aixos.Mem.Kmem	Type	55
IBM.PSSP.aixos.Mem.Kmem	NodeNum	54
IBM.PSSP.aixos.Mem.Real	NodeNum	56
IBM.PSSP.aixos.Mem.Virt	NodeNum	57
IBM.PSSP.aixos.PagSp	NodeNum	60
IBM.PSSP.aixos.Proc	NodeNum	61
IBM.PSSP.aixos.VG	NodeNum	50
IBM.PSSP.aixos.VG	VG	51
IBM.PSSP.aixos.cpu	CPU	44
IBM.PSSP.aixos.cpu	NodeNum	43
IBM.PSSP.aixos.pagsp	NodeNum	58
IBM.PSSP.aixos.pagsp	Name	59
IBM.PSSP.pm	NodeNum	3

Note:

ivElement_description is a particular message number in the set specified by *rmMessage_set*. Refer to Table 6 on page 236.

D.4.2 Why Is It Useful?

This information is useful when you want to know what kind of Instance Vectors the -e option of the pmandef command needs. Follow these instructions:

1. In *ivResource_name*, find the Resource Class that your Resource Variable belongs to.
2. *ivElement_name* is the Instance Vector that you need when you use the pmandef command with the -e option.

D.5 Event Management Resource Variable Class

This class contains information about the Resource Variable class.

You can get this information by using the following command:

```
# SDRGetObjects EM_Resource_Variable | more
```

You must use this command with the more operand because it shows more than 300 Resource Variables.

If you are interested in a particular Resource Class, such as IBM.PSSP.pm, use the following command:

```
# SDRGetObjects EM_Resource_Variable rvClass==IBM.PSSP.pm
```

D.5.1 What Can You Get?

The following is a default implementation:

Table 10. Event Management Resource Variable Class

Class	**sample 1**	**sample 2**
rvName	IBM.PSSP.pm.Errlog	IBM.PSSP.pm.User_state1
rvDescription	1	2
rvValue_type	State	State
rvData_type	SBS	SBS
rvInitial_value		
rvClass	IBM.PSSP.pm	IBM.PSSP.pm
rvPTX_name		
rvPTX_description		
rvPTX_min		
rvPTX_max		
rvPredicate		
rvEvent_description		
rvLocator	NodeNum	NodeNum
rvDynamic_instance	0	0
rvIndex_vector		

D.5.2 Why Is It Useful?

To help you choose one of the 300 available Resource Variables, refer to Table 10.

If you pick a Resource Variable and want to know its description, use *rvDescription*. Refer to Table 6 on page 236.

Creating your own monitor also means that you have to find proper options for the pmandef command. You can use *rvName* with the -e option for a Resource Variable. If *rvData_type* is SBS, you may need to look up Event Management Structured Byte String SDR class with the -e option for a predicate.

D.6 Event Management Structured Byte String Class

This class contains information about the Instance Vector for a particular Resource Variable.

You can learn about it by using the following command:

```
# SDRGetObjects EM_Structured_Byte_String
```

D.6.1 What Can You Get?

The following is a default implementation:

Table 11. Event Management Structured Byte String Class

sbsVariable_name	sbsField_name	sbsField_type	sbsField_SN	sbsField_init_val
IBM.PSSP.Prog.pcount	CurPIDCount	long	0	
IBM.PSSP.Prog.pcount	PrevPIDCount	long	1	
IBM.PSSP.Prog.pcount	CurPIDList	cstring	2	
IBM.PSSP.Prog.xpcount	CurPIDCount	long	0	
IBM.PSSP.Prog.xpcount	PrevPIDCount	long	1	
IBM.PSSP.Prog.xpcount	CurPIDList	cstring	2	
IBM.PSSP.SP_HW.Node.lcd1	STRING	cstring	0	
IBM.PSSP.SP_HW.Node.lcd2	STRING	cstring	0	
IBM.PSSP.pm.Errlog	sequenceNumber	cstring	0	
IBM.PSSP.pm.Errlog	errorID	cstring	1	
IBM.PSSP.pm.Errlog	errorClass	cstring	2	
IBM.PSSP.pm.Errlog	errorType	cstring	3	
IBM.PSSP.pm.Errlog	alertFlagsValue	cstring	4	
IBM.PSSP.pm.Errlog	resourceName	cstring	5	
IBM.PSSP.pm.Errlog	resourceType	cstring	6	
IBM.PSSP.pm.Errlog	resourceClass	cstring	7	
IBM.PSSP.pm.Errlog	errorLabel	cstring	8	
IBM.PSSP.pm.User_state1	STRING	cstring	0	

Note:

*1 is *sbsField_init_val.*

This shows you that Resource Variable *sbsVariable_name* uses a Structured Byte String. Field Serial Number *sbsField_SN* is named *sbsField_name* and its variable type is *sbsField_type*. Its initial value is *sbsFiled_init_val*.

D.6.2 Why Is It Useful?

If you monitor a Resource Variable that uses data type Structured Byte String, you need to know the contents of Structured Byte String, and you need to specify Predicate in the pmandef command with the -e option. X@0 represents *sbsField_name* with *sbsField_SN* equal to 0. X@1 represents *sbsField_name* with *sbsField_SN* equal to 1, and so on.

D.7 Problem Management Daemon (pmand)

D.7.1 SDR Class

There is one SDR class for pmand, pmandConfig.

You can learn about it by using the following command:

```
# SDRGetObjects pmandConfig
```

The following is a list of the members of this class:

- pmEventid
- pmNodenumber
- pmTargetType
- pmTarget
- pmRvar
- pmIvec
- pmPred
- pmCommand
- pmCommandTimeout
- pmHandle
- pmTrapid
- pmPPSlog
- pmThrottle
- pmRearmPred
- pmRearmTrapid
- pmRearmPPSlog
- pmRearmCommand
- pmRearmCommandTimeout
- pmUsername
- pmPrincipal
- pmHost
- pmActivated
- pmText
- pmRearmText
- pmUserLabel

D.7.2 Commands

pmanctrl Controls the Problem Management subsystem.

```
Usage: pmanctrl {-a|-s|-k|-d|-c|-t|-o|-r|-h}
   -a Add Problem Management subsystem to this partition.
   -s Start Problem Management subsystem in this partition.
   -k Stop Problem Management subsystem in this partition.
   -d Delete Problem Management subsystem from this partition.
   -c Remove Problem Management subsystem from all partitions
      (but do not remove SDR objects).
   -t Start Problem Management subsystem trace in this partition.
   -o Stop Problem Management subsystem trace in this partition.
   -r Refresh the Problem Management subsystem on all nodes in this
      partition.  (This option currently does nothing.)
   -h Display this usage statement.
```

pmandef Defines events and resulting actions to Problem Management.

To subscribe to an event and associate a set of actions with that event, use the following:

```
Usage: pmandef -s HandleName -e ResourceVariable:InstanceVector:Predicate
   [-r RearmPredicate]
   [-c EventCommand] [-C RearmCommand]
   [-t EventTrapid] [-T RearmTrapid]
   [-l EventLogText] [-L RearmLogText]
   [-x EventCmdTimeout] [-X RearmCmdTimeout]
   [-U UserName] [-m Label]
   [-h { Host1[,Host2,...] | - | local } | -N NodeGroup | -n NodeRange]
```

To deactivate or activate a Problem Management subscription, use the following:

```
Usage: pmandef {-d|-a} {HandleName | all}
   [-h { Host1[,Host2,...] | - | local } | -N NodeGroup | -n NodeRange]
```

To query or remove a Problem Management subscription, use the following:

```
Usage: pmandef {-q|-u} {HandleName | all}
```

pmanquery Queries the System Data Repository (SDR) for a description of a Problem Management subscription.

```
Usage: pmanquery -n {HandleName | all} [-k {Kerberos_principal | all}]
   [-q|-a|-d|-t]
```

D.8 Problem Management Resource Monitor Daemon (pmanrmd)

The pmand daemon provides sophisticated Resource Variables to monitor your SP system, but you may want to monitor your unique resources. That is what pmanrmd is used for. You can create up to 16 of your own Resource Variables

D.8.1 SDR Class for pmanrmd

There is one SDR class for pmanrmd, pmanrmdConfig.

You can learn about it by using the following command:

```
# SDRGetObjects pmanrmdConfig
```

The following is a list of the members of this class:

pmrmNodenumber The node number of the host on which pmanrmd is to instantiate the Resource Variable being defined.

pmrmTargetType Either NODE_LIST, NODE_RANGE, or NODE_GROUP.

pmrmTarget This specifies the set of nodes on which the daemons are to be configured. It can be a list of hostnames, a node range as specified on the hostlist command, or the name of a node group. A predefined NODE_LIST, CWS, is provided. It refers to the Control Workstation.

pmrmRvar This specifies the fully-qualified Resource Variable name in the resource class user_state.

pmrmCommand This command provides the Resource Variable value, consisting of its stdout.

pmrmSampInt This specifies the sampling interval in seconds.

D.8.2 Configuration File

The following is a sample configuration file:

/spdata/sys1/pman/pmanrmd.conf

D.8.3 Resource Variables

Sixteen user-configurable Resource Variables are provided:

IBM.PSSP.pm.User_state*nn*, where *nn*=1-16

D.8.4 Commands

pmanrmdloadSDR

This command reads a pmanrmd configuration file and loads the information into the System Data Repository (SDR).

```
Usage:
   pmanrmdloadSDR filename
```

pmanrminput

```
Usage:
   pmanrminput [-h host] [-a argument] -g group_name
   pmanrminput [-h host] [-a argument] -s subsystem_name
   pmanrminput [-h host] [-a argument] -p subsystem_pid
```

Appendix E. The IBM.PSSP.Prog.pcount Resource Variable

IBM.PSSP.Prog.pcount represents processes running a specified program on behalf of a specified user. The Resource Variable′s Instance Vector specifies the program name (ProgName), user name (UserName), and node number (NodeNum) of interest. The Resource Variable′s value is a structured byte string that includes:

- A count of the current number of processes running the program for the user (field serial number 0: long data type)
- A count of the previously known number of processes that had been running the program for the user (field serial number 1: long data type)
- A comma-separated list of the process identifiers (PIDs) of the processes currently running the program for the user (field serial number 2: character string data type)

For example, assume the six processes shown in the following ps output are running the biod program on Node 5:

```
# ps -ef | grep -v grep | grep biod
    root  7786  8040   0 20:53:34      -  0:00 /usr/sbin/biod 6
    root  8040  5624   0 20:53:34      -  0:00 /usr/sbin/biod 6
    root  8300  8040   0 20:53:34      -  0:00 /usr/sbin/biod 6
    root  8558  8040   0 20:53:34      -  0:00 /usr/sbin/biod 6
    root  8816  8040   0 20:53:34      -  0:00 /usr/sbin/biod 6
    root  9074  8040   0 20:53:34      -  0:00 /usr/sbin/biod 6
#
```

The value of the IBM.PSSP.Prog.pcount instance represented by the Instance Vector ProgName=biod;UserName=root;NodeNum=5 would include a current process count of 6 and a current process list of 7786,8040,8300,8558,8816,9074. The value of the previous process count would depend on the history of processes running the biod program on Node 5.

A change in the value of an IBM.PSSP.Prog.pcount instance either indicates that fewer processes are running the program represented by the Resource Variable instance or more processes are running the program. Which condition is indicated can be determined by comparing the value of the current process count with the value of the previous process count.

To be informed whenever the set of processes running the biod program for the root user changes on Node 5, you could register for an event specified as follows:

```
    Resource Variable Name: IBM.PSSP.Prog.pcount
    Instance Vector:        ProgName=biod;UserName=root;NodeNum=5
    Predicate:              X@0 != X@1
```

To be informed only when no processes are running the biod program for the root user on Node 5, you could register for an event specified as follows:

```
Resource Variable Name: IBM.PSSP.Prog.pcount
Instance Vector:        ProgName=biod;UserName=root;NodeNum=5
Predicate:              X@0 == 0
```

E.1 Limitations

The IBM.PSSP.Prog.pcount Resource Variable is designed to be used to monitor programs that are expected to have long lifetimes. If it is used to monitor a program that runs for only a few seconds, it is possible that not all processes that run the program will be detected.

E.1.1 Instance Vector Wildcarding

Neither the ProgName nor the UserName Instance Vector element may be wildcarded.

The NodeNum Instance Vector element may be wildcarded.

E.2 Related Resource Variable IBM.PSSP.Prog.xpcount

There is another Resource Variable closely related to IBM.PSSP.Prog.pcount. It is named IBM.PSSP.Prog.xpcount. These two Resource Variables differ in the following way:

- The pcount Resource Variable represents all processes running a specified program, regardless of why the processes are running the program.
- The xpcount Resource Variable represents only those processes that are running a specified program as a result of having called one of the exec() routines.

A typical process runs a program because it called an exec() routine specifying the program. However, a process may be running a program because it inherited it from its parent process, and it never called an exec() routine to execute another program. Some daemons, including biod, do this.

For example, the ps output that follows shows that only the process whose PID is 8040 called an exec() routine to run the biod program. All the other processes running biod do not have the SEXECED process flag (200000) set; they inherited the biod program from their parent process—the process with PID 8040.

```
# ps -e -o ″flag,pid,ppid,comm″ | grep biod
   40001  7786  8040 biod
  240001  8040  5624 biod
   40001  8300  8040 biod
   40001  8558  8040 biod
   40001  8816  8040 biod
   40001  9074  8040 biod
#
```

Use the IBM.PSSP.Prog.pcount Resource Variable to monitor processes that inherit programs.

Use the IBM.PSSP.Prog.xpcount Resource Variable to monitor only processes that explicitly call an exec() routine to run a program.

Using IBM.PSSP.Prog.pcount to monitor certain programs can cause problems. For example, if a program creates child processes that run other programs, registering for an event using pcount may generate unintended events.

Consider the inetd program as an example. Its purpose is to spawn other service daemons on request. As part of the process, inetd calls fork() to create a child process. That child process starts out running the inetd program, but quickly calls an exec() routine to run the appropriate daemon (such as telnetd). Therefore, for brief periods of time, more than one process will be running the inetd program.

If you use pcount to monitor inetd, events might be generated showing these child processes running inetd for small periods of time. To avoid such false events, it would be better to use xpcount to monitor inetd.

As you can see, you should know how a program operates before deciding how to monitor it. In most cases, it is probably appropriate to use the xpcount Resource Variable to monitor a program. However, if the program to be monitored is inherited by long-running processes that do not call an exec() routine, pcount might be more appropriate.

Appendix F. The IBM.PSSP.Prog.xpcount Resource Variable

IBM.PSSP.Prog.xpcount represents processes running a specified program on behalf of a specified user. The Resource Variable′s Instance Vector specifies the program name (ProgName), user name (UserName), and node number (NodeNum) of interest. The Resource Variable′s value is a structured byte string that includes:

- A count of the current number of processes running the program for the user (field serial number 0: long data type)
- A count of the previously known number of processes that had been running the program for the user (field serial number 1: long data type)
- A comma-separated list of the process identifiers (PIDs) of the processes currently running the program for the user (field serial number 2: character string data type)

For example, assume the process shown in the following ps output is running the inetd program on Node 5:

```
# ps -ef | grep -v grep | grep inetd
    root  7218  5624   0   Jul 18      -  0:00 /usr/sbin/inetd
#
```

The value of the IBM.PSSP.Prog.xpcount instance represented by the Instance Vector ProgName=inetd;UserName=root;NodeNum=5 would include a current process count of 1 and a current process list of 7218. The value of the previous process count would depend on the history of processes running the inetd program on Node 5.

A change in the value of an IBM.PSSP.Prog.xpcount instance either indicates that fewer processes are running the program represented by the Resource Variable instance, or more processes are running the program. Which condition changed the value can be determined by comparing the value of the current process count with the value of the previous process count.

To be informed whenever the set of processes running the inetd program for the root user changes on Node 5, you could register for an event specified as follows:

```
    Resource Variable Name: IBM.PSSP.Prog.xpcount
    Instance Vector:        ProgName=inetd;UserName=root;NodeNum=5
    Predicate:              X@0 != X@1
```

To be informed only when no processes are running the inetd program for the root user on Node 5, you could register for an event specified as follows:

```
    Resource Variable Name: IBM.PSSP.Prog.xpcount
    Instance Vector:        ProgName=inetd;UserName=root;NodeNum=5
    Predicate:              X@0 == 0
```

F.1 Limitations

The IBM.PSSP.Prog.xpcount Resource Variable is designed to be used to monitor programs that are expected to have long lifetimes. If it is used to monitor a program that runs for only a few seconds, not all processes that run the program may be detected.

F.1.1 Instance Vector Wildcarding

Neither the ProgName nor the UserName Instance Vector element may be wildcarded.

The NodeNum Instance Vector element may be wildcarded.

F.2 Related Resource Variable IBM.PSSP.Prog.pcount

There is another Resource Variable closely related to IBM.PSSP.Prog.xpcount. It is named IBM.PSSP.Prog.pcount. These Resource Variables differ in the following way:

- The pcount Resource Variable represents all processes running a specified program, regardless of why the processes are running the program.
- The xpcount Resource Variable represents *only* those processes that are running a specified program as a result of having called one of the exec() routines.

A typical process runs a program because it called an exec() routine specifying the program. However, a process may be running a program because it inherited it from its parent process, and it never called an exec() routine to execute another program. Some daemons, including biod, do this.

For example, the ps output that follows shows that only the process whose PID is 8040 called an exec() routine to run the biod program. All the other processes running biod do not have the SEXECED process flag (200000) set; they inherited the biod program from their parent process—the process with PID 8040.

```
# ps -e -o "flag,pid,ppid,comm" | grep biod
   40001  7786  8040 biod
  240001  8040  5624 biod
   40001  8300  8040 biod
   40001  8558  8040 biod
   40001  8816  8040 biod
   40001  9074  8040 biod
#
```

Use the IBM.PSSP.Prog.pcount Resource Variable to monitor processes that inherit programs.

Use the IBM.PSSP.Prog.xpcount Resource Variable only to monitor processes that explicitly call an exec() routine to run a program.

Using IBM.PSSP.Prog.pcount to monitor certain programs may cause problems. For example, if a program creates child processes that run other programs, registering for an event using pcount may generate unintended events.

Consider the inetd program as an example. Its purpose is to spawn other service daemons on request. As part of the process, inetd calls fork() to create a child process. That child process starts out running the inetd program, but quickly calls an exec() routine to run the appropriate daemon (such as telnetd). Therefore, for brief periods of time, more than one process will be running the inetd program.

If you use pcount to monitor inetd, events might be generated showing these child processes running inetd for small periods of time. To avoid these false events, it would be better to use xpcount to monitor inetd.

As you can see, you should know how a program operates before you decide how to monitor it.

In most cases, it is probably appropriate to use the xpcount Resource Variable to monitor a program. However, if the program to be monitored is inherited by long-running processes that do not call an exec() routine, pcount might be more appropriate.

Appendix G. SNMP-Related Request For Comments

RFC1910 “User-Based Security Model for SNMPv2”
G. Waters

RFC1909 “An Administrative Infrastructure for SNMPv2”
K. McCloghrie

RFC1907 “Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)”
SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose and S. Waldbusser
(Obsoletes RFC 1450)

RFC1906 “Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)”
SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose and S. Waldbusser
(Obsoletes RFC 1449)

RFC1905 “Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)”
SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose and S. Waldbusser
(Obsoletes RFC 1448)

RFC1904 “Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)”
SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose and S. Waldbusser
(Obsoletes RFC 1444)

RFC1903 “Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)”
SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose and S. Waldbusser
(Obsoletes RFC 1443)

RFC1901 “Introduction to Community-Based SNMPv2”
SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose and S. Waldbusser

RFC1503 “Algorithms for Automating Administration in SNMPv2 Managers”
K. McCloghrie and M. Rose

RFC1461 “SNMP MIB Extension for Multiprotocol Interconnect over X.25”
D. Throop

RFC1450 “Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)”
J. Case, K. McCloghrie, M. Rose, and S. Waldbusser
(Obsoleted by RFC 1907)

RFC1449 “Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)”
J. Case, K. McCloghrie, M. Rose, and S. Waldbusser
(Obsoleted by RFC 1906)

RFC1448 “Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)”
J. Case, K. McCloghrie, M. Rose, and S. Waldbusser
(Obsoleted by RFC 1905)

RFC1447 “Party MIB for Version 2 of the Simple Network Management Protocol (SNMPv2)”
K. McCloghrie and J. Galvin

RFC1446 “Security Protocols for Version 2 of the Simple Network Management Protocol (SNMPv2)”
J. Galvin and K. McCloghrie

RFC1445 “Administrative Model for Version 2 of the Simple Network Management Protocol (SNMPv2)”
J. Galvin and K. McCloghrie

RFC1444 “Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)”
J. Case, K. McCloghrie, M. Rose, and S. Waldbusser

RFC1443 “Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)”
J. Case, K. McCloghrie, M. Rose, and S. Waldbusser

RFC1442 “Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)”
J. Case, K. McCloghrie, M. Rose, and S. Waldbusser

RFC1420 “SNMP over IPX”
S. Bostock

RFC1419 “SNMP over AppleTalk”
G. Minshall and M. Ritter

RFC1418 “SNMP over OSI”
M. Rose

RFC1382 “SNMP MIB Extension for the X.25 Packet Layer”
D. Throop

RFC1381 “SNMP MIB Extension for X.25 LAPB”
D. Throop and F. Baker

RFC1352 “SNMP Security Protocols”
J. Galvin, K. McCloghrie, and J. Davin

RFC1351 “SNMP Administrative Model”
J. Davin, J. Galvin, and K. McCloghrie

RFC1303 “A Convention for Describing SNMP-Based Agents”
K. McCloghrie and M. Rose

RFC1298 “SNMP over IPX”
R. Wormley and S. Bostock
(Obsoleted by RFC 1420)

RFC1283 “SNMP over OSI”
M. Rose
(Obsoleted by RFC 1418)

RFC1270 “SNMP Communications Services”
F. Kastenholz

RFC1228 “SNMP-DPI: Simple Network Management Protocol Distributed Program Interface”
G. Carpenter and B. Wijnen
(Obsoleted by RFC 1592)

RFC1227 “SNMP MUX Protocol and MIB”
M.T. Rose

RFC1215 “Convention for Defining Traps for Use with the SNMP”
M.T. Rose

RFC1187 “Bulk Table Retrieval with the SNMP”
M.T. Rose, K. McCloghrie, and J.R. Davin

This memo reports an interesting family of algorithms for bulk table retrieval using the Simple Network Management Protocol (SNMP). It describes an Experimental Protocol for the Internet community, and requests discussion and suggestions for improvements.

It does not specify a standard for the Internet community. Refer to the current edition of the *IAB Official Protocol Standards* for the standardization state and status of this protocol.

RFC1161 “SNMP over OSI”
M.T. Rose
(Obsoleted by RFC 1418)

This memo defines an experimental means for running the Simple Network Management Protocol (SNMP) over OSI transports.

It does not specify a standard for the Internet community.

RFC1157 “Simple Network Management Protocol (SNMP)”
J.D. Case, M. Fedor, M.L. Schoffstall, and C. Davin
(Obsoletes RFC 1098)

This RFC is a re-release of RFC 1098, with a changed “Status of this Memo” section plus a few minor typographical corrections. It defines a simple protocol by which management information for a network element may be inspected or altered by logically remote users. [STANDARDS-TRACK]

RFC1098 “Simple Network Management Protocol (SNMP)”
J.D. Case, M. Fedor, M.L. Schoffstall, and C. Davin
(Obsoletes RFC 1067) (Obsoleted by RF1157)

This RFC is a re-release of RFC 1067, with a changed “Status of this Memo” section. It defines a simple protocol by which management information for a network element may be inspected or altered by logically remote users. Together with companion memos that describe the structure of management information along with the initial management information base, this document provides a simple, workable architecture and system for managing TCP/IP-based internets and, in particular, the Internet.

RFC1089 “SNMP over Ethernet”
M.L. Schoffstall, C. Davin, M. Fedor, and J.D. Case

This memo describes an experimental method by which the Simple Network Management Protocol (SNMP) can be used over Ethernet

MAC layer framing instead of the Internet UDP/IP protocol stack. This specification is useful for LAN-based network elements that support no higher layer protocols beyond the MAC sublayer.

G.1 How to Get RFCs

There are many sites in the Internet where you can get these RFCs. Here are some pointers:

- ftp://ds.internic.net/rfc
- ftp://nis.nfs.net/internet/documents/rfc
- ftp://src.doc.ic.ac.uk/computing/internet/rfc
- ftp://wuarchive.wustl.edu/doc/rfc

Appendix H. How to Get the Examples in This Book

The examples in this publication are available on a diskette that is included with paper copies of the book, at an FTP site, and via the World Wide Web.

H.1 Diskette Version

The files are stored in tar format. The examples will extract to the location ./SPmonitoring/..., where ... includes directories like src, bin, and lib. The recommended location for the examples is /usr/local/SPmonitoring.

To extract the examples from the diskette and place them in the recommended location, use these commands after inserting the diskette into the drive on the RS/6000:

```
#
# mkdir /usr/local
# cd /usr/local
# tar xvf /dev/rfd0
#
```

Figure 132. Installing Examples from Diskette to the Recommended Location

H.2 FTP Site

The files are available for anonymous FTP from www.redbooks.ibm.com. To retrieve the files using FTP, you must have access to the Internet. If you do, use the following procedure:

```
#
# mkdir /usr/local
# cd /usr/local
# ftp www.redbooks.ibm.com
ftp> bin
ftp> cd /redbooks/SG244873
ftp> get SPmonitoring.tar.Z
ftp> quit
#
# uncompress SPmonitoring.tar.Z
# tar xvf SPmonitoring.tar
rm SPmonitoring.tar
#
```

Figure 133. Installing Examples to the Recommended Location Using FTP

H.3 WWW Site

The examples can also be downloaded using the World Wide Web. The URL www.redbooks.ibm.com provides details on the procedure.

Note:

These examples have been tested on pre-release versions of POWERparallel System Support Programs Version 2.2. They may not be suitable for use in a particular environment, and are not supported by IBM in any way. IBM is not responsible for your use of these examples.

Appendix I. Special Notices

This publication is intended to help IBM customers, Business Partners, IBM System Engineers and other RS/6000 SP specialists who are involved in POWERparallel System Support Programs Version 2.2 projects, including education of RS/6000 SP professionals responsible for installing, configuring, and administering PSSP Version 2 Release 2. The information in this publication is not intended as the specification of any programming interfaces that are provided by POWERparallel System Support Programs. See the PUBLICATIONS section of the IBM Programming Announcement for POWERparallel System Support Programs for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
AIX/ESA
AS/400
BookManager
ES/9000
IBM
LoadLeveler
NetView
OS/2
POWERparallel
PROFS
RS/6000
SP
System/390

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

Other trademarks are trademarks of their respective companies.

Appendix J. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

J.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 263.

- *RS/6000 SP PSSP 2.2 Technical Presentation*, SG24-4868
- *RS/6000 SP High Availability Infrastructure*, SG24-4838
- *Examples of Using NetView for AIX, V4*, SG24-4515

J.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

J.3 Other Publications

These publications are also relevant as further information sources:

- *RS/6000 SP: Administration Guide*, GC23-3897
- *RS/6000 SP: Installation and Migration Guide*, GC23-3898
- *RS/6000 SP: Diagnosis and Messages Guide*, GC23-3899
- *RS/6000 SP: Command and Technical Reference*, GC23-3900
- *RS/6000 SP: System Planning Guide*, GC23-3902
- *Examples of Using NetView for AIX*, GG24-4327
- *RS/6000 SP: Event Management Programming Guide and Reference*, SC23-3996
- *RS/6000 SP: Group Services Programming Guide and Reference*, SC28-1675
- *NetView for AIX User's Guide for Beginners V4*, SC31-8158 (available in softcopy only)
- *NetView for AIX Administrator's Guide V4*, SC31-8168 (available in softcopy only)

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://www.redbooks.ibm.com

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications
Publications Customer Support
P.O. Box 29570
Raleigh, NC 27626-0570
USA

IBM Publications
144-4th Avenue, S.W.
Calgary, Alberta T2P 3N5
Canada

IBM Direct Services
Sortemosevej 21
DK-3450 Allerød
Denmark

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
(+45) 48 14 2207 (long distance charge)	Outside North America

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number

- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

ACL	Access Control List
AIX	Advanced Interactive Executive
AMG	Adapter Membership Group
ANS	Abstract Notation Syntax
APA	All Points Addressable
API	Application Programming Interface
BIS	Boot-Install Server
BSD	Berkeley Software Distribution
BUMP	Bring-Up Microprocessor
CP	Crown Prince
CPU	Central Processing Unit
CSS	Communication Subsystem
CWS	Control Workstation
EM	Event Management
EMAPI	Event Management Application Programming Interface
EMCDB	Event Management Configuration Database
EMD	Event Manager Daemon
EPROM	Erasable Programmable Read Only Memory
FIFO	First In - First Out
GB	Gigabytes
GL	Group Leader
GS	Group Services
GSAPI	Group Services Application Programming Interface
hb	heart beat
HPS	High Performance Switch
hrd	host respond daemon
HSD	Hashed Shared Disk
IBM	International Business Machines Corporation
IP	Internet Protocol
ISB	Intermediate Switch Board
ISC	Intermediate Switch Chip
ITSO	International Technical Support Organization
JFS	Journal File System
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light Emitter Diode
LRU	Least Recently Used
LSC	Link Switch Chip
LVM	Logical Volume Manager
MB	Megabytes
MIB	Management Information Base
MPI	Message Passing Interface
MPL	Message Passing Library
MPP	Massively Parallel Processors
NIM	Network Installation Manager
NSB	Node Switch Board
NSC	Node Switch Chip
OID	Object ID
ODM	Object Data Manager
PE	Parallel Environment
PID	Process ID
PROFS	Professional Office System
PSSP	Parallel System Support Program
PTC	Prepare to Commit
PTPE	Performance Toolbox Parallel Extensions
PTX/6000	Performance Toolbox/6000
RAM	Random Access Memory
RCP	Remote Copy Protocol
RM	Resource Monitor
RMAPI	Resource Monitor Application Programming Interface
RPQ	Request for Product Quotation
RSI	Remote Statistics Interface
RVSD	Recoverable Virtual Shared Disk
SBS	Structured Byte String
SDR	System Data Repository
SMP	Symmetric Multiprocessors
SNMP	System Network Management Protocol
SPDM	SP Data Manager

SPMI	System Performance Measurement Interface
SRC	System Resource Controller
SSI	Single System Image
TS	Topology Services
TCP/IP	Transmission Control Protocol / Internet Protocol
UDP	User Datagram Protocol
VSD	Virtual Shared Disk
VSM	Visual System Management

Index

A

abbreviations 267
acronyms 267
Adapter Membership group 5
  algorithm 6
  prepare to commit 6
AIX error log 72, 84, 174, 197
aixos 73
Alert attribute 174
AMG
  See Adapter Membership group
  See Topology Services
API
  See Application Program Interfaces
Application Program Interfaces
  configuration of Resource Monitor 128
  example of use 98
  example Resource Monitor 119
  example Resource Monitor httprtA 119
  helpful hints 97
  LPP requirements 97
ASN 48, 49
availability 27

B

bibliography 261
bos.net.tcp.server 194
BSD syslog 72, 84

C

commands
  dspcat 75
  errpt 200
  errupdate 200
  haemcfg 51, 53
  haemctrl 54
  haemloadcfg 52
  haemrcpcdb 51
  hagsgr 15
  hagsmg 12
  hagsreap 10
  lssrc 185
  Perspectives 138
  pmanctrl 90
  pmandef 75, 83, 90
  pmanquery 83, 86, 90
  pmanrmdloadSDR 80, 90, 92
  pmanrminput 77, 82, 90
  runcat 42
  SDR_test 156
  SDRArchive 156
  SDRCreateObjects 52
commands *(continued)*
  SDRDeleteObjects 92, 156
  SDRGetObjects 9, 73, 92, 143, 149, 156, 161
  SDRListClasses 46, 156
  SDRListFiles 156
  SDRRetrieveFile 156
  SDRWhoHasLock 156
  snmpget 194
  snmpinfo 194
  sp_configd 197
  spevent 138
  sphardware 138
  spmon 218
  startsrc 184
  stopsrc 185
  su 78
  xmpeek 37
condition
  components of 152
  default predefined conditions 170
condition name 168
condition value 169
control desk 199, 213
core dump file 10
Counter 18, 67
CP
  See crown prince
  See Topology Services
CPU_Idle_Monitor 87, 160
crown prince 5
cssMembership 15
CurPIDCount 162
CurPIDList 162

D

daemons
  haemd 15, 22
  hagsd 8, 9, 12, 13
  hagsglsmd 15
  hatsd 7, 9
  hbd 66
  hrd 65
  pmand 63, 72
  pmanrmd 72
  sendmail 165
  sp_configd 174, 176, 201
directories
  /spdata/sys1/ha/cfg 51, 56
  /usr/lib/nls/msg 75
  /usr/lpp/bin 138
  /usr/lpp/ssp/config/snmp_proxy 182
  /usr/lpp/ssp/samples 27
  /usr/OV/conf 181
  /usr/OV/conf/C 187

directories *(continued)*
  /var/adm/sulog 78
  /var/ha 201
  /var/ha/lck/hags.tid.<syspar_name> 12
  /var/ha/log 10, 21
  /var/ha/soc/hats 7
  /var/rmp 198
  /var/tmp 201
downstream_neighbor 6
Dynamic Data Supplier (DDS) 36
dynamic workspace 217

E

EM_Condition 46, 169
EM_Instance_Vector 46, 75, 80
EM_Resource_Class 39, 46, 52, 73
EM_Resource_Monitor 47, 52, 74
EM_Resource_Variable 33, 47, 52, 73, 81
EM_Structured_Byte_String 47, 81
EMAPI 17, 65, 174
  See also ?
EMCDB 51, 52, 53, 59
  See also Event Management Configuration Database
enMembership 4, 15
errnotify 198
error stack 159
errpt 200
errupdate 200
event 31, 58, 59
event definition
  authorizations needed 139, 156
  components of 153
  CPU_Idle_Monitor 160
  creating 139, 141
  definition page 141
  deleting 139, 156
  Event Definition name 142
  managing 137
  modifying 139, 156
  name 142, 154
  realMemLowEvent 142
  rearm-command 147
  rearm-event 147
  registering 139, 144
  response options 145
  select resources 144
  storage 154
  target nodes 147
  unregistering 139, 144
  varFullEvent 146
  viewing 139
Event Definitions pane 141, 146
event generation 57
Event Management
  client programs 29
  daemon 21
  Event Management 28, 68

Event Management *(continued)*
  generation 57
  haemd daemon 15
  instantiation vector 18
  notification 58
  overview 16
  predicate 17
  predicates 61
  registration 58
  subsystem 19
  wildcarding 58
Event Management Configuration Database
  accepting join requests 57
  in the group 57
  out of the group 57
  peer count 56
  peer daemon status 57
  rejecting join requests 57
  version number 56
Event Management daemon 29
Event Manager API example 98
Event Notification Log 146
Event Perspective
  invoking 138, 140
  predefined condition 142
  terminology considerations 152
  tool bar 140
  using 139
event registration 58
events 137, 139
example applications
  configuration of Resource Monitor 128
  constructing EMAPI clients 98
  example Resource Monitor 119
  example Resource Monitor httprtA 119
  getemv 111
  helpful hints 97
  how to get examples 257
  ideas on designing and writing 233
  LPP requirements 97
  lsemv 101
  measuring response time of httpd 226
  monemv 115
example event definitions
  monitoring a daemon 165
  monitoring CPU usage 160
  monitoring file system size 148
  monitoring memory usage 140
example programs
  configuration of Resource Monitor 128
  constructing EMAPI clients 98
  example Resource Monitor 119
  example Resource Monitor httprtA 119
  getemv 111
  helpful hints 97
  how to get examples 257
  ideas on designing and writing 233
  LPP requirements 97

example programs *(continued)*
- lsemv 101
- measuring response time 226
- monemv 115

expression 139, 144, 147, 151, 152, 154, 161, 165, 168
- *See also* predicate

extensibility 67

F

FIFO 201
file changed monitor 94
file system monitor 93
files
- /etc/logmgmt.acl 197
- /etc/services 179
- /etc/snmpd.conf 179
- /etc/snmpd.peers 181
- /etc/sysctl.pman.acl 156
- /usr/OV/conf/ovsnmp.conf 180
- /var/ha/log/gs.default.<syspar_name> 9
- /var/ha/soc/em.clsrv.<syspar_name> 22
- /var/ha/soc/em.RM.<rmname>.<syspar_name> 22
- /var/ha/soc/em.rmsrv.<syspar_name> 22
- /var/ha/soc/hagsd.<syspar_name> 9
- .klogin 84
- .kshrc 158
- .startup_script 138
- $HOME/.$USER&Events 146
- errlog_entry 198
- harm_des.cat 75
- ibmSPMIB.defs 182
- ibmSPMIB.my 182
- pmanrmd.conf 78
- rmapi_smp 42
- rmapi_smp.cat 42
- rmapi_smp.msg 42
- snmp_trap_gen 198
- snmp.log 203
- sp_configd.log 201
- sysctl.pman.acl 83
- trapd.conf 187
- trapd.log 182

G

getemv 111
GL
- *See* group leader
- *See* Topology Services

group leader 5
Group Services
- connection with Topology Services 7
- core dump file 10
- cssMembership 15
- csstMembership 65
- daemon 8
- domain 10
- enMembership 15, 65

Group Services *(continued)*
- failure leave 14
- group state value 14
- ha_em_peers 15
- hagsd 8, 9
- hagsglsmd 15
- hagsglsmd daemon 15
- hagsgr command 15
- hagsmg 12
- hb_init 10
- HostMembership 15, 65
- log file space 10
- log files 9
- MetaGroup 12
- name server 10
- name server protocol 10
- namespace 10
- nomination messages 11
- overview 8
- provider 14
- subscriber 14
- SwitchGroupie 15
- voting protocol 15

Group Services API
Group Services domain
group state value 14
GSAPI 10
- *See also* Group Services API

H

ha_em_end_session() 102
ha_em_peers 15
ha_em_receive_response() 106
ha_em_restart_session() 107
ha_em_send_command() 105
ha_em_start_session() 102
haemcfg 51, 52, 53
haemctrl 54
haemloadcfg 52
haemrcpcdb 51
hagsd 8, 9
hagsglsmd 15
hagsgr 15
hagsmg 12
hagsreap 10
handling SIGPIPE 102
hardmon 22
harm_des.cat 75
harmld 63
harmpd 63
hatsd 7, 9
hb_init 10
hbd 66
- *See also* heartbeat

HDS 137
heartbeat 3
- daemon 66

High Availability Cluster Multiprocessing 28
High Availability Infrastructure
hmrmd 63
host responds
  daemon 65
host_response 65
HostMembership 4, 15
hostResponds 167
hrd 65
  See also host responds
httprtA 119

I
IBM.PSSP.aixos.CPU 73, 161
IBM.PSSP.aixos.FS 151
IBM.PSSP.harmld 64
IBM.PSSP.harmpd 64
IBM.PSSP.hmrmd 64
IBM.PSSP.Membership.LANAdapter.state 65
IBM.PSSP.Membership.Node.state 65
IBM.PSSP.pm 80
IBM.PSSP.pmanrmd 64, 80
IBM.PSSP.Prog.pcount 162
IBM.PSSP.Prog.xpcount 162
IBM.PSSP.Response.Host.state 65
IBM.PSSP.Response.Switch.state 65
IBM.PSSP.SP_HW.Node.keyModeSwitch 208
ibmProd 177
ibmSP.MIB 180
ibmSPConfig 177
ibmSPEMVariable 177
ibmSPErrlogVars 177
inherited process 163
Instance Vector 44, 58, 68
  name-value pair 44
instantiation vector 18
Internet 256
invoking
  Event Perspective 140
  Hardware Perspective 138
  Perspectives 138

K
Kerberos principal 141, 156
keyNotNormal 167

L
leave, failure 14
LoadLeveler 64
lsemv 101
lssrc 185

M
Makefile 98
Management Information Base
  GET 174
  GET NEXT 174
  ibmProd 177
  ibmSP description 189
  ibmSP viewing 194
  ibmSP.MIB 180
  ibmSPConfig 177, 189
  ibmSPEMNodeDepVarsTable 193
  ibmSPEMNodeIndepVarsTable 193
  ibmSPEMVariable 177
  ibmSPEMVariables 192
  ibmSPEMVarValuesTable 193
  ibmSPErrlogVars 177, 191
  MIB-2 175
  NetView for AIX browser 196
  SET 174
  snmpget command 194
  snmpinfo command 194
  understanding 175
membership 14
MetaGroup 12
MIB 174
  See also Management Information Base
modifier 62
monemv 115
monitors
  resource monitors for response times 225
  response time 226
  user response time 225

N
name server 10
namespace 10
NetView for AIX
  control desk 199, 213
  customizing traps 186
  dynamic workspace 217
  MIB browser 196
  monitoring server key wwitch 207
  monitoring SP resources 205
  SNMP subagent (sp_configd) 173
  trap-setting 214
network
NFS daemons 14
NMG
  See Node Membership group
  See Topology Services
Node Membership group 4, 5
nodeEnvProblem 167
nodeNotReachable 167
nodePowerDown 167
nodePowerLED 167
nodeSerialLinkOpen 167
nomination messages 11
notification 58, 59

O

observation 30, 57
observation interval 43, 58

P

pageSpaceLow 167
PATH variable 138
pcount 165
peer count 56
peer daemon status 57
Perspectives 59
  error message help 158
  error stack 159
  event definition 139
  help 159
  invoking 138
  Launch Pad 138, 139
  monitoring a daemon 165
  monitoring CPU usage 160
  monitoring file system size 148
  monitoring memory usage 140
  monitoring the health of your system 167
  overview 137
  tasks performed by 137
  terminology 152
Perspectives GUI
  See Perspectives
Phoenix 3
pipe 201
pman 71
  See also Problem Management subsystem
pmanctrl 85, 86, 90, 91, 92
pmand 63, 72
  See also Problem Management daemon
pmandConfig 86, 91
pmandef 75, 76, 81, 82, 83, 90, 91
pmanquery 83, 86, 90, 91
pmanrmd 72
  See also Problem Management Resource Monitor daemon
pmanrmd.conf 78
pmanrmdConfig 77, 78, 92
pmanrmdloadSDR 80, 90, 92
pmanrminput 77, 82, 90, 92
pmHandle 152
predicate 17, 48, 51, 57, 58, 59, 61
  default 68
  modifier 62
PrevPIDCount 162
Problem Management 71
Problem Management daemon 77, 91
  Commands 91
Problem Management Resource Monitor daemon 77, 92
  Commands 92
Problem Management subsystem 71
  automating performance correcting tasks 225

Problem Management subsystem *(continued)*
  create your own monitor 72
  file changed monitor 94
  file system monitor 93
  pmanctrl 90
  pmandConfig 86, 91
  pmandef 75, 83, 90
  pmanquery 83, 86, 90
  pmanrmd.conf 78
  pmanrmdloadSDR 80, 90, 92
  pmanrminput 82, 90
  pmHandle 152
  Problem Management subsystem 154
  process monitor 94
  server monitor 94
  sysctl.pman.acl 83
  unsubscribe 87
process monitor 94
provider 14
proxy agent 174, 176
proxy client 67, 68, 69
proxy clients 19
PSSP Log 146, 154
PTX/6000 36
PTX/6000 Manager 36

Q

Quantity 18

R

realMemLow 167
realMemLowEvent 142
rearm expression 147, 151, 152, 154, 165
rearm predicate 152
rearm-command 147, 161
rearm-event 147
recovery 28
relational expression 61
Reliable Messaging 4
Reliable Messaging library 7
reporting interval 43, 58
resource 168
resource attributes 38
Resource Class 43
  rcClass 43, 44
  rcObservation_interval 43, 44
  rcReporting_interval 43, 44
  rcResource_monitor 43, 44
Resource Monitor 27, 28, 29
  aixos 66
  client 31
  client type 30, 63
  extensibility 67
  harmld 63
  harmpd 63
  hmrmd 63
  membership 65

Resource Monitor *(continued)*
  modes 30
  objectives 29
  pmand 63
  pull mode 30
  push mode 30
  response 65
  rmArguments 48
  rmConnect_type 48, 49
  rmMessage_file 48, 49
  rmMessage_set 48, 49
  rmName 48, 49
  rmPath 48, 49
  rmPTX_asnno 48
  rmPTX_description 48, 49
  rmPTX_prefix 48, 49
  rvPTX_asnno 49
  server 31
  server type 30, 63
Resource Monitor API
Resource Variable 30, 31, 67, 68, 144, 149, 150, 151, 160, 161, 162, 165, 166, 171
  attributes 38
  Counter 18, 31
  definition 48
  dynamically instantiated 165
  EM_Resource_Variable 33
  IBM.PSSP.aixos.CPU 73, 161
  IBM.PSSP.aixos.FS 151
  IBM.PSSP.pm 80
  IBM.PSSP.pm.User_state[1-16] 77
  IBM.PSSP.pmanrmd 80
  IBM.PSSP.Prog.pcount 162
  IBM.PSSP.Prog.xpcount 162
  IBM.PSSP.SP_HW.Node.keyModeSwitch 208
  Instance Vector 44
  messages 38, 67
  observation interval 43
  pcount 162, 166, 245
  predicates 61
  Quantity 18, 31, 67
  reporting interval 43
  rvClass 39, 41
  rvData_type 39, 41
  rvDescription 39, 40
  rvDynamic_instance 40, 41
  rvEvent_description 40
  rvIndex_vector 40
  rvInitial_value 39
  rvInitial_value. 41
  rvLocator 40
  rvName 39, 40
  rvPredicate 40
  rvPRX_max 41
  rvPTX_description 39, 41
  rvPTX_min 41
  rvPTX_name 39, 41
  rvValue_type 39, 40

Resource Variable *(continued)*
  SBS 62
  State 18, 31
  types 31
  user-configurable 72, 77
  vendor name 39
  xpcount 162, 166, 245, 246, 249
Resource Variable definition 48, 51
Resource Variable type
  float 33
  long 33
  Structured Byte String 33
resources
  See system resources
response time
  example of resource monitor to measure 226
  monitoring 225
  responding to performance problems 225
  setting goals 225
  ways to improve example 233
RFCs 253
RMAPI 22, 27, 67
  See also Resource Monitor API
rmapi_smp.cat 42
rmapi_smp.msg 42
routing maps 4
runcat 42

S

SBS
  See Structured Byte String
SDR 143, 146, 152, 156
SDR classes
  EM_Condition 46, 143, 169
  EM_Instance_Vector 46
  EM_Resource_Class 39, 46, 52, 161
  EM_Resource_Monitor 41, 52
  EM_Resource_Variable 43, 46, 52, 149
  EM_Structured_Byte_String 46
  pmandConfig 86, 91
SDR Syspar 51
SDR_test 156
SDRArchive 156
SDRCreateObjects 40, 43, 49, 52
SDRDeleteObjects 92, 156
SDRGetObjects 42, 45, 58, 73, 92, 143, 149, 150, 153, 156, 161
SDRListClasses 156
SDRListFiles 156
SDRListObjects 46
SDRRetrieveFile 156
SDRWhoHasLock 156
sendmail 165
server monitor 94
SEXECED process flag 163
shared memory 35, 36, 38
shared memory segment 7

signal handler for SIGPIPE 102
SIGPIPE handler 102
Simple Network Management Protocol
  agent 175
  client 175
  manager 175
  server 175
  snmp.log 203
  snmpget command 194
  snmpinfo command 194
singleton 5
SMUX
  See SNMP Multiplexor
SNMP 173, 175
  See also Simple Network Management Protocol
SNMP Agent (sp_configd)
  community name 179
  configure 184
  overview 173
  snmp_trap_gen 197
  sp_configd.log 201
  subagent 176
SNMP Multiplexor
  peers 179
  ports 179
  understanding 176
SNMP trap 72, 84, 146, 161, 174, 179
SNMP trap 10001 78
SNMP trap 10003 212
SNMP trap 1006 160
SNMP Traps 146
snmp.log 203
snmpget 194
snmpinfo 194
SP monitoring
  monitoring a daemon 165
  monitoring CPU usage 160
  monitoring file system size 148
  monitoring memory usage 140
  monitoring the health of your system 167
  monitoring user response time 225
  resource monitors for response times 225
SP Perspectives
  See Perspectives
SP Perspectives GUI
  See Perspectives
sp_configd 176, 197, 201
SP_ports 22
spevent 138
sphardware 138
SPMI 36
  See also System Performance Measurement Interface
spmon 218
staging area 51
startsrc 184
State 18
stopsrc 185
Structured Byte String
  byte string 34
  character string 34
  float 34
  long 34
  serial number 34
su 78
subscriber 14
Summary of RMAPI example program 132
switch_response 65
SwitchGroupie 15
switchNotReachable 167
switchResponds 167
sysctl.pman.acl 83
Syspar_ports 9, 22
Syspars pane 141
System Membership Groups 4
system partition 67
System Performance Measurement Interface 36, 39, 48
system resources 28, 29, 30

T
tmpFull 168
topology 3, 4
Topology Services 3
  /var/ha/soc/hats directory 7
  algorithm 6
  connection with Group Services 7
  crown prince 5
  daemon 7
  dependency 5
  domain 5
  downstream_neighbor 6
  group leader 5
  hatsd 7, 9
  heartbeats 3
  NMG 5
  Node Membership 4
  Node Membership Group 5
  Reliable Messaging 4
  routing maps 4
  shared memory segment 7
  singleton 5, 6
  System Membership Groups 4
  topology 3, 4
  Topology Manager 3
  UNIX Domain Sockets 7
  upstream_neighbor 6

U
UDS
  See Unix Domain Sockets
Unix Domain Sockets 7
  used by Topology Services 7

upstream_neighbor 6
user response time
  See response time
users
  setting performance goals for 225
  user response time 225
util.c 98
util.h 98

V
varFullEvent 146
variable value 168
voting protocol 15
VSD 137
VSM 137

X
xmpeek 37
xpcount 165

ITSO Redbook Evaluation

RS/6000 SP Monitoring: Keeping It Alive
SG24-4873-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redeval@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____________

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

____

____

____

____

What other redbooks would you like to see published?

____

____

____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

____

____

____

____

____

Printed in U.S.A.

SG24-4873-00