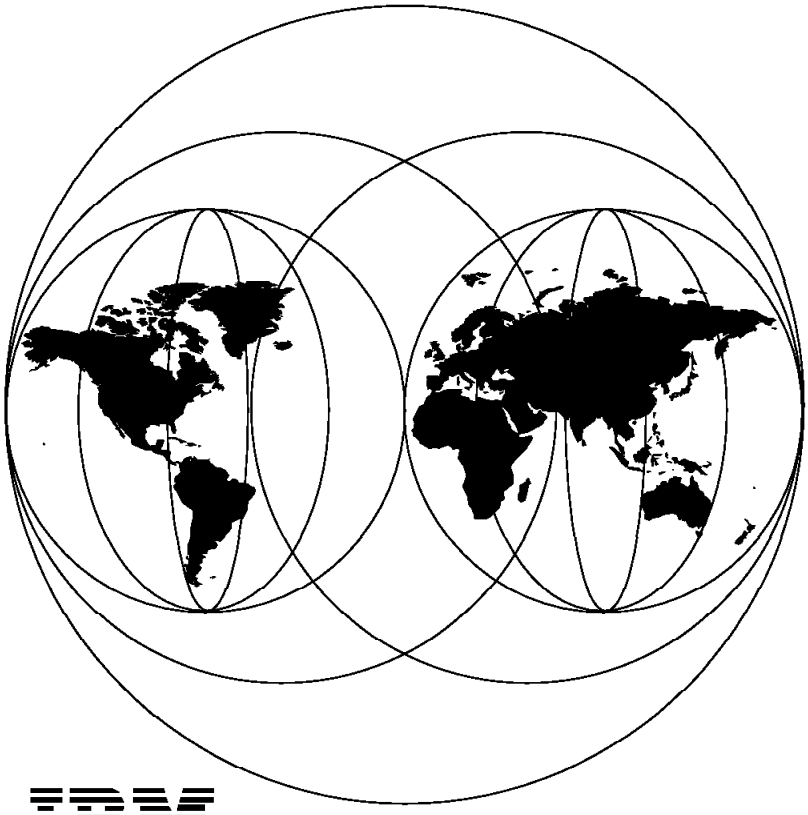
SG24-4838-00

RS/6000 SP High Availability Infrastructure

November 1996



IBM

International Technical Support Organization
Poughkeepsie Center

International Technical Support Organization

SG24-4838-00

RS/6000 SP High Availability Infrastructure

November 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special Notices” on page 313.

First Edition (November 1996)

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

Tables

Preface

This redbook provides detailed coverage of the High Availability Infrastructure functions that were made available with the newest release of PSSP 2.2 software. The book is organized in the form of a technical presentation. It includes mid-size foils and related text for each foil in the document.

This redbook was written for IBM customers, Business Partners and IBM technical and marketing professionals to provide them with a detailed presentation of the different functions and components that are meant to be the foundation of the whole strategy of highly available clusters across a multivendor environment.

The book focuses on the following topics:

- PSSP 2.2 High Availability Overview
- Topology Services
- Group Services
- Resource Monitor
- Event Management
- Problem Management
- Virtual Shared Disk and Recoverable Virtual Shared Disk
- Performance Toolbox Parallel Extension

A good knowledge of AIX Version 4 and RS/6000 SP is assumed.

How This Redbook Is Organized

This redbook contains 326 pages. It is organized as follows:

- Chapter 1, “Introduction to High Availability Infrastructure”

  This provides an overview of the High Availability Infrastructure of PSSP Version 2 Release 2. It uses a pictorial diagram to illustrate how the various components of the infrastructure fit together and the inter-relationships that exist within the structure.

- Chapter 2, “Topology Services”

  This focuses on the new PSSP Version 2 Release 2 topology services which are the foundation of the high availability infrastructure. It provides details on the currently supported adapter networks, namely Ethernet and switch (CSS) networks.

- Chapter 3, “Group Services”

  This chapter discusses the rudimentary concepts of group services, which is one of the critical components of the high availability infrastructure supported by PSSP Version 2 Release 2. It introduces the definition of groups and the various protocols associated with the group services component.

- Chapter 4, “Resource Monitors”

  This chapter describes the Resource Monitor, which is one of the new high availability services of PSSP Version 2 Release 2 used to monitor the software and hardware components of a system. The chapter provides rudimentary definitions and basic presentations of Resource Monitors and how they relate to Event Management and Performance Toolbox Parallel Extension.

- Chapter 5, “Event Management”

  This chapter describes Event Management, which is one of the new high availability services of PSSP Version 2 Release 2. It also provides an introductory presentation on the Resource Monitor and its relationship with other components. The presentation describes the concepts and architectural design and operational strategy.

- Chapter 6, “Problem Management”

  This chapter describes Problem Management, which is one of the new high availability services of PSSP Version 2 Release 2. It describes the concepts and how to exploit this function in the RS/6000 SP system environment. It provides some examples of the commands and how they are used when events are triggered in the system.

- Chapter 7, “VSD/RVSD”

  This chapter describes the latest release of Virtual Shared Disk (VSD), which is one of the clients that exploits the new high availability services of PSSP Version 2 Release 2.

- Chapter 8, “Performance Toolbox Parallel Extensions”

  This chapter describes the Performance Toolbox Parallel Extension (PTPE). It is one of the features of PSSP Version 2 Release 2. This chapter contains a detailed presentation of this tool and a description of how it is used in a scalable environment for performance measurement of the various system components.

- Appendix A, “Resource Class Definition”

  This appendix contains a list of the Event Management Resource Classes that are defined in the SDR as EM_Resouce_Class.

- Appendix B, “ibmSP MIB”

  This appendix contains specific object types that are supplied for the ibmSP MIB.

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.

Endy Chiakpo is a Project Leader at the International Technical Support Organization, Pougkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of RS/6000 SP. He holds a B.S. degree in Physics and a Master of Science degree in Electrical Engineering from Syracuse University New York. Before joining the ITSO, Endy worked in the IBM Poughkeepsie Lab in New York, USA.

Peter Kes is a Project Leader at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of RS/6000 SP. Before joining the ITSO in 1996, Peter worked in AIX Systems in The Netherlands, as a System Engineer.

Adrian Demeter is an RS/6000 SP specialist at the IT Availability Services department of IBM Czech, in Prague (Czech Republic). He is responsible for RS/6000 SP and SMP support in the country, and the first RS/6000 SP and SMP installations in the country were done by him. He has prepared and taught other SP and non-SP education courses for customers in the Czech environment. He has worked at IBM for 3 years. He has 4 years of experience in UNIX and system engineering. He holds a degree in Electrical Engineering and Technical Cybernetics from Czech Technical University (CVUT) Prague.

Robert Gambs is a Technical Marketing Support Specialist at the AIX Systems Center in Westlake, Texas (USA). He is responsible for education, planning, installation, customer support, and various other aspects of RS/6000 SP, general RS/6000, and AIX technology. He co-authored the RS/6000 SP Implementation Workshop, available from IBM Education and Teaching. Robert holds a bachelor degree in Computer Science with emphasis in Physics from Texas A&M at Commerce, formerly known as East Texas State University. He has been with IBM for over four years.

Franz Gerharter-Lueckl is an AIX specialist at IBM Austria. He holds an engineering degree for a five years study at the TGM Technical School for communications engineering and electronics. Before joining IBM in 1989, Franz worked as project leader and programmer for a software house. At IBM, he was on assignment at the T.J. Watson Research Center for 20 months. During that time, he was also responsible for the initial releases of LoadLeveler for Sun Solaris (5765-227) and SGI Irix 5 (5765-228). His areas of expertise include RS/6000 SP, benchmarks, networking, porting software to HP, SUN, SGI, and AIX ESA. He is a co-author of two redbooks: *PSSP Version 2 Technical Presentation*, SG24-4542; and *Scientific and Technical Computing Overview*, SG24-4541. Franz teaches “TCP/IP in a Multivendor Environment” classes for IBM Education.

Yoram Gnat is a senior RS/6000 market specialist in Israel. He has 15 years of experience in UNIX operating systems. He has worked at IBM for 11 years, following AIX from Version 2, through Version 3, and up to Version 4. He holds a Ph.D. degree in Physics from Tel Aviv University, participated in experimental research projects in high energy physics and published several research studies. His areas of expertise include AIX, scientific/engineering programming and optimizations, parallel programming, and TCP/IP communications. He now also teaches the C programming language and the AIX Operating System.

Yann Guerin is a team leader at the EMEA Parallel Solutions Support Center (PSSC), in Montpellier (France). He is responsible for RS/6000 SP education courses and for the coordination of the system administrators team on the other RS/6000 SP activities of the PSSC (benchmarking, briefing, demonstrations, customized solutions). He has 12 years of experience in UNIX and project management.

Hisashi Shirai is an advisory IT specialist at IBM Japan Systems Engineering Co., Ltd. in Makuhari (Japan). He is mainly responsible for high availability (HA) of RS/6000 systems, including RS/6000 SP. He has 4 years of experience in AIX, and has worked as a team leader of an HA technical support group for two

years. He is also a co-author of the redbook *Implementing High Availability on RS/6000 SP*, SG24-4742.

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

Marcelo Barrios
International Technical Support Organization, Poughkeepsie Center

IBM PPS Lab Poughkeepsie:
Mike Browne
Deepak Advani
Joseph Banas
Dr. Gili Mendel
Dr. Aruna Ramanan
Michael Schmidt
Ken Briskey
Stephen Tovcimak
James Gilman
Dennis Jurgensen
Tim Race
Robert Gensler Jr.
Peter Badovinatz
Paul Bildzok
Skip Russell
Richard Ferri
Mark Gurevich
Steven Cangemi
William Wajda
Margaret Moran
Bernard King-Smith
Patrick Meehan
John Simpson

IBM US Dallas System Center:
Mark Venator

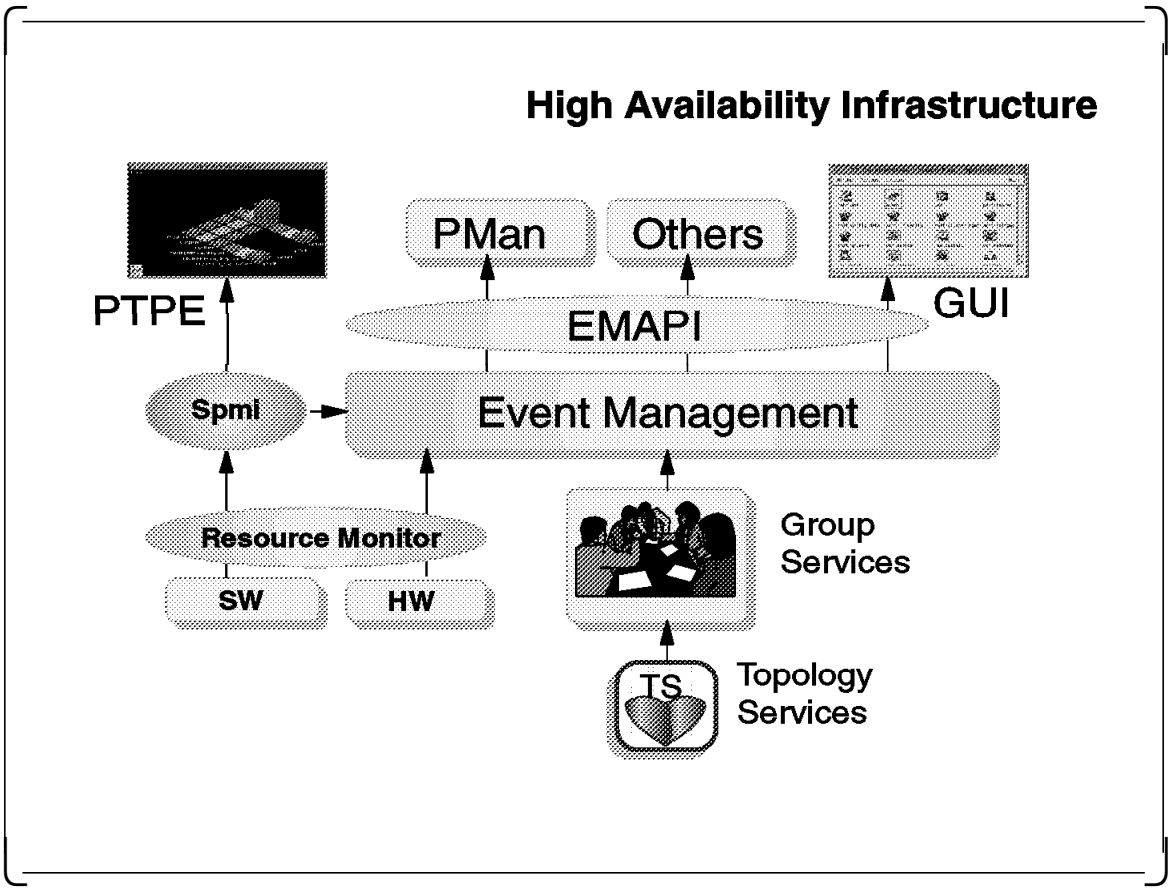
Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Introduction to High Availability Infrastructure



This chapter provides an overview of the IBM High Availability Infrastructure components and the interrelationships that exist within their internal structure. This infrastructure is made up of three key distributed subsystem components, namely:

- Topology Services
- Group Services
- Event Management Services

1.1 Topology Services

Topology Services is the foundation of the entire infrastructure and it coordinates adapter membership and node membership information in the system. This information is in turn provided to the other subsystem components such as Group Services.

1.1.1 Adapter Membership

The two types of adapter networks that are currently supported in this release of PSSP 2.2 are:

- Ethernet Membership
- Switch Membership (CSS)

enMembership The Ethernet membership is considered ″UP″ if the node is accessible through its Ethernet adapter.

cssMembership The high-speed switch membership is considered to be ″UP″ if the node is accessible through its switch adapter.

1.1.2 Host Membership

The host membership provides a view of the state of the processors (nodes) in the system. A node is considered ″UP″ if there are at least one or more communication paths to the node. A node is considered ″DOWN″ if there are no communication paths to it. Refer to Chapter 2, “Topology Services” on page 5 for more details.

1.2 Group Services

The Group Services component of the high availability infrastructure provides a distributed coordination, messaging, and synchronization service. It supports subsystems for their coordination of recovery actions but it does not actually perform the recovery actions. In this release of PSSP, Group Services is exploited by two other components:

- Recoverable Virtual Shared Disk (RVSD)
- Event Management

A group is defined as a named collection of processes and the name Group Services could be interpreted as process services. This could be a collection of processes that are executing on different nodes. Any process may create a new group, or join an existing group. To create or join an existing group requires a root access authority.

The high availability services is an open infrastructure that can be exploited by most users. Group Services provides a set of Application Programming Interfaces (API) that will enable Independent Software Vendors (ISVs) or other end users to make use of this infrastructure. Refer to Chapter 3, “Group Services” on page 25 for more details.

1.3 Event Management

The comprehensive Event Management subsystem monitors system resources and generates events when a resource changes state. The state of change in a resource is detected by the application of a predicate. A predicate is akin to a C program expression (such as X==0). When the predicate is true, an event ID is generated. There are three main components associated with Event Management, namely:

- Resource Monitor
- Clients
- Event Management Daemon (EM daemon)

The Resource Monitor supplies resource variables to the Event Management Daemon through the Resource Monitor Application Programming Interface (RMAPI). The Resource Monitor does the actual monitoring of system resources.

The clients are programs that define and receive events or query the current state of resources. This is done through the use of the Event Manager

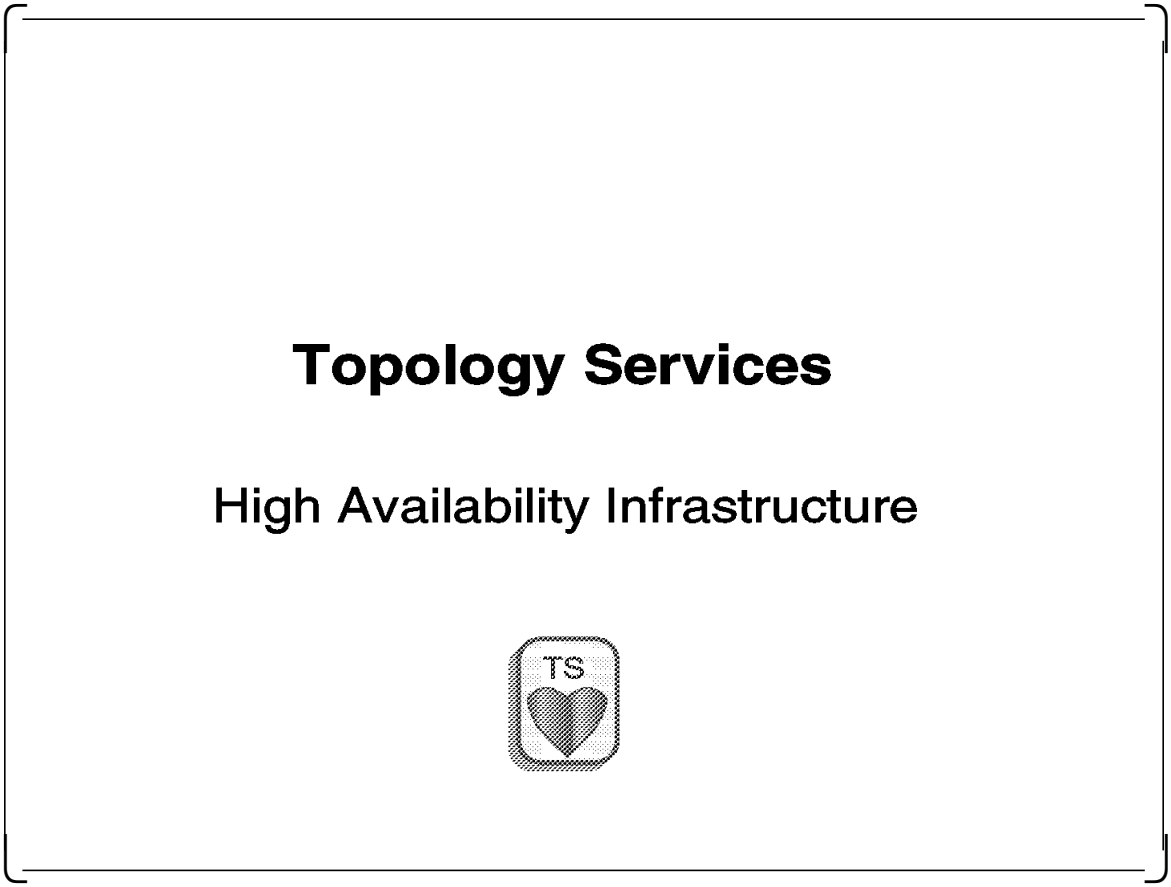
Application Interface (EMAPI). The clients are usually local to the EM Daemon with some exceptions. One good example of a client to Event Management is the Problem Management described in Chapter 6, “Problem Management” on page 189.

The Event Management Daemon provides the following major functions:

- Generates events
- Answers queries
- Forwards requests and replies among nodes within an operational domain

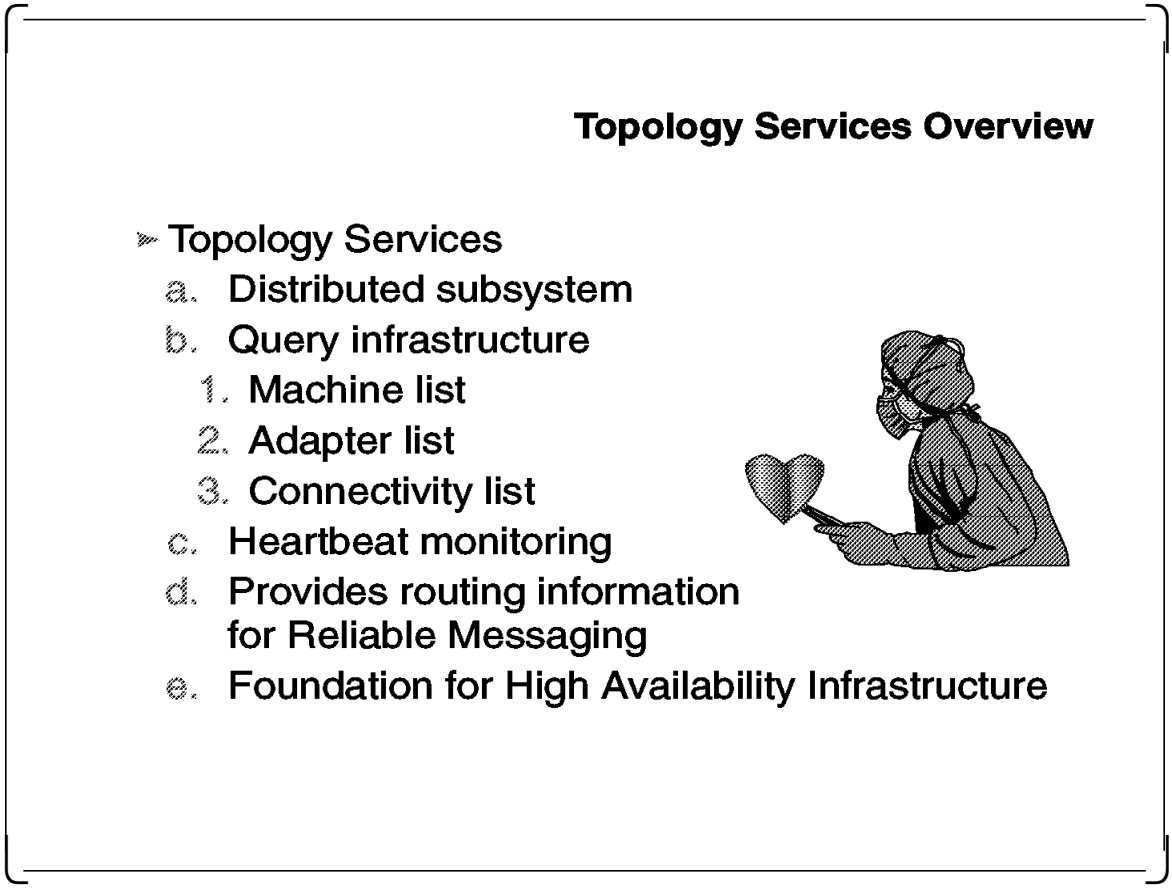
Refer to Chapter 5, “Event Management” on page 135 for more details.

Chapter 2. Topology Services



This chapter describes the Topology Services subsystem of the new PSSP 2.2 High Availability Infrastructure.

2.1 Topology Services Overview



The Topology Services subsystem provides the foundation for the PSSP 2.2 High Availability Infrastructure. This subsystem is distributed across all nodes in an environment and maintains availability information about the nodes and adapters. This availability information allows Topology Services to provide routing paths to the Reliable Messaging subsystem. Group Services subscribes to Topology Services for changes in the availability status of nodes and adapters.

Topology Services has a dependency on the Control Workstation for SDR information about the environment. It currently monitors the administrative Ethernet network and a switch network (if present); other network adapters will be supported in the future.

The Topology Services daemon is `hatsd` and is administered by the `hatsctrl` program. The log file is located in:

/var/ha/log/hats.*<DayofMonth>.<hhmmss>.<partition #>* .

2.1.1 hatsctrl

The `hatsctrl` program contains the following parameters:

-a Configures Topology Services for a node by adding an entry to the SRC subsystem, updating the SDR, and modifying */etc/services*.

-d Removes Topology Services from SRC, SDR, and /etc/services for a node.

-c Removes Topology Services configuration for all nodes.

-s Starts hatsd.

-k Stops hatsd.

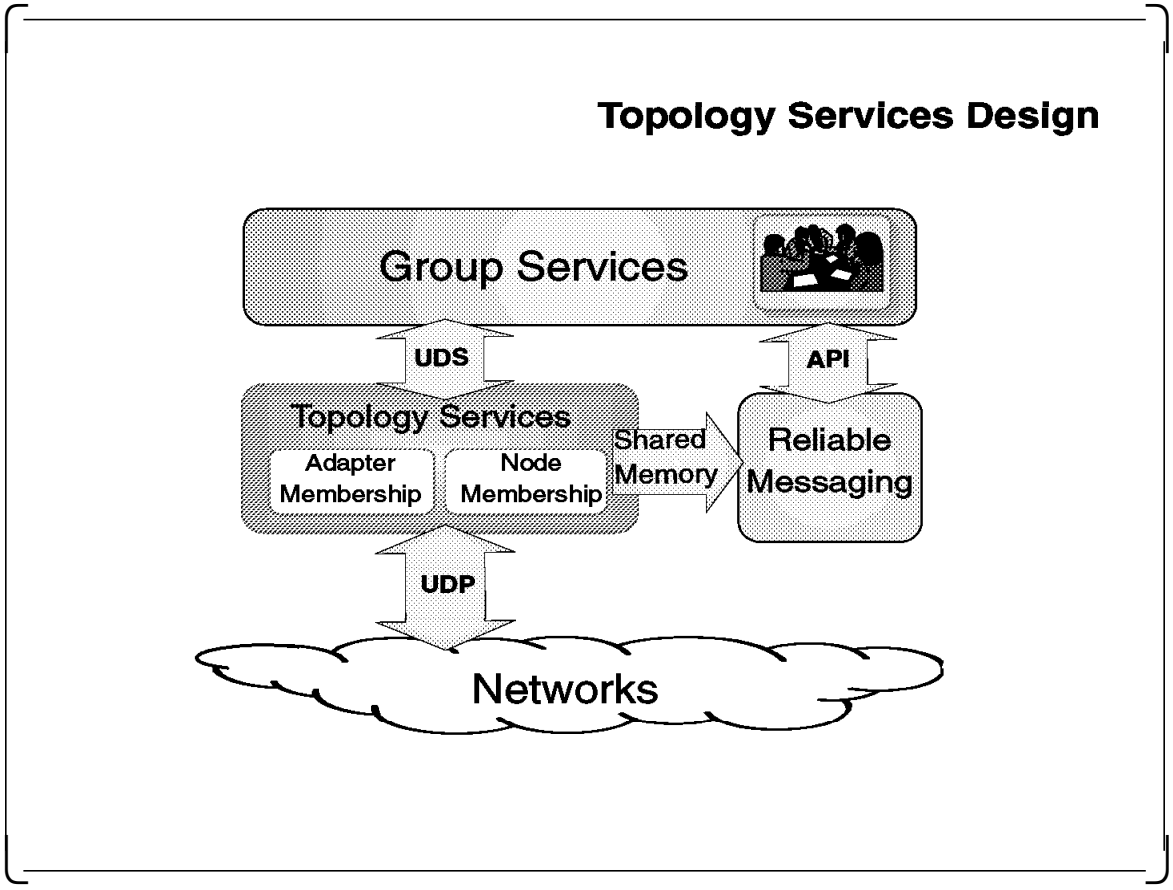
-t Turns ON tracing for hatsd.

-o Turns OFF tracing for hatsd.

-r Rebuilds the configuration and refreshes the Topology Services daemon

-h Displays the usage information.

2.2 Topology Services Design



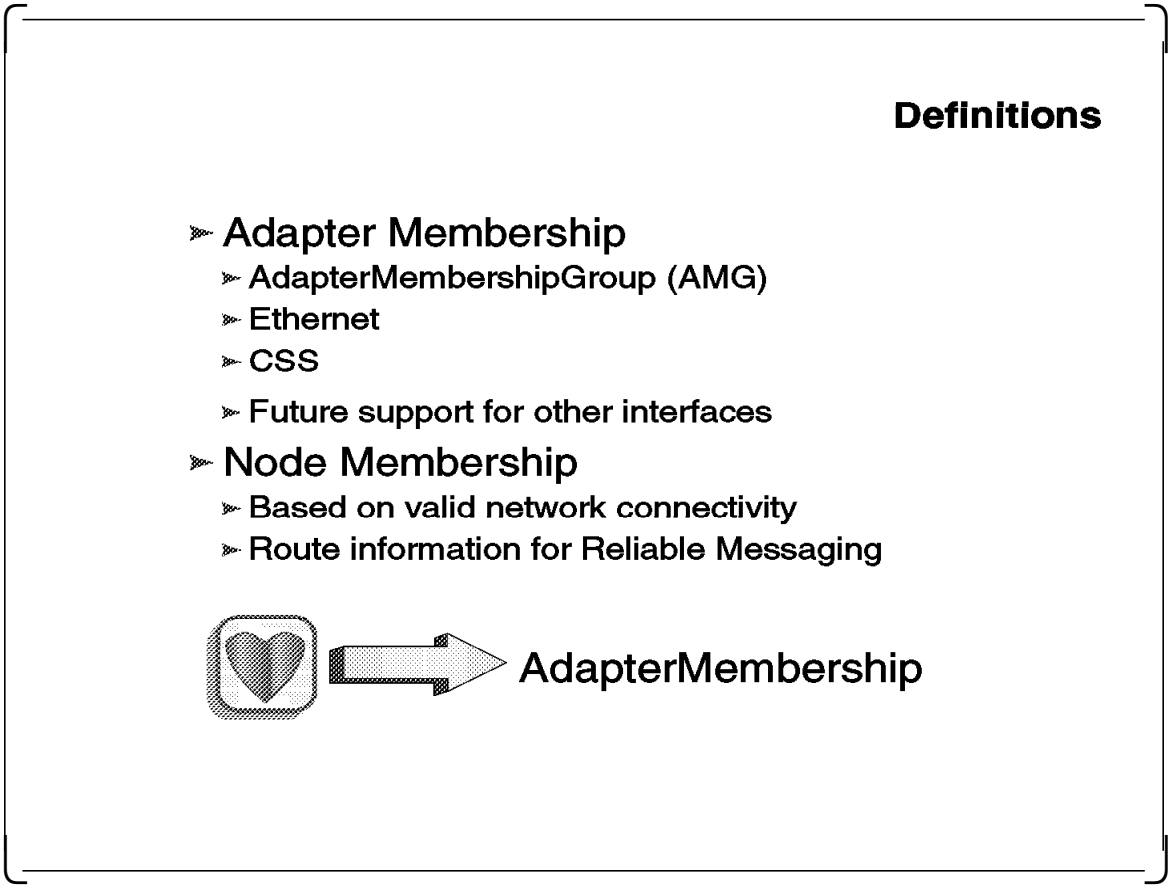
Topology Services provides availability details to Group Services and routing information to Reliable Messaging. The Topology Services daemons communicate between nodes to monitor availability and maintain topology descriptions.

Topology Services' interaction with Group Services is accomplished with UNIX Domain Stream Sockets, so that Group Services may detect when a Topology Service has died prematurely. Group Services subscribes to Topology Services for status changes in the availability of nodes and adapters in the environment.

From the availability information, Topology Services can build Network Connectivity Tables to provide routing information to Reliable Messaging. These tables are maintained in shared memory so that Reliable Messaging may query them.

The communication between Topology Services daemons is accomplished by using unreliable data packets. Therefore, these daemons must always validate the source of these packets and acknowledge receipt to provide a reliable form of communication.

2.3 Definitions



The following are several terms that will be used to describe the different concepts of Topology Services.

Adapter Membership describes the process of monitoring the availability of the different network adapters defined in the environment. Adapter Membership Group is an association of adapters that establish their relationship for monitoring and routing purposes.

Currently, Topology Services only supports Ethernet and switch adapters, but it will eventually support all types of network adapters.

Node Membership is the process for maintaining node availability information based on Adapter Membership. The routes and monitoring provided by Adapter Membership define which nodes can communicate across their different adapters. If adapters on different nodes are in an Adapter Membership Group, then the nodes are able to communicate. Nodes may also communicate indirectly by routing across several different Adapter Membership Groups. This will be discussed in later sections.

Definitions (cont.)

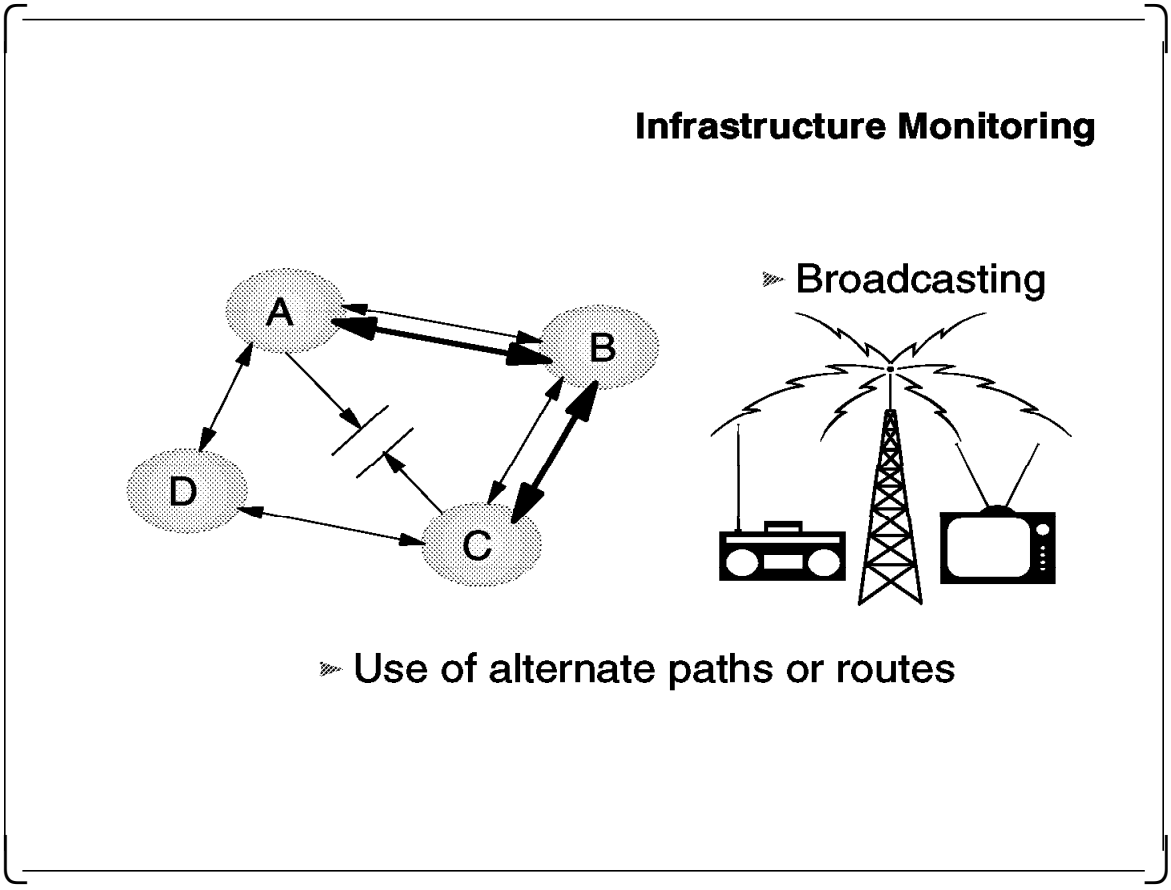
- Group Leader (GL)
  - Manager of a group
  - Processes node connectivity
  - Updates members on connectivity
  - Node with highest IP address
- Crown Prince (CP)
  - Will rule after GL dies
  - Node with second-highest IP address
- Singleton
  - Adapter Membership group with one member

The *Group Leader* is the adapter of the highest IP address within an Adapter Membership Group. The Group Leader is responsible for maintaining the topology and connectivity information of a group and distributing that information to the other members. The Group Leader will also attempt to recruit other adapters into the group, or to merge with groups which have a Group Leader with a higher IP address.

The *Crown Prince* monitors the availability of the Group Leader and assumes the leadership role when the Group Leader fails. The Crown Prince has the second-highest IP address in the group.

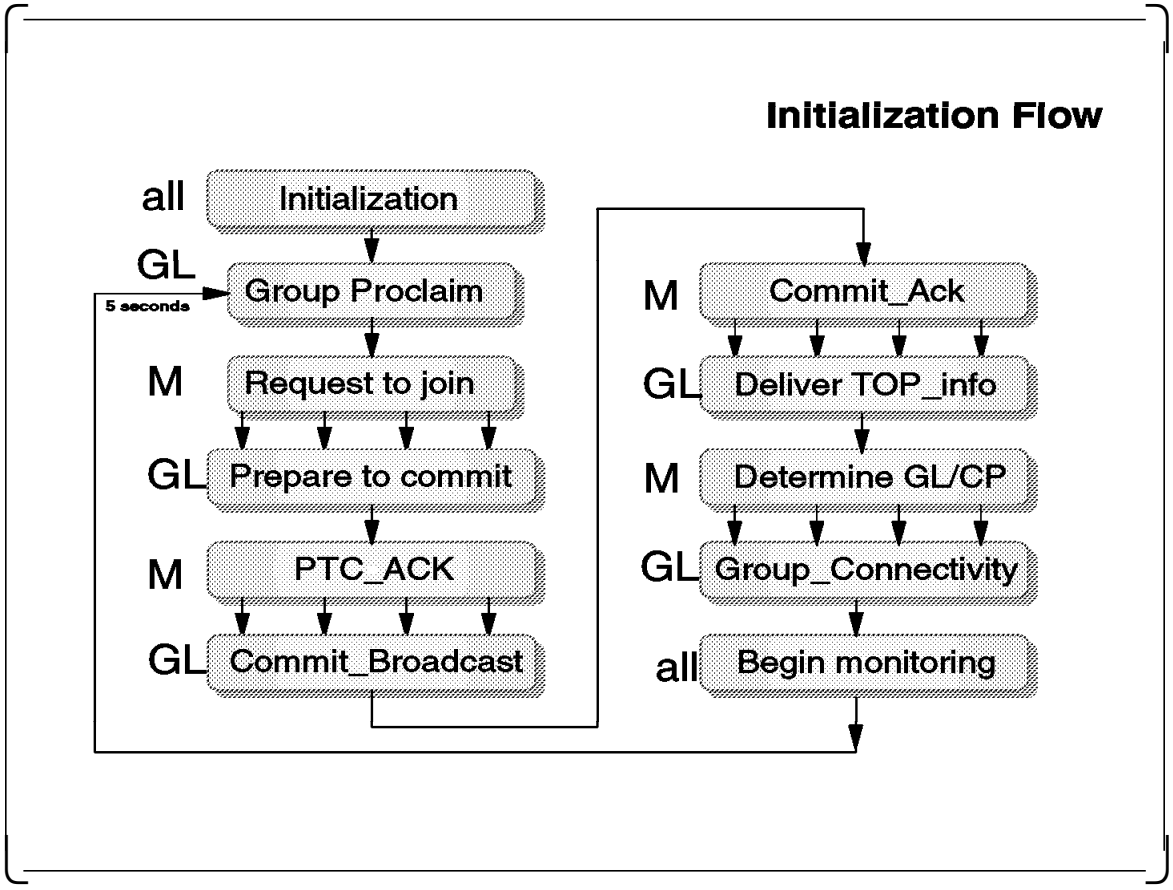
A *Singleton* group is an Adapter Membership Group which contains only one member. All adapters initialize into a Singleton before they join a larger group.

2.4 Monitoring and Routing



Topology Services' two main goals are to monitor node and adapter availability and to provide routing information to the Reliable Messaging subsystem. These functions work together with availability monitoring to perform validation on routes between nodes and to update that information in the subsystem's topology. Thus, Reliable Messaging will always have usable routes based on the latest attempt to send heartbeats over the different segments of that route.

2.5 Initialization Flow



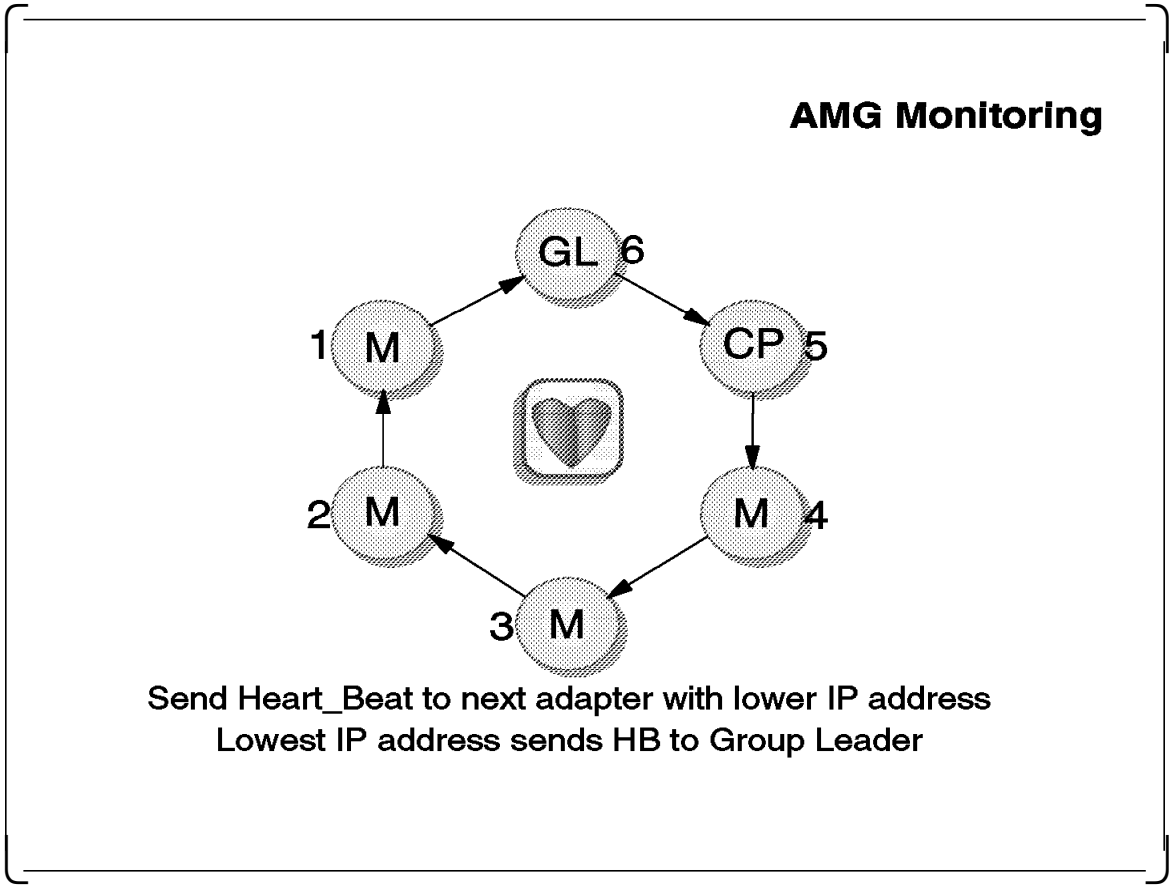
Adapter Membership Groups (AMG) are created and maintained using the following algorithm:

1. Each node acquires the machine list from the Control Workstation.
2. Each adapter initializes itself into a Singleton group.
3. Group Leaders periodically send proclamations to all lower priority adapters.
4. Lower priority Group Leaders respond to proclamations made by the higher priority Group Leaders with a request to join that group.
5. The Group Leader recognizes requests to join made by lower priority Group Leaders, and incorporates their group topology.
6. The Group Leader notifies all members of the newly forming group to Prepare to Commit the new group topology.
7. Group members acknowledge receipt of the Prepare to Commit message.
8. The Group Leader waits for all acknowledgements, then sends a message to commit the new topology.
9. Members acknowledge receipt of the commit message.
10. The Group Leader then delivers the new topology to all members.
11. Members will then determine the new Group Leader, Crown Prince, and Neighbors.
12. All members begin sending and receiving heartbeats from neighbors.

Group Leaders will continue to periodically send Group Proclaims to any lower priority adapters that are in their configuration but are not members of their group.

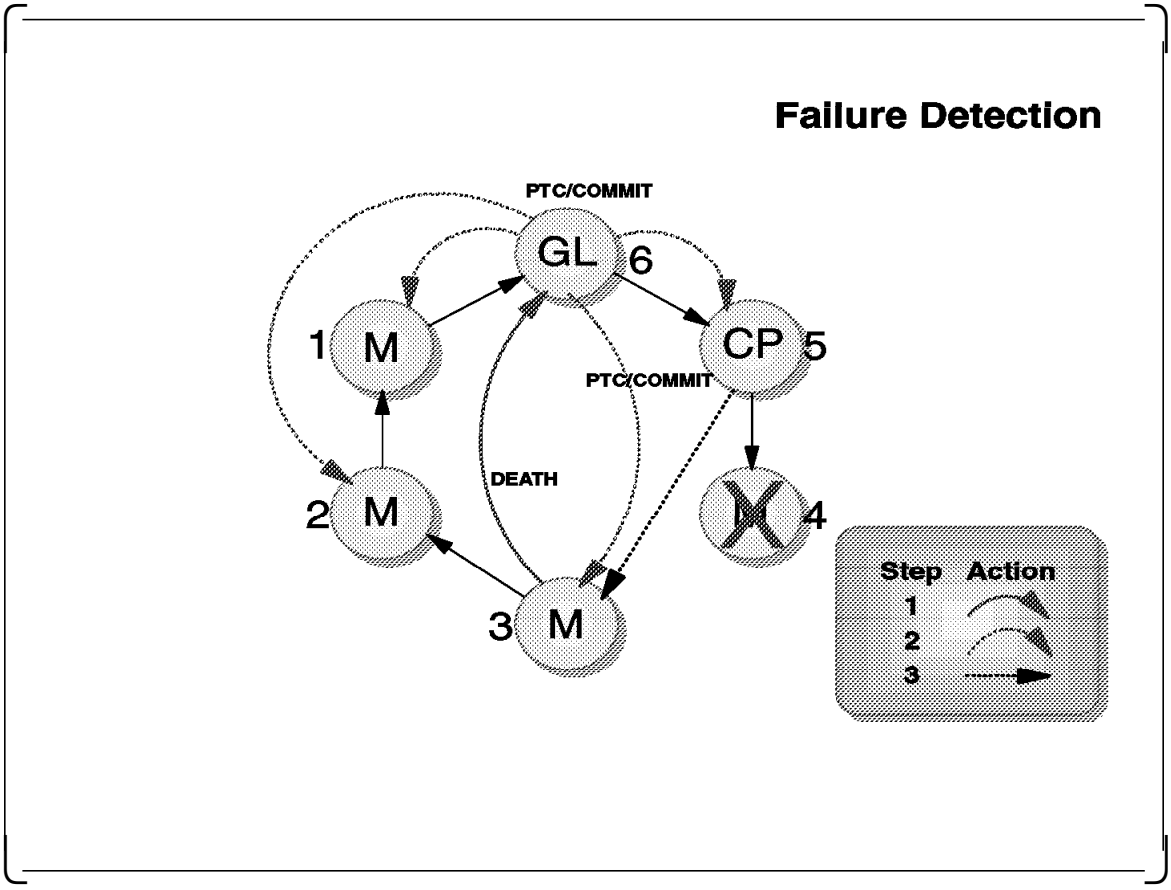
The monitoring process is only interrupted when modifications to the group topology are being made, as in the case of an adapter failure.

2.6 AMG Monitoring



Heartbeats are sent from one adapter to the adapter in the group with the next lower IP address. The adapter with the lowest priority will send its heartbeat to the Group Leader. All adapters will monitor the neighbor immediately above them and report any failures to the Group Leader.

2.7 Failure Detection

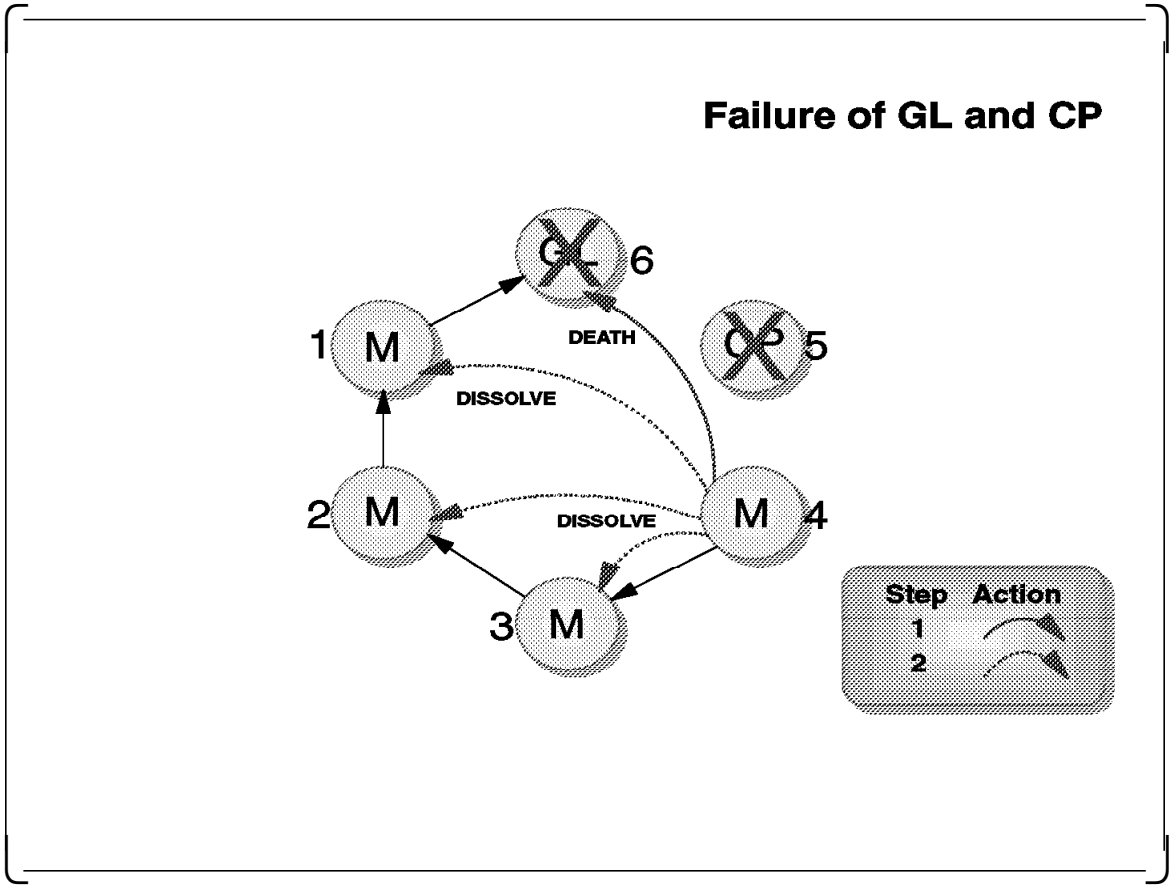


This diagram is an example of how the failure of an adapter or node is managed within an Adapter Membership Group.

1. Due to either adapter or node failure, Adapter 4 fails to send a HEART_BEAT message to Adapter 3.
2. Adapter 3 waits until it has missed four HEART_BEAT messages from Adapter 4 and then sends a DEATH_IN_FAMILY message to the Group Leader to indicate the death of Adapter 4. This is step 1 in the foil.
3. Adapter 3 knows that the Group Leader should be sending a Prepare to Commit (PTC) message shortly to update the group, but if Adapter 3 does not see that message after a time, it will send a DISSOLVE_GROUP message to all members so they will reinitialize themselves and create a new AMG.
4. The Group Leader will send a PTC to all members once it has been notified of the failure of Adapter 4. This is step 2 in the foil.
5. Once all PTC_ACK messages have been received, the Group Leader will send a COMMIT and the updated topology information to all members.
6. Each member will update their topology information and determine the Group Leader, Crown Prince, and Neighbor adapters. This is step 3 in the foil.

If the Group Leader is the adapter which failed, the Crown Prince will assume the role of Group Leader and notify the other members of the new topology.

2.8 Failure of Group Leader and Crown Prince

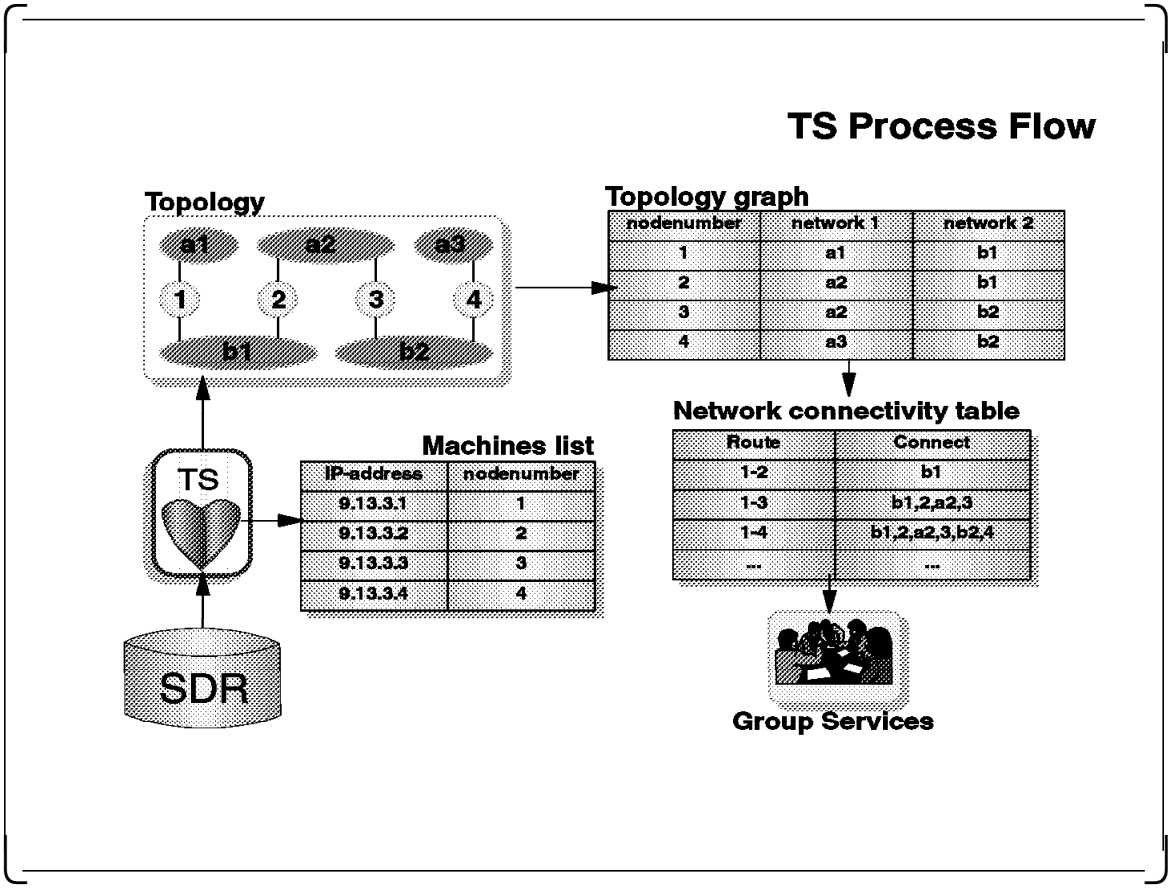


The failure of both the Group Leader and the Crown Prince is a rare event, but it is possible. It would be handled in the following manner:

1. Since the Crown Prince has died, it will not detect the death of the Group Leader.
2. Adapter 4 will detect the death of the Crown Prince after eight seconds and attempt to notify the Group Leader. This is step 1 in the foil.
3. Adapter 4 expects the Group Leader to respond with a PTC to all members. When no PTC arrives after approximately five seconds, Adapter 4 realizes something is critically wrong and notifies the other members in the group to DISSOLVE. This is step 2 in the foil.
4. Each member verifies that the DISSOLVE message came from a valid group member and then proceeds to reinitialize and form a new group which will not include the old Group Leader and Crown Prince.

As is described here, the last solution an AMG can take when the Group Leader fails to respond to changes in topology is to dissolve the group and reinitialize all adapters. Once reinitialized, all adapters will form a new group.

2.9 Topology Services Data Flow

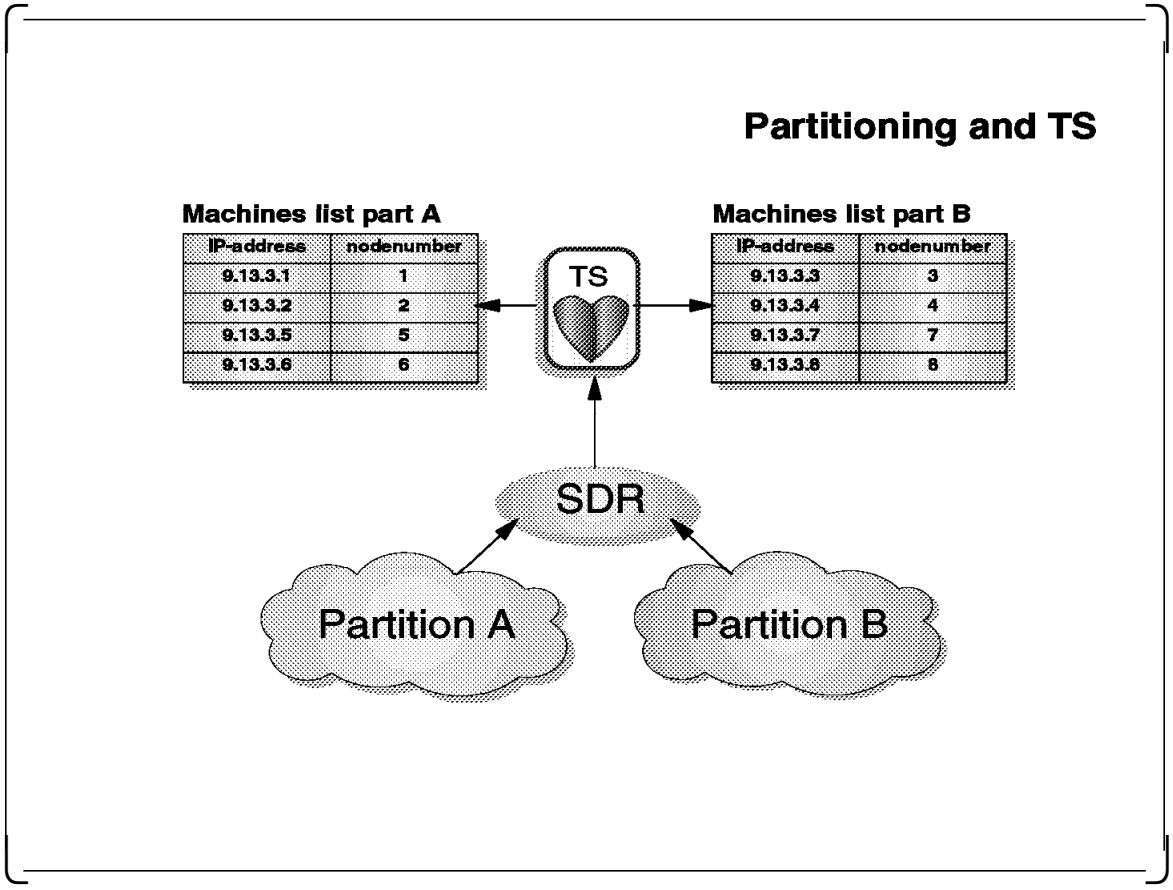


Topology Services utilizes and creates several levels of information concerning the adapters and nodes in the environment. Starting with the SDR, each piece is used to create the next piece, until ultimately a complete Network Connectivity Table is formed for use by Reliable Messaging. The SDR provides node number and adapter information about the RS/6000 SP environment, which the Topology Services builds into a machine list. The machine list is specific to a partition and always contains the Control Workstation. The SDR is the reason Topology Services has a dependency on each node being able to communicate with the CWS.

Topology Services uses the machine list to build Adapter Membership Groups on the various networks. With the information from the Adapter Membership Groups, Topology Services can build a topology table and graph to indicate node connectivity in the environment. Group Services is mainly interested in the changes to nodes and adapters as they are added and removed from the topology data.

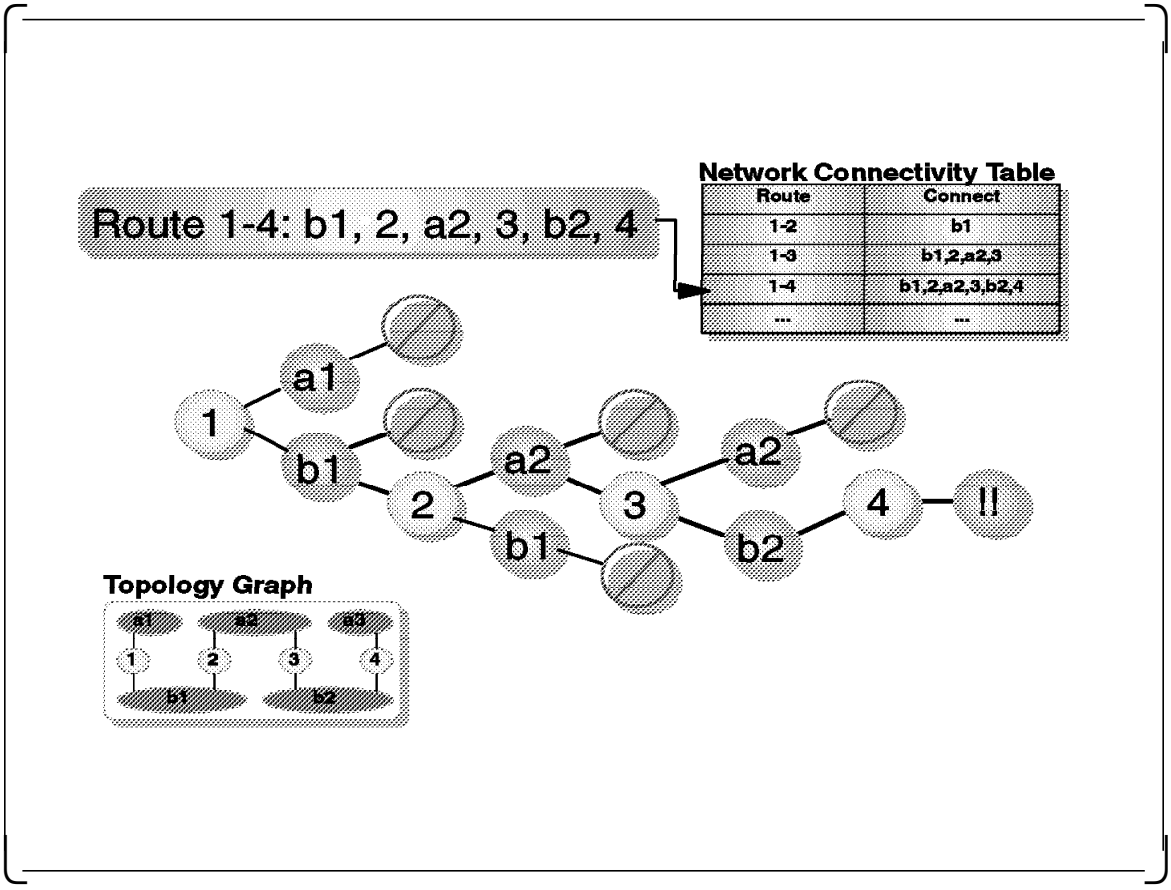
From the node connectivity, the Network Connectivity Table is developed for use by Reliable Messaging.

2.10 Partitioning and Topology Services



Separate machine lists will be built for each partition in the RS/6000 SP environment. The Control Workstation will be included in both machine lists because each will need to access the SDR. Additionally, the Control Workstation will run one instance of the Topology Services daemon for each partition so that it may separate its functions in each partition.

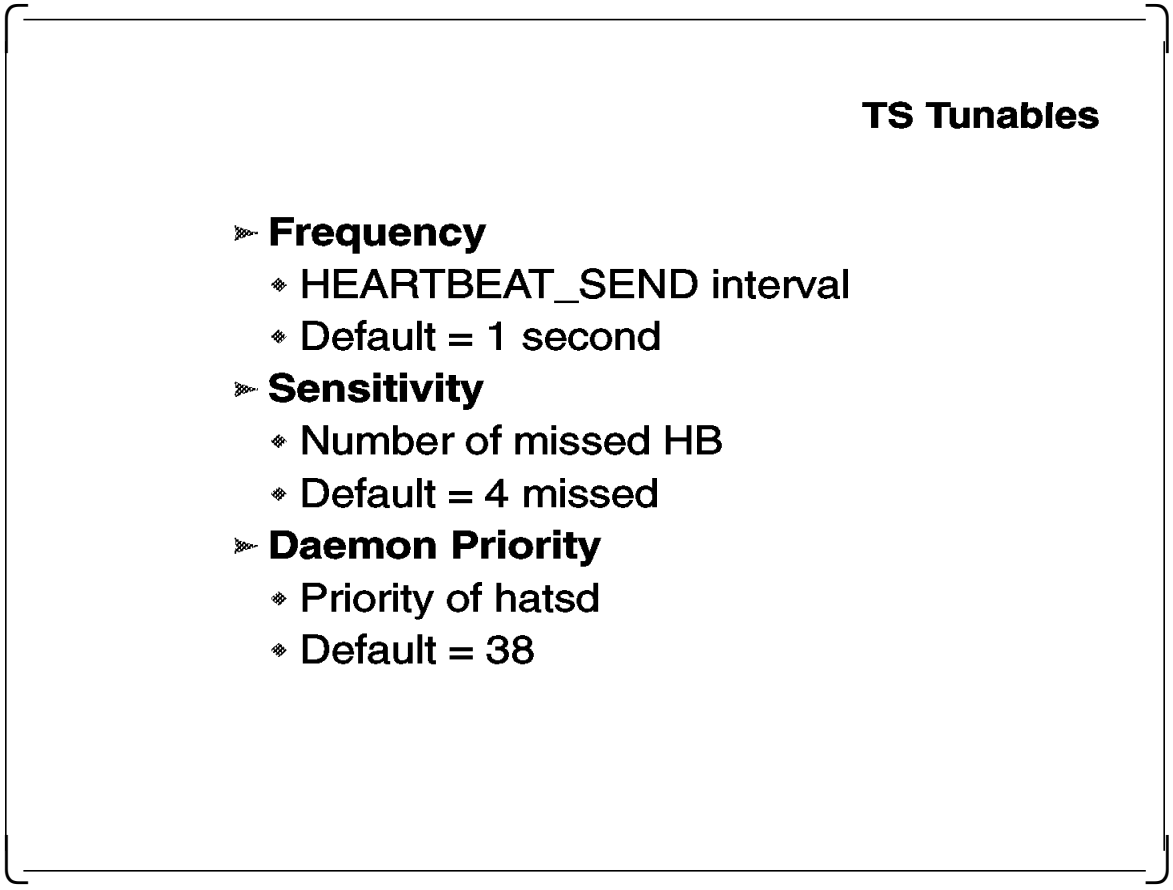
2.11 Network Connectivity



The Network Connectivity Table is derived from a breadth-first search performed on the Topology Graph to establish the shortest path between any two nodes. The diagram illustrates this algorithm for the node pair 1-4. At each node, the local AMGs are checked to see if the destination node is a member. If it is not, the search proceeds to another node in the AMG and checks there for the destination node in those AMGs. This process is repeated until a route is found from the source node to the destination.

The Network Connectivity Table is maintained by Topology Services in a section of shared memory so that Reliable Messaging is able to query it when necessary.

2.12 Topology Services Tunables



Topology Services has three parameters which are tunable, but one of these parameters affects several other parameters that are used in monitoring. Each of these tunable values are maintained in the TS_Config object within the SDR. The Topology Services daemon must be refreshed in order for any changes made to these values to take effect.

Frequency sets the rate at which heartbeats are sent. The heartbeat send rate is in seconds and affects the majority of intervals used in Topology Services.

Sensitivity indicates the number of heartbeats that must be missed for an adapter to be declared unavailable.

Run_FixPri and *FixPri_Value* set the priority value of the Topology Services daemon, `hatsd`. Run_FixPri indicates whether to use the value assigned to FixPri_Value or the default priority.

Following is a list of the various intervals and thresholds used in the Topology Services subsystem.

HEARTBEAT_SEND

This is the interval at which heartbeats are sent. It is modifiable by the frequency value in TS_Config. The default is one second.

HEARTBEAT_RECV

This is the interval at which heartbeats are declared missing. The default is 2 * HEARTBEAT_SEND.

STABLE_WAIT

This is the interval during which a group must have no membership changes before the group becomes stable. The default is 10 * HEARTBEAT_SEND.

JOIN

This is the interval to wait for a PTC after a JOIN_REQUEST. The default is 5 * HEARTBEAT_SEND.

JOIN_BATCH

This is the interval a Group Leader waits for additional JOIN_REQUESTs, after receiving the first, before processing the requests. The default is 2 * HEARTBEAT_SEND.

PTC_ACK

This is the interval a Group Leader will wait for an acknowledgement to a PTC. The default is 2 * HEARTBEAT_SEND.

COMMIT_ACK

This is the interval to wait for an acknowledgement to a COMMIT. The default is 2 * HEARTBEAT_SEND.

COMMIT_BROADCAST_ACK

This is the interval a Group Leader will wait for an acknowledgement to a COMMIT_BROADCAST. The default is 2 * HEARTBEAT_SEND.

COMMIT

This is the interval to wait for a COMMIT following a PTC. The default is 5 * HEARTBEAT_SEND.

PROCLAIM

This is the interval between GROUP_PROCLAIM messages. The default is 5 * HEARTBEAT_SEND.

NODE_CONNECTIVITY

This is the interval between NODE_CONNECTIVITY messages. The default is 10 * HEARTBEAT_SEND.

GROUP_CONNECTIVITY

This is the interval between GROUP_CONNECTIVITY messages. The default is 10 * HEARTBEAT_SEND.

NODE_REACHABILITY

This is the interval to wait after receiving a GROUP_CONNECTIVITY before determining node reachability. The default is 1 * HEARTBEAT_SEND.

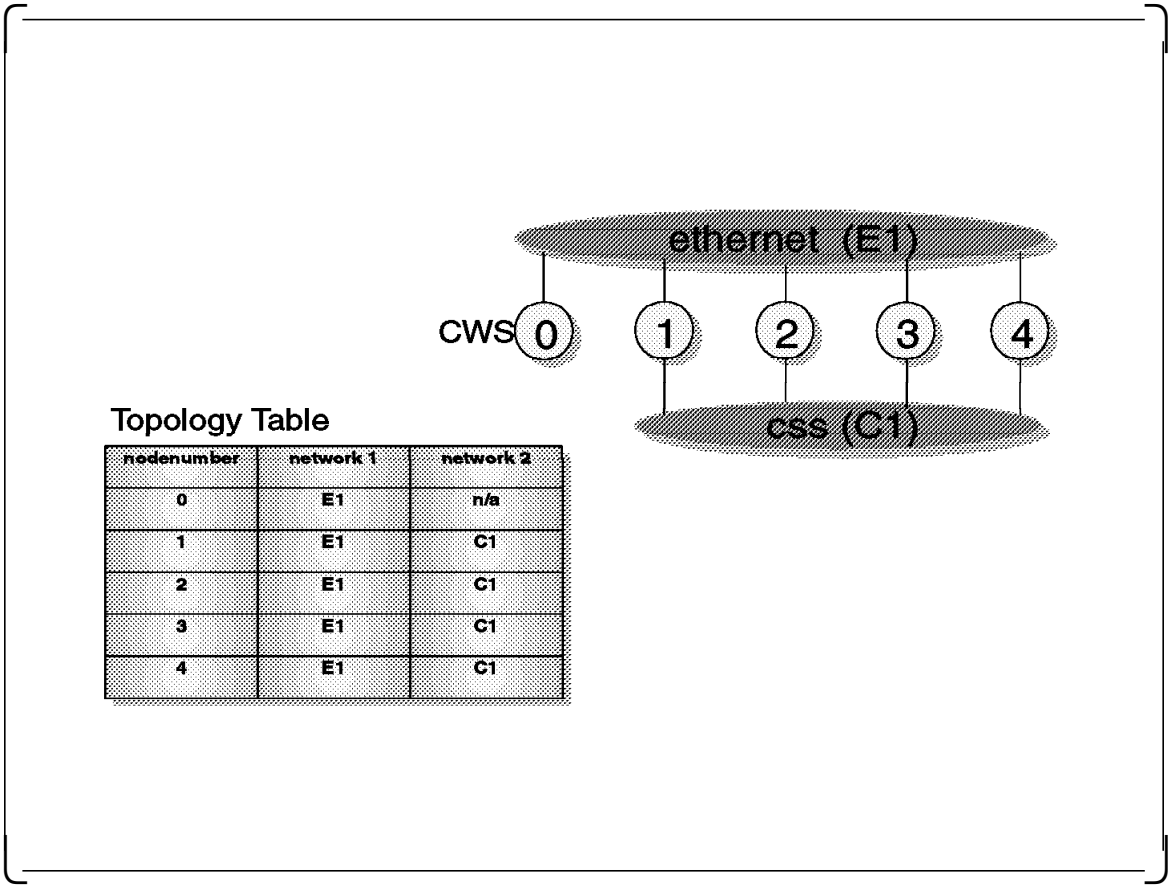
FIBRILLATE

This is the threshold at which an adapter is declared dead. It is modifiable by the sensitivity value of TS_Config. The default is four missed heartbeats.

CSS_DOWN

This is the interval to wait after FIBRILLATE threshold on a CSS adapter before declaring it dead. The default value is 120 seconds.

2.13 Topology for RS/6000 SP



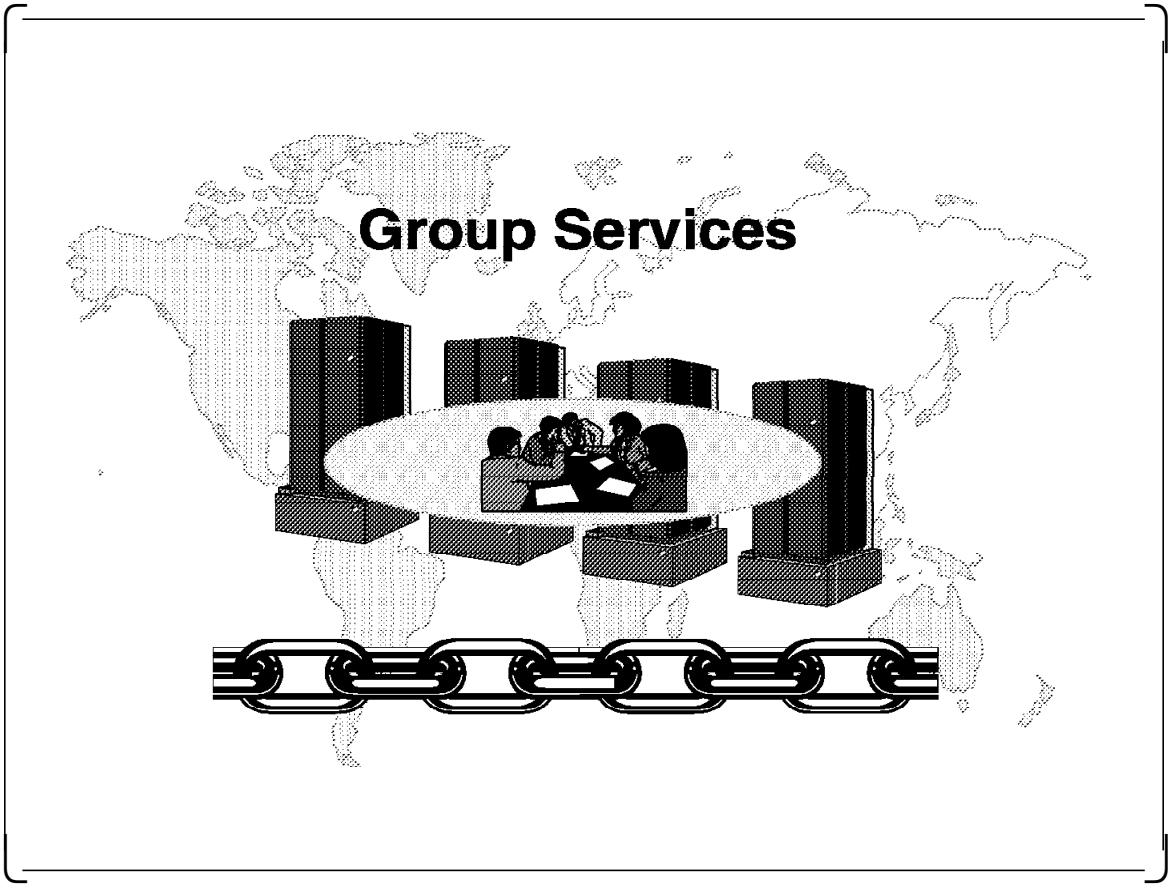
Topology Table

nodenumber	network 1	network 2
0	E1	n/a
1	E1	C1
2	E1	C1
3	E1	C1
4	E1	C1

This diagram illustrates an example of Topology Services with the currently supported adapter networks in an RS/6000 SP environment. It consists of an Ethernet adapter network connecting all the nodes including the Control Workstation, and a switch network (CSS) which only connects the nodes, excluding the Control Workstation.

There will be two different group leaders, one for the Ethernet adapter network and the other for the switch network (CSS). The group leader is the node with the highest IP address in its respective network. It is possible to have one node serving as the group leader for both networks, and in some cases the group leaders could be on different nodes (depending on the location of the node with the highest IP address in the networks).

Chapter 3. Group Services



Group Services (GS) is a distributed subsystem of the IBM Parallel System Support Programs (PSSP) V2.2 on the RS/6000 SP. It is one of three subsystems in the PSSP Version 2 Release 2 that provides a set of high availability services. For information about the other high availability subsystems, Event Management and Topology Services, refer to Chapter 5, “Event Management” on page 135 and Chapter 2, “Topology Services” on page 5.

The Group Services subsystem provides a distributed coordination and synchronization mechanism to other subsystems or to multicomputer (that is, multinode) applications. You are not required to use Group Services, but it is an effective way for multicomputer applications or subsystems to exploit the distributed consistency mechanism to achieve a useful level of fault-tolerance and high availability.

Table of Contents

- Group Services introduction
- Group Services functional overview
- Group Services Programming Interfaces (GSAPI)
- Group Services administrations

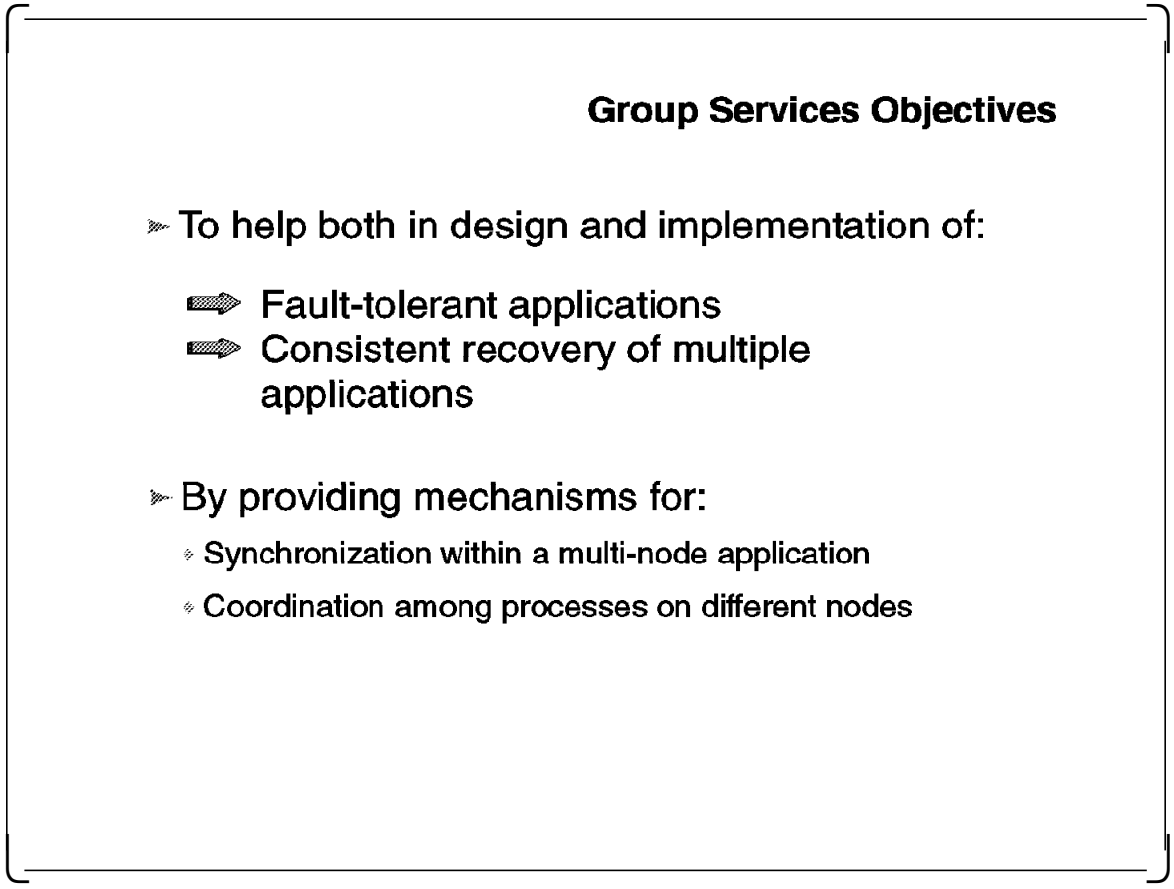
This chapter covers the following topics on Group Services:

- Introduction to Group Services
- Functional overview of Group Services
- The Group Services Application Programming Interfaces (GSAPI)
- Administration of Group Services

3.1 Group Services Introduction

This section gives an overview of Group Services.

3.1.1 Group Services Objectives



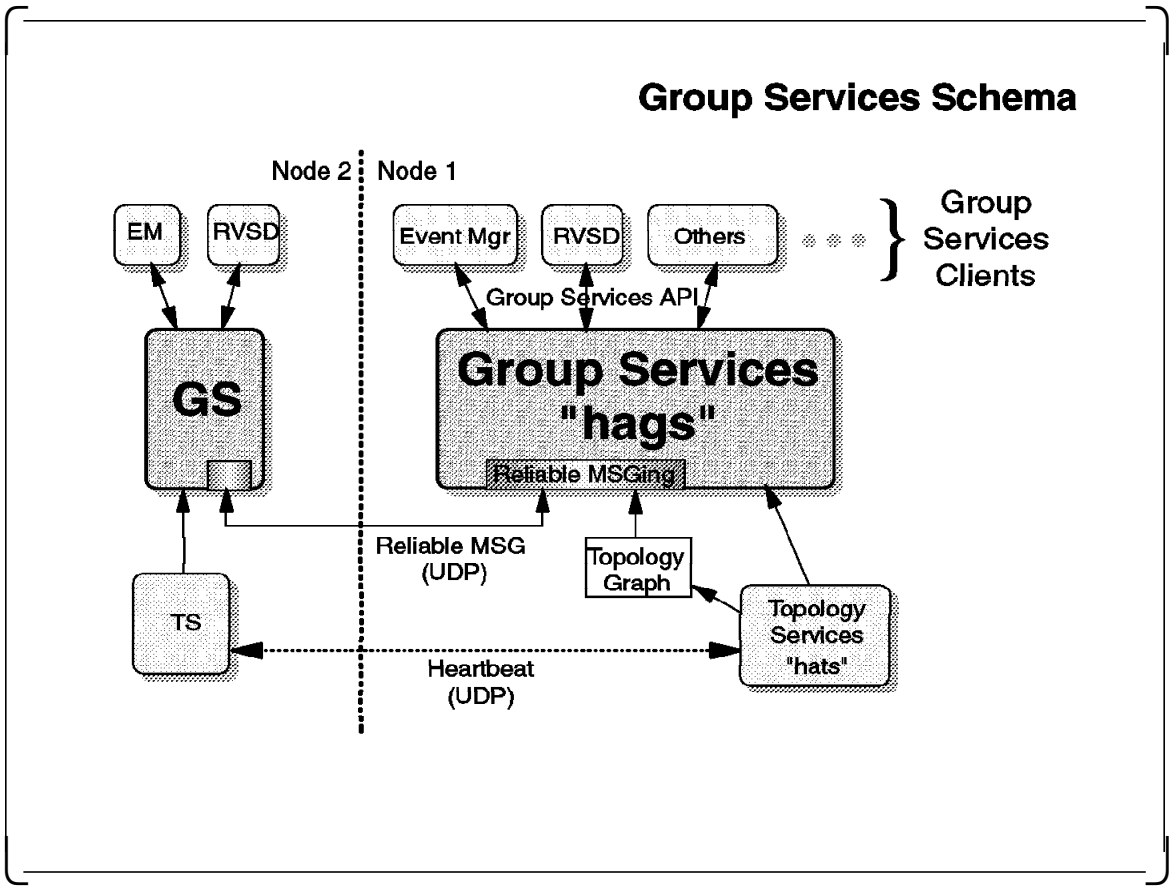
Applications that can best exploit a multicomputer environment such as RS/6000 SP typically consist of several cooperating processes running on multiple nodes. These applications are usually vulnerable to node, process, network, and storage device failures which arise from a combination of hardware failures, software failures, resource exhaustion, and operator error.

To make such applications *highly available*, some solution techniques are required. Such solutions should include the following requirements, as well as hardware techniques such as multitailed disks and IP-address takeover:

- The ability to detect component (process and node) failures
- The ability to recover from communication partitions
- The coordination of activities among the processes of an application
- The coordination of activities between applications

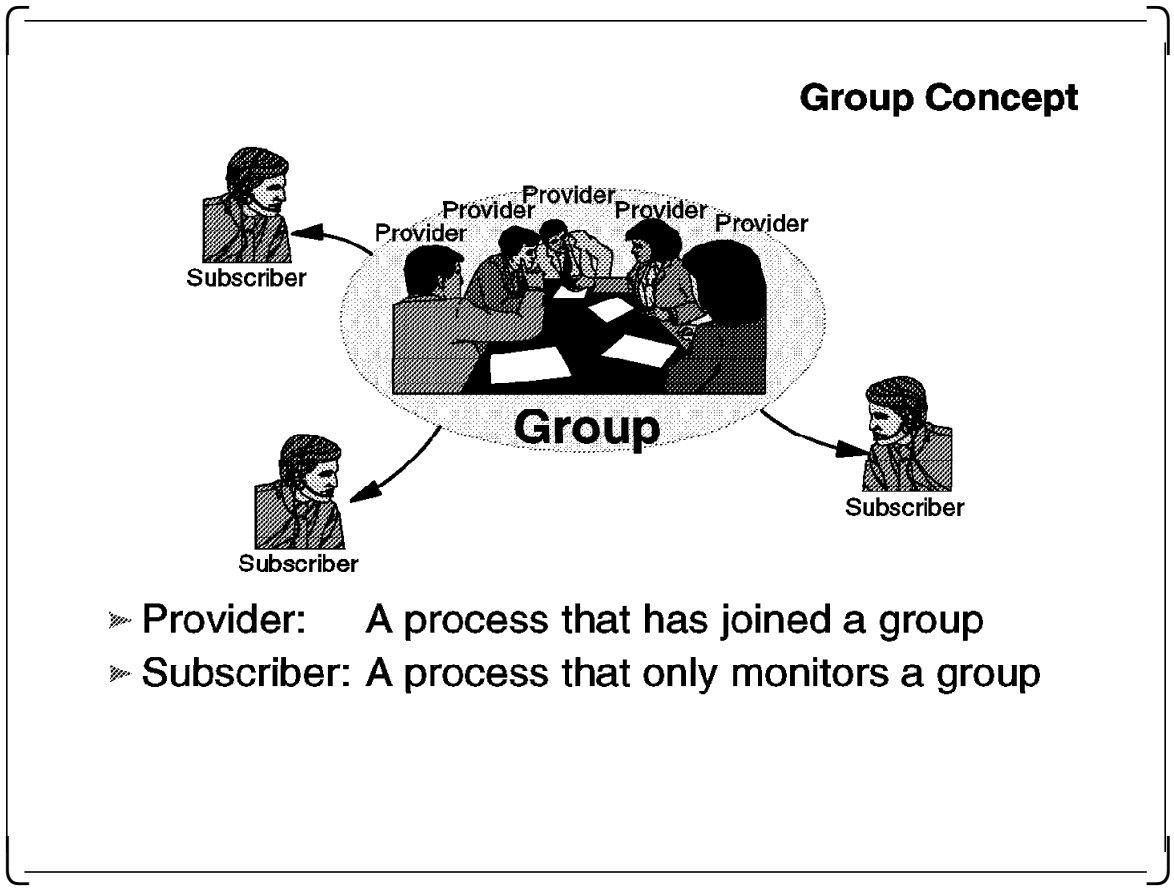
The Group Services are intended to satisfy these requirements by providing multicomputer applications or subsystems (which are referred to as *clients*) with a general purpose facility for coordinating and monitoring changes to the state of an application that is running on a set of nodes.

3.1.2 Group Services Schema



A Group Services daemon (hagsd) runs on each RS/6000 SP node and on the Control Workstation. If there is more than one system partition, then one daemon is executed on the Control Workstation for each partition. Group Services daemons exchange *reliable messages* implemented on UDP/IP for their own protocol messages through the live communication paths. These paths are provided by Topology Services through the *topology graph* on the memory. For more information on topology graph, see Chapter 2, “Topology Services” on page 5.

3.1.3 Group Concept

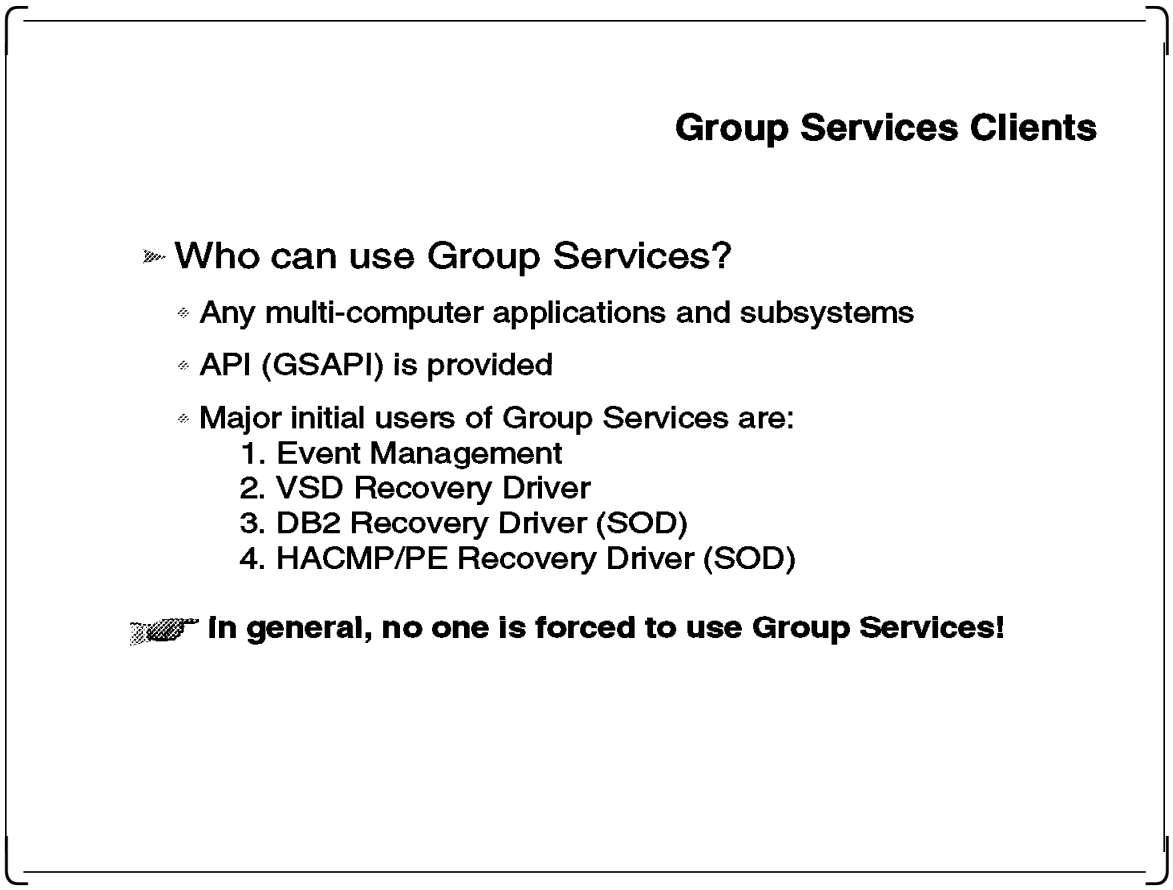


A multicomputer application that may consist of multiple processes running on multiple RS/6000 SP nodes can use any service which the Group Services subsystem provides by forming a *group*.

Any process in a Group Services domain (which is an RS/6000 SP partition in which Group Services daemons run) can create a new group, and any process in the domain may ask to become a member of a group to fully enjoy the functions that Group Services provide. Such a member of a group is referred to as a *provider*.

On the other hand, any process in the domain can ask to monitor the group. This request is called a *subscribe* request or *subscribing* to the group. If a subscribe request is successful, the process becomes a *subscriber*. A subscriber is only notified of the group's activities. A subscriber is not listed in the membership of a group.

3.1.4 Group Services Clients



The term “Group Services clients” is used to refer to both providers and subscribers. Any process of multicomputer applications or subsystems can be a Group Services client.

If you are writing a new application or updating an existing application, you can use the *Group Services Application Programming Interfaces* (GSAPI) to make the application act as a Group Services client. By using this GSAPI, you can:

- Coordinate among peer processes.
- Create a “bulletin board” to display the state of your application to other applications.
- Subscribe to changes in the state of other applications.

The initial release of PSSP Version 2 Release 2 contains the two major Group Services clients, which are:

- Event Management
- Recoverable Virtual Shared Disk

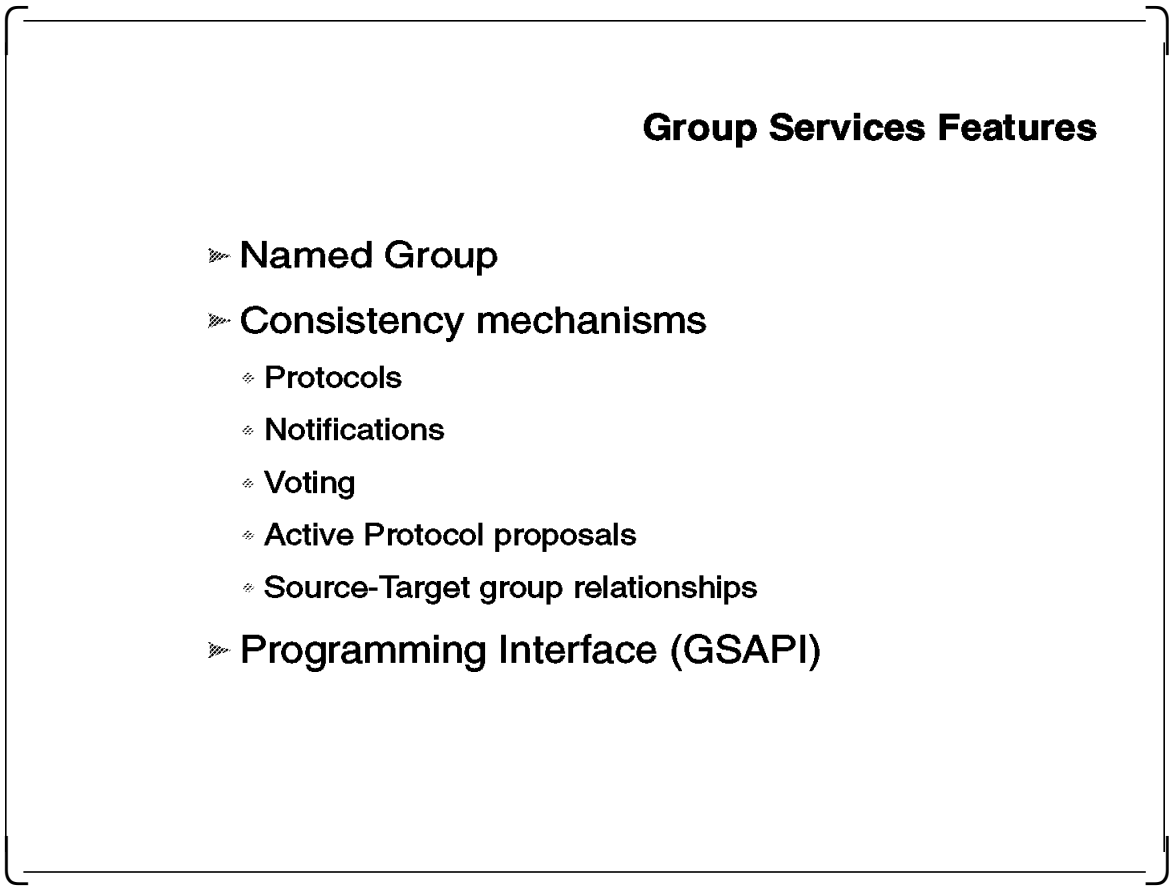
The other Group Services clients which are currently planned are:

- High Availability Cluster Multi-Processing Parallel Edition (HACMP/PE)
- DB2 Parallel Edition

Note that no applications or subsystems are forced to use Group Services, and the Group Services subsystem does not perform subsystem recovery by itself. By using Group Services abstractions, however, application developers can avoid having to develop their own synchronization and commit protocols, which tend to introduce some complexity, errors, and expensive duplications into their codes.

In addition, application developers are free to use only those abstractions that are suitable for their application, and do not have to buy into the entire toolkit of Group Services. For example, developers of applications that have historically relied on their own methods for fault-tolerance, such as database servers and transaction monitors, are free to restrict their use of Group Services interfaces to interapplication coordination services.

3.1.5 Group Services Features



Group Services has several features for implementing synchronization within an application and coordination among applications, as follows:

1. Group

   A group is a collection of individual processes (providers) which need to provide a distributed service. More details on groups are available in 3.2.1, “Group” on page 36.

2. Consistency mechanisms

   A Group Services subsystem provides various mechanisms that coordinate membership and state value changes within a group, as follows:

   - Protocols

     These are the coordination mechanisms used to modify the group data (that is, membership list and state value). Group Services provides these protocols to allow the group members to propose changes, to be notified of proposed or actual changes, and to react to changes.

   - Notification

     This is an act of informing a group member of a matter needing attention, (such as a group state change or proposed change), or of group membership changes (which would be caused by the loss of a node containing a group member). Group Services guarantees that all

notifications in a group are presented to all members of the group in the *same order*, by which the consistency of the group data is guaranteed.

- Voting

  This is a mechanism to mediate group member changes (such as join or departure) and group state value changes.

- Active protocol proposals

  Group Services guarantees that only one protocol can be executed at any time; this guarantees the consistency of the group data.

- Source-Target group relationships

  Group Services provides some level of synchronization between groups, and between the consistency mechanisms within a group.

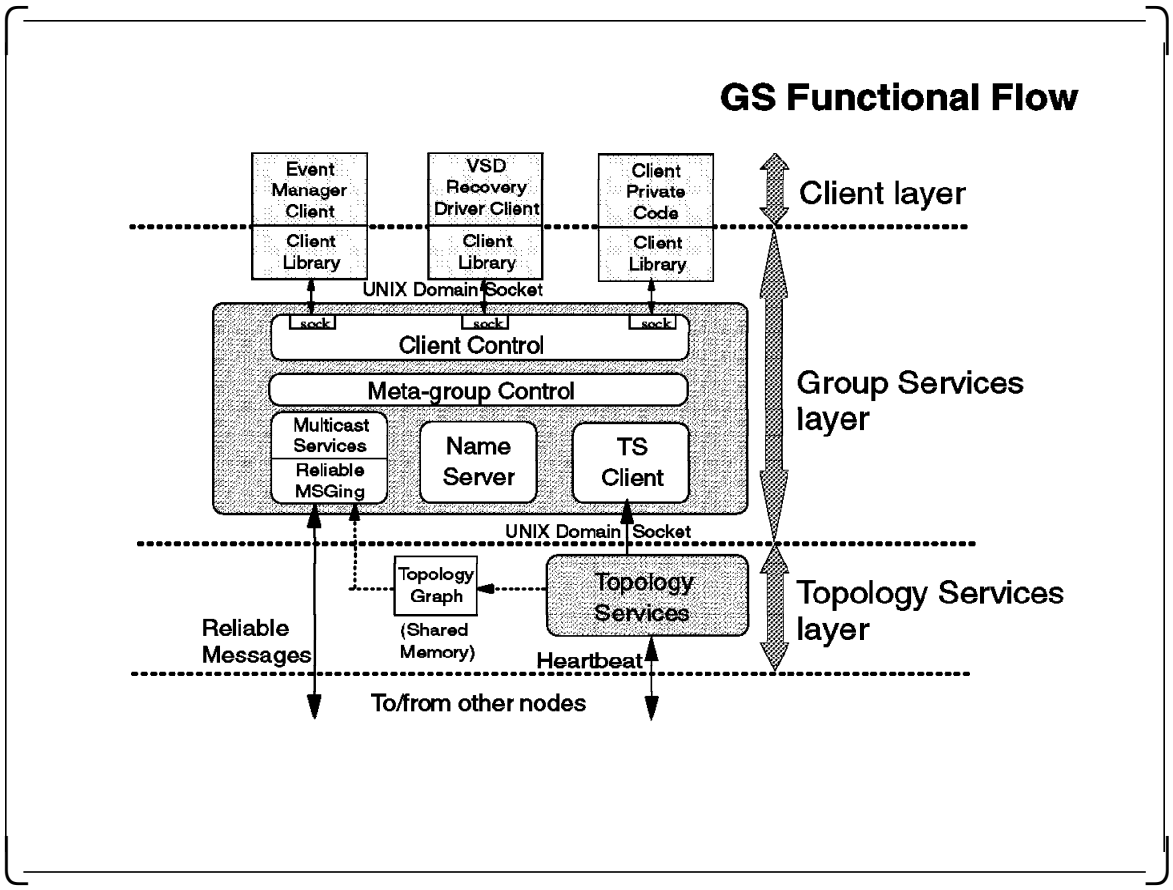
These items are further explained in 3.2, “Group Services Functional Overview” on page 36.

Note: Group Services does not force any notion of *quorum* on client groups, because the requirements for quorum are significantly different for each group. Subsystems using Group Services can implement their own quorum handling.

3. Group Services Application Programming Interfaces (GSAPI)

   Any consistency mechanism described in this section can be used by client processes through the GSAPI. This topic is covered in 3.3, “Group Services Application Programming Interface (GSAPI)” on page 78.

3.1.6 Group Services Functional Flow



This foil illustrates how Group Services clients can get services from a Group Services subsystem and how a Group Services subsystem works internally.

Group Services client processes, including the Group Services library code, establish Unix domain socket connections with Group Services daemons after completing the *init* request. For the meaning of this request, see 3.3.2, “ha_gs_init()” on page 80.

A Group Services subsystem consists of several modules, such as:

- Client control module

  This module handles the connection with Group Services clients. It also accepts requests from GS clients and manages them.

- Meta-group control module

  Meta-groups are the collection of the Group Services daemons that support the user groups. This module manages the behavior of Group Services daemons.

- Topology Services client module

  Group Services receives the services from Topology Services through this module.

- Name Server module

  This module controls Group Services namespace. For details on GS namespace, see 3.2.1.3, “Group Namespace” on page 42.

- Reliable messaging module

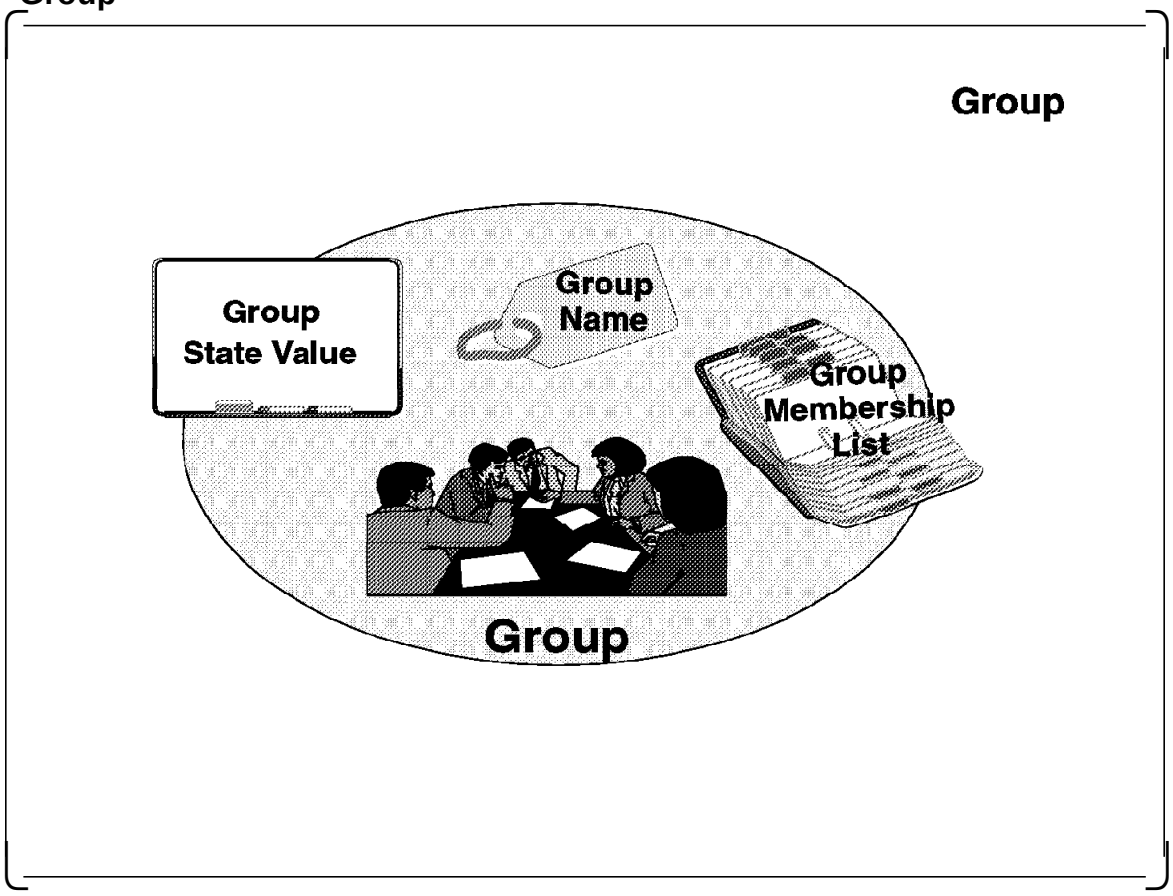
  This module provides reliable sequenced delivery of datagrams between Group Services daemons. It uses the “Topology Graph” on the memory (which is dynamically updated by Topology Services) to select the network route.

Group Services itself uses Topology Services for maintaining system topology information, which includes node and LAN adapter membership information. For more details of the functions of Topology Services, see Chapter 2, “Topology Services” on page 5.

3.2 Group Services Functional Overview

This section provides an overview of the Group Services functions.

3.2.1 Group



A *group* is a collection of individual processes, which are also called *members* or *providers*. A group may have its members on multiple nodes, and each node may have multiple members.

For each group, the Group Services subsystem maintains the following three group state data:

- *A name*

  A token to uniquely identify each group in the system.

- *A membership list*

  A list of one or more providers. In a group, each provider is identified by its identifier, which consists of an *instance ID* and the node number on which the provider is running.

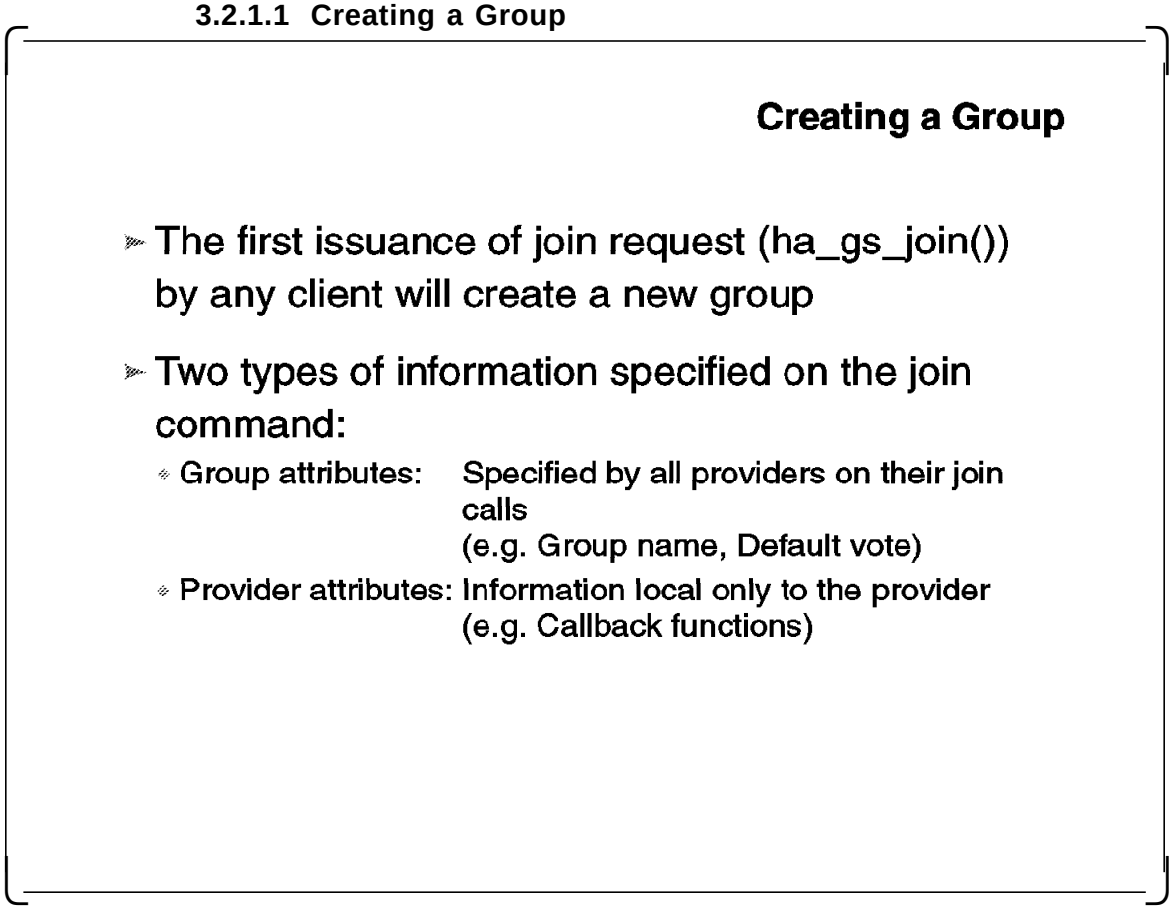
  The Group Services subsystem maintains the list in order of age. The oldest provider (that is, the first provider to join the group) is at the top of the list, and the youngest is at the bottom. All of the group′s providers and subscribers see the same ordering of the list.

  Note: The *instance ID* of a provider is defined by the provider itself when it joins a group. The ID must be unique on a node for the group.

- *A group state value*

  A byte field whose length is between 1 and 256 bytes. The state value of a group is not interpreted by the Group Services subsystems. The default value is four byte “0”s (zeros).

3.2.1.1 Creating a Group



Typically, an application that uses Group Services defines one or more group names that are known from the beginning to all of the processes that are part of the application.

During initialization of the application, each process asks to join the group. On the receipt of the first join request (*ha_gs_join* subroutine call), the Group Services subsystem creates the group. The subsequent join requests result in new providers joining the group.

The first join request creates the group and defines the attributes of the group as follows, according to the information specified in the join request:

- The name of the group
- An application-defined version code
- The number of phases (one or multiple) for join and failure leave protocols
- A time limit for voting in each phase of an n-phase protocol

  This is the number of seconds within which each provider must submit its vote for each phase of an n-phase join or failure leave protocol. “0” may be specified, if no time limit is desired.
- A default vote

  It is used as a proxy for a provider that fails, or that fails to vote in time (if a time limit is used). The value of a default vote should be *approve* or *reject*.

- A batch control value to specify how requests may be batched

  Join requests may be batched with other join requests, failure leave requests may be batched with other failure leave requests, or both or neither may be specified.

- Attributes related to a source-target relationship, if any:
  - The name of the source group for this group, which implicates this group as a target group.
  - The number of phases to use for the source-reflection protocols, which run in the target-group when the source-group changes its state value
  - The voting phase time limit for source-reflection protocols, if they are n-phase

  For details on source-target relationship, see 3.2.6, “Source-Target Group Relationships” on page 73.

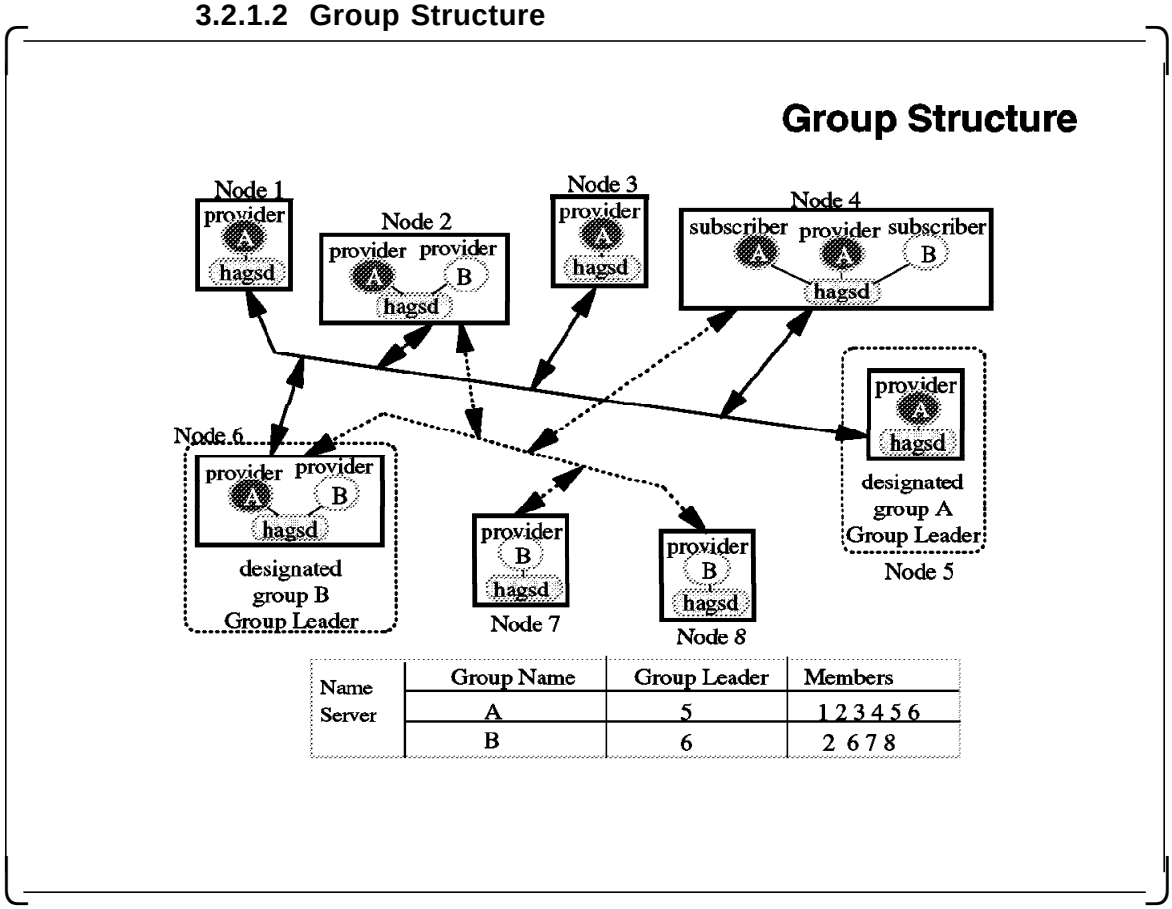
All subsequent requests to join this group also include group attributes, and the information must match the group′s established attributes. Otherwise, the subsequent join request is rejected.

A join request also includes the following attributes of the provider who issues it:

- The instance number to be used by this provider
- A local name for this provider (optional)
- A set of callback functions

Join requests are discussed in detail later in this chapter.

3.2.1.2 Group Structure



In this chart, we refer to the Group Services daemon on the nodes as hagsd:. For each group registered with Group Services, one hagsd will be designated as the *Group Leader* (GL) for that group. Group Services will maintain replicated data across the multiple nodes on which the providers reside, so that if the node containing the GL fails, a new GL for that group is elected from the remaining nodes.

The chart illustrates a logical structure of two groups, group A and group B. For group A, node 5 is acting as the GL. On node 6, the hagsd acts as the GL for group B, but simply as an hagsd for group A. Nodes 2 and 4 have multiple clients, so the hagsd daemons deal with operations for both groups.

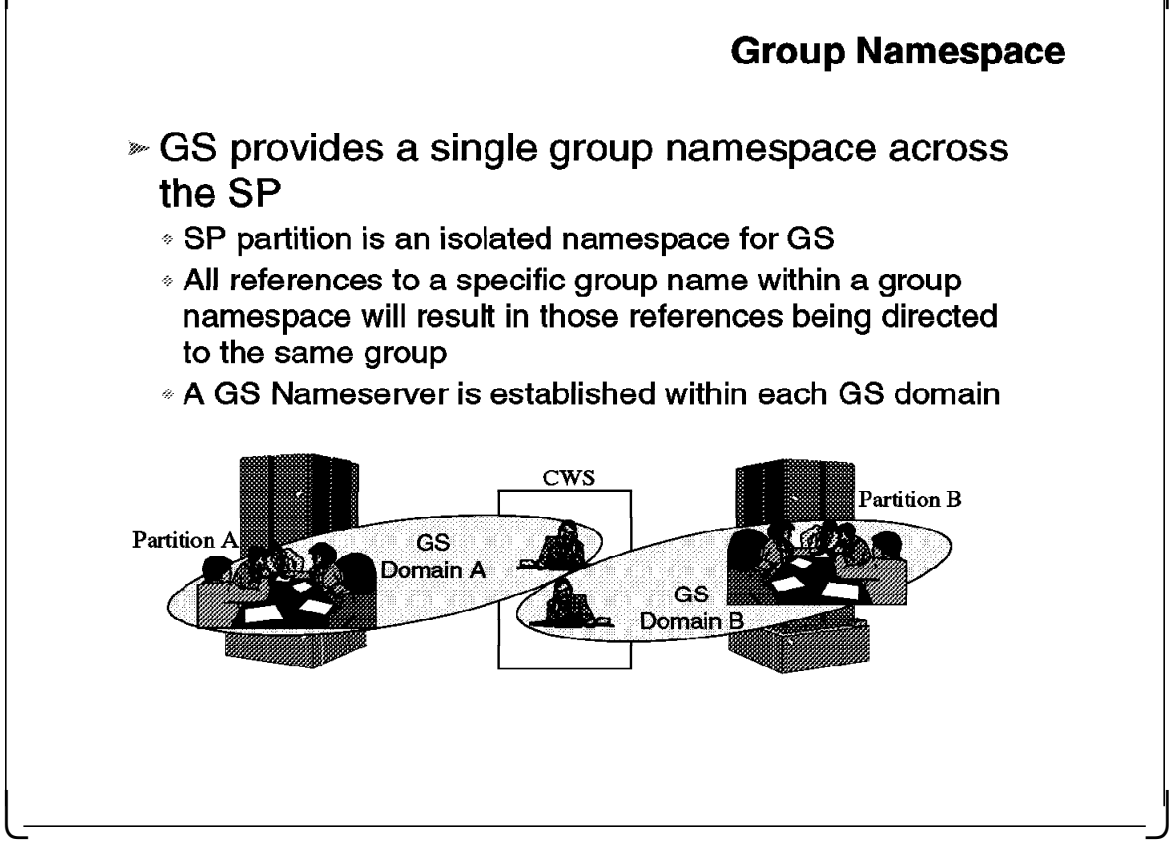
Each GL is responsible for acting as the coordinator for all operations of each group as follows:

- Managing the group information:
  - Membership lists
  - State information
- Managing the group meta-data:
  - Nodes (that is, addresses) of clients
  - Group protocol information
  - Number of phases for providers joining or leaving the group
  - Source-Target information for the group
  - Default votes for providers
  - If reflecting multiple membership change requests, batching is allowed

- Serializing “simultaneous” protocol proposals
- Recovery actions whenever a group member or hagsd is lost due to process or node failure:
  - If the GL is lost, a new GL must be elected from remaining hagsd daemons.
  - If hagsd is lost, the hagsd needs to be recreated, and the client data from that node needs to be retrieved from the GL for each group that had a member on the node where the hagsd was lost.

Note: These functions of GL are *invisible* to the Group Services clients.

3.2.1.3 Group Namespace



It is vital that Group Services be able to keep track of the groups that its clients want to form. To do this, a Group Services nameserver is established within each domain. A domain is the set of running nodes within each RS/6000 SP system partition. The GS nameserver is responsible for keeping track of all client groups created in that domain. The lowest-numbered running node in the partition will be elected as a GS nameserver.

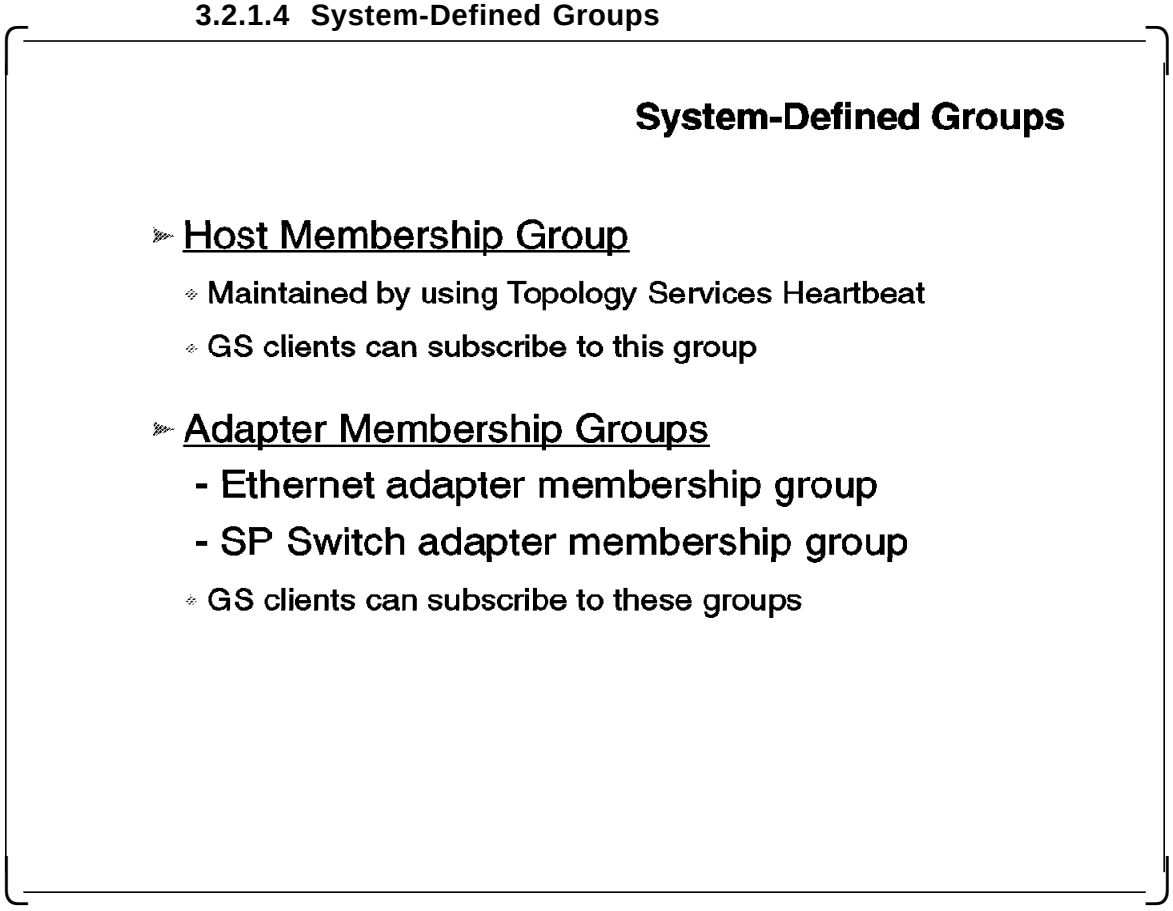
To ensure that only one node becomes a GS nameserver, the following process is followed:

1. When each GS daemon is connected to Topology Services, it waits for Topology Services to inform it of which nodes are currently running in this system partition.
2. Based on the input from Topology Services, each GS daemon finds the lowest-numbered running node in the partition. The daemon compares its own node-number to the lowest-numbered node.
   a. If a node *is* the lowest-numbered node, then it waits for all other running nodes (listed in the input from Topology Services) to “grovel” to it. This node is the “putative GS nameserver.” It also waits for a 20 second coronation timer to expire. This node is in the “NS::ENsState:KAscend” state.
   b. If a node is *not* the lowest-numbered node, then it starts sending “grovel” messages to the lowest-numbered node. It will resend this message every 5 seconds. This node is in the “NS::ENsState:KGrovel” state.

3. Once all running nodes have groveled to the putative GS nameserver, and its 20 second coronation timer has expired (popped), it then sends an "insert" message to the nodes. All nodes then must acknowledge (ACK) the insert. At this point, the putative GS nameserver becomes the established GS nameserver (the "NS::ENsState:kBecomeNS" state), and it sends a "commit" message to all nodes (which then go to the "NS::ENsState:kCertain" state).
4. At this point, the Group Services domain is established, and requests by clients to join or subscribe to groups will be processed.

Note: The "NS::ENsState:xxxxxxx" states are mentioned for use in interpreting the hagsns command output in 3.4.3.1, "Other Utilities" on page 104.

3.2.1.4 System-Defined Groups



The Group Services subsystem provides several system-defined groups to which GS clients can subscribe to keep track of hardware status. The system-defined groups are:

- Host Membership Group

  The Group Services subsystem keeps track of node status to determine when nodes are no longer reachable. Accordingly, a fully isolated or failed node triggers notifications to all groups that have one or more providers on the failed node or nodes.

- Adapter Membership Group (AMG)

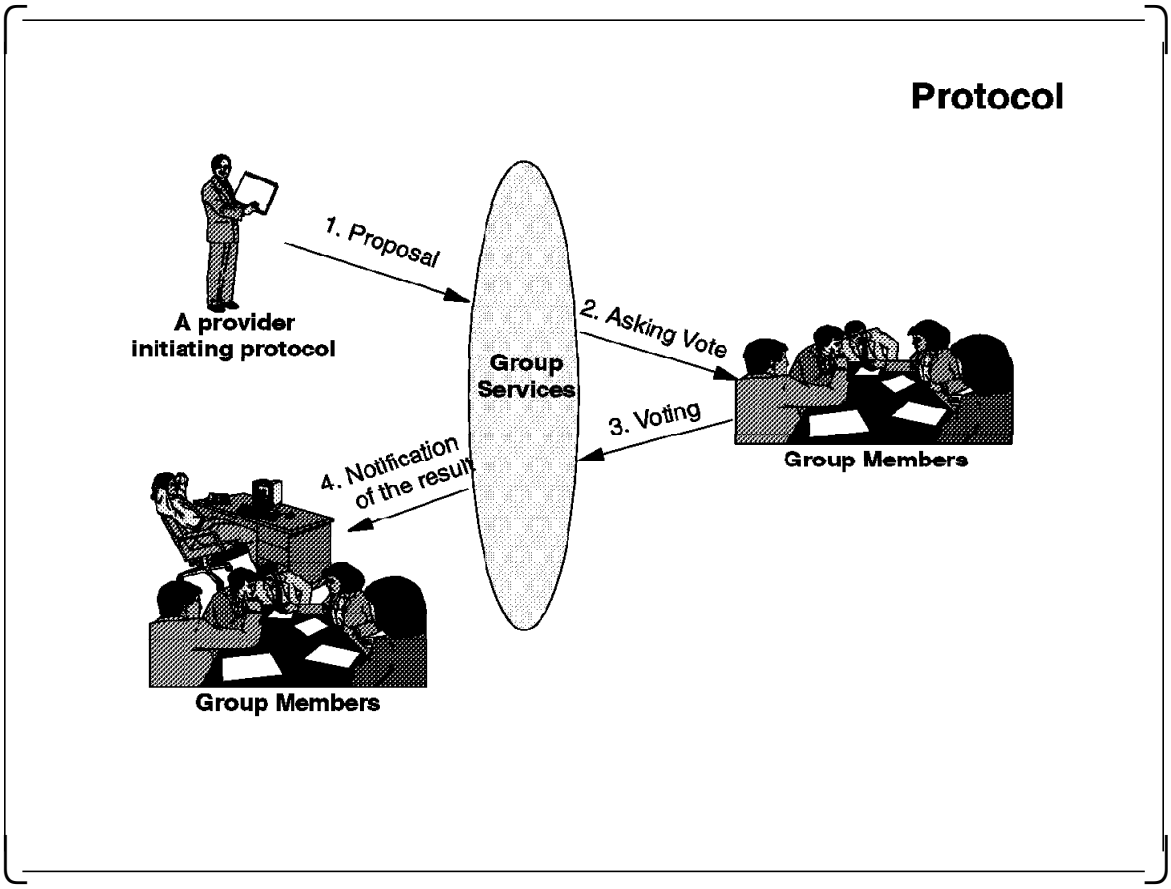
  The Group Services subsystem also keeps track of the status of Ethernet and SP Switch adapters, and the status of the adapters is reflected by the Group Services subsystem in the following two system-defined groups:

  – Ethernet Adapter Membership Group

  – SP Switch Adapter Membership Group

  By subscribing to these groups, a GS client can obtain adapter membership information.

3.2.2 Protocols



The work of the Group Services subsystem is accomplished through a variety of *protocols*. A protocol is the mechanism which coordinates membership and state value changes within a group.

Typically, a protocol is executed in the following sequence:

1. Proposal

   Any provider of a group or the Group Services itself can initiate a protocol, or *propose* a protocol. If multiple providers submit proposals at the same time, then only one proposal will be accepted by Group Services. The other proposals will be returned asynchronously to the providers that proposed them, with the *HA_GS_COLLIDE* error code.

2. Asking vote

   If the protocol requires a voting process, (in other words, if the protocol is an n-phase protocol), then the Group Services asks each member of the group to *vote*. Each provider, including the provider who proposed the protocol, will be notified (n-phase protocol notification), and all providers that have received the notifications are expected to submit a voting response (Approve, Reject, or Continue). Voting process will be repeated n-1 times in the n-phase protocol.

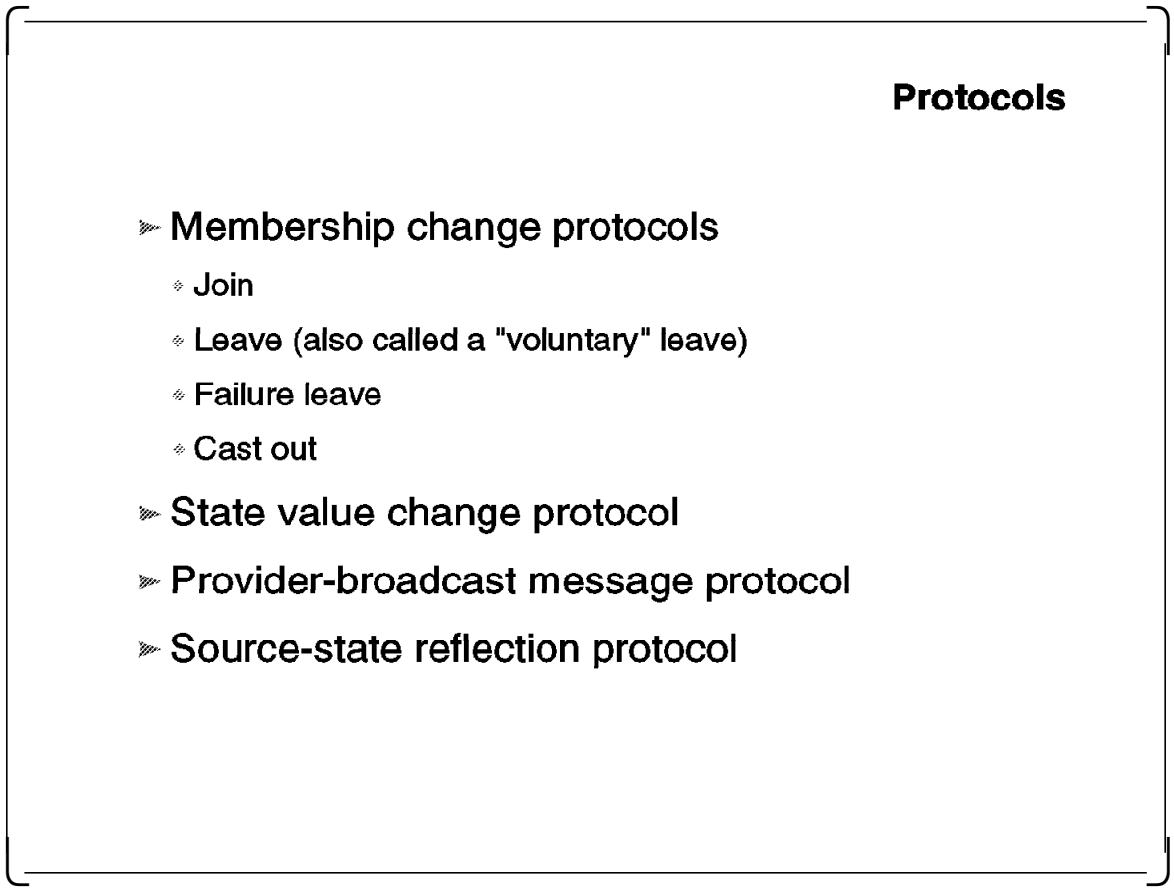
   If the protocol is a one-phase protocol, then the protocol is automatically approved. In this case, no voting process occurs.

3. Voting

   Group Services uses a flexible n-phase voting protocol to mediate provider joins and leaves, and state value changes. Applications or subsystems can tailor their synchronization and coordination requirements by choosing the number of phases needed to complete membership or state changes. When a protocol is proposed, the proposal contains an indication of whether it is one-phase or n-phase. This is discussed in detail in 3.2.4, “Voting” on page 66.

4. Notification of the result

   At the end of the protocol, the Group Services subsystem notifies the providers of the results of the protocol, that is, its approval or rejection. The updated membership list and state value, if any, are also sent to the providers.



Protocols can be grouped into three categories:

- Membership change protocol

  These protocols are used when a provider wants to join or leave a group. If approved, the membership of the group changes. Membership change protocols include:

  - Join
  - Leave (also called ″voluntary leave″)
  - Failure leave
  - Cast-out

  Each type of protocol is discussed in the following sections.

- State value change protocol

  This protocol is used when a provider wants to change the state value of the group, but wants to leave the membership unchanged.

- Provider-broadcast message protocol

  If this protocol is proposed as one-phase, it allows a provider to broadcast a message to all other providers in the group, without voting.

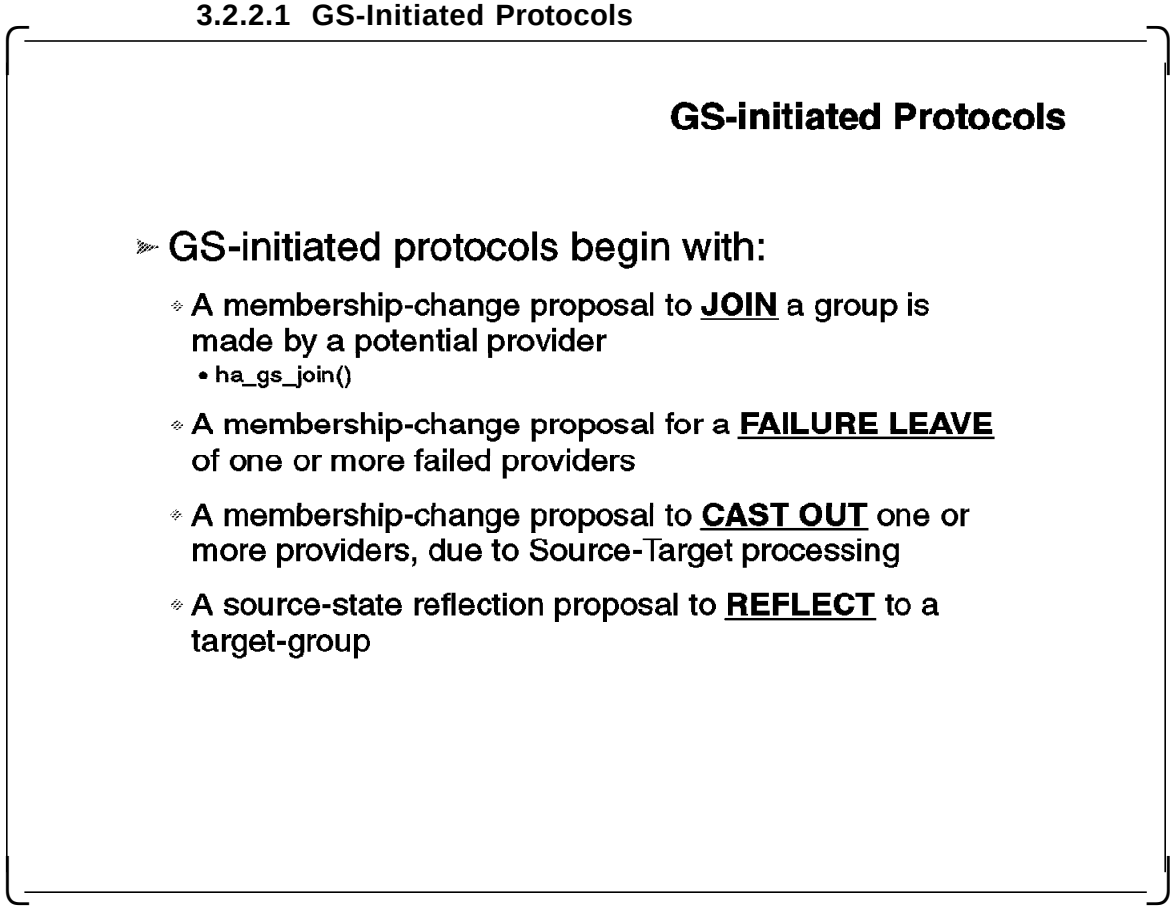
  If this protocol is proposed as n-phase, it allows a provider to broadcast a message to the other providers in the group, and also initiates the standard voting phases.

  This protocol does not affect the group membership.

- Source-state reflection protocol

  This protocol is initiated by Group Services when a source-group has modified its state value in certain cases. In these cases, target-groups are notified of the change in the source-group′s state value through this protocol. For details about source- or target- group, see 3.2.6, “Source-Target Group Relationships” on page 73.

3.2.2.1 GS-Initiated Protocols



As mentioned earlier, *protocol proposals* are made by either a provider of a group or the Group Services subsystem itself.

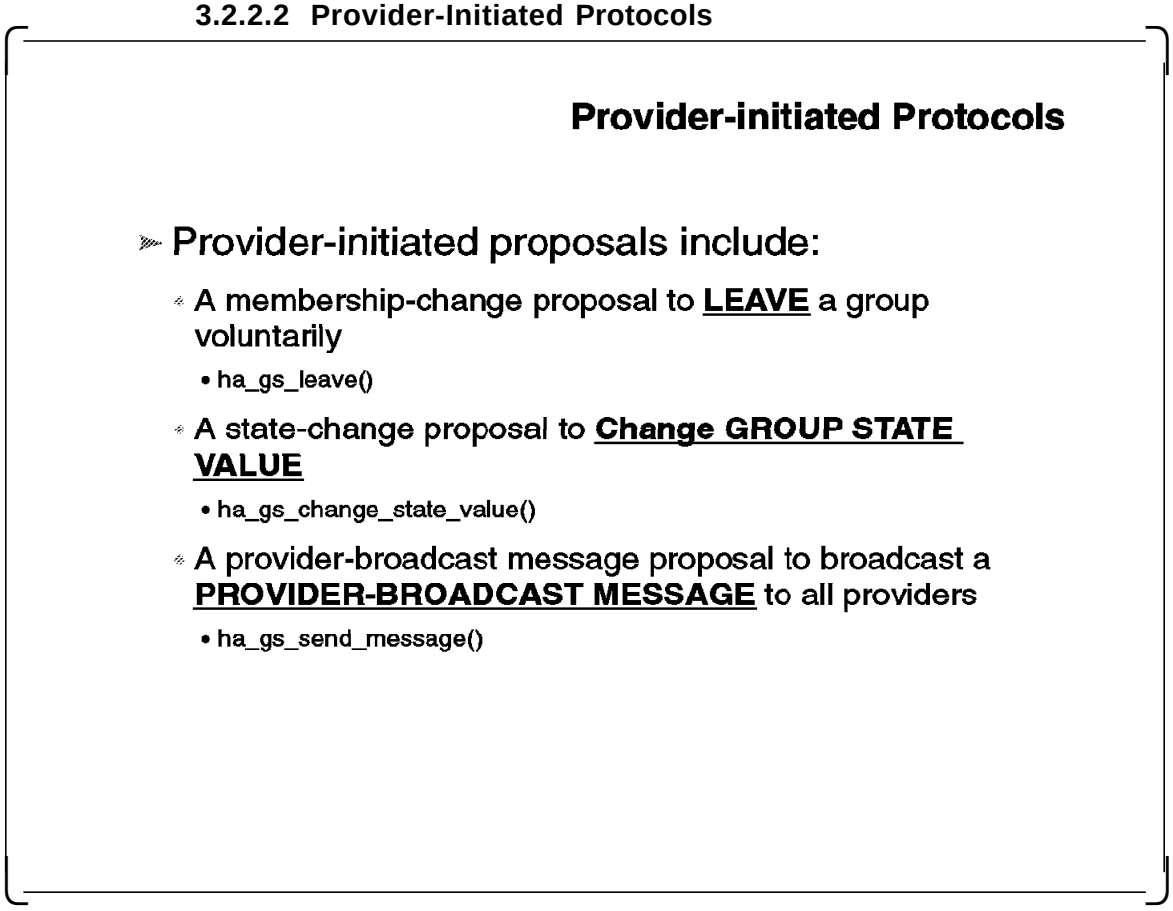
The Group Services-initiated protocol proposals cover the following situations:

- A membership change proposal to *join* a group made by a potential provider
- A membership change proposal for a *failure leave* of one or more failed providers
- A membership change proposal to *cast out* of one or more providers, due to *Source-Target* processing
- A source-state reflection proposal to *reflect* to a target-group when its source-group changes its state value through a non-membership change protocol

The number of phases for these protocols is determined as follows:

- For a *join* proposal, the provider must specify either a one-phase or an n-phase join protocol in the group attributes.
- For a *failure leave* and *cast out* proposals, the Group Services subsystem uses the same number of phases that was specified when the group was created, that is, the number of phases that the first join request specified.
- For *source-state reflection* protocols, the Group Services subsystem uses the *source-reflection-phases* setting to control the number of phases in the group attributes.

3.2.2.2 Provider-Initiated Protocols

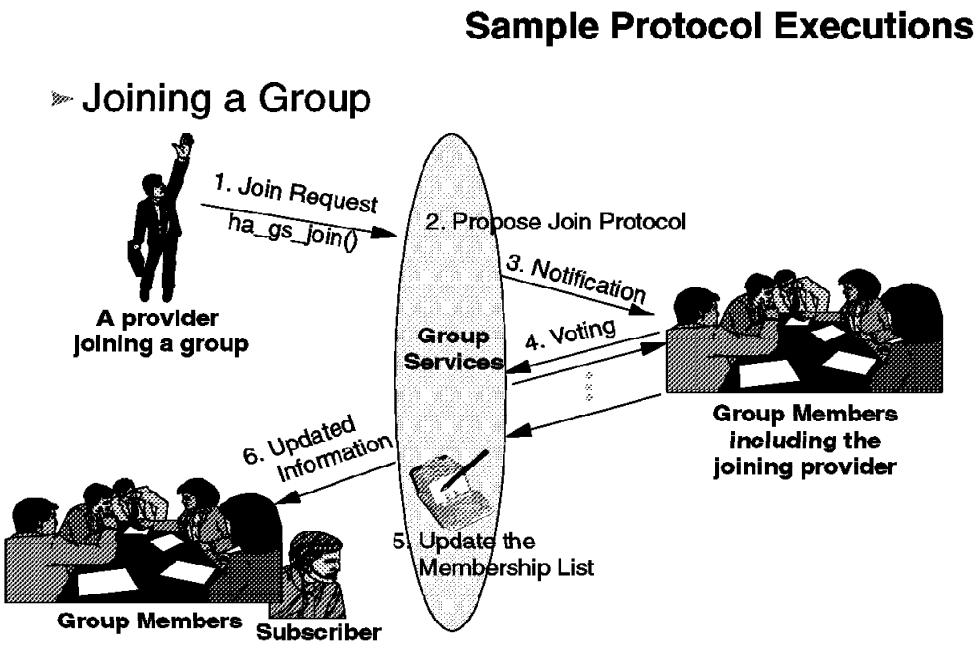


Provider-initiated proposals are for the following:

- A membership-change proposal to *leave* a group is made by a provider in the group.
- A membership-change proposal to *change the group state value* is made by a provider in the group.
- A provider-broadcast message proposal to *broadcast a provider-broadcast message* to all providers is made by a provider in the group.

A provider calls a Group Services Application Programming Interface (GSAPI) subroutine to propose one of these protocols, specifying the number of phases for the protocol.

3.2.2.3 Sample Protocol Executions--Join



The next four foils, starting with this page, show the typical proceedings for the following protocols:

- Join
- Change group status value
- Leave
- Failure leave

A typical scenario for the join protocol is as follows:

1. A provider of the group issues a join request, that is, a *ha_gs_join()* subroutine call.
2. The Group Services subsystem initiates a join protocol.
3. When the Group Services subsystem initiates a join protocol, it notifies all providers, including the “old” providers that are already in the group, as well as the providers asking to join the group.

   Note that if this is the first join request by the first provider of the group, the Group Services subsystem creates the new group before notifying.

   Membership changes may be batched, which means that multiple providers can be joined to a group for the cost of a single n-phase (or one-phase) protocol execution, rather than having to repeat the exercise once for every new provider. This can result in significant performance gains on groups with a large number of providers, especially if they tend to join simultaneously.

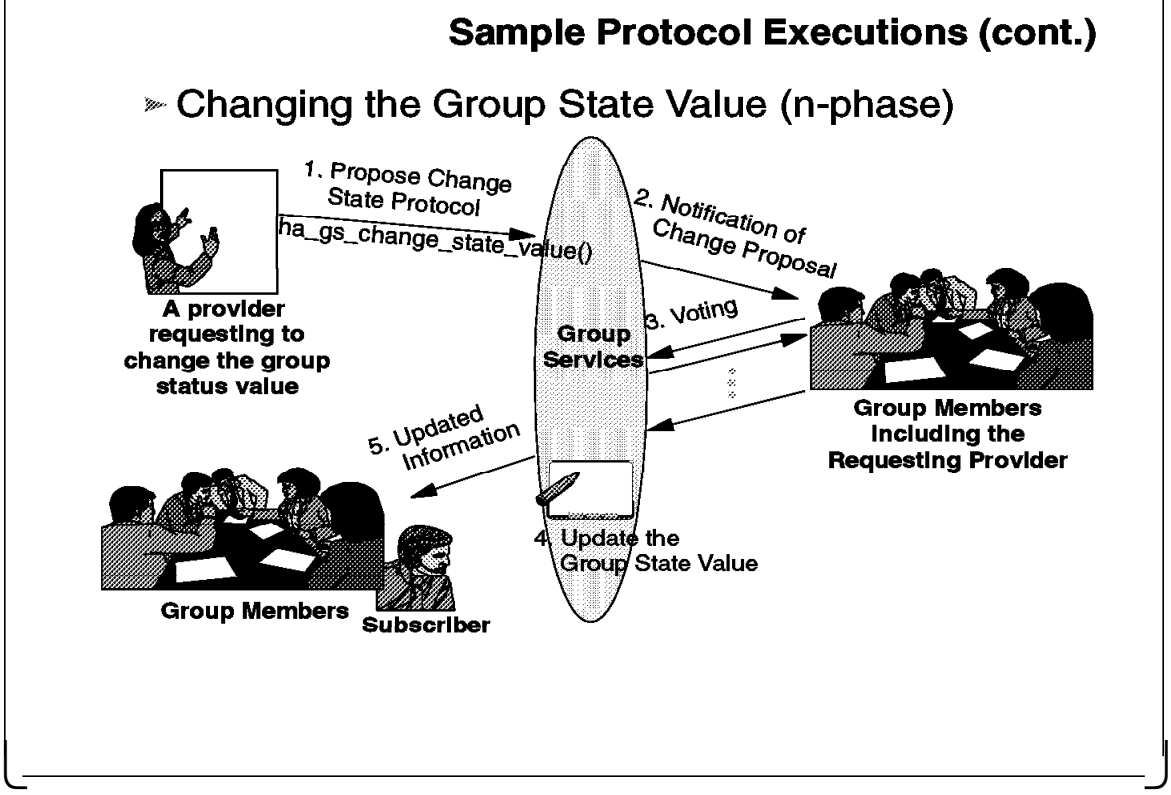
4. Once the Group Services subsystem has initiated the protocol, the providers execute it as described in 3.2.2, “Protocols” on page 45

5. Once voting has completed and the protocol has been approved, the membership list is changed. If the join is rejected for any reason, the membership remains unchanged.

6. All the providers receive an updated membership list, or state value, or both. They may also receive a provider-broadcast message, if one was included in the final vote. Subscribers may receive the updated membership list, state value, or both, depending on their subscription request.

   If a join request is rejected, the rejected provider receives a notification that its application to join the group has been rejected. The existing providers also receive the notification. The subscribers receive no notification.

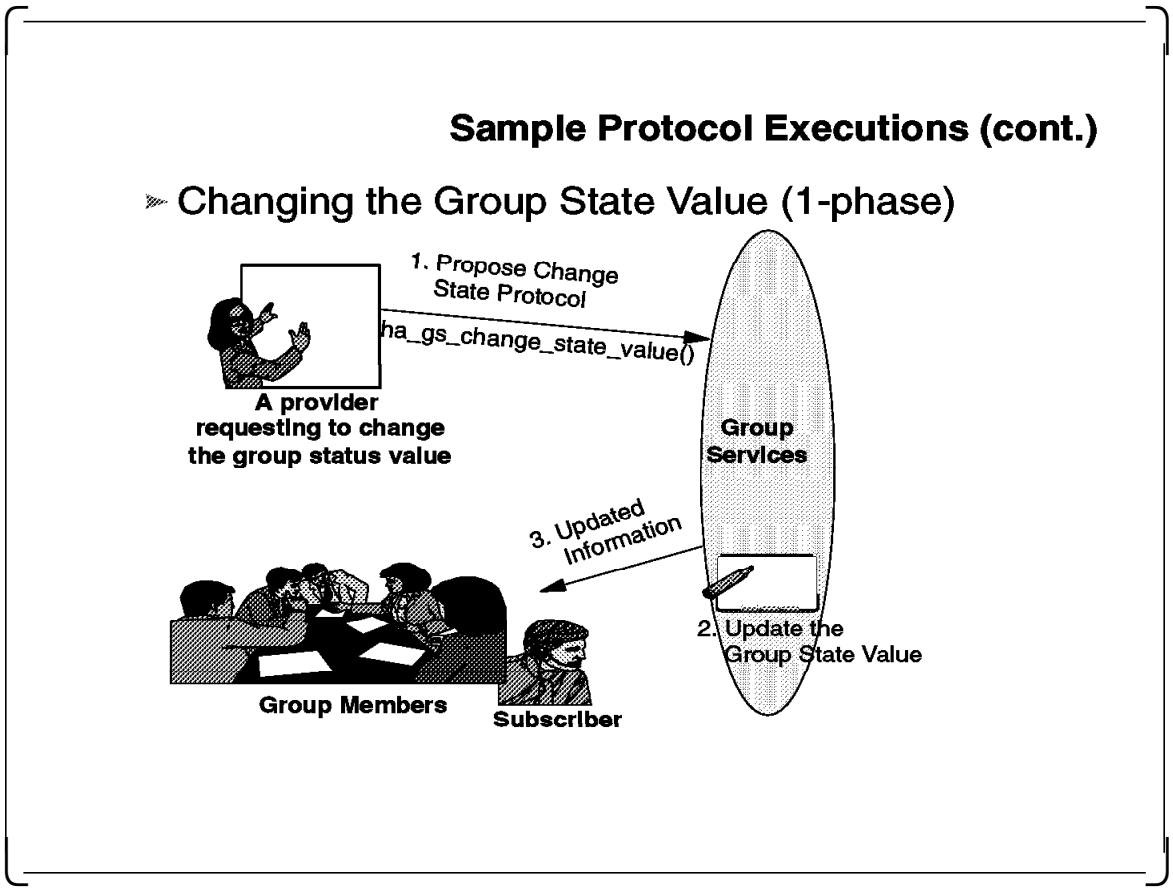
3.2.2.4 Sample Protocol Executions--Change State Value



A typical scenario for a state value change protocol is as follows:

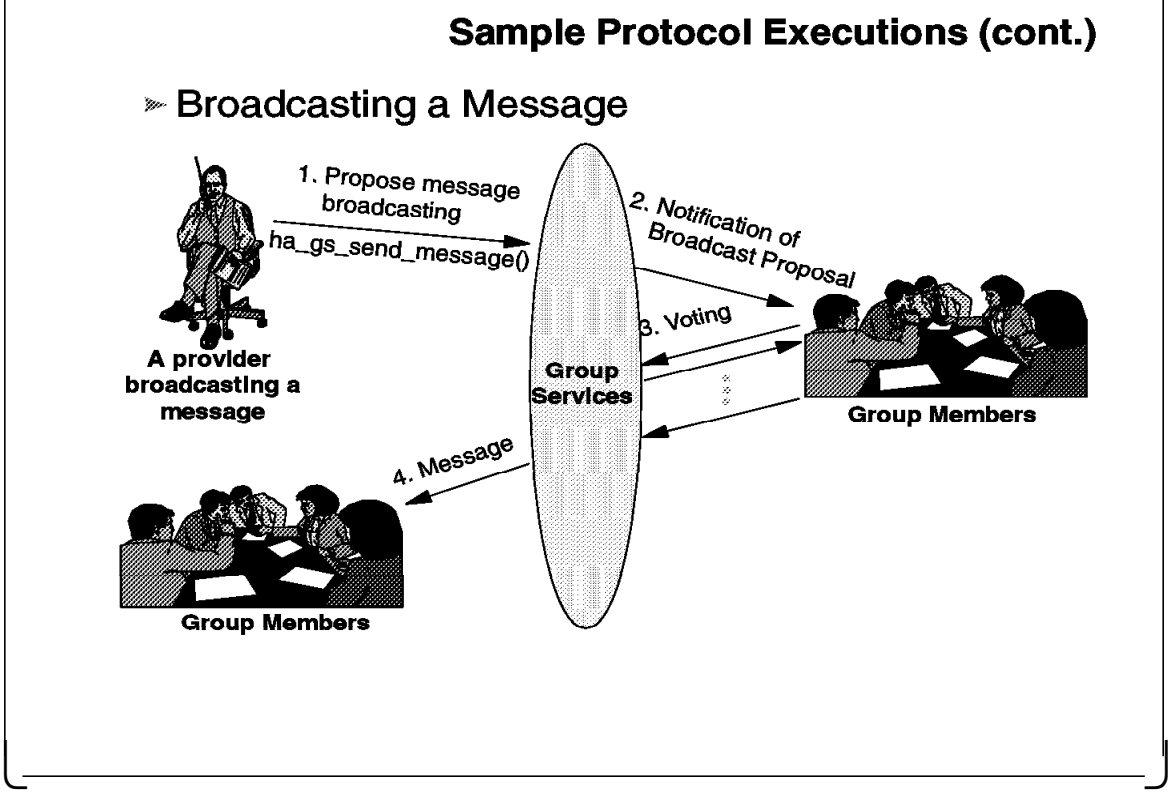
1. A provider calls a *ha_gs_change_state_value()* subroutine to propose a state value change protocol.
2. The Group Services notifies all providers in the group that voting is requested for a state value change protocol, if the protocol is an n-phase protocol. If the protocol is a one-phase protocol, the proposal is automatically approved.
3. The voting process continues.
4. The Group Services subsystem updates the group status value according to the request, if the protocol has been approved. The status value reverts to its value at the beginning of the protocol, if the protocol is rejected.
5. If a state value change is approved, the providers and subscribers receive the updated state value. In addition, if the group is a source-group, its target-groups also receive notification of the change.

   If the protocol is rejected, the providers receive notification of the rejection. The subscribers receive no notification, and nothing is reflected to any target-groups, if they exist.



This illustration shows the typical scenario for a state value change protocol with one-phase, that is, without voting. The protocol is automatically approved.

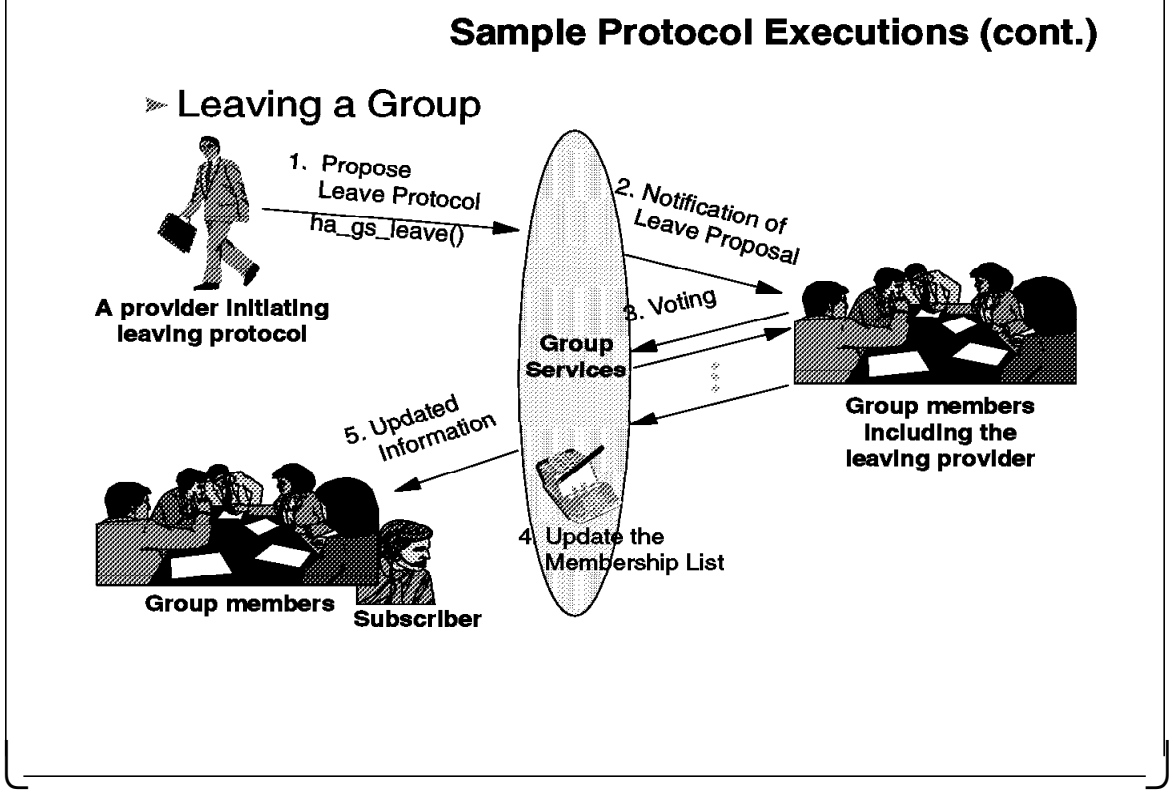
3.2.2.5 Sample Protocol Executions--Broadcast



A typical scenario for a broadcast protocol is as follows:

1. A provider calls a *ha_gs_send_message()* subroutine to propose a leave protocol.
2. The Group Services notifies all providers in the group that voting is requested for a state value change protocol, if the protocol is an n-phase protocol. If the protocol is a one-phase protocol, the proposal is automatically approved.
3. The voting process proceeds (unless it is a one-phase protocol).
4. If it is a one-phase protocol, the message contained in the proposal is broadcasted to all providers. Subscribers receive no notification. If it is an n-phase protocol:
   - If it is approved, and if the group state value was changed during the voting phases, the providers and subscribers receive the updated state value.
   - If it is approved, but the group state value was not changed during voting phases, the providers receive notice that the protocol is completed. Subscribers receive no notification.
   - If it is rejected, the state value reverts to its value at the beginning of the protocol. The providers receive notification of the rejection. The subscribers receive no notification.

3.2.2.6 Sample Protocol Executions--Leave

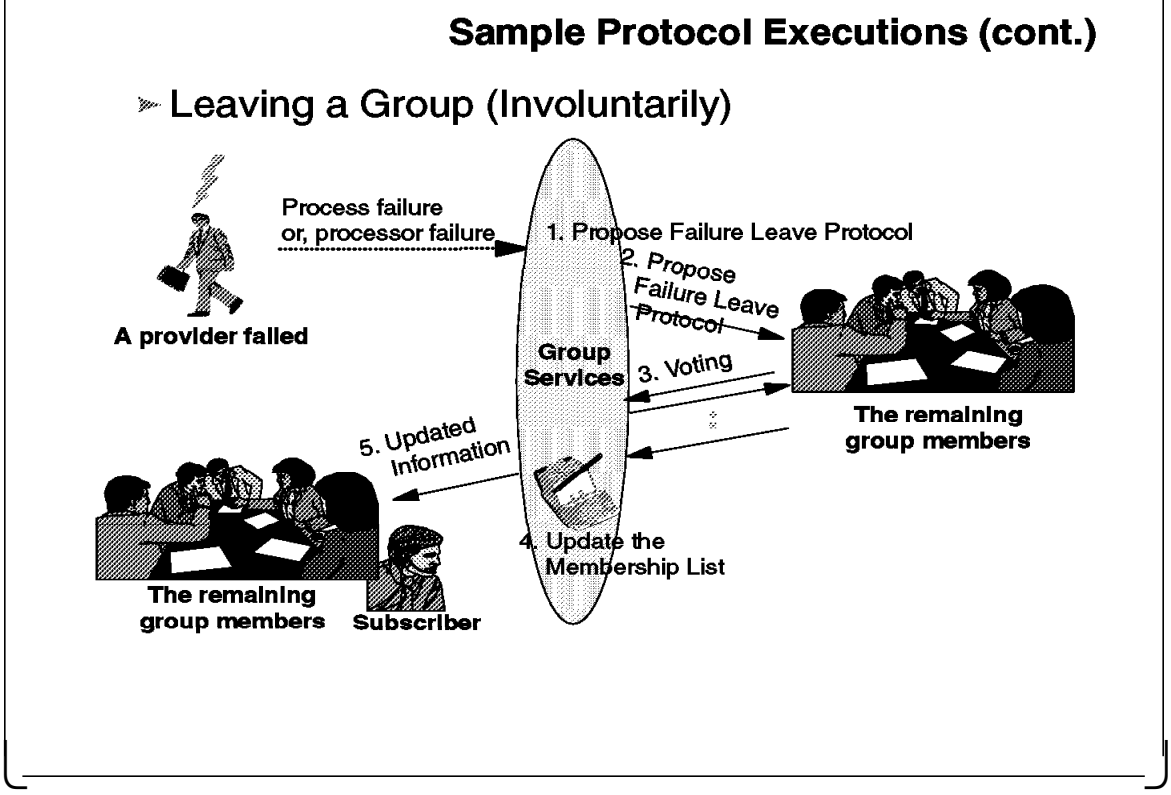


A typical scenario for a leave protocol is as follows:

1. A provider calls a *ha_gs_leave()* subroutine to propose a leave protocol.
2. The Group Services notifies all providers in the group that voting is requested for a state value change protocol, if the protocol is an n-phase protocol. If the protocol is a one-phase protocol, the proposal is automatically approved.
3. The voting process continues.
4. The Group Services subsystem updates the membership list in both cases where the protocol has been approved and rejected.
5. If a leave protocol is approved, all remaining providers receive the updated membership list, updated state value, or both. Subscribers receive the updated membership list, updated state value, or both, based on their subscription. The providers targeted by the protocol are sent a final notification informing them that they are out of the group. However, the Group Services subsystem does not verify that they receive it.

   If the protocol is rejected, it ends the execution of the protocol. However, the provider who proposed the leave is still removed from the group; that is, the membership list is updated to show the removal of the targeted providers. The providers and subscribers receive this updated list. The state value reverts to its value at the beginning of the protocol.

3.2.2.7 Sample Protocol Executions--Failure Leave



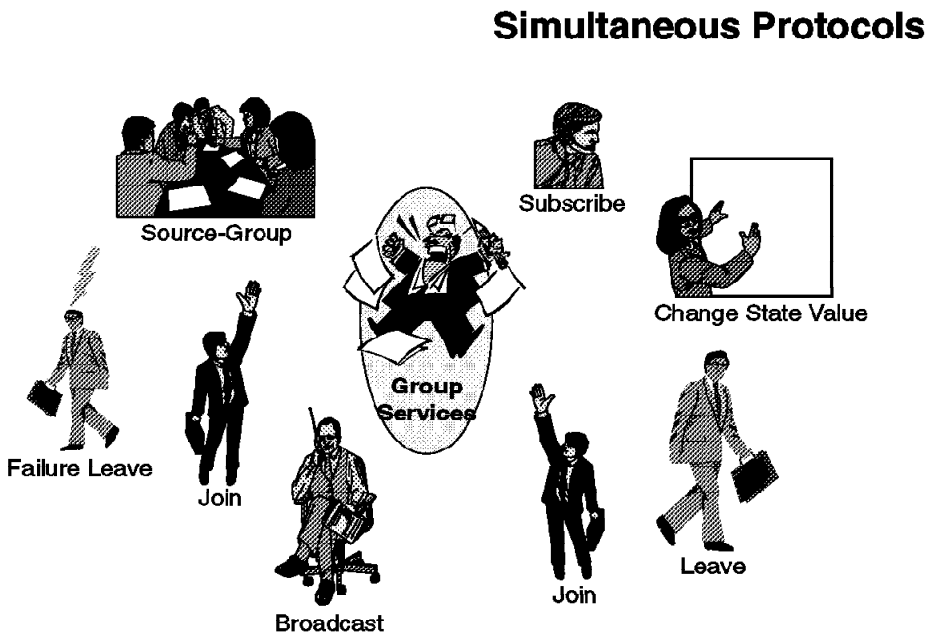
A typical scenario for a failure leave protocol is as follows:

1. A Group Services subsystem detects the providers failure by one of or both of the following events:
   - The loss of stream socket connection with the failed providers
   - The processor failure notification from Topology Services subsystem
2. The Group Services subsystem initiates a failure leave protocol.
3. When the Group Services subsystem initiates a failure leave protocol, it notifies the remaining providers of the protocol proposal.

   Multiple failing providers may be batched into a single failure leave protocol.
4. Once the Group Services subsystem has initiated the protocol, the providers execute it as described in 3.2.2, “Protocols” on page 45.
5. Once voting has completed and the protocol approved, the membership list is changed. If any provider *explicitly* votes to REJECT a failure leave proposal, the execution of the protocol stops. The membership list is updated to show the removal of the targeted providers. The state value reverts to its value at the beginning of the protocol.
6. All the providers receive an updated membership list, state value, or both. They may also receive a provider-broadcast message, if one was included in the final vote. Subscribers receive the updated membership list, state value, or both, depending on their subscription request.

If the protocol is rejected, the remaining providers and subscribers receive the updated list.

3.2.2.8 Simultaneous Protocols

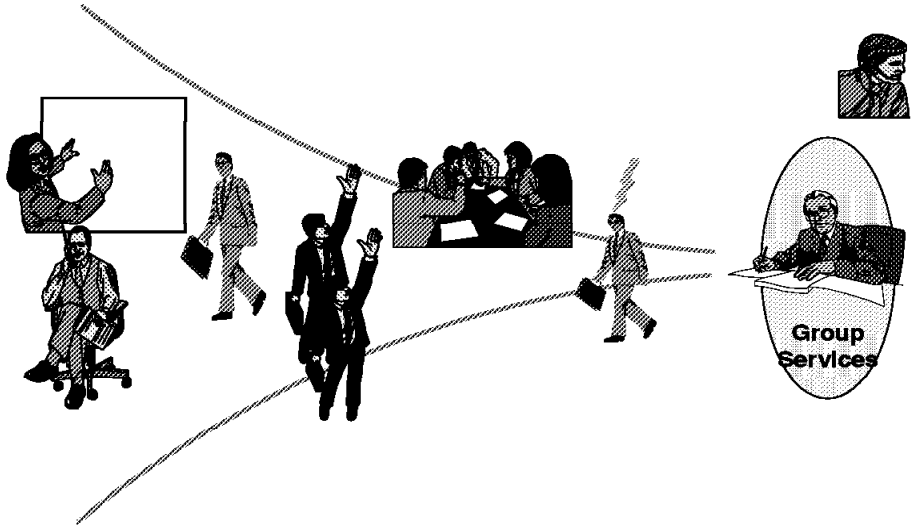


Because there are multiple providers (including the Group Services subsystem itself), there are multiple proposals, and proposals may arrive simultaneously. However, the Group Services subsystem does not execute more than one protocol at a time in one group, except in the following cases:

- The Group Services subsystem collects all of the joins during the lag time between the GSAPI subroutine call by a provider and the actual broadcast of any resultant notification for that subroutine, and issues a single notification.
- Similarly, the Group Services subsystem batches together multiple failure leaves or cast-outs into a single protocol.

Note: Group Services subsystem can handle multiple protocols independently if each of them belongs to a different group.

Simultaneous Protocols (cont.)



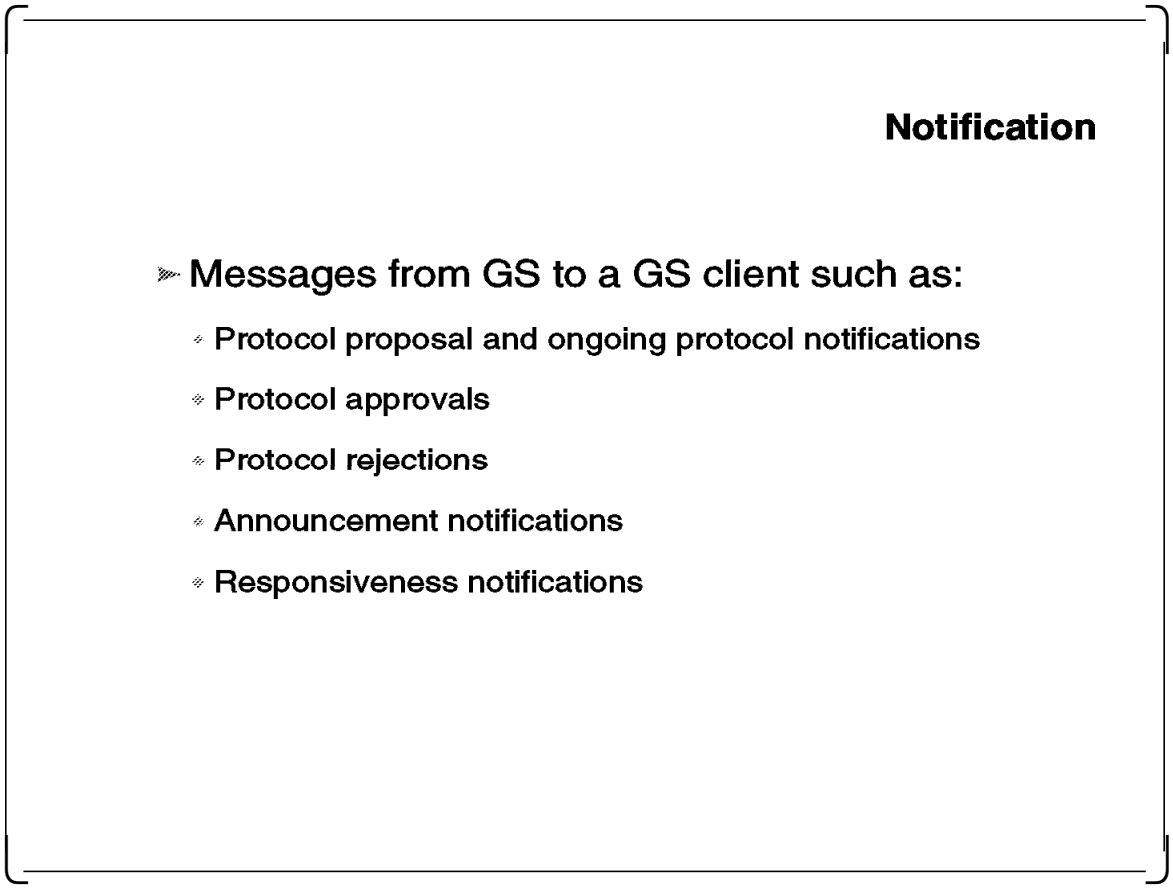
The Group Services subsystems handles simultaneous proposals as follows:

- In general, the first proposal to be made after an executing protocol completes is the one that is chosen to execute next. If multiple providers all attempt to submit proposals, the Group Services subsystem chooses one *arbitrarily*.
- For provider-initiated proposals, all proposals that are not chosen to be executed immediately are returned to the providers, with an asynchronous collision error code (HA_GS_COLLIDE).
- All the Group Services subsystem-initiated proposals remain pending until they have been executed within the group (see 3.2.5, “Active Protocol Proposal” on page 72). No provider-initiated proposals are accepted until all of the pending Group Services subsystem-initiated proposals have been executed. A provider that attempts to submit a proposal receives a synchronous or asynchronous collision error code.
- When choosing among multiple proposals, the Group Services subsystem chooses a proposal based on the following priority order:
  1. Failure leaves and cast-outs
  2. Source-state reflection
  3. Joins
  4. Leaves
  5. State value change and provider-broadcast message protocols

Within these categories, if there are multiple simultaneous proposals of the chosen type, the Group Services subsystem arbitrarily chooses one of them.

- No provider is allowed to cycle invisibly. If a provider should fail and then restart and try to join the group, the Group Services subsystem ensures that the leave of that provider is proposed before the subsequent join of that provider.
- A rejected provider-initiated protocol is not automatically resubmitted. The provider must resubmit the protocol, if it is required.

3.2.3 Notifications



The *notification* is a formatted message delivered to the client through the Group Services client library code (see the foil in 3.1.6, “Group Services Functional Flow” on page 34). These messages include the following types of notification:

- Protocol proposal and ongoing protocols

  These notifications are sent to the providers of a group to indicate that a multi-phase protocol has been proposed or is in progress (that is, n-phase protocol notifications).

  As a response to these notifications, the Group Services subsystem typically expects a vote. There are three types of proposals for which these notifications are sent:

  - Membership change protocol proposals
  - State value change protocol proposals
  - Provider_broadcast message protocol proposals

- Protocol approvals

  These notifications are sent to the providers of a group to indicate that a proposed membership or state value change has been approved. It is also sent to the subscribers of the group.

- Protocol rejections

  These notifications are sent to the providers of a group to indicate that a proposed membership or state value change has been rejected. Subscribers are not notified when proposals are rejected.

- Announcement

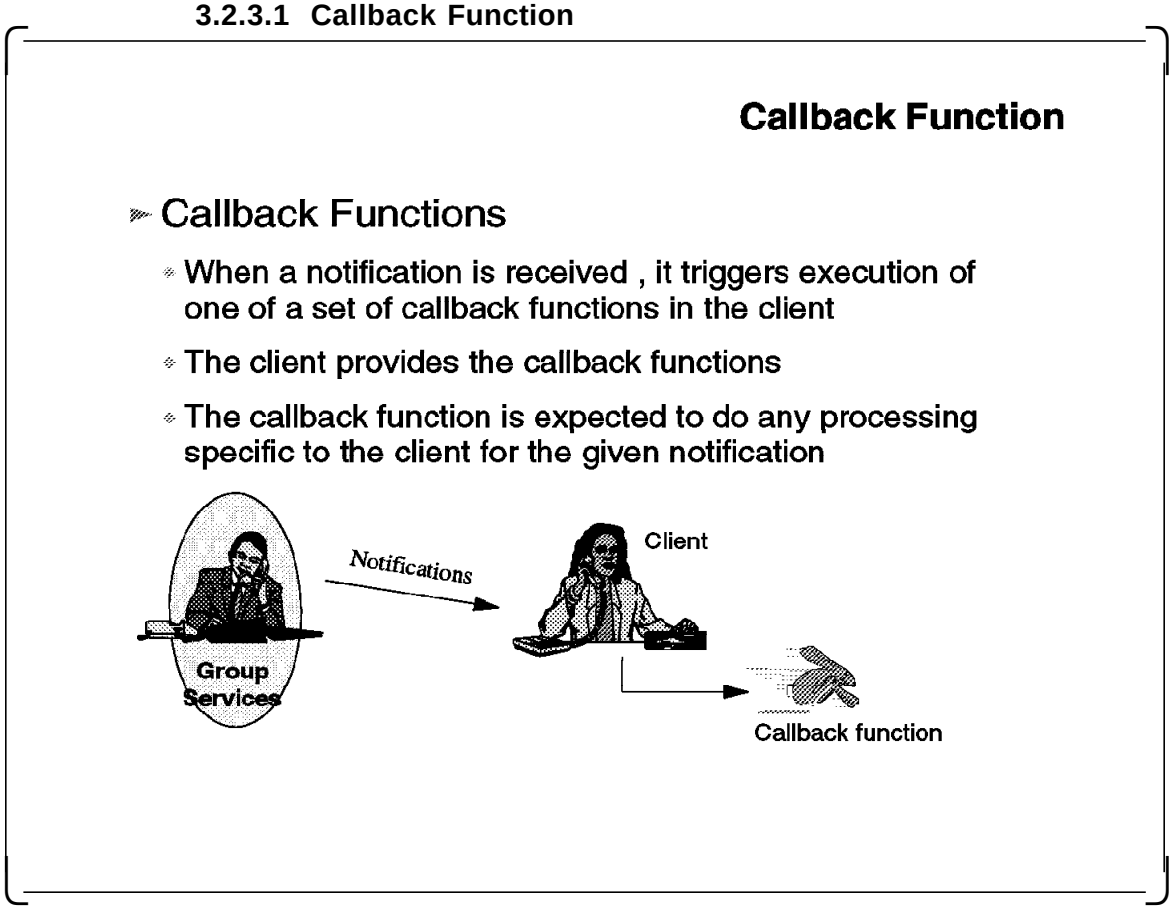
  These notifications are sent to the providers of a group to announce an item of interest within the group. They include warnings that the individual providers are unresponsive or the group is being dissolved.

- Responsiveness

  These notifications are sent to each of the providers of a group to determine whether the provider is active. A provider that does not respond to this responsiveness check within the time limit (the value of which is specified in the *ha_gs_init()* call) is considered to be unresponsive.

All messages are sent in a fault tolerant-manner; that is, providers and subscribers are guaranteed to receive notifications despite any single network failure.

3.2.3.1 Callback Function



Within the client, Group Services handles work by *callback functions*. Each notification, described in 3.2.3, “Notifications” on page 62, will result in one of the callbacks being executed to handle that notification. When the client library code has detected that it has received a notification message from Group Services, it reads the message and determines the type of the notification. It will then execute the callback function registered to handle that type of notification.

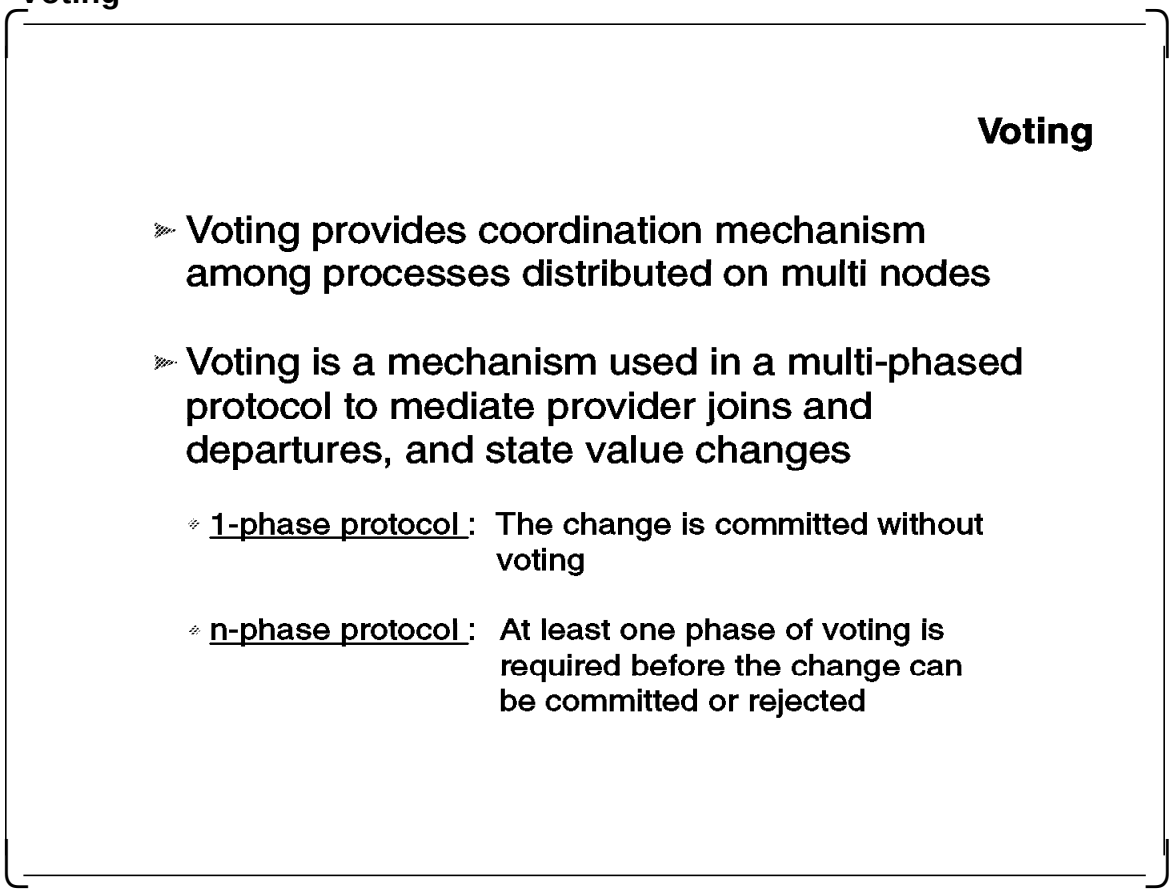
When a GS client initializes itself with a *ha_gs_init* subroutine call, it may specify a callback function that will handle responsiveness checks, and a callback function that will handle a delayed error delivered asynchronously from Group Services. The other callback functions must be registered at the time the client attempts to become a provider with a *ha_gs_join* subroutine call, or a subscriber with a *ha_gs_subscribe* subroutine call.

The callback functions that a GS client can define, and the subroutine calls used to register the callbacks, are listed in Table 1 on page 65.

Table 1. List of Callback Functions

Callback Function	Description	GSAPI Subroutine used to register
Announcement_callback	This callback executes when a group *announcement notification* is received. The announcements provide detailed information of abnormal conditions (other than complete failure) affecting one or more providers in the group.	ha_gs_init()
Delayed_err_callback	This callback executes when a *delayed error notification* arrives from the Group Services subsystem. Delayed errors are asynchronous errors that occur when a GS client has submitted a protocol proposal, and the Group Services subsystem later discovers a problem with the proposal.	ha_gs_init()
N-phase_callback	This callback is executed when an *n-phase protocol proposal notification* is delivered from Group Services. In response to this notification, it is expected that the provider will vote on the proposal by calling the ha_gs_vote() subroutine.	ha_gs_join()
Protocol_approved_callback	This callback is executed when a *protocol approved notification* is received. This notification is delivered after the n-phase protocol has been approved by voting and after all one-phase protocols (which are automatically approved).	ha_gs_join()
Protocol_rejected_callback	This callback is executed when a *protocol rejected notification* is received, which occurs after an n-phase protocol has been rejected by voting.	ha_gs_join()
Responsiveness_callback	The responsiveness callback routine is executed when a *responsiveness notification* is received from Group Services. This callback routine is called at intervals; therefore, in addition to responding to the Group Services subsystem, the GS client can also use this routine to perform validity checks on its own operation or its environment.	ha_gs_init()
Subscriber_callback	This callback is executed when *subscription notifications* are received. A subscription notification is delivered when a protocol is approved in a group to which the process is subscribed and the protocol modifies the group's membership or state value.	ha_gs_subscribe()

3.2.4 Voting

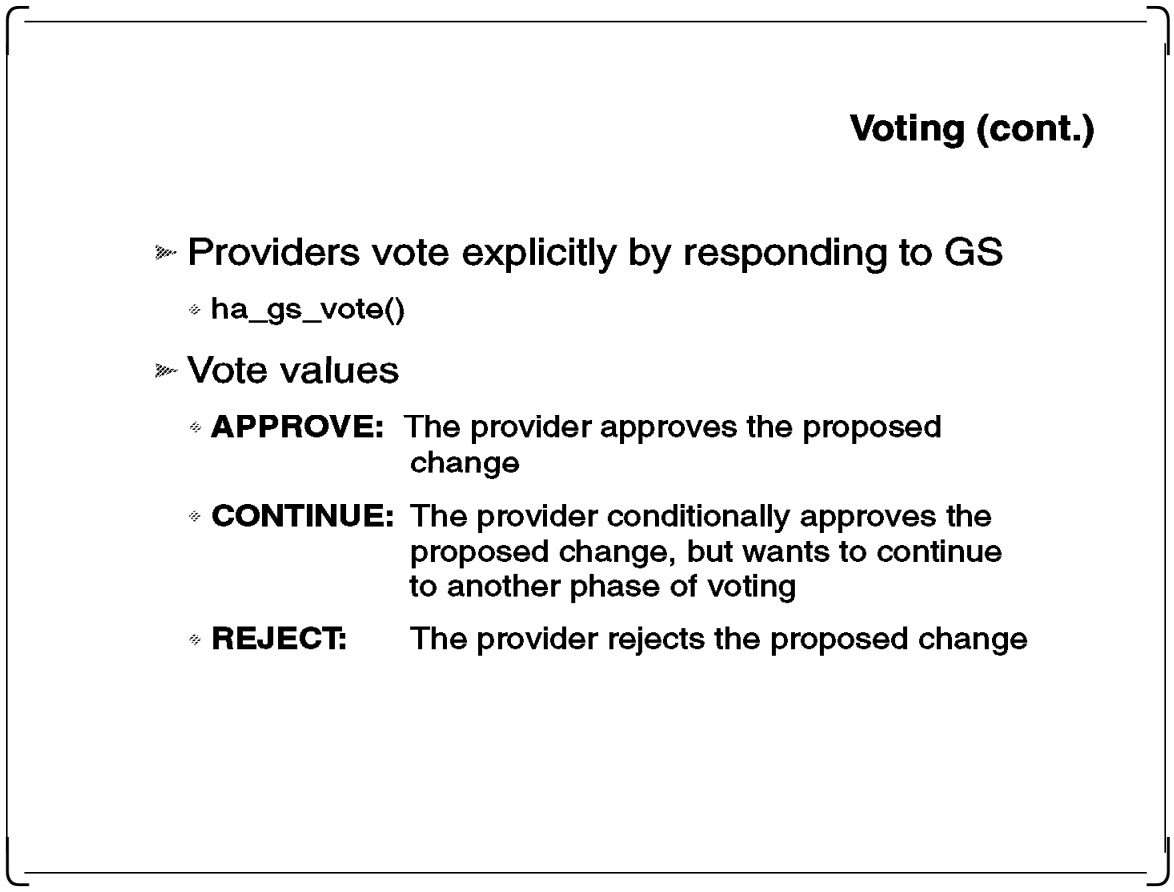


Voting is a mechanism to allow group providers to coordinate any actions requiring synchronization among them. For example, the voting phase allows each provider to run scripts, issue commands to manipulate resources, or display graphics on the screen.

Group Services uses a flexible n-phase voting protocol to mediate provider joins and departures and state value changes. Programmers can tailor their applications, which have different synchronization and coordination requirements, by choosing the number of phases needed to complete a membership or state changes:

- A *one-phase* protocol is the special case where no voting is allowed. Here, the proposed membership or state value change is automatically approved and committed, without voting.
- An *n-phase* protocol puts the group through at least one phase of voting before the change can be committed.

After the decision to approve or reject the protocol is made, Group Services notifies all of the participating providers of the outcome of the agreement (that is, whether the protocol was approved or rejected).



When a provider receives an n-phase protocol notification message, it is being asked to vote on the approval of the proposed change. The provider submits a vote with *ha_gs_vote()* call. Voting can occupy any number of phases, based on the desires of the providers.

For each phase, each provider must provide one of the following vote values:

- APPROVE

  The provider approves the proposed change. If all providers vote to Approve the proposal in the same voting phase, the protocol is approved and the protocol will be committed.

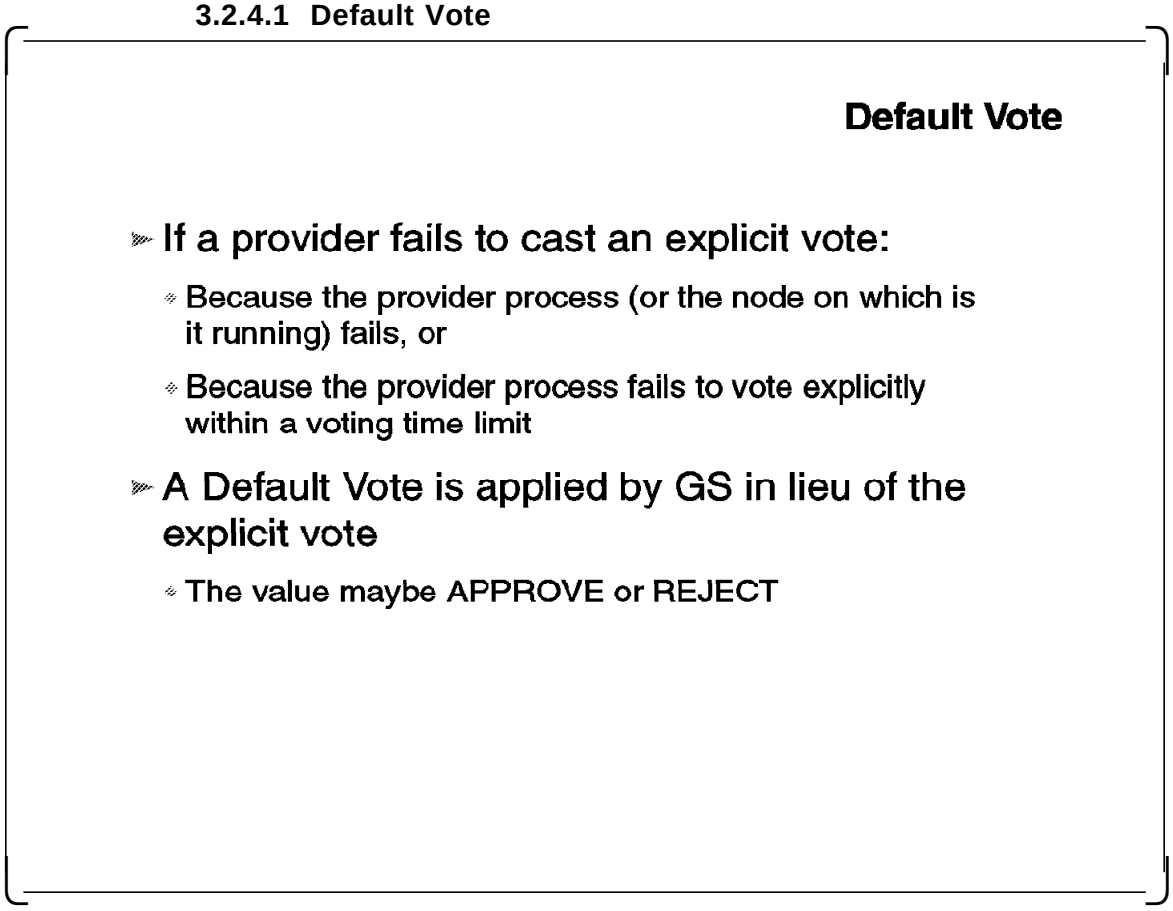
- CONTINUE

  The provider conditionally approves the proposed change, however it wants to continue to another phase of voting. If at least one provider chooses this value, the protocol proceeds to another voting phase.

- REJECT

  The provider rejects the proposed change. If at least one provider votes to reject the proposal, the protocol is rejected and ends, regardless of any other votes.

3.2.4.1 Default Vote



Providers normally vote *explicitly* by responding to Group Services with an *ha_gs_vote()* call. Default votes are made *implicitly* by Group Services as follows.

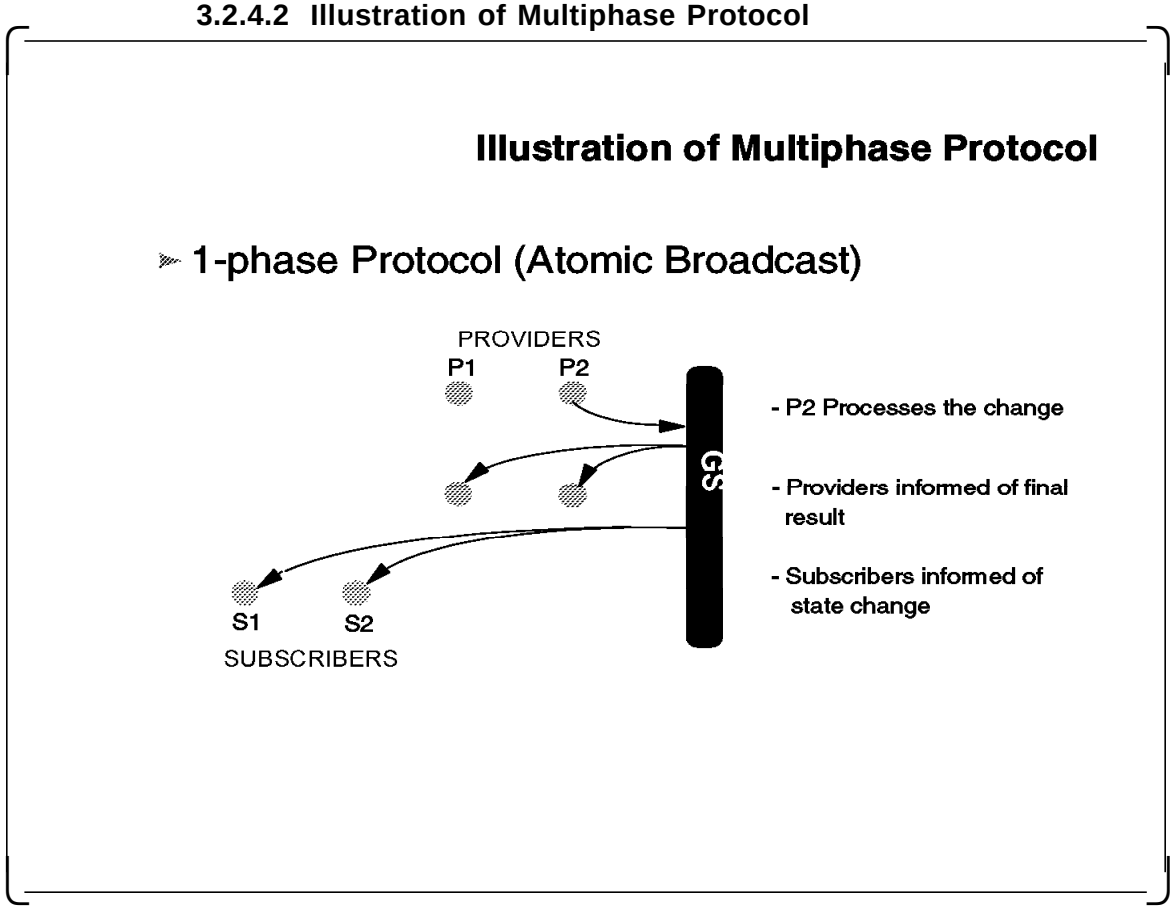
A provider may fail to cast an explicit vote for one of the following reasons:

- The provider process (or the node on which it is running) fails
- The provider process fails to vote explicitly within a group-specified *voting time limit*

In this case, default votes are applied as if they were specified by the providers, except in the case of failure leave protocol, in which case the membership list will be updated to show the removal of the targeted providers regardless of the default vote value.

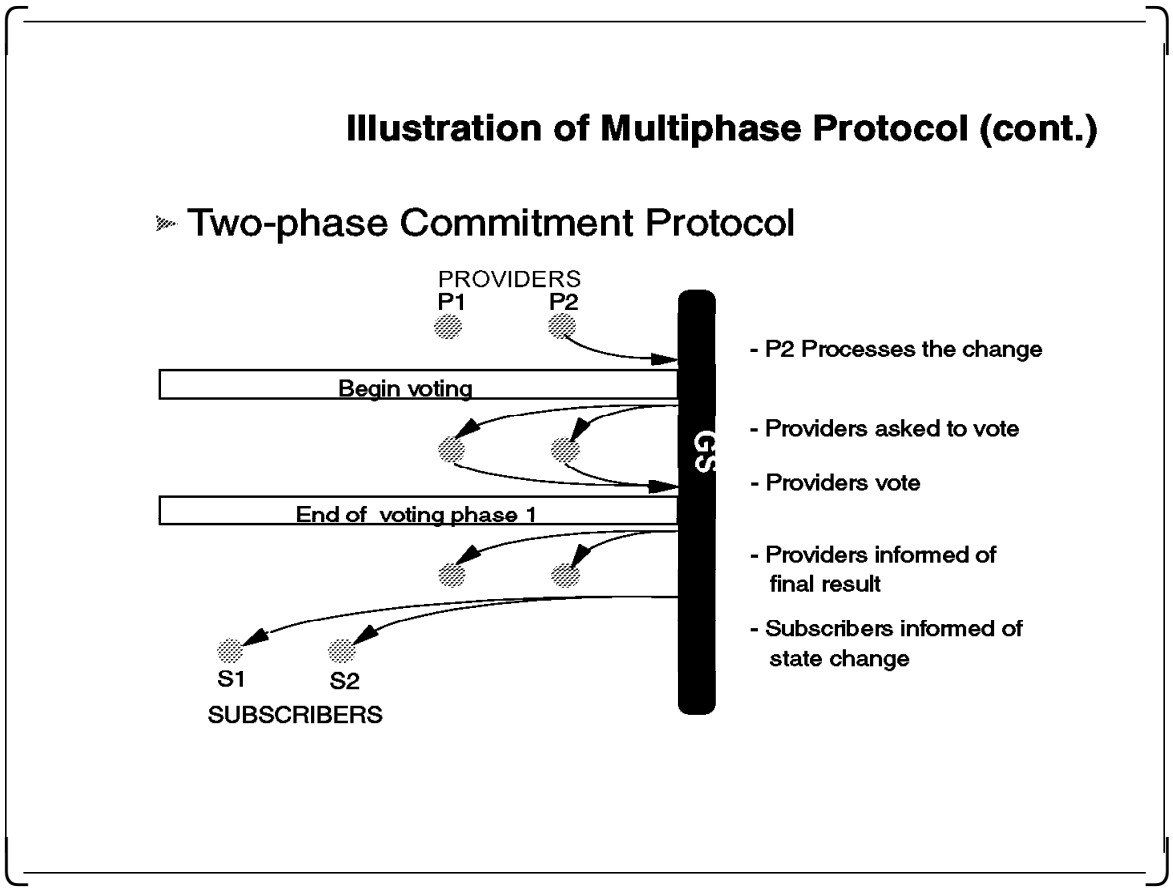
The default votes may be either APPROVE or REJECT.

3.2.4.2 Illustration of Multiphase Protocol

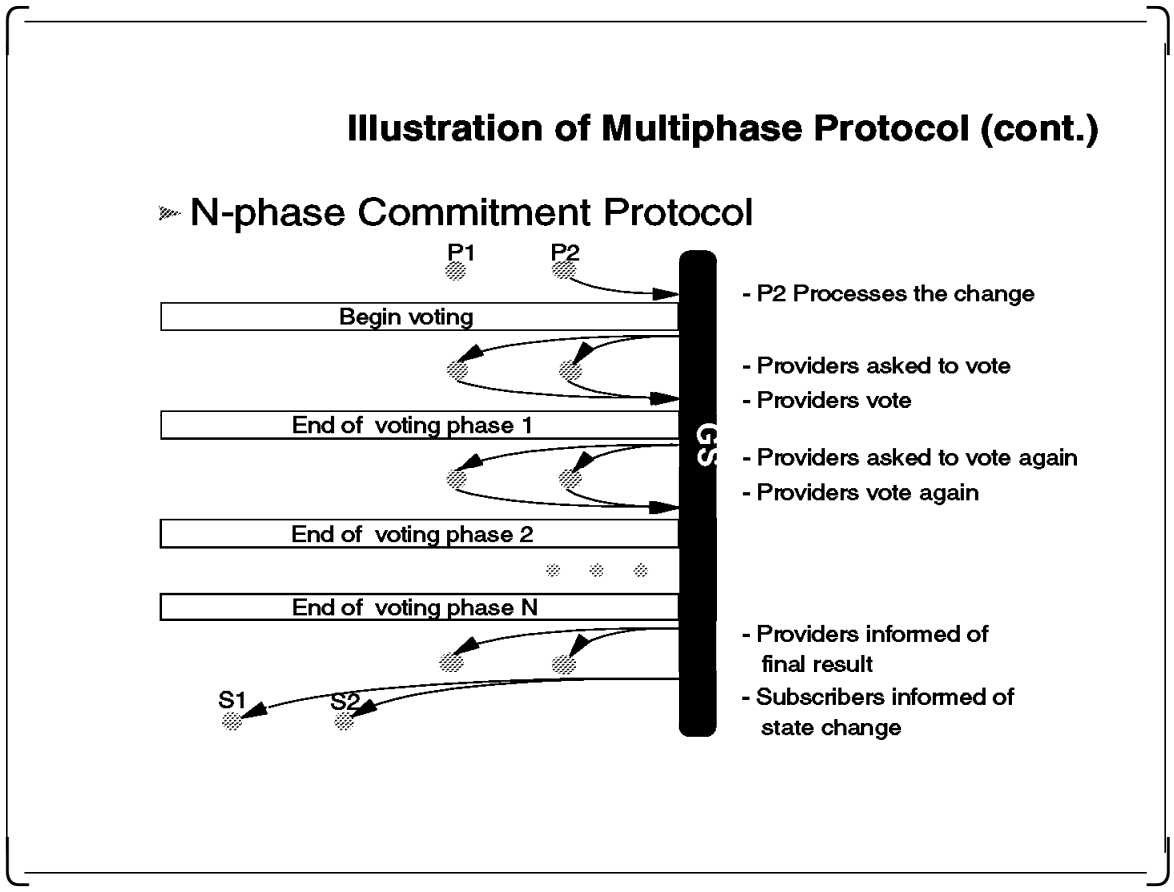


The next three foils illustrate a state change protocol for a group with two providers (P1 and P2) and two subscribers (S1 and S2). P2 proposes a change to the group's state value, and specifies whether the change requires voting phases or is handled as a single broadcast. Note that although Group Services is a cluster-wide distributed service, it is presented in the figure as a logically central service.

A one-phase protocol is invoked when a provider submits a state change requesting that there be no voting by the providers. The first phase of such a protocol is also the last phase of the protocol. A one-phase protocol can be used as reliable ordered broadcast; that is, one where all providers are guaranteed to receive that broadcast message, and all providers receive the broadcast messages in the *same order*.



A two-phase state change protocol is essentially the two-phase commit protocol with a reliable “coordinator.”

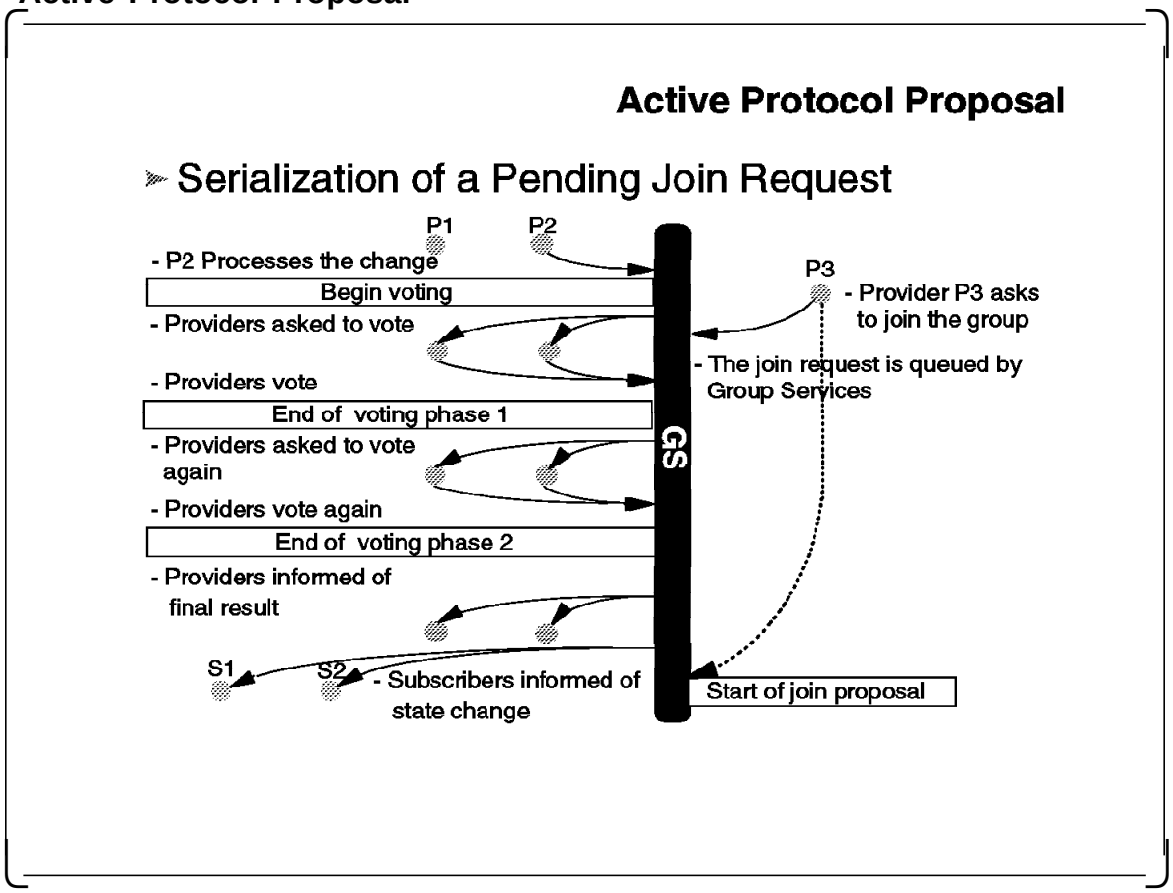


An n-phase state change protocol gives the providers the framework to perform n-1 rounds of barrier synchronization. For example, a 4-phase protocol yields three rounds of barrier synchronization: at the end of voting phases one, two, and three.

Attention

It is the responsibility of the providers in a group to determine the level of consistency that is required for managing changes to the group membership and state value. Programmers should decide the number of phases to be taken for each protocol according to the strictness of the synchronous consistency level required.

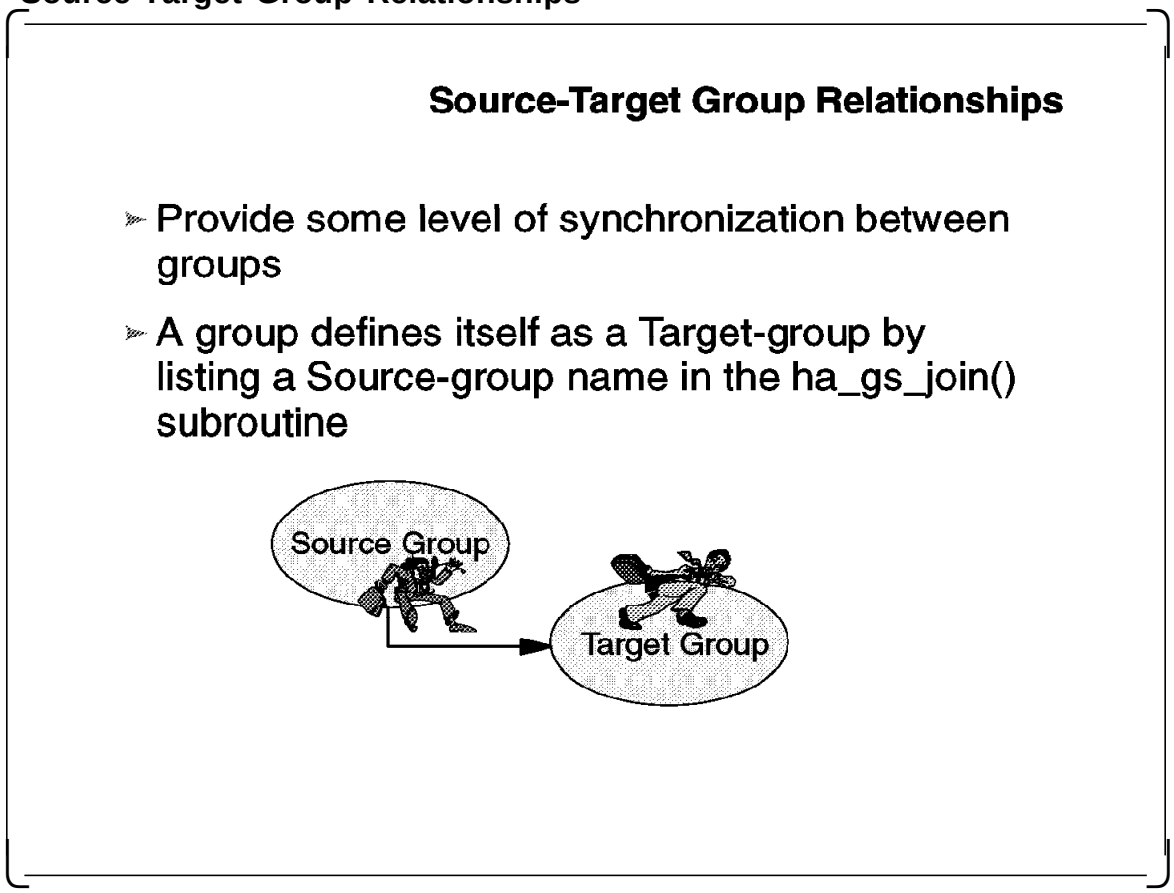
3.2.5 Active Protocol Proposal



Group Services guarantees that only *one* protocol affecting the group's membership or state value can be executed at any time. If more than one proposal is made within the group simultaneously, then the Group Services subsystem chooses one for execution, and returns the others to the providers that submitted them. It is a responsibility of providers to resubmit them for execution, if appropriate.

This processing is modified for providers joining or leaving the group. In these cases, the membership protocol to deal with the join or the leave is held until the currently running protocol has been approved or rejected, and then the membership change protocol is started immediately.

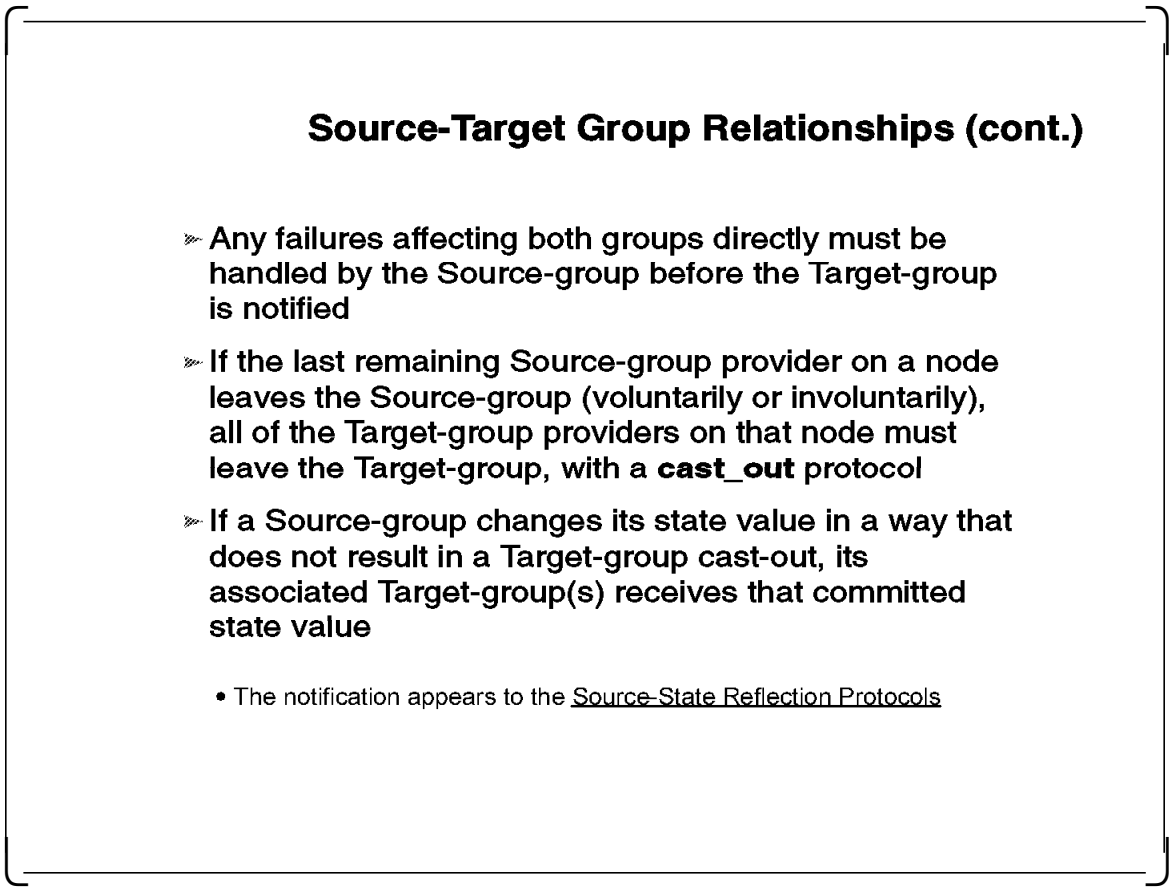
3.2.6 Source-Target Group Relationships



The Source-Target facility is intended to provide some level of synchronization between groups. Normally Group Services does not support inter-group services, beyond the subscription features. In some cases, however, two groups are interdependent to the point where one group must ensure that another group has handled certain situations in a serialized fashion.

If a node crashes, then all groups with providers on that node will be given a membership change proposal “simultaneously,” and this will cause each group to immediately begin reacting to this membership change. However, certain subsystems may not want to process the membership change *until after another group has completed processing this change*. Such a relationship exists, for instance, between the disk recovery subsystem and a distributed database application on the RS/6000 SP, where the database application must wait for the disk recovery subsystem to recover from a node failure before it can begin its recovery. The Source-Target facility can be used in such a case.

A group defines itself as a target-group by listing a source-group name in the set of group attributes specified on the *ha_gs_join()* subroutine by each target-group provider. A source-group is not notified that it has been “sourced” by any groups.



Group Services implement source-target relationships by applying to source-target groups a set of special behaviors of join and leave processing, as follows:

- For every node on which a target-group provider wants to run, there must exist a source-group provider.
- There may be multiple source-group and target-group providers on a node.

  If there is no source-group provider on a node, a potential target-group provider is not allowed to join the target group, and no membership change is proposed.

  - A source-group may have any number of target-groups.
  - A target-group may source only one group.
- If the last remaining source-group provider on a node leaves the source-group, whether voluntarily or involuntarily, then all target-group providers on that node must leave the target-group.

  When the source-group has approved the leave protocol of the last provider, a membership change is proposed to the target-group as a *cast_out* of the affected providers from the target-group. As a failure leave, the cast_out protocol cannot be rejected.
- If a target-group is running a protocol, and a source-group provider process fails on a node that also contains a target-group provider, the source-group runs a failure leave protocol.

  In this case, only the process of the source-group provider has failed, not the node on which it is running. However, once the source-group completes its

leave protocol, the target-group provider may no longer validly belong to the target-group.

Therefore, the Group Services subsystem considers the target-group providers that will be cast-out as having “failed” during the protocol, and treats them accordingly.

Whatever the outcome of the target-group′s running protocol, once it ends, the Group Services subsystem immediately initiates a cast-out protocol for the target-group.

- If a node fails instead of the last source-group provider on a node, then it is handled in the same way as if the source-group provider itself had failed, with the source-group completing its protocol before the target-group is notified. In this case, the target-group receives a cast-out protocol, rather than a failure leave.
- If a source-group changes its state value during protocols that do not result in a target-group cast-out (for example, through a state value change protocol or a voting response during any other n-phase protocol), its associated target-groups receive the committed state value.

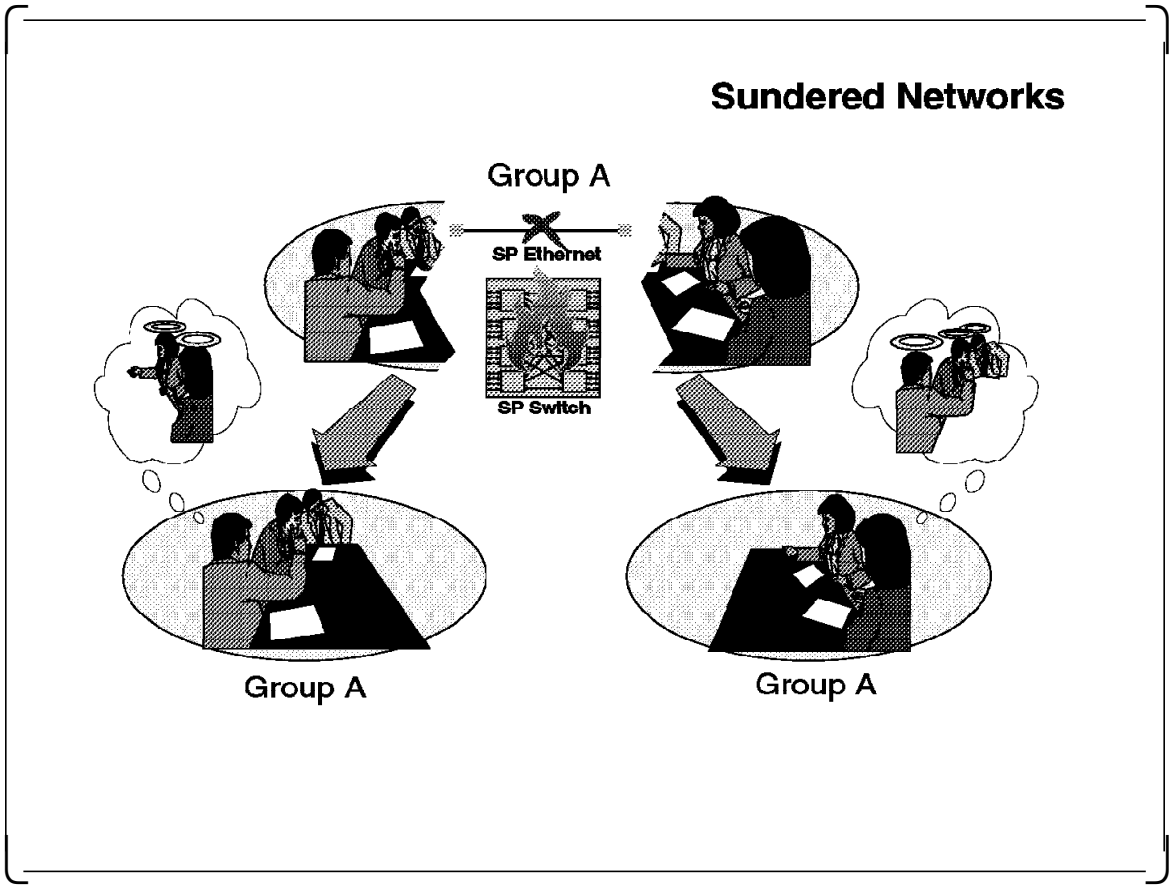
  The notification appears to the target-group as a *source-state reflection protocol*. The number of phases and a voting time limit are controlled by values specified in the group attributes.

  Since the source-state reflection protocol is initiated by Group Services, it is always initiated before any pending provider-initiated protocols for the group. In addition, there is no interface for a provider to request this protocol. It is automatically initiated as a consequence of a source-group′s state value change.

Attention

For a more detailed information about this topic, be sure to refer to *Group Services Programming Guide and Reference*, GC28-1675 before writing an application that will use source-target relationships.

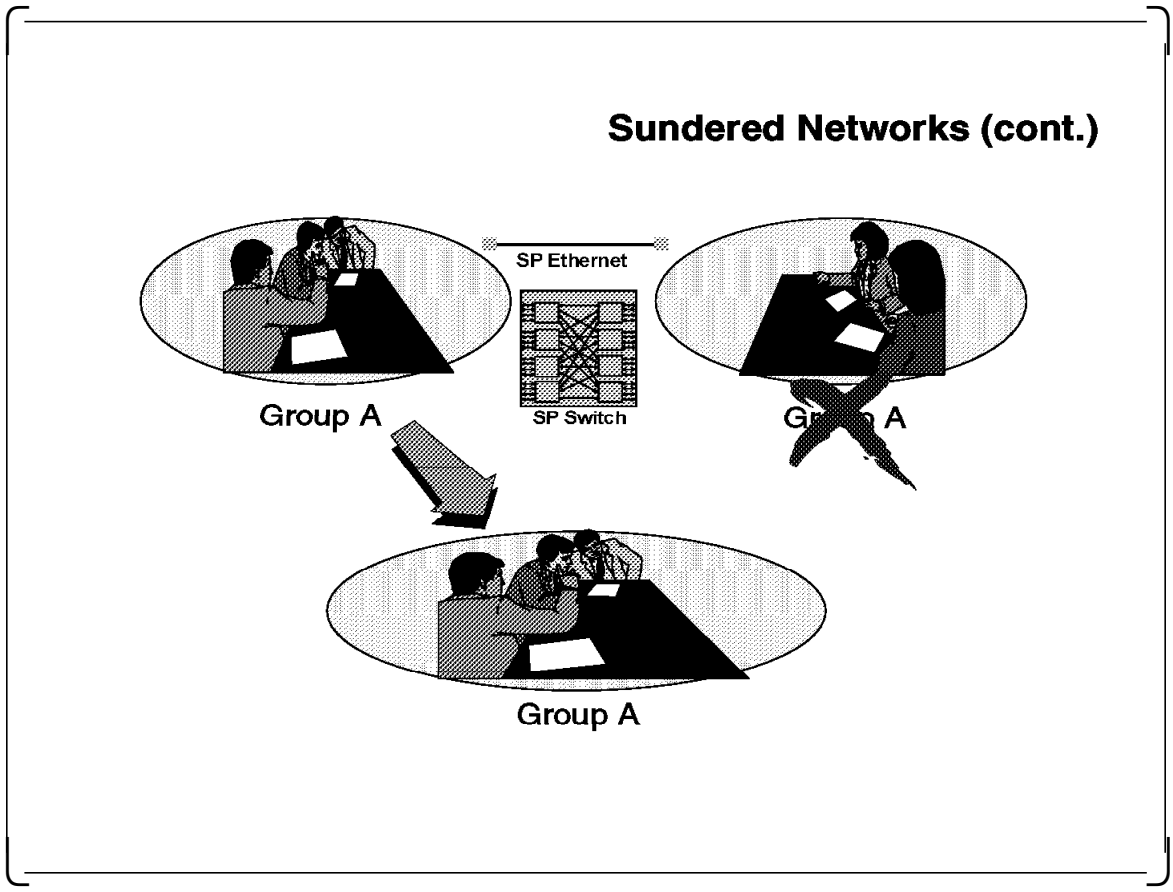
3.2.7 Sundered Networks



The Group Services subsystem provides a single group namespace within each system partition. It is possible for even an RS/6000 SP partition with multiple networks to become split, given the right set of multiple network failures. In this case of *sundered namespace*, the nodes become split in a way that they can no longer communicate with any nodes on the other side of the split. However, it is possible for each sundered portion to maintain enough information to reconstruct the groups that were in existence previously, at least those groups that still have members within any particular portion.

When a namespace is sundered, it is possible to get two instances of what should be one group. For example, in a sundered network, two nodes that own the two tails of a twin-tailed disk could end up on separate sides of the split. Since the processes of the subsystem coordinating the disk on each node would believe that the other process had disappeared, the process might want to activate its tail, which could lead to data corruption. As this example shows, it is important that each group determine if it needs a form of quorum, and use the quorum to guide the group when it is ready to perform its services.

Although the Group Services subsystem does not provide a quorum mechanism, it does provide some assistance to a group when a network is sundered. When a system partition is sundered, the providers receive membership protocol proposals from Group Services that all of the providers on the “other side” of the split have failed. The providers can then execute those protocols as they normally would, taking into account such factors as quorum to protect resources as necessary.

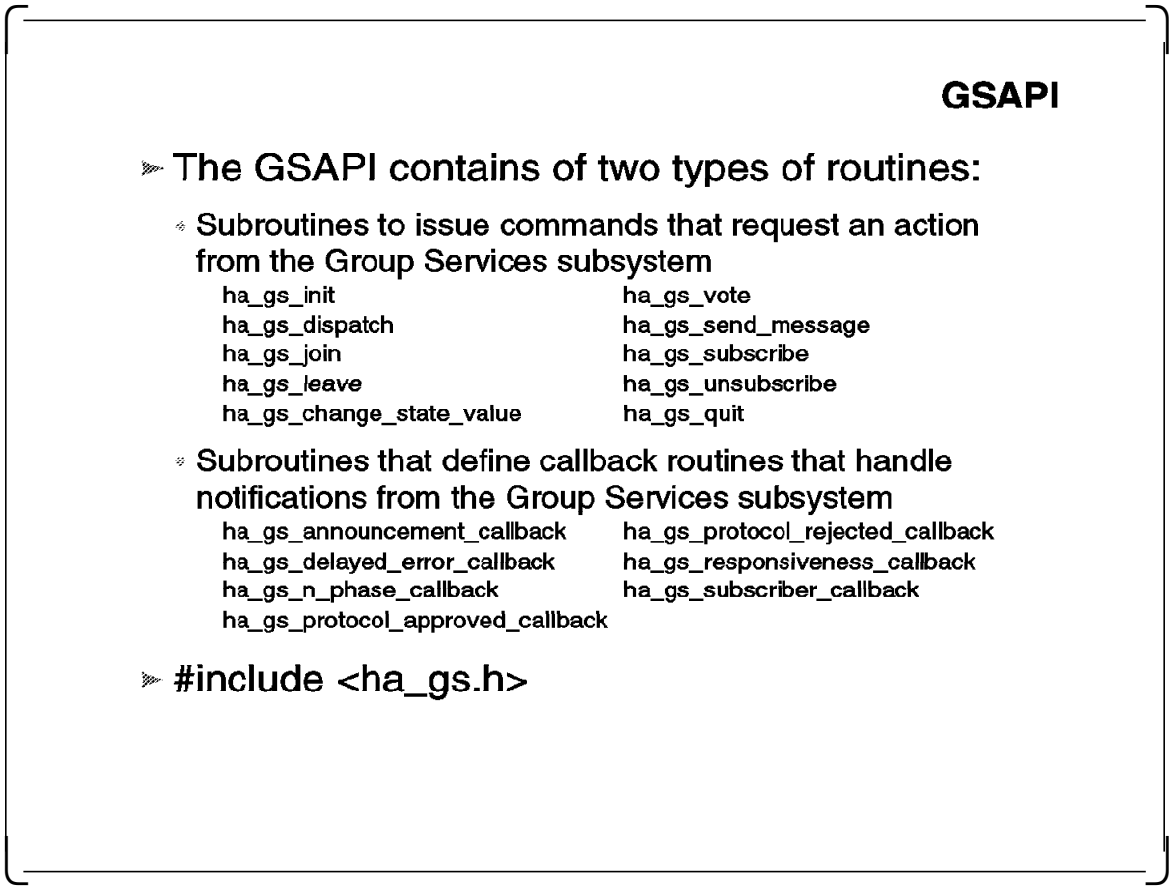


If an RS/6000 SP partition is sundered, it needs to merge again, once the errors that caused the split are repaired. In such a case, the following scenario takes place:

1. When two separate Group Services domains discover each other, then they will determine which domain has the most nodes.
2. The domain with the most nodes will “survive,” and will continue with no notifications.
3. The domain with the fewest nodes will force all clients connected to Group Services on nodes in the “smallest” domain to lose their connections to Group Services:
   - This will be done by closing the sockets connecting the clients to Group Services.
   - This forces those clients to reinitialize, and to rejoin (or resubscribe) to their groups.
4. When these clients reconnect, they will be brought into the fully reconnected domain.

3.3 Group Services Application Programming Interface (GSAPI)

3.3.1 GSAPI Routines



The Group Services Application Programming Interface (GSAPI) contains two types of subroutines:

1. Subroutines to issue commands that request an action from the Group Services (GS) subsystem, which are:

Subroutine	Action
ha_gs_init	Registers with Group Services.
ha_gs_dispatch	Checks for notifications.
ha_gs_join	Joins a group as a provider.
ha_gs_leave	Leaves a group (as a provider).
ha_gs_change_state_value	Proposes a change to the group′s state values.
ha_gs_vote	Votes on a proposed change to a group′s membership or state value by approving, rejecting, or continuing.
ha_gs_send_message	Sends data to all of the providers in the group.

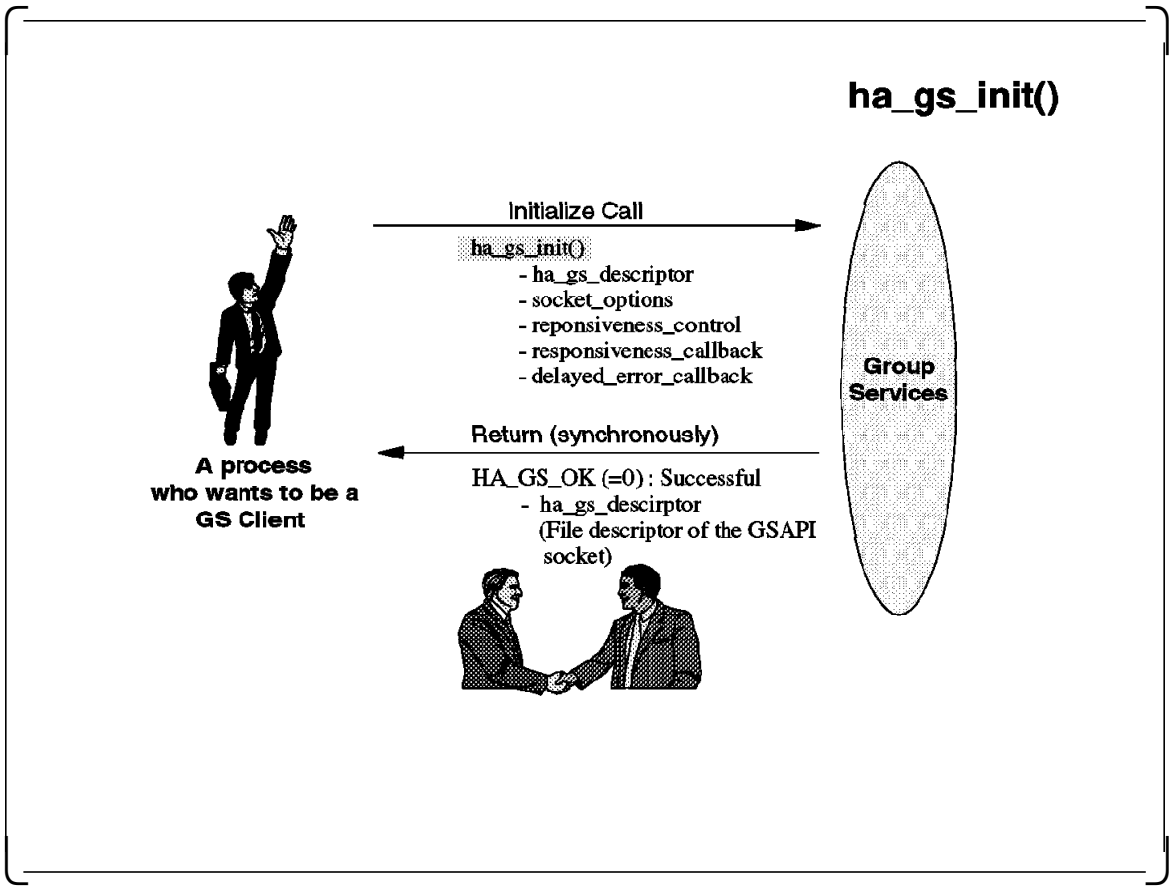
ha_gs_subscribe	Subscribes to a group.
ha_gs_unsubscribe	Unregisters as a subscriber to a group.
ha_gs_quit	Terminates the connection to the Group Services subsystem.

2. Subroutines that define callback routines that handle notifications from the Group Services subsystem, which are:

Subroutine	Response
ha_gs_announcement_callback	Responds to an announcement that: • One or more providers failed a responsiveness check. • One or more providers that previously failed responsiveness checks are now responding successfully. • The group is being dissolved. • The Group Services daemon has died or is about to die. • The voting time limit has expired.
ha_gs_delayed_error_callback	Handles an asynchronously presented error.
ha_gs_n_phase_callback	Responds to a request for a vote on a proposed join, leave, cast-out, failure leave, or group state change request.
ha_gs_protocol_approved_callback	Responds to a notification that a proposal has been approved.
ha_gs_protocol_rejected_callback	Responds to a notification that a proposal has been rejected.
ha_gs_responsiveness_callback	Responds to a responsiveness check.
ha_gs_subscriber_callback	Receives a notification that a subscribed-to group′s membership or state has been changed.

For the complete description of each GSAPI subroutine, refer to *Group Services Programming Guide and Reference*, GC28-1675. In addition, a sample Group Services client program called *sample_schg.c*, which is provided with PSSP Version 2 Release 2, would greatly help you to understand the GSAPI usage and to write your own Group Services client application. You can find the sample program source files in the */usr/lpp/ssp/samples/hags* directory after installing PSSP Version 2 Release 2.

3.3.2 ha_gs_init()



The next nine foils (including this page) illustrate the function and the usage of each GSAPI subroutine. Note that not all subroutines are covered.

The *ha_gs_init* subroutine is used by a process to register itself with GSAPI. The subroutine establishes a connection between the GSAPI and the process which has called this subroutine. This subroutine returns synchronously. After the successful completion of this call, the process will be a GS client which may become a provider or a subscriber to one or more groups.

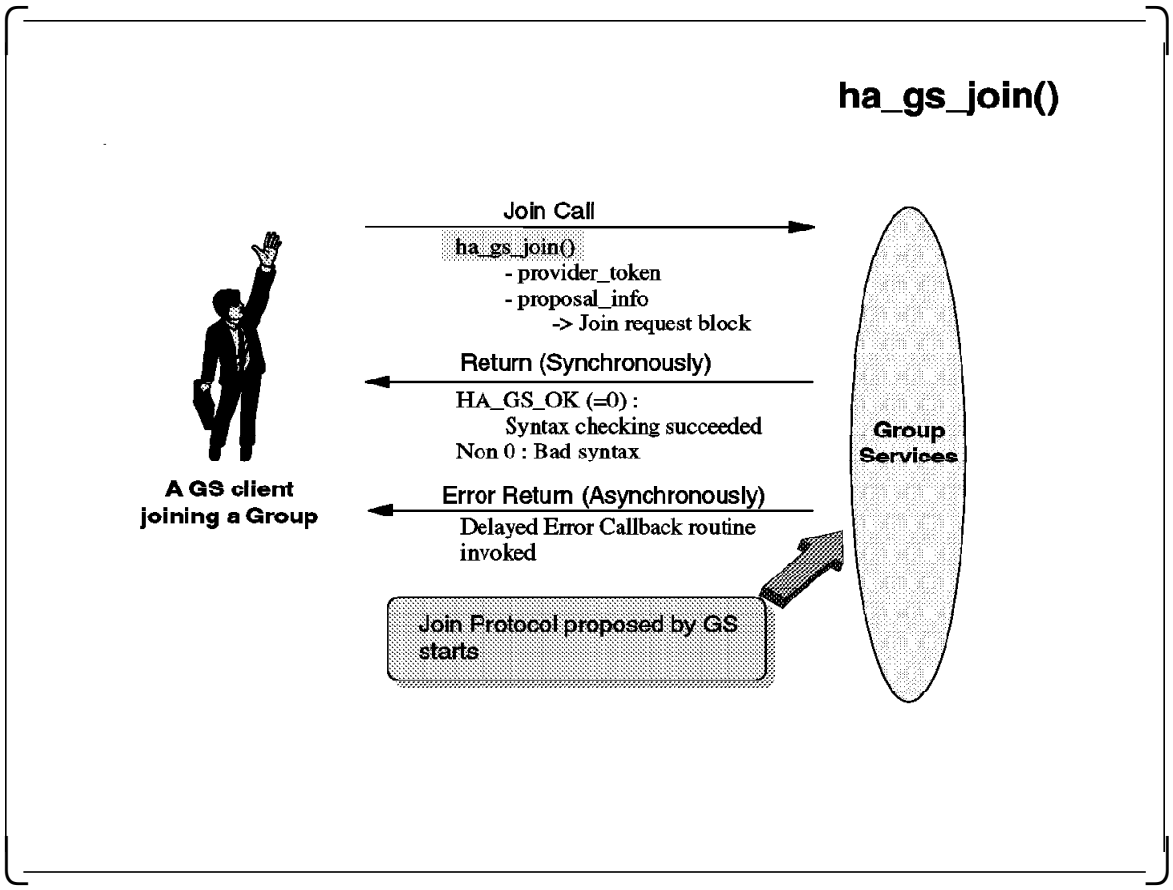
The common error case is **HA_GS_CONNECT_FAILED**, which means that the GS daemon is not yet accepting requests from GS clients. The clients should retry the init request call in this case.

Only processes with **root** authority are allowed to register themselves with the GSAPI.

The *responsiveness control* structure is used to specify whether the process wants the GSAPI to check it periodically for responsiveness and, if so, the protocol to be used. The GSAPI can always detect the actual exit (intentional or otherwise) of all GS clients. However, this check allows the GS client to perform any periodic validity checks on its own operation or environment that might be needed. If the GS client fails a responsiveness check and it is joined to any groups as a provider, the other providers in the groups receive an announcement that a provider has failed its responsiveness protocol.

The responsiveness protocol is executed only when the GS client is *idle* (*idle* means the client is not involved in a running protocol).

3.3.3 ha_gs_join()



The *ha_gs_join* subroutine is used by a GS client to join a group as a provider. If the named group does not already exist, it is created.

Information about the join request is supplied through the *join request block*. This structure contains the pointers to the *group attributes block*, which includes the attributes of the group discussed in 3.2.1.1, “Creating a Group” on page 38.

Upon receipt of the join request, Group Services checks the group attributes. If the named group already exists, it checks to see that the input group attributes match those that have already been established for the group. If they do not match, the **HA_GS_BAD_GROUP_ATTRIBUTE** error number is returned asynchronously by the delayed error callback routine.

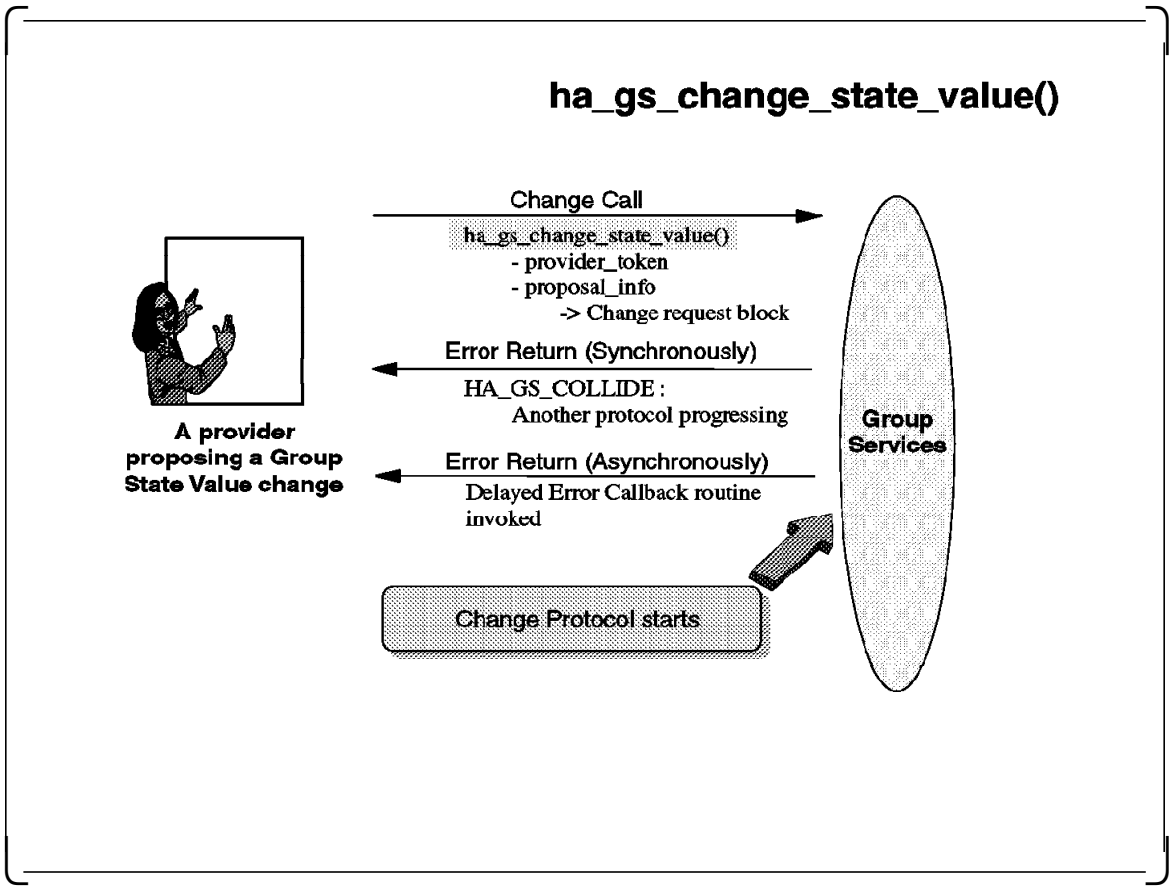
If the join request is for a new group, Group Services uses the attributes specified on the join request to establish the new group’s attributes.

If the asynchronous checks succeed, Group Services initiates a membership change protocol within the group to enable the provider to join. For the details of join protocol, see 3.2.2.3, “Sample Protocol Executions--Join” on page 51.

If this subroutine is successful, the *provider_token* field is set to the token that identifies this provider’s connection to the group. The GS client must pass a copy of this token onto all subsequent GSAPI calls that refer to the group. The

GSAPI, in turn, passes a copy of the token to the GS client onto all subsequent callbacks that refer to the group.

3.3.4 ha_gs_change_state_value()



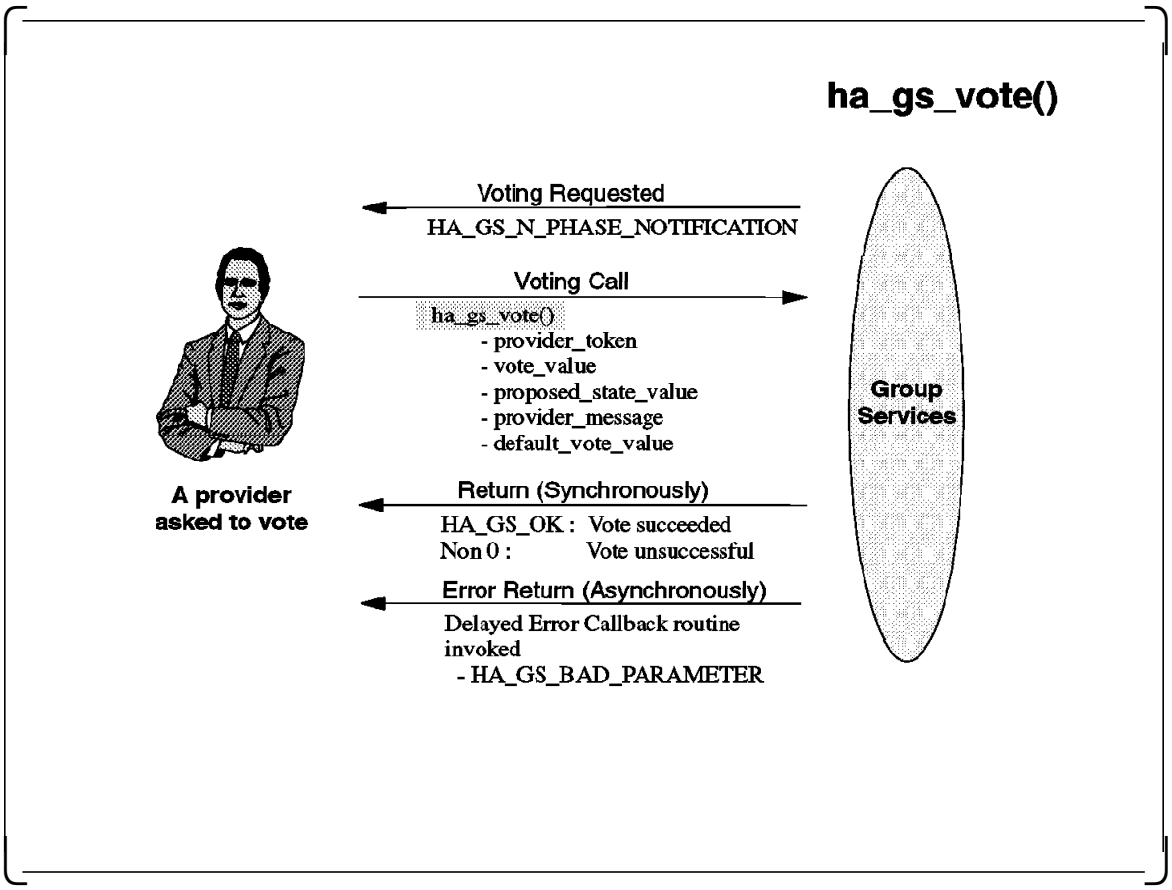
The *ha_gs_change_state_value* subroutine is used by a provider of a Group Services group to propose a change to the group's state value.

If another protocol is already in progress, the **HA_GS_COLLIDE** error number is returned when the error is detected. Otherwise, the proposal will initiate a protocol within the group. For details of a change state value protocol, see 3.2.2.4, "Sample Protocol Executions--Change State Value" on page 53.

Information about the state change request is supplied through the *state change request block*. This structure includes the following fields:

- **gs_num_phases** field specifies whether the state change protocols are to be n-phase or one-phase protocols.
- **gs_time_limit** field contains the voting phase time limit, in seconds. If "0" is specified, no time limit is enforced.
- **gs_new_state** field points to a buffer that contains the proposed new value for the group's state.

3.3.5 ha_gs_vote()



The *ha_gs_vote* subroutine is used by a provider of a Group Services group to submit its vote on a proposal during a voting phase of an executing protocol.

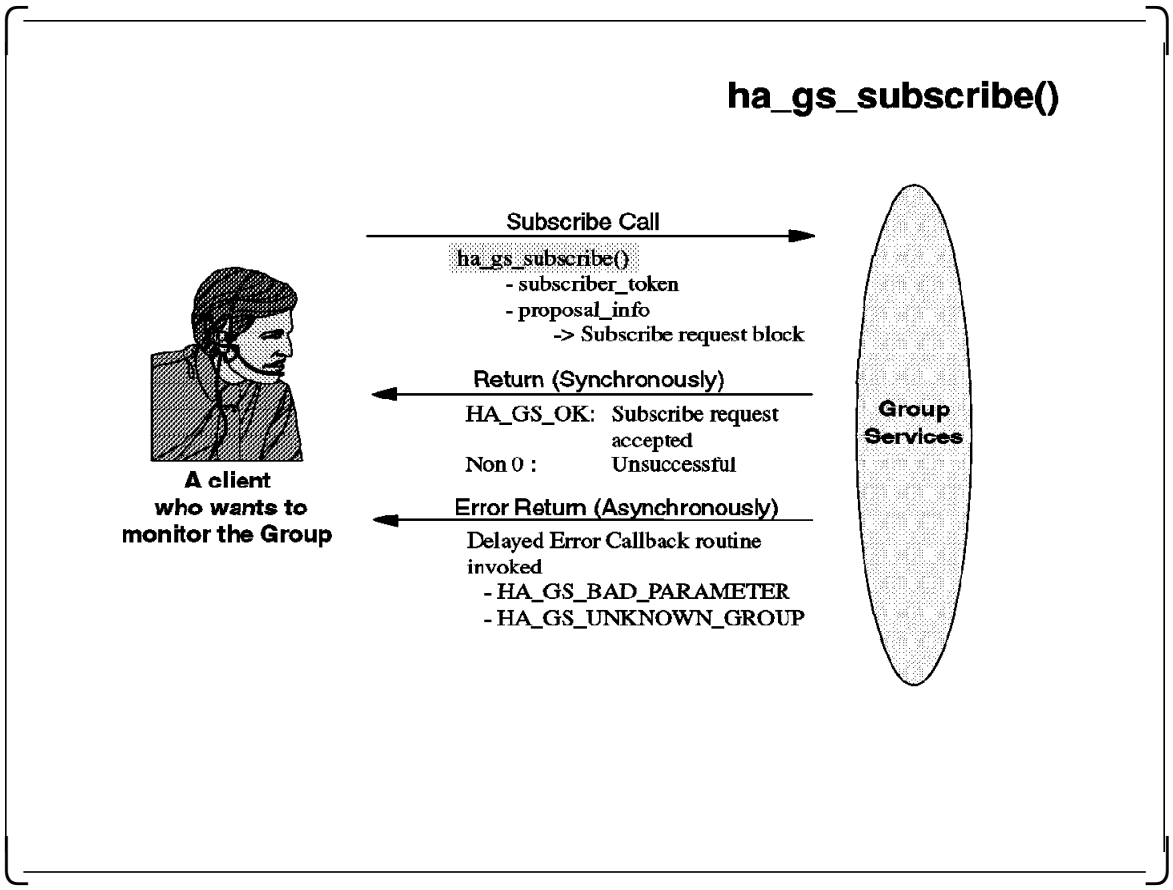
When an application has selected an n-phase protocol, providers are expected to vote on proposed changes to the group. When a vote is requested, the n-phase callback routine is called for each of the providers, and each of the providers is expected to return a vote using this subroutine within the time limit (if a time limit was specified) previously established by the group.

The value of vote_value can take APPROVE, CONTINUE, or REJECT.

If multiple providers, in the same voting phase, submit state value changes, provider messages, or both, the Group Services subsystem chooses *only one of each*. Therefore, if different providers submit different values, the Group Services subsystem will arbitrary choose the values. Due to this, a group should use one of the following two approaches to get consistent submissions of state value changes, provider messages, or both:

- All providers submit same value.
- Only one provider submits values.

3.3.6 ha_gs_subscribe()



The *ha_gs_subscribe* subroutine is used by a GS client to register as a subscriber for a Group Services group. If the named group does not already exist, the **HA_GS_UNKNOWN_GROUP** error number is returned asynchronously.

Note that subscribers are known only to the Group Services subsystem. The providers of the group and the other subscribers of the group are unaware of any of the subscribers to the group.

If this subroutine is successful, the *subscriber_token* field is set to the token that identifies this subscriber's connection to the group. The GS client must pass a copy of this token on all subsequent GSAPI calls that refer to the group. The GSAPI, in turn, passes a copy of the token to the GS client on all subsequent callbacks that refer to the group.

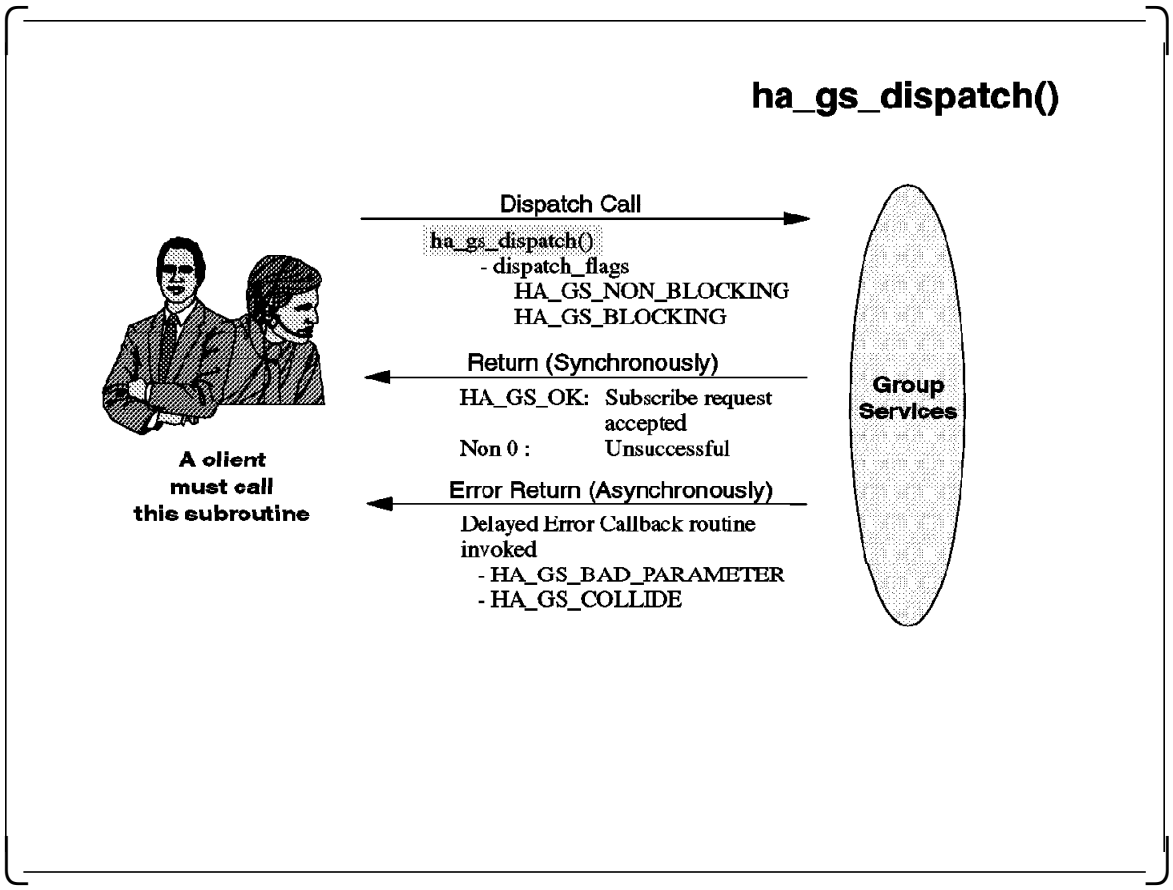
To receive notifications for changes in the list of active nodes or adapters in the system, the subscriber should specify one of the following constants as the group name in the **gs_subscription_group** field in the *subscribe request block*:

- **HA_GS_HOST_MEMBERSHIP_GROUP**, for subscriptions to the host membership group
- **HA_GS_ENET_MEMBERSHIP_GROUP**, for subscriptions to the Ethernet adapter membership group

- **HA_GS_CSS_MEMBERSHIP_GROUP**, for subscriptions to the SP Switch adapter membership group

Notifications for host or adapter membership look the same as notifications for any other group; each active node or adapter is represented as a provider.

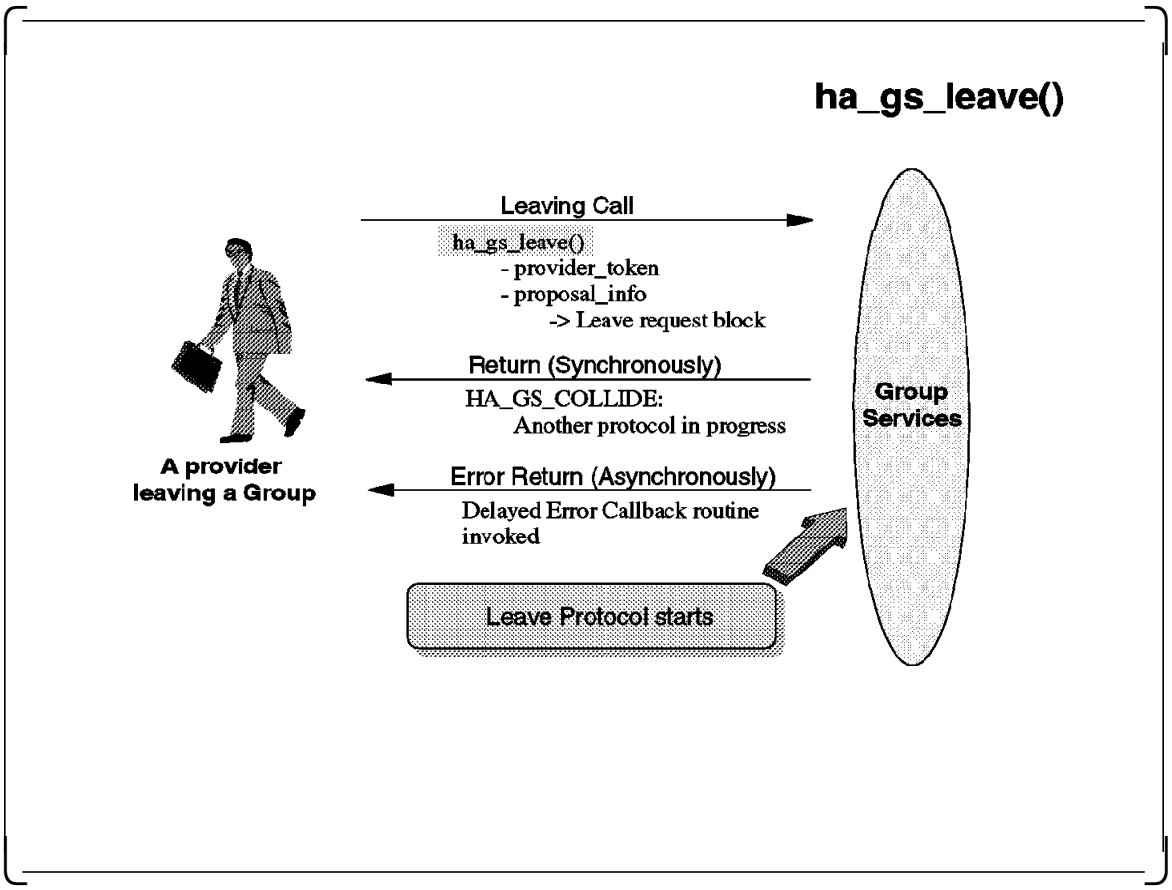
3.3.7 ha_gs_dispatch()



The *ha_gs_dispatch* subroutine is used by a process to handle messages (also known as *notifications*) from GSAPI. The flags provided on input determine whether the GSAPI performs **select** on the GSAPI socket and whether the GSAPI blocks the socket if there are no messages to be read.

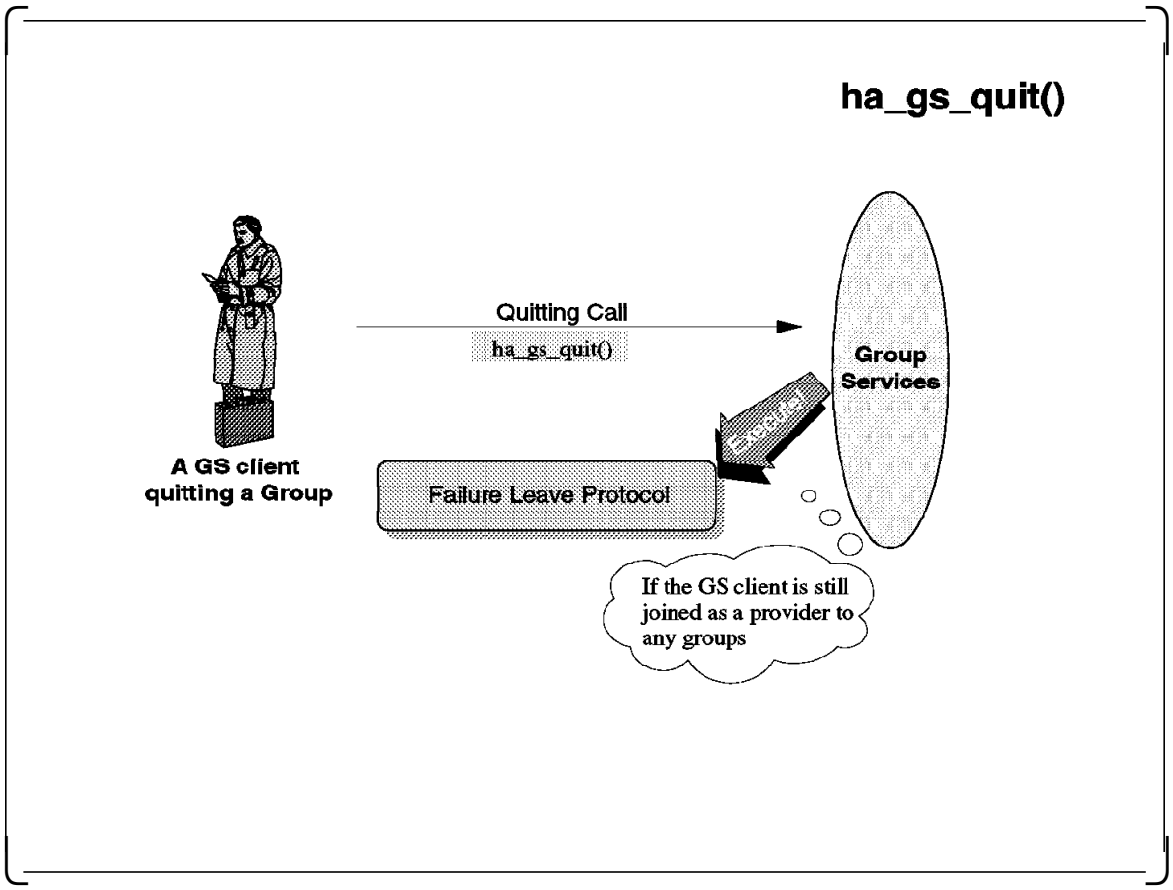
The GS client must call this subroutine regularly to communicate with the GSAPI.

3.3.8 ha_gs_leave()



The *ha_gs_leave* subroutine is used by a provider of a Group Services group to leave the group.

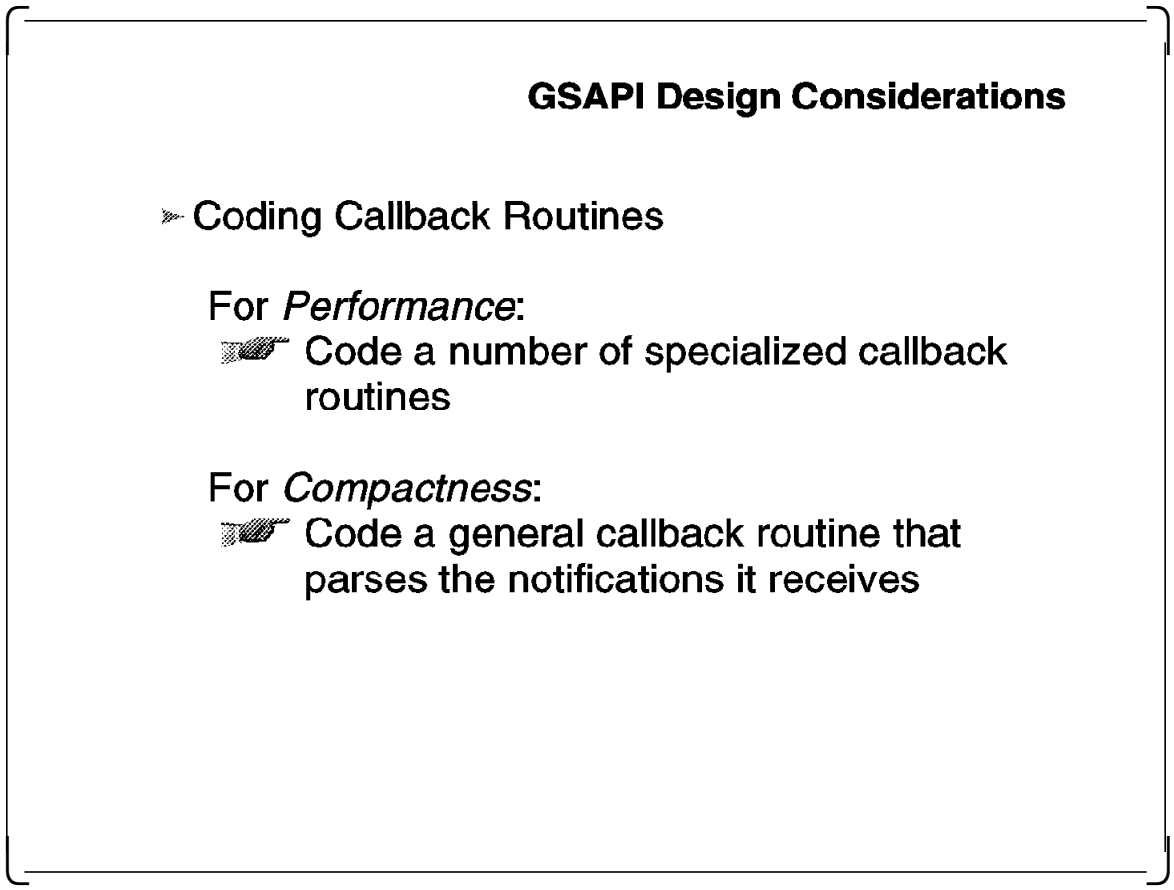
3.3.9 ha_gs_quit()



When a GS client no longer needs to use the Group Services subsystem, it should call the *ha_gs_quit* subroutine to terminate its connection to the Group Services subsystem. This allows Group Services to release the resources associated with the GS client.

If the GS client is still joined as a provider to any groups, the Group Services subsystem notifies the groups that the provider has failed, and the groups execute a failure leave protocol.

3.3.10 GSAPI Design Considerations



The GSAPI provides a number of separate callback routines, each of which expects to receive a different type of notification. However, each notification block, whose pointer each callback routine receives from each notification, also specifies its type. For example, the *n-phase notification block*, which is passed from n-phase notification to the n-phase callback routine, has the *ha_gs_notification_type* field, which is expected to contain a value of HA_GS_N_PHASE_NOTIFICATION.

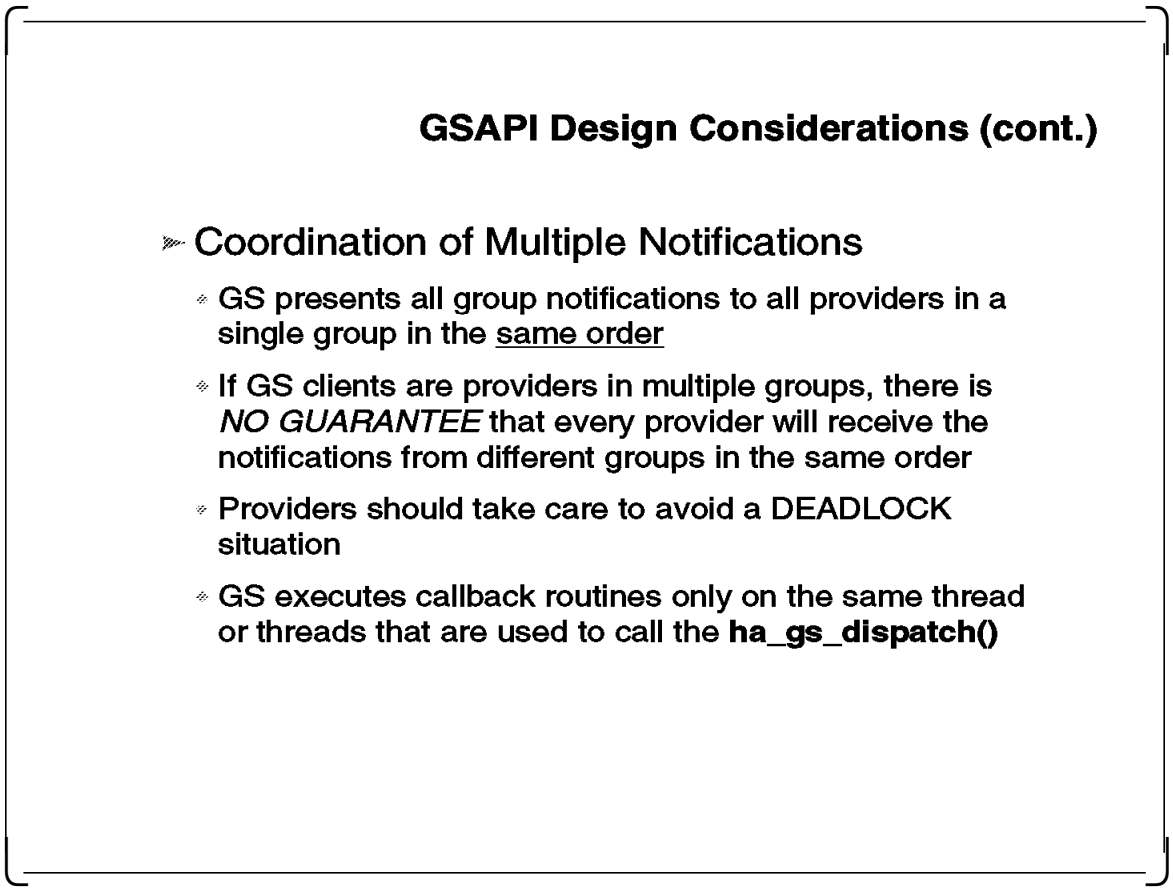
This design allows you to code callback routines using either of the following two strategies, or a combination of the two:

1. Code a number of specialized callback routines.

   This reduces the amount of checking each callback routine must perform when it receives a notification. You could use this approach if performance and path length are considerations when your application handles a notification.

2. Code a general callback routine that parses the notifications it receives.

   This reduces the number of callback routines you need to code, but increases the amount of work each must do to determine the type of notification it has received.



The following discussion of multiprocessing considerations applies to all callback routine designs:

- The Group Services subsystem presents all notifications to all providers in a single group in the same order (but *not* at the same time). The providers should try to execute the same callback routines in the same order.

  However, only for n-phase protocols does the Group Services subsystem verify that all of the group's providers have reached the same execution point before continuing to the next notification (as discussed in 3.2.4.2, "Illustration of Multiphase Protocol" on page 69). In other cases, the providers may not receive and react to the notifications at the same time.

- If GS clients are providers in multiple groups, there is no guarantee that every provider will receive the notifications from different groups in the same order.

- For multi-threaded clients, it is assumed that the callback routines are thread-safe and reentrant (if the same callback routines are specified for multiple groups), so that the client can process notifications by executing the callback routines for more than one group at a time.

  For single-threaded providers, if they are acting as providers for multiple groups, they must also be coded to handle simultaneously executing protocols in all groups.

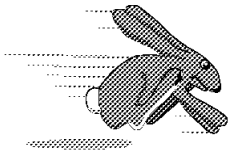
- In all cases where GS clients are acting as providers in multiple groups, it is the responsibility of the providers to ensure that they do not create deadlock situations across groups. An example of a deadlock that could occur is when one provider blocks before voting, waiting for another provider to take

some action, and the second provider is blocked on another group protocol, waiting for the first provider to take some action.

- The Group Services subsystem executes callback routines only on the same thread or threads that are used to call the *ha_gs_dispatch* subroutine. The only exception is the case of a group dissolution notification, for which the announcement callback routine is called on any thread that is available.

GSAPI Design Considerations (cont.)

- Performance Considerations
  - Minimize the number of groups, providers, and nodes
  - Minimize the size of the provider-broadcast messages
  - Select the batching option for join requests
  - The actions taken during the barriers in an n-phase protocol should be idempotent



To get the best performance from your application, design your application according to the following guidelines:

- Minimize the number of groups, providers, and nodes your application requires.

  The greater the number of groups, the number of providers joined to those groups, and the number of nodes across which each group is spread, the longer it will take to coordinate your application's activities.

- Minimize the size of the provider-broadcast messages that your application uses.

  The larger the messages, the greater the load on the network, particularly when a message must be broadcast to every provider in a group and a large number of subscribers as well.

- Select the batching option to allow the Group Services subsystem to batch multiple join requests and failure leaves together (see 3.2.2.3, "Sample Protocol Executions--Join" on page 51 and 3.2.2.6, "Sample Protocol Executions--Leave" on page 56).

- The actions taken during the barrier synchronizations that are imposed by an n-phase protocol should be *idempotent*, that is, they should be designed so that they can execute one or more times without damage.

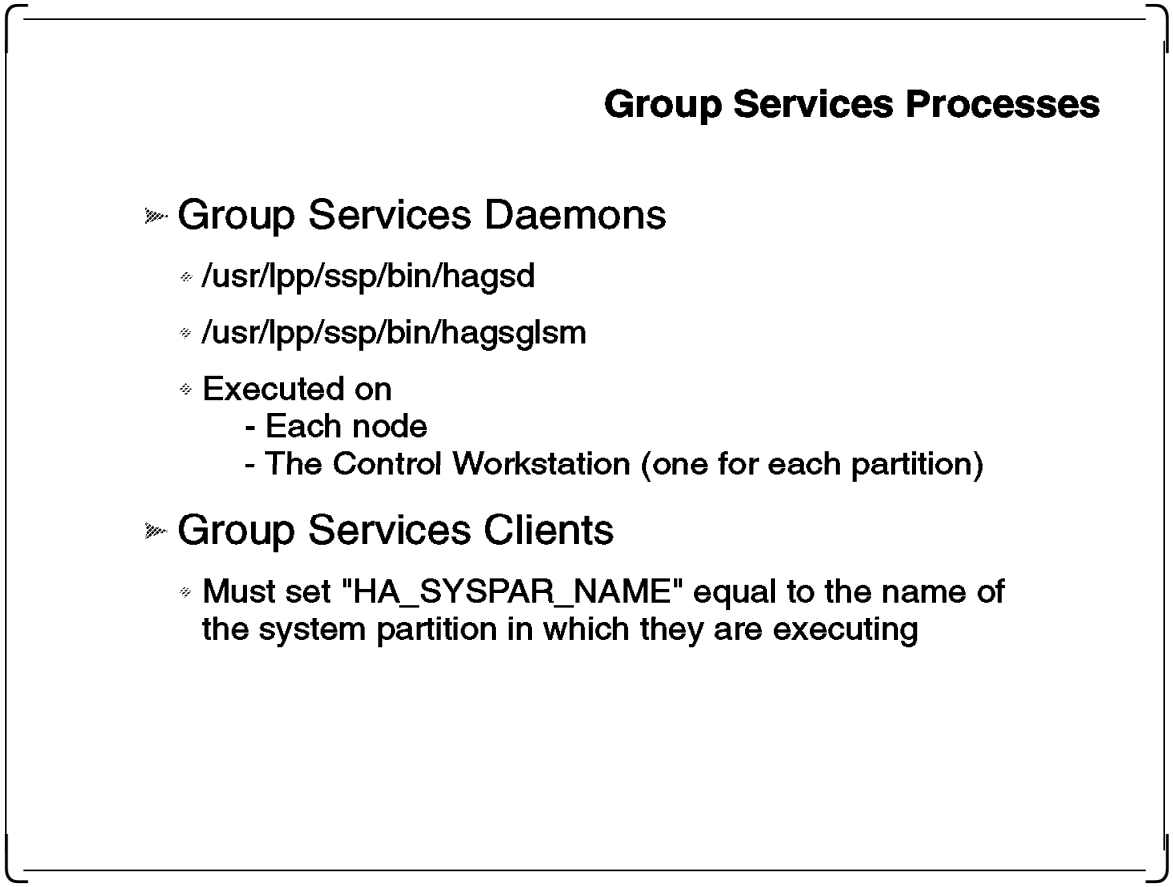
  Suppose that the providers of an application have been taking external actions during the barriers that are imposed by a multi-phase protocol, and that these actions must be completed for the application to be operational. Now suppose that the protocol is terminated because of a failure.

If the actions that the providers have already taken are not designed to be idempotent, the providers must explicitly undo the actions before they restart the protocol. Such an undo phase can be expensive and may require additional phases of coordination among the providers.

3.4 Group Services Administrations

This section discusses the administration topics associated to Group Services.

3.4.1 Group Services Processes



Group Services consists of two subsystems, *hags* and *hagsglsm*. They are controlled by the System Resource Controller (SRC), which is an AIX feature that provides mechanisms to define and control subsystems. The two subsystems belong to a subsystem group called *hags*.

The Group Services subsystems (hags and hagsglsm) are associated with the following daemons each:

- hagsd

  The hagsd daemon provides most of the services of the Group Services subsystem, which provides a general purpose facility for coordinating and monitoring changes to the state of an application that is running on a set of nodes.

  One instance of the hagsd deamon (hagsd.syspar_name) executes on the Control Workstation for each system partition. An instance of the hagsd daemon (hagsd) also executes on every node of a system partition.

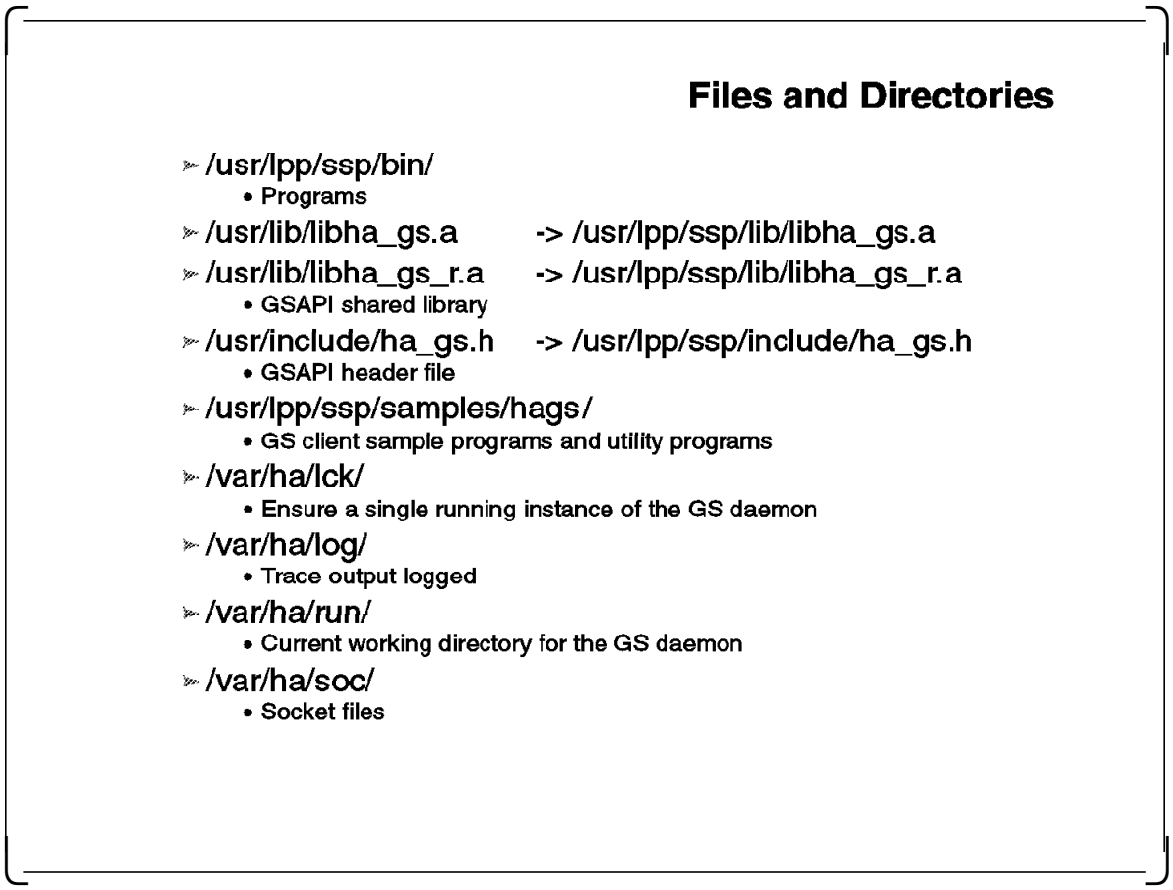
- hagsglsmd

  The hagsglsmd daemon provides global synchronization services for the SP Switch adapter membership group.

  One instance of the hagsglsmd daemon (hagsglsmd.syspar_name) executes on the Control Workstation for each system partition. Another instance of the hagsglsmd daemon (hagsglsmd) also executes on every node of a system partition.

Client processes need to communicate with a Group Services daemon (hags) that is executing on the same node on which the client processes are executing. The client processes accomplish this through the Group Services Application Programming Interface (GSAPI), using a Unix domain socket. All client processes that intend to use Group Services services through the GSAPI must set the environment variable called “HA_SYSPAR_NAME” equal to the system partition name in which they are executing. This must be done on the nodes, as well as on the Control Workstation, and the variable must be set before the client process executes the *ha_gs_init()* subroutine call.

3.4.2 Files and Directories



The Group Services subsystem uses the following directories:

- */usr/lpp/ssp/bin/*

  This directory contains the main executable files of the Group Services subsystem, which are:

  hagsd Group Services subsystem daemon

  hagsglsmd Group Services subsystem daemon

  hagsctrl Group Services subsystem control script

  hagscl Utility command (See 3.4.3.1, “Other Utilities” on page 104.)

  hagsgr Utility command (See 3.4.3.1, “Other Utilities” on page 104.)

  hagsmg Utility command (See 3.4.3.1, “Other Utilities” on page 104.)

  hagsns Utility command (See 3.4.3.1, “Other Utilities” on page 104.)

  hagspbs Utility command (See 3.4.3.1, “Other Utilities” on page 104.)

  hagsvote Utility command (See 3.4.3.1, “Other Utilities” on page 104.)

- */usr/lpp/ssp/lib/libha_gs.a*

  This is a GSAPI shared library used by a GS client program to obtain the services of the Group Services subsystem. This is for non-thread-safe programs.

- */usr/lpp/ssp/lib/libha_gs_r.a*

  This is a GSAPI shared library for thread-safe GS client programs.

- */usr/lpp/ssp/include/ha_gs.h*

  This is a header file for GSAPI.

- */usr/lpp/ssp/samples/hags/*

  This directory includes a GS client sample program and its related files. For more details on the sample program, see *Group Services Programming Guide and Reference*, GC28-1675.

- */var/ha/lck/*

  In this directory, the following directories are to ensure a single running instance of each Group Services daemon and to establish an "instance number" for each invocation of the daemons:

  ```
  hags.tid.syspar_name
  hagsglsm.tid.syspar_name
  ```

  syspar_name is the name of the system partition.

- */var/ha/log/*

  In this directory, Group Services daemons will write trace output during operation. The names of the files are in the following formats:

  ```
  hags_<node_number>_<instance_number>.syspar_name
  hagsglsm_<node_number>_<instance_number>.syspar_name
  ```

  where,

  - < node_number> is the node number on which the daemon is executing.
  - < instance_number> is the instance number of the daemon.
  - syspar_name is the name of the system partition.

  The daemon will limit the log size to a pre-established number of lines (by default 10000) and when this limit is reached, it will append ".bak" to the log and begin a new log. If a ".bak" file already exists, it will be removed, and the current log will then be renamed.

- */var/ha/run/*

  In this directory, the following directories are created:

  ```
  hags.syspar_name
  hagsglsm.syspar_name
  ```

  where, syspar_name is the system partition name. These directories are the current working directories for the Group Services daemons. Any core files created when the Group Services daemons abnormally terminate are placed in these directories. Whenever the Group Services daemons start, they rename any core files to:

  ```
  core_<node_number>.<instance_number>
  ```

  where,

  - < node_number> is the node number on which the daemons are executing.
  - < instance_number> is the instance number of the daemons.

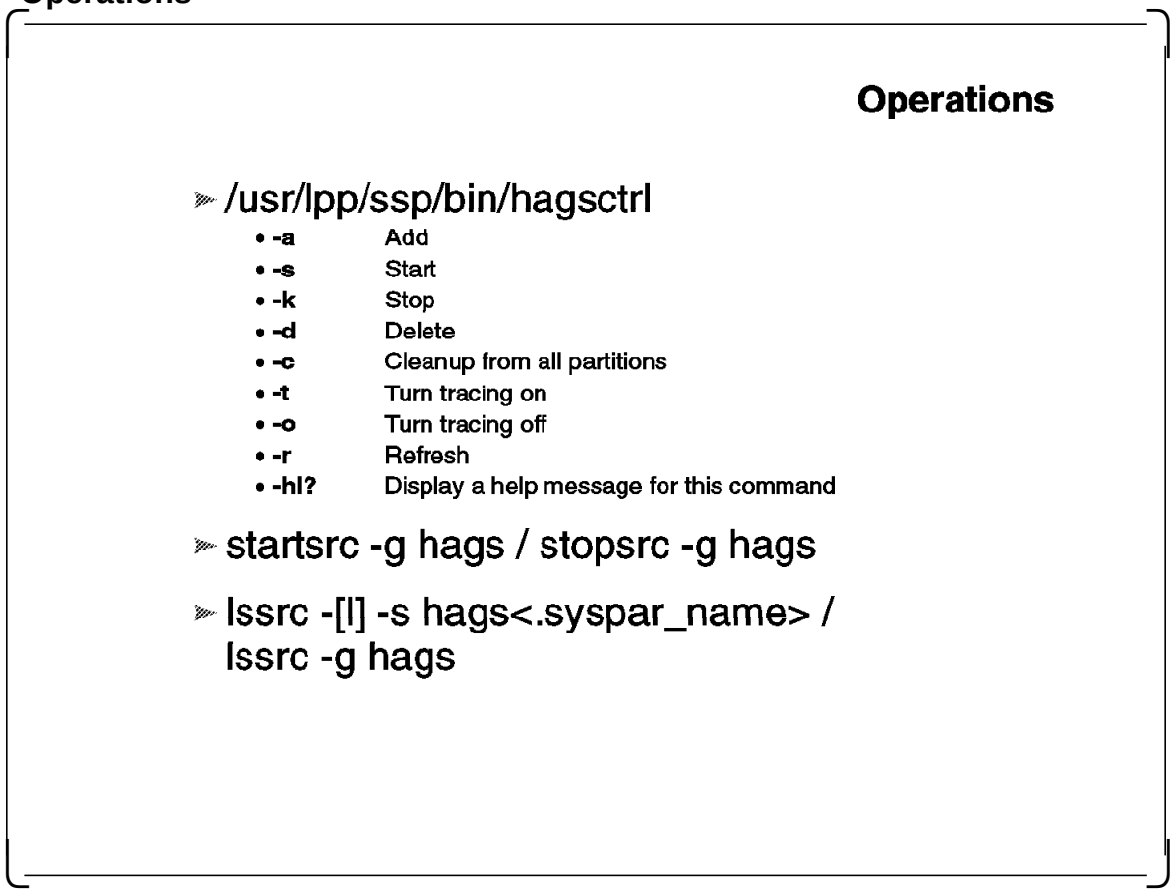
- */var/ha/soc/*

  This directory contains the socket files. The following socket file that is used by the GSAPI to connect to the Group Services daemon is located in this directory:

  `hagsdsocket.syspar_name`

  `syspar_name` is the system partition name.

3.4.3 Operations



The `hagsctrl` is the utility script command used to control the operations of the Group Services subsystems.

The `hagsctrl` script is not normally executed from the command line. It is normally called by the `syspar_ctrl` command during installation of the system, boot or reboot of individual nodes, and partitioning or re-partitioning of the system.

The purpose of the `hagsctrl` script is to add (configure) the Group Services subsystems to a system partition. It can also be used to remove the subsystems from a system partition. The `hagsctrl` script when combined with these options performs the following functions:

-a Add the subsystems.

-s Start the subsystems.

-k Stop the subsystems.

-d Delete the subsystems.

-c Clean the subsystems, that is, delete them from all system partitions.

-t Turn tracing on for the subsystems.

-o Turn tracing off for the subsystems.

-r Refresh the subsystems.

-h|? Display a help message for this command.

Note that except for the “clean” function, hagsctrl affects the Group Services subsystems in the current system partition, that is, the partition specified by the SP_NAME environment variable. For more information about the hagsctrl command, refer to *Command and Technical Reference*, GC23-3900.

Since the Group Services subsystems are controlled by SRC, you could use the startsrc and stopsrc SRC commands to start and stop the Group Services subsystems as follows:

```
startsrc -g hags
stopsrc -g hags
```

Also, by using the lssrc SRC command as follows, operational status can be obtained from the Group Services daemons:

```
lssrc [-l] -s hags.<syspar_name>
lssrc [-l] -s hagsglsm.<syspar_name>
lssrc -g hags
```

If you use the lssrc command with the “-l” flag for the *hags* subsystem, the following status information is written to standard out:

- The number of currently connected clients, and their process-IDs (PIDs)
- The status of the Group Services domain
- The node-number on which the GS nameserver is executing

  The following line in the output sample indicates that the GS nameserver is *node 0* (the Control Workstation):

  ```
  Domain established by node 0.
  ```

- Statistics for client groups with providers or subscribers on this node

The output of the lssrc command with “-l” on a “regular” node (not the Control Workstation) is as follows:

```
# lssrc -l -s hags
Subsystem         Group            PID     Status
 hags             hags             10054   active
 2 locally-connected clients.  Their PIDs:
 11848 12378
 HA Group Services domain information:
 Domain established by node 0.
 Number of groups known locally: 2
                    Number of   Number of local
 Group name         providers   providers/subscribers
 cssMembership            6           1           0
 ha_em_peers              7           1           0
```

For the same domain, the output from the Control Workstation is as follows:

```
# lssrc -l -s hags.sp2cw0
Subsystem         Group            PID     Status
 hags.sp2cw0      hags             16164   active
 1 locally-connected clients.  Their PIDs:
 18808
 HA Group Services domain information:
 Domain established by node 0.
 Number of groups known locally: 2
                     Number of   Number of local
 Group name          providers   providers/subscribers
 cssMembership             6           0           1
 ha_em_peers               7           1           0
```

The only differences in the examples are the subsystem names “hags” on a node and “hags.sp2cw0” on the Control Workstation, and the “number of local providers/subscribers” for *cssMembership* group.

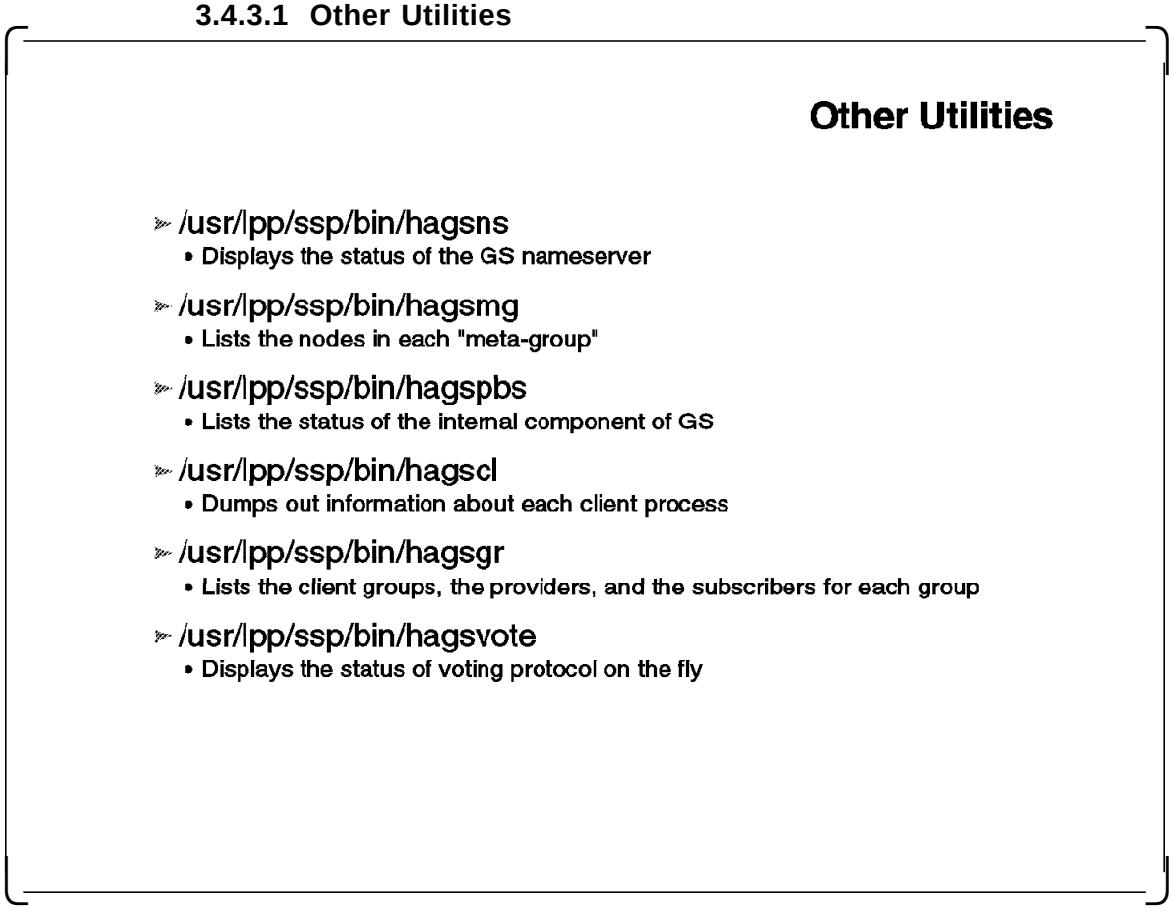
If the `lssrc` command times out, then the Group Services daemon is probably unable to properly connect to Topology Services. You can check the status of the Topology Services subsystem by issuing the following command:

```
lssrc -g hats
```

If the status of the Topology Services subsystem is “inoperative,” then you need to start Topology Services by issuing the following command on each node:

```
hatsctrl -s
```

3.4.3.1 Other Utilities



The following utility commands can be used to monitor the Group Services daemons and groups and to get information on how to trouble-shoot the Group Services subsystems when they are not working properly:

- **/usr/lpp/ssp/bin/hagsns**

  This command provides an SRC-based interface to interrogate the GS daemon as to its complete GS nameserver status. The status may vary significantly, depending upon the status of the various GS daemons across a system partition. The following description does not intend to completely describe all possible outputs, but it does provide some outputs of this command to help you understand how this command can be used.

  Assume we have a system partition with two nodes and the Control Workstation:

  Partition (CWS) name: sp2cw0
  Nodes in partition: 1 (sp2n01) and 5 (sp2n05)

  Assume that all nodes have just booted. On the Control Workstation, we see:

```
# lssrc -l -s hags.sp2cw0
Subsystem         Group            PID     Status
 hags.sp2cw0      hags             50942   inoperative
 1 locally-connected clients.  Their PIDs:
 52636
 HA Group Services domain information:
 Domain not established.
 Number of groups known locally: 0
```

Here, we see that the domain is not established. The hagsns command should show the following:

```
# hagsns -s hags.sp2cw0
We are: 0.28 domaindId = 0.Nil noNS = 1 inRecovery = 0
NS::ENsState(2):kAscend protocolInProgress = NS::ENsProtocol(0):kNoProtocol
outstandingBroadcast = NS::ENsBroadcast(0):kNoBcast
Process started on Aug 20 20:59:13, (0:0:28) ago.  HB connection took (0:0:0).
Our current epoch of certainty started on Aug 20 20:59:13, (0:0:28) ago.
3 UP nodes: 0 1 5
Coronation Timer has popped!
Domain not established for (0:0:28).  Waiting to hear from these 2 nodes: 1 5
```

The line that states:

```
We are: 0.28 domaindId = 0.Nil noNS = 1 inRecovery = 0
```

specifies that we have no GS nameserver (noNS = 1), but we are *not* recovering (inRecovery = 0). Therefore, we have not yet established a GS nameserver. The domaindId of 0.Nil also means we have not yet established a GS nameserver.

Note: For the process of establishing the GS nameserver, see 3.2.1.3, “Group Namespace” on page 42.

The lines that state:

```
NS::ENsState(2):kAscend protocolInProgress = NS::ENsProtocol(0):kNoProtocol
outstandingBroadcast = NS::ENsBroadcast(0):kNoBcast
```

specify that this node is in the Ascend GS nameserver state, meaning it is the lowest-numbered node, and is waiting to collect grovels from all nodes, so that it can establish itself as the GS nameserver. There is no protocol in progress.

The line that states:

```
3 UP nodes: 0 1 5
```

specifies that Topology Services has told the GS daemon that 3 nodes (including the Control Workstation) are running.

The line that states:

```
Coronation Timer has popped!
```

specifies that the coronation timer has popped, so this node is waiting solely to collect grovels from all nodes, so that it can establish itself as the GS nameserver.

The line that states:

```
Domain not established for (0:0:28).  Waiting to hear from these 2 nodes: 1 5
```

specifies how long this GS daemon has been waiting (HH:MM:SS). It then lists the nodes from which it is waiting for grovels (in this case, all other nodes in the system partition).

Assuming we then start Group Services on node 1, we get the following output:

```
# hagsns -s hags.sp2cw0
We are: 0.28 domaindId = 0.Nil noNS = 1 inRecovery = 0
NS::ENsState(2):kAscend protocolInProgress = NS::ENsProtocol(0):kNoProtocol
outstandingBroadcast = NS::ENsBroadcast(0):kNoBcast
Process started on Aug 20 21:43:07, (0:7:2) ago.  HB connection took (0:0:0).
Our current epoch of certainty started on Aug 20 21:43:07, (0:7:2) ago.
3 UP nodes: 0 1 5
Coronation Timer has popped!
Domain not established for (0:7:2).  Waiting to hear from these 1 node: 5
```

Now the GS daemon on the Control Workstation is waiting only for node 5.

On node 1, the following output from the hagsns command looks a little different from the output on node 5:

```
# hagsns -s hags.sp2cw0
We are: 1.15 domaindId = 0.Nil noNS = 1 inRecovery = 0
NS::ENsState(2):kGrovel protocolInProgress = NS::ENsProtocol(0):kNoProtocol
outstandingBroadcast = NS::ENsBroadcast(0):kNoBcast
Process started on Aug 20 21:50:02, (0:1:21) ago.  HB connection took (0:0:0).
Our current epoch of certainty started on Aug 20 21:50:02, (0:1:21) ago.
3 UP nodes: 0 1 5
Domain not established for (0:1:21).  Currently waiting for node 0
```

Here, we are in the Grovel state, with no protocol in progress.

Once Group Services is started on node 5 (with `startsrc -s hags`), then Group Services quickly forms a domain and node 0 establishes itself as the GS nameserver. The following output on the Control Workstation is basically what was previously shown, with the difference being the list of groups:

```
# hagsns -s hags.sp2cw0
We are: 0.28 domaindId = 0.28 noNS = 0 inRecovery = 0
NS::ENsState(7):kBecomeNS protocolInProgress = NS::ENsProtocol(0):kNoProtocol
outstandingBroadcast = NS::ENsBroadcast(0):kNoBcast
Process started on Aug 20 21:43:07, (0:11:51) ago.  HB connection took (0:0:0).
Initial NS certainty on Aug 20 21:54:55, (0:0:3) ago, taking (0:11:47).
Our current epoch of certainty started on Aug 20 21:54:55, (0:0:3) ago.
3 UP nodes: 0 1 5
1.1 ha_em_peers: GL: 0 seqNum: 0 theIPS: 0 1 5 lookupQ:
2.1 cssMembershp: GL: 1 seqNum: 0 theIPS: 1 lookupQ:
```

The following list in this output shows the client groups that have had providers on nodes join them in this Group Services domain:

```
1.1 ha_em_peers: GL: 0 seqNum: 0 theIPS: 0 1 5 lookupQ:
2.1 cssMembershp: GL: 1 seqNum: 0 theIPS: 1 lookupQ:
```

where:

<group_identifier> Is an identifier given to each Group Services group. It is internally used by Group Services (for example, 1.1, 2.1).

<group_name> Is the external name given by the Group Services clients when they join the group (for example, ha_em_peers).

< G L > Is the node number of the node on which a ″group leader″ is executing. The group leader (also called *primary Group Services daemon*) controls the functioning of the group. See 3.2.1.2, “Group Structure” on page 40.

<SeqNum> Is a sequence number for messages within each group. Each group maintains its own sequence of message broadcasts. The value shown here is the group's sequence number when it got an update message from its group leader. A group leader updates this field at the GS nameserver when it initially becomes group leader, and periodically thereafter (about every 20,000 group messages it sends). It is an internal to Group Services.

<theIPS> Is a "list" of nodes which have expressed an interest in a client group. Normally this results because a client process has asked to join or subscribe to a group. A node expresses interest in a group by sending a "lookup" request to the domain's GS nameserver, and is placed on the theIPS list when the GS nameserver sends out the response to the lookup. It is an internal to Group Services.

<lookupQ> Is a "list" of nodes that have sent in lookup requests, but to which the GS nameserver has not yet sent out a response. There is only a lookup queue when there is no group leader for the group, which can happen when the group is first being formed, or when the group leader's node fails, and group leader recovery is in progress.

At the same time, the hagsns output on node 1, which is not the GS nameserver node, is as follows:

```
# hagsns -s hags
We are: 1.15 domaindId = 0.28 noNS = 0 inRecovery = 0
NS::ENsState(6):kCertain protocolInProgress = NS::ENsProtocol(0):kNoProtocol
outstandingBroadcast = NS::ENsBroadcast(0):kNoBcast
Process started on Aug 20 21:50:02, (0:12:54) ago.  HB connection took (0:0:0).
Initial NS certainty on Aug 20 21:54:55, (0:8:1) ago, taking (0:4:52).
Our current epoch of certainty started on Aug 20 21:54:55, (0:8:1) ago.
3 UP nodes: 0 1 5
```

Note that no group information is contained in node 1's output. Since it is not the nameserver, it does not collect that information.

- **/usr/lpp/ssp/bin/hagsmg**

  This command provides the ordered list of nodes in each "meta-group." A meta-group can be thought of as a "shadow" to each client group, with the following two exceptions:

  1. *ZtheNameServerXY* meta-group: This is the group used by the GS nameserver to manage the domain, and should include all active nodes in the system partition.
  2. *theGROVELgroup* meta-group: This group is an internal use only group that never contains any nodes.

The output to hagsmg appears as follows:

```
# hagsmg -s hags
2.1 cssMembership: 1 5
1.1 ha_em_peers: 1 5 0
0.Nil ZtheNameServerXY: 1.15 5.6 0.29
0.Nil theGROVELgroup:
```

Similar to hagsns, the first column provides the internal “group_identifier” for each meta-group. The group_name is the external group name, with the two exceptions as noted above. Following the group name is the list of nodes that have been inserted into each meta-group. The first node in the list is the group leader (GL) for each meta-group. This node controls the operation of the group by choosing the next protocol to be executed.

The hagsmg output will list *only* the meta-groups to which that node has been inserted. On node 0 (the Control Workstation), which knows nothing about Switch adapter membership group (cssMembership), the output is as follows:

```
# hagsmg -s hags.sp2cw0
1.1 ha_em_peers: 1 5 0
0.Nil ZtheNameServerXY: 1.15 5.6 0.29
0.Nil theGROVELgroup:
```

The nodes are listed in group leader takeover order, which reflects the order in which they joined the group. If node 1 fails, node 5 will takeover as GS nameserver, as well as group leader for the ha_em_peers group.

- **/usr/lpp/ssp/bin/hagspbs**

  This command lists the status of the *broadcast services* subcomponent, an internal component of Group Services. The output of this command on node 1 when the domain is established is as follows:

```
# hagspbs -s hags
2.1 cssMembership: HWM 3 LWM 3 weAreTheGL
  pendingAckCount=0 kNotExpectingAcks  pendingRecoverCount=0
        1: HWM=0: lastType1=Nil
        6: HWM=3: lastType1=Nil
1.1 ha_em_peers: HWM 15 LWM 15 weAreTheGL
  pendingAckCount=0 kNotExpectingAcks  pendingRecoverCount=0
        1: HWM=1: lastType1=Nil
        5: HWM=15: lastType1=14
        0: HWM=15: lastType1=14
0.Nil ZtheNameServerXY: HWM 7 LWM 7 weAreTheGL
  pendingAckCount=0 kNotExpectingAcks  pendingRecoverCount=0
        1.15 kUp: needDSM: HWM=1: lstType1=Nil
        5.6 kUp: HWM=7: lastType1=Nil
        6.29 kUp: HWM=7: lastType1=Nil
0.Nil theGROVELgroup: HWM Nil LWM Nil
  pendingAckCount=0 kNotExpectingAcks  pendingRecoverCount=0
```

At the same time, there is no activity on node 5, as shown in the following output of hagspbs command:

```
# hagspbs -s hags
2.1 cssMembership: HWM 3 LWM 2
  pendingAckCount=0 kNotExpectingAcks  pendingRecoverCount=0
        1: HWM=Nil: lastType1=Nil
        6: HWM=1: lastType1=Nil
1.1 ha_em_peers: HWM 15 LWM 14
  pendingAckCount=0 kNotExpectingAcks  pendingRecoverCount=0
        1: HWM=3: lastType1=Nil
        5: HWM=3: lastType1=Nil
        0: HWM=12: lastType1=Nil
0.Nil ZtheNameServerXY: HWM 7 LWM 6
  pendingAckCount=0 kNotExpectingAcks  pendingRecoverCount=0
        1.15 kUp: HWM=Nil: lstType1=Nil
        5.6 kUp: HWM=Nil: lastType1=Nil
        6.29 kUp: HWM=6: lastType1=Nil
0.Nil theGROVELgroup: HWM Nil LWM Nil
  pendingAckCount=0 kNotExpectingAcks  pendingRecoverCount=0
```

The output lists each group_identifier and group_name. It then lists the "broadcast sequence numbers" for each group, which consists of a HighWaterMark (HWM) and a LowWaterMark (LWM) for each group.

In general, these sequence numbers should match for each group on each node that is inserted in to that group. However, due to delays of messages getting across the networks, some nodes may not have yet seen all messages. Assuming the system partition eventually quiets down, all nodes should eventually catch up. Note that these sequence numbers will *not* normally match those displayed by the hagsns command.

The other lines of output have too many possible permutations to go into now. If one or more groups are running a series of protocols (due to nodes boots or failures) you can see a large amount of output here. If a group is hung, or the domain seems hung with a protocol in progress, run this command on the group′s leader′s node to see which node (or nodes) it may be waiting for.

- **/usr/lpp/ssp/bin/hagscl**

  This command allows you to dump out information about each client process currently connected to Group Services. If you remember, the lssrc -l -s hags command lists the process-IDs (PIDs) of the connected clients. The hagscl command with long option (-l) allows you to get additional information as follows:

```
# hagscl -l -s hags
Client Control layer summary:
  Number of clients connected: 2
  Cumulative number of clients connected: 2
  Total number of client requests: 2
  Number of client hash table conflicts: 0

--------------------------------------------------
Client: socketFd[9] pid[17076]Total number of Clients: 2
 Client initialized: pid: 17076
Number of local providers/subscribers: 1/0
Responsiveness information for Client: socketFd[9] pid[17076]
Type[ type[HA_GS_PING_RESPONSIVENESS]] interval[60] response time limit[5]
Checks done/bypassed[58/0] lastResponse[OK]]
Results(good/bad/late)[58/0/0]
Membership list:
slot    info
0       [ provider Member token[0] Client: socketFd[9] pid[17076]ProviderId[1/5]
--------------------------------------------------
Client: socketFd[9] pid[9824]Total number of Clients: 2
 Client initialized: pid: 9824
Number of local providers/subscribers: 1/1
Responsiveness information for Client: socketFd[9] pid[9824]
Type[ type[HA_GS_PING_RESPONSIVENESS]] interval[3630] response time limit[10]
Checks done/bypassed[0/0] lastResponse[OK]]
Results(good/bad/late)[0/0/0]
Membership list:
slot    info
0       [ subscriber Member token[0] Client: socketFd[10] pid[9824]]
0       [ provider Member token[0] Client: socketFd[10] pid[9824]ProviderId[0/5]
```

This output expands to describe a number of items:

- The following lines that show:

  ```
  Client: socketFd[9] pid[9824]Total number of Clients: 2
   Client initialized: pid: 9824
  ```

  are mostly just a repeat of what `hagscl -s hags` says, listing the PID and socket file descriptor. However, information on whether or not the client is “initialized” is also added. “Initialized” means that the client has successfully executed *ha_gs_init()* call.

- The following line shows how many groups this client has joined or subscribed to:

  ```
  Number of local providers/subscribers: 1/1
  ```

  In this case, the client has joined one group and subscribed to one group.

- The following lines show some information about *responsiveness*, which is a periodic check that a GS daemon performs on its clients (see *Group Services Programming Guide and Reference, GC28-1675* for how to specify this on the *ha_gs_init()* call):

  ```
  Responsiveness information for Client: socketFd[9] pid[9824]
  Type[ type[HA_GS_PING_RESPONSIVENESS]] interval[3630] response time limit[10]
  Checks done/bypassed[0/0] lastResponse[OK]]
  Results(good/bad/late)[0/0/0]
  ```

  Here, it echoes the parameters given on *ha_gs_init()* (type/interval/time limit) and also displays the number of checks done. In this case, the interval is 3630 seconds, and the client has not been connected that long.

– The following lines show the membership list:

```
Membership list:
slot  info
0     [ subscriber Member token[0] Client: socketFd[10] pid[9824]]
0     [ provider Member token[0] Client: socketFd[10] pid[9824]ProviderId[0/5]
```

This client is subscribed to one group and joined to another. The ProviderId is this client's provider identification in its group. This can be matched with the hagsgr output to determine the groups the client actually cares about.

- **/usr/lpp/ssp/bin/hagsgr**

This command lists the client groups, and the providers and subscribers for each group. Similar to hagsmg, it only lists the groups to which the node (where the command is issued) is joined, subscribed, or both.

The output of hagsgr command is as follows:

```
# hagsgr -s hags
  Number of: groups: 3
Group slot # [0] Group name[HostMembership] group state[Not Inserted |]
Providers[]
Local subscribers[]

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

Group slot # [2] Group name[!SwitchGroupie!] group state[Idle |]
Providers[[0/1][0/5]]
Local subscribers[[10/5]]

Group slot # [3] Group name[cssMembership] group state[Inserted |Idle |]
Providers[[0/1][0/5]]
Local subscribers[]
```

Group slot is the index of the group in an internal Group Services table.

The format for providers listed is the following:

[provider_instance_number / provider_node_number]

where

– "Provider_instance_number" is specified by the client process when it joins a group.
– "Provider_node_number" is the node_number on which the client process is executing.

The providers are shown in the order in which they joined the group, therefore, the oldest provider is listed first, and the youngest last.

The format for subscribers is similar:

[socket_file_descriptor / subscriber_node_number]

where:

– "Socket_file_descriptor" is the GS daemon's socket file descriptor that connects it to the client process that subscribed to the group. Note that you can use this along with the output from hagscl to determine the PID of the subscriber, since a node will display only the local subscribers (those actually executing on the node on which the hagsgr command is executed).

- "Subscriber_node_number" is the node_number on which the subscriber process is executing, which is always the local node.

The group state field indicates the status of that group, and it is a collection of separate states concatenated with the "OR" sign:

1. If it includes "Not Inserted," then that group is not currently active on this node, and should have no providers or subscribers.
2. If it includes "Insert Pending," then it is attempting to become inserted into the group. It has sent the GS nameserver a "lookup" and is awaiting the response, or the response is received, and it is awaiting to be inserted into the meta-group. You have to use `hagsmg` or `hagspbs` to determine the exact step. This should normally be a temporary state.
3. If it includes "Inserted," then that group is currently active on this node, and it may have providers, subscribers, or both.

   If "Inserted" is included, then one of the following is also present:

   - "Idle" indicates the group is not currently running a protocol.
   - "Running Protocol" indicates the group is running a protocol. The `hagsgr` with the long option allows you to see what this protocol is.
   - "Needs Priming" indicates a node is attempting to become fully active in a group, and is waiting to find out the current state (membership and group state value) of the group. This is normally temporary.
4. Other temporary states include "Waiting for BroadcastSent" and "Resending Requests." The former indicates the node is in the midst of sending a broadcast, and the latter indicates that the group's GL has failed and the Group Services subsystem is recovering to a new GL.

It is also possible to specify `hagsgr` for a specific group:

```
# hagsgr -a ha_em_peers -s hags
  Number of: groups: 3
Group slot # [1] Group name[ha_em_peers] group state[Inserted |Idle |]
Providers[[1/1][1/5][1/0]]
Local subscribers[]
```

You can also use the long option, which may result in an awesomely verbose output. The output sample and its explanation are too deep into the internal of Group Services to go into in this book.

- **/usr/lpp/ssp/bin/hagsvote**

  This command only displays information of groups in the midst of voting protocols. If no groups are doing anything, you get something similar to the following:

```
# hagsvote -s hags
  Number of: groups: 4
Group slot # [0] Group name[HostMembership] voting data:
No protocol is currently executing in the group.
---------------------------------------------------

Group slot # [2] Group name[ha_em_peers] voting data:
No protocol is currently executing in the group.
---------------------------------------------------

Group slot # [3] Group name[!SwitchGroupie!] voting data:
No protocol is currently executing in the group.
---------------------------------------------------

Group slot # [3] Group name[!cssMembership!] voting data:
No protocol is currently executing in the group.
---------------------------------------------------
```

However, the rest of the output was captured when at least one group was busy.

The hagsvote output also differs depending whether it is issued on the GL node for a group, or on a non-GL node. It will display a summary of the collected vote responses if it is on the GL, as well as a list of nodes that have and have not voted. On non-GL nodes, it can only list the local providers, and whether they have voted or not.

The short option displays only summary data, while the long option will display data for each provider (on all nodes) and for all providers of the group (on the GL node). The hagsgr with the long option shows an active "join protocol" as follows:

```
# hagsvote -l -s hags
Currently executing protocol: SJoinProtocol: requested by: [ provider Member tok
en[0]  [remote provider] ProviderId[1/0]]
SVProtocol: state[Submitted+Executing+ExecutingPostBroadcast+ExecutingNeedsVotes
+ExecutingVoteSubmitted+Continuing]
ProtocolToken[11/22]
[proposer:] provider Member token[0]  [remote provider] ProviderId[1/0]]
][group:Group name[ha_em_peers]]
Number phases[2] this phase[1]
summary code[0] time limit[60]

[Batching allowed]
Changing count [1] local changing count [0] changers removed count [0]
Have [32] changing member slots, list:
SProvider(ProviderId[1/0]conditionalListPosition[0]
SVSuppMember:  token[0] status[NotIn ]
 supp ptr : 0x0 group ptr: 0x3004b358 groupListPosition: -1 nodeListPosition: -1
Need Vote/Vote Yet[0/0]
 Last Request:0x30055a58
 [votingParticipant])[end SProvider]

----------------------------
```

On node 0 (the Control Workstation), which is not the GL, the hagsvote output with the short option followed by the hagsvote output with the long option are as follows:

```
# hagsvote -a ha_em_peers -s hags.sp2cw0
  Number of: groups: 2
Group name[ha_em_peers] voting data:
Not GL in phase [1] of an n-phase protocol of type[Join].
Local voting data:
Local provider count [1] Number not yet voted [0](vote submitted).
 Given vote:[Approve vote]Default vote:[No vote value]
```

Here,

`Given vote:[Approve vote]Default vote:[No vote value]`

indicates that this provider voted to approve the protocol, and did not submit a default vote value on its vote.

```
# hagsvote -l -a ha_em_peers -s hags.sp2cw0
  Number of: groups: 2
Group name[ha_em_peers] voting data:
Not GL in phase [1] of an n-phase protocol of type[Join].
Local voting data:
Local provider count [1] Number not yet voted [0](vote submitted).
 Given vote:[Approve vote]Default vote:[No vote value]
ProviderId      Voted?  Failed? Conditional?
[1/0]   Yes     No      Yes
```

The long option breaks this down by individual providers, and lists each provider's voting status, failure status, and conditional status. A conditional provider is the provider involved in a running join protocol.

On the GL node (node 1), the output is as follows:

```
# hagsvote -a ha_em_peers -s hags
  Number of: groups: 5
Group name[ha_em_peers] voting data:
GL in phase [1] of an n-phase protocol of type[Join].
Local voting data:
Local provider count [1] Number not yet voted [0](vote submitted).
 Given vote:[Approve vote]Default vote:[No vote value]
Global voting data:
Number of nodes in group [3] Number not yet voted [1]
Given vote:[Approve vote]Default vote:[No vote value]
```

This output initially gives data for its local providers (as does a non-GL node), which indicates that this node's provider has voted. It then adds the `Global voting data:`, which here summarizes how many nodes need to vote, and how many have yet to vote. In this case, one node remains to vote. Also, this summarizes the results of the collected voting (`Approve vote`) so far.

The output with the long option (-l) on node 1 is as follow:

```
# hagsvote -l -a ha_em_peers -s hags
  Number of: groups: 5
Group name[ha_em_peers] voting data:
Not GL in phase [1] of an n-phase protocol of type[Join].
Local voting data:
Local provider count [1] Number not yet voted [0](vote submitted).
 Given vote:[Approve vote]Default vote:[No vote value]
ProviderId      Voted?  Failed? Conditional?
[1/0]   Yes     No      Yes
[1/1]   Yes     No      No
Global voting data:
Number of nodes in group [3] Number not yet voted [1]
Given vote:[Approve vote]Default vote:[No vote value]
Nodes that have voted [0 1 ]
Nodes that have not voted [5 ]
```

This breaks down the voting to all providers in that have so far submitted votes, and also breaks it down by node. The line:

`Nodes that have not voted [5 ]`

shows that the provider on node 5 has not voted.

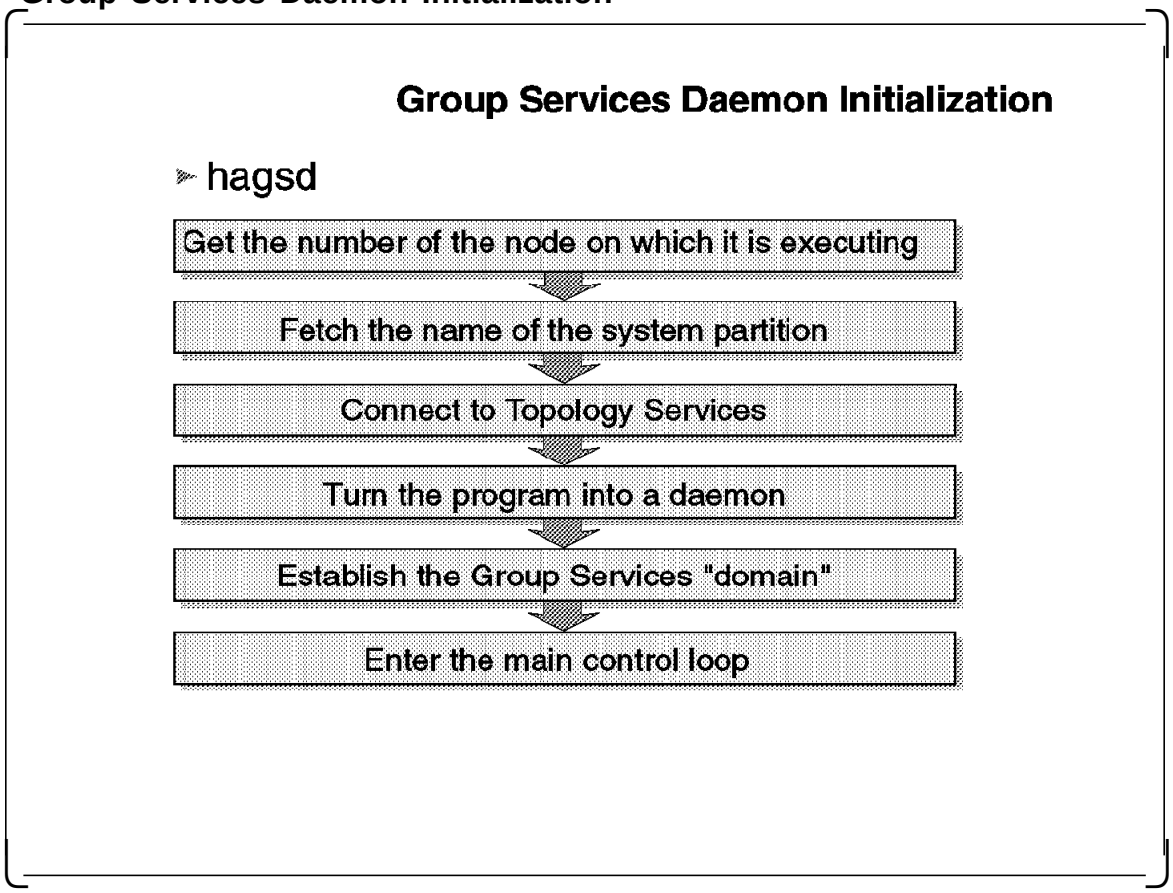
The hagsvote outputs on node 5 are as follows:

```
# hagsvote -a ha_em_peers -s hags
  Number of: groups: 4
Group name[ha_em_peers] voting data:
GL in phase [1] of an n-phase protocol of type[Join].
Local voting data:
Local provider count [1] Number not yet voted [1](vote submitted).
 Given vote:[No vote value]Default vote:[No vote value]
```

```
# hagsvote -l -a ha_em_peers -s hags
  Number of: groups: 4
Group name[ha_em_peers] voting data:
Not GL in phase [1] of an n-phase protocol of type[Join].
Local voting data:
Local provider count [1] Number not yet voted [0](vote submitted).
 Given vote:[Approve vote]Default vote:[No vote value]
ProviderId      Voted?  Failed? Conditional?
[1/0]   Yes     No      Yes
[1/5]   No      No      No
```

Note that Global voting data is not shown here, since this node is not a GL.

3.4.4 Group Services Daemon Initialization



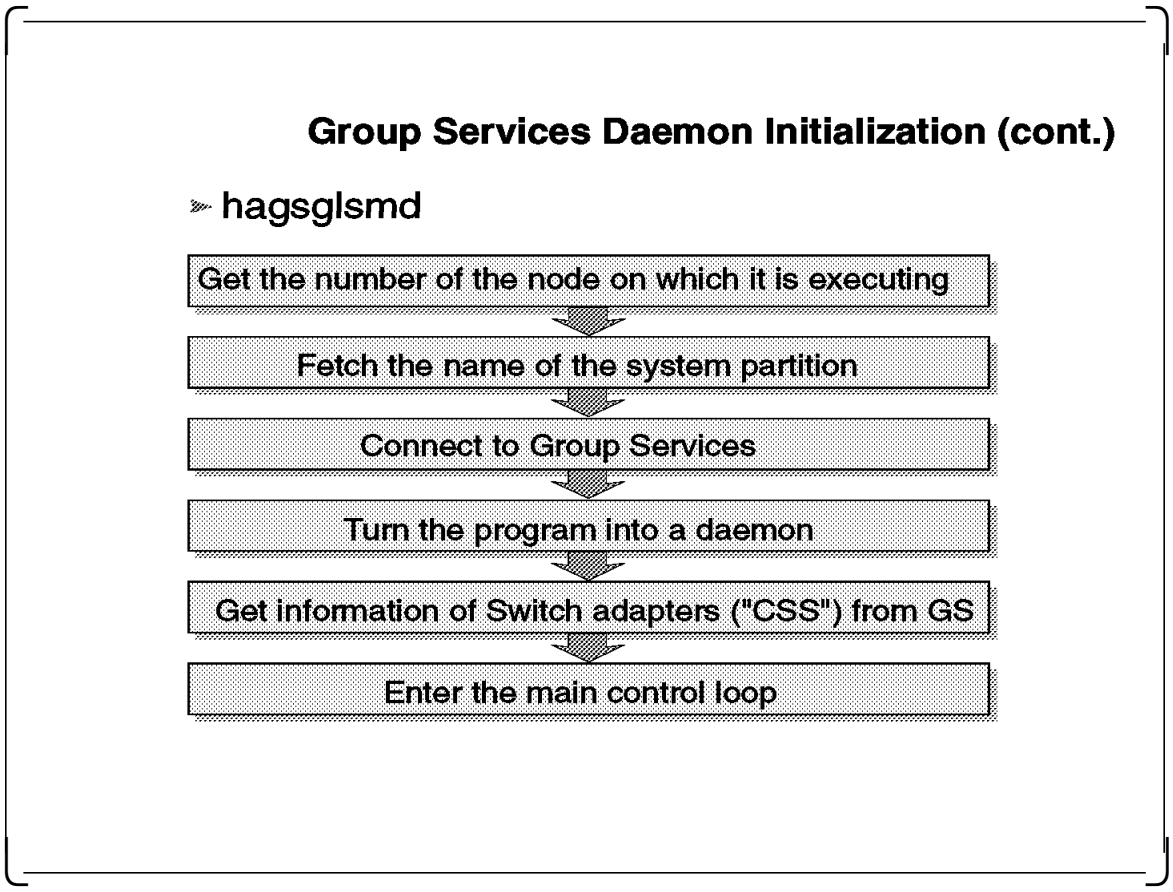
Normally, the Group Services daemons are started by an entry in the /etc/inittab file, with the `startsrc` command. If necessary, the GS daemons can be started using the `hagsctrl` script or the `startsrc` command directly.

The `hagsd` daemon executes the following steps during its initialization:

1. Get the number of the node on which it is executing using the /usr/lpp/ssp/install/bin/node_number command. Node 0 is the Control Workstation.
2. Fetch the name of the system partition.
3. Connect to Topology Services. If the connection cannot be made because the Topology Services subsystem is not running, then the connection attempt is scheduled to be made again in one second. This is repeated until the connection to Topology Services is established. Group Services periodically writes an AIX error log until this connection is established. Until this time, no clients may connect to Group Services
4. Turn the program into a daemon. This includes establishing communications with the SRC subsystem (in order to return a status information at the request of SRC commands).
5. Establish the Group Services “domain.” The domain is equivalent to the set of nodes within the RS/6000 SP system partition in which a daemon is executing. At this point, one of the `hagsd` daemons establishes itself as the *GS Nameserver*. For the process of establishing GS nameserver, see 3.2.1.3,

"Group Namespace" on page 42. Until the domain is established, no client requests to join or subscribe to groups will be processed.

6. Enter the main control loop. This loop waits for requests from clients, messages from other Group Services daemons, messages from Topology Services, and requests from the SRC for status.



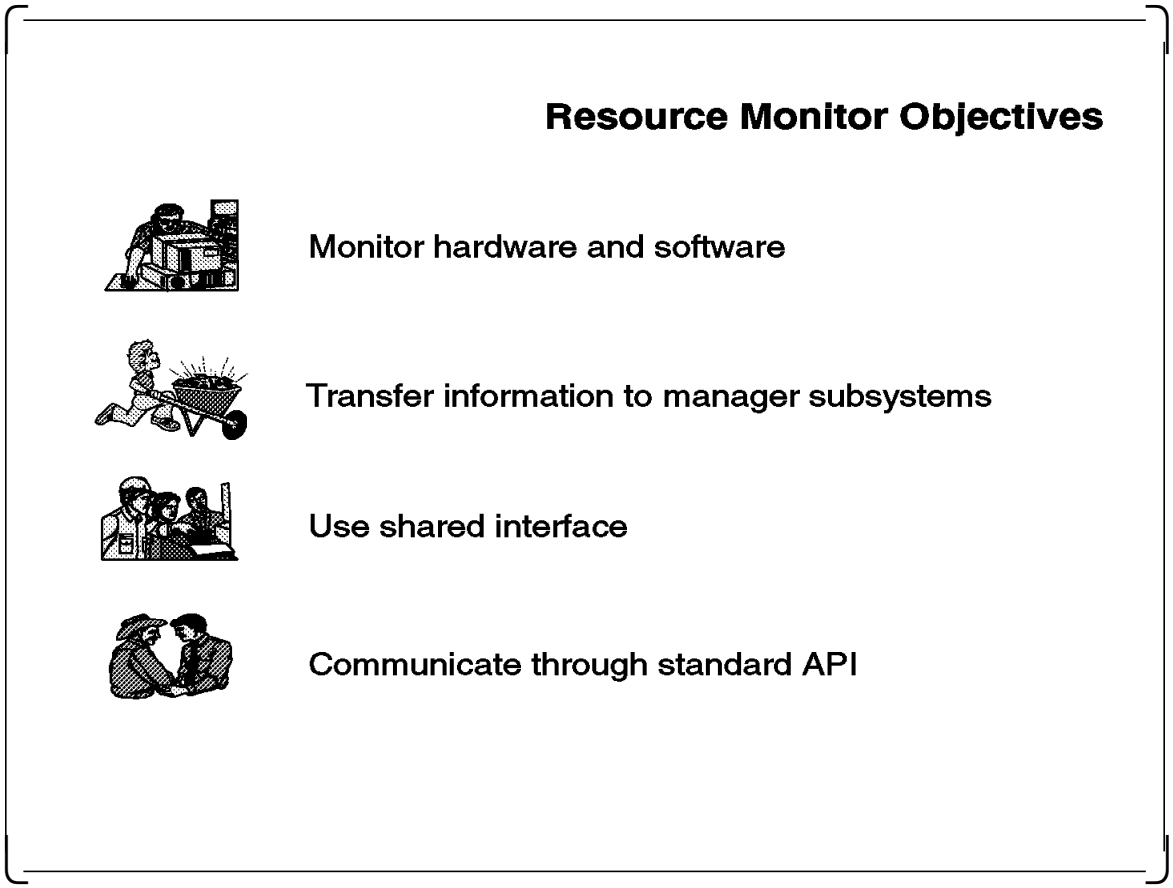
The hagsglsmd daemon executes the following steps during its initialization:

1. Get the number of the node on which it is executing using the /usr/lpp/ssp/install/bin/node_number command. Node 0 is the Control Workstation.
2. Fetch the name of the system partition.
3. Connect to Group Services. If the connection cannot be made because the Group Services subsystem is not running, then the connection attempt is scheduled to be made again in one second. This is repeated until the connection to Group Services is established. Group Services periodically writes an AIX error log entry until this connection is established.
4. Turn the program into a daemon. This includes establishing communications with the SRC subsystem (in order to return a status information at the request of SRC commands).
5. Get the information about the High Performance Switch (HPS) or SP Switch adapters (“CSS”) from the hagsd daemon.
6. Enter the main control loop. This loop waits for requests from clients, messages from other Group Services daemons, messages from Topology Services, and requests from the SRC for status.

Chapter 4. Resource Monitors

This chapter describes the Resource Monitors and gives an explanation of their basic terminology and architecture.

4.1 Resource Monitor Objectives

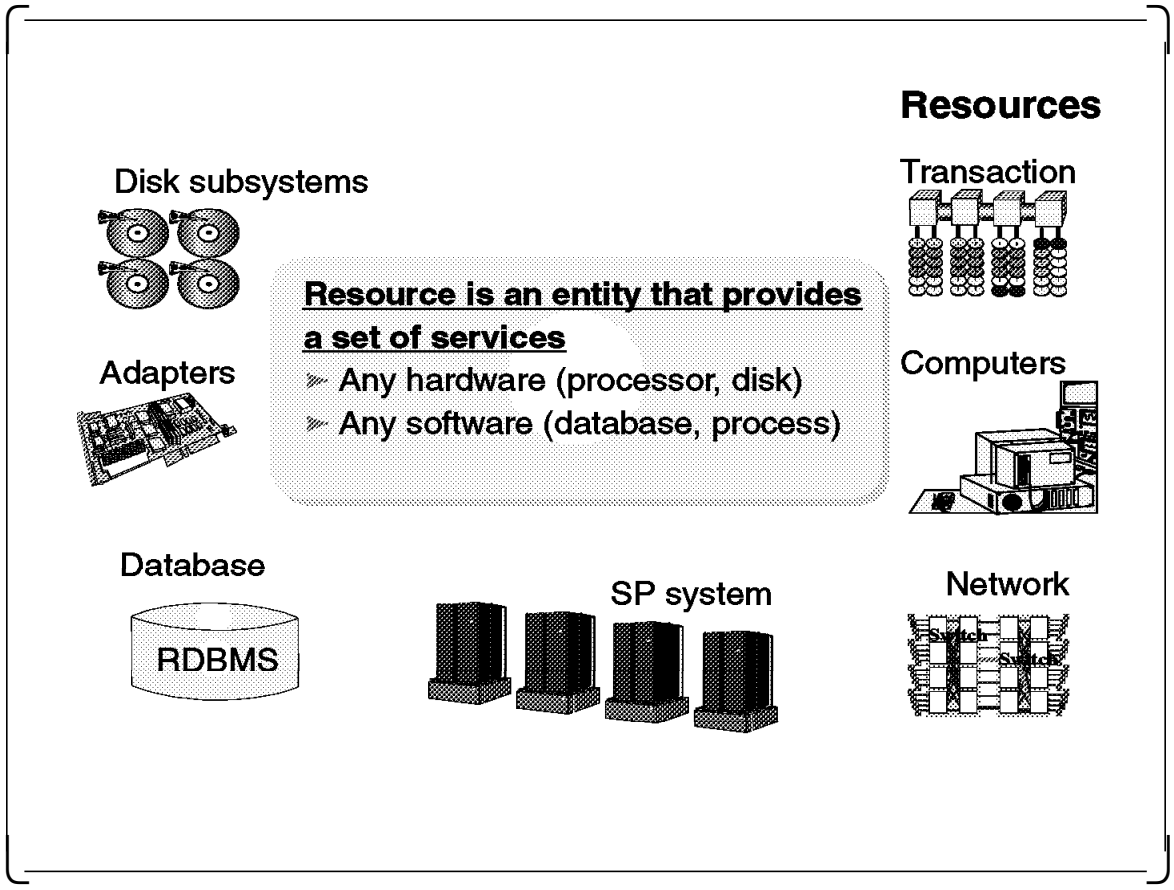


Most of the computer users monitor the hardware and software components of a system. They often find that the hardware is different than expected, the software does not behave as expected, and the user application does not use the system resources in an efficient manner. The results of monitoring lead to the user's desire to optimize the system.

The monitored results must be transferred to a higher level of software that processes the data. The data transfer must be highly efficient, because a monitor must not load the system with a heavy computing load.

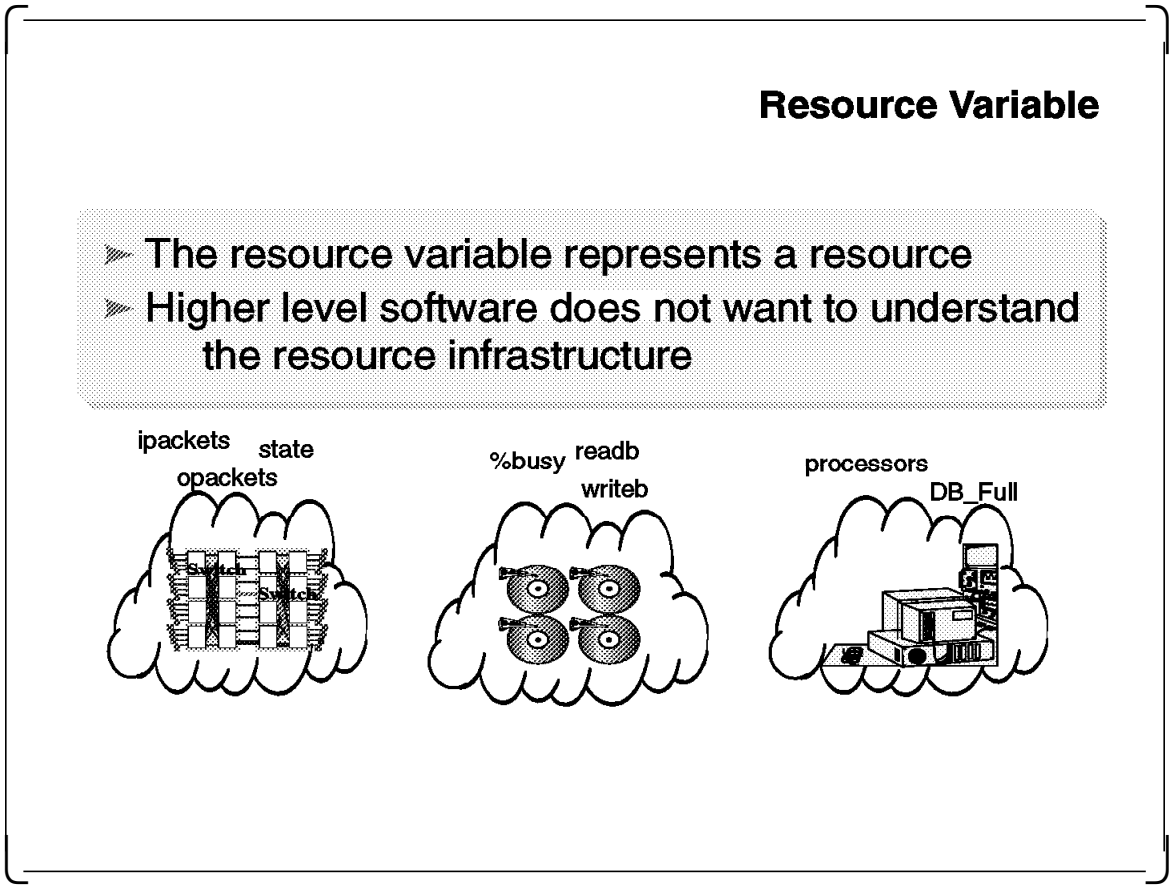
The data obtained from a monitor program is usually distributed to more than one higher level software through shared memory. Therefore, the interface must be able to be shared. This interface is published in the standard documentation delivered with the product. Using a standard interface also allows independent software vendors to create their software applications and use the standard system resources in order to achieve the best system performance.

4.2 Resources



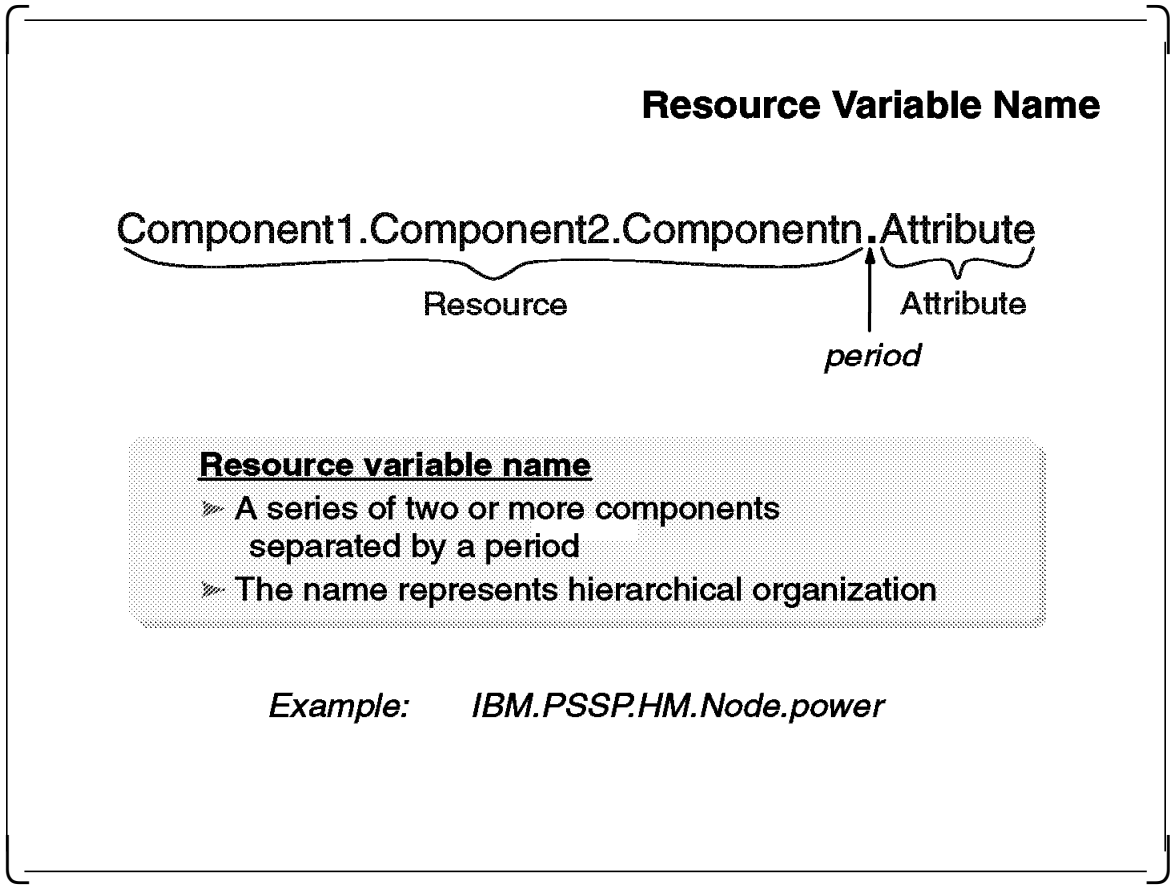
A *resource* is an entity in the system that provides a set of services. Examples of resources include hardware entities (such as processors, disk drive, memory, and adapters), and software entities (such as database subsystems, programs, and filesystems).

4.3 Resource Variable



A higher level structure does not want to understand the internal structure of a resource. Every resource in a system can be represented by variables. The variables representing a resource are called *resource variables*. A resource variable is simply the representation of an attribute of a resource. Any resource can be represented by one or more resource variables. For example, the SP Switch can be represented by input packets, output packets, state, transmit queue, receive queue, and so on. A representation of a disk can be the number of write blocks, read blocks, percentage of disk busy, and so on. Resource variables concerning the CPU types of resources may be the number of processors on-line, the processor load, and so on. A database application may produce variables on the database load, database full, and so on.

4.4 Resource Variable Name



A resource variable name is a series of two or more components separated by a period. Each component is a character string that begins with an alpha character. Any component other than the first component 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.

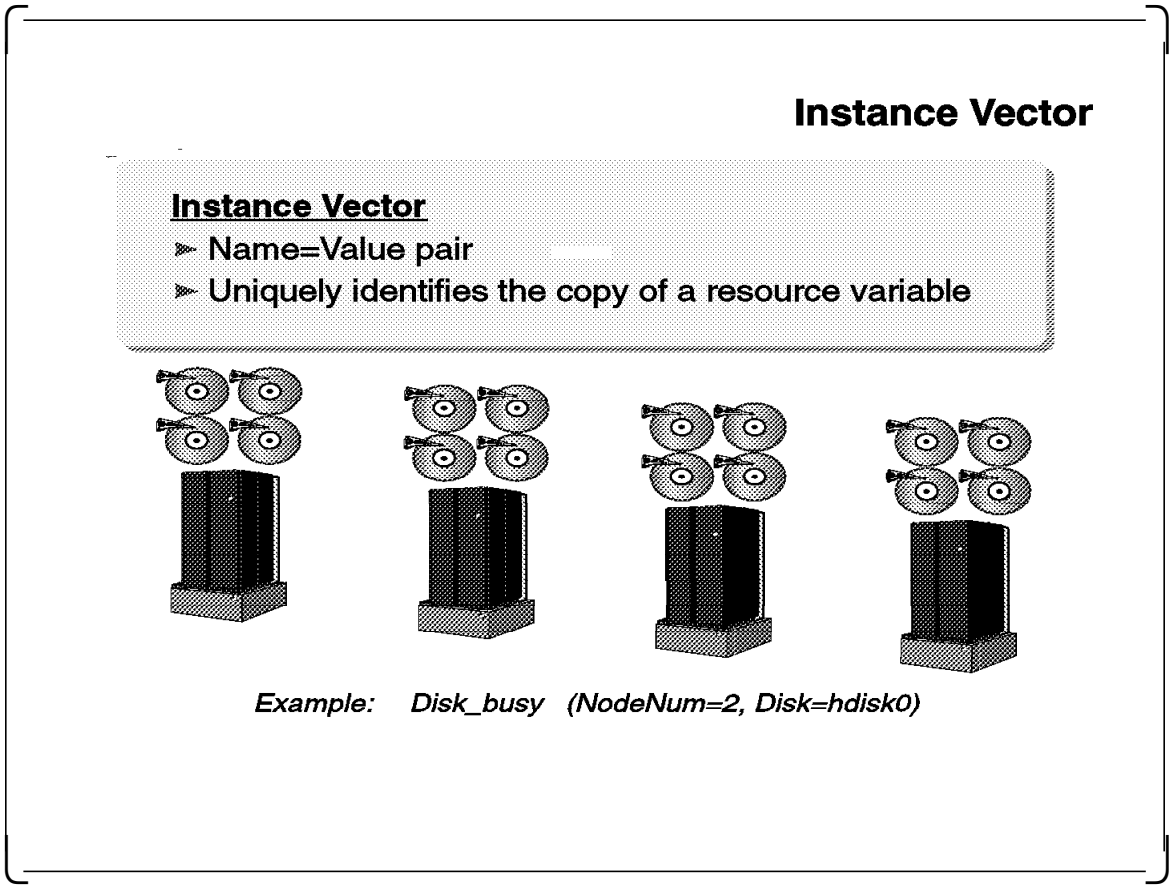
The name represents a hierarchical organization, from the general to the specific.

An example of a resource variable is as follows:

```
IBM.PSSP.PRCRS.procs_online
IBM.PSSP.Prog.pcount
IBM.PSSP.Response.Host.state
IBM.PSSP.Response.Switch.state
```

In order to avoid collisions in the naming of resource variables, the first component of each variable name is the name of the vendor that supplied the subsystem. As you can see in the preceding example, for IBM subsystems the vendor name is *IBM*. In an IBM subsystem, the second component of resource variables is the product name containing the subsystem.

4.5 Instance Vector



Most of the 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 than one instance of a database, and certainly more than one node.

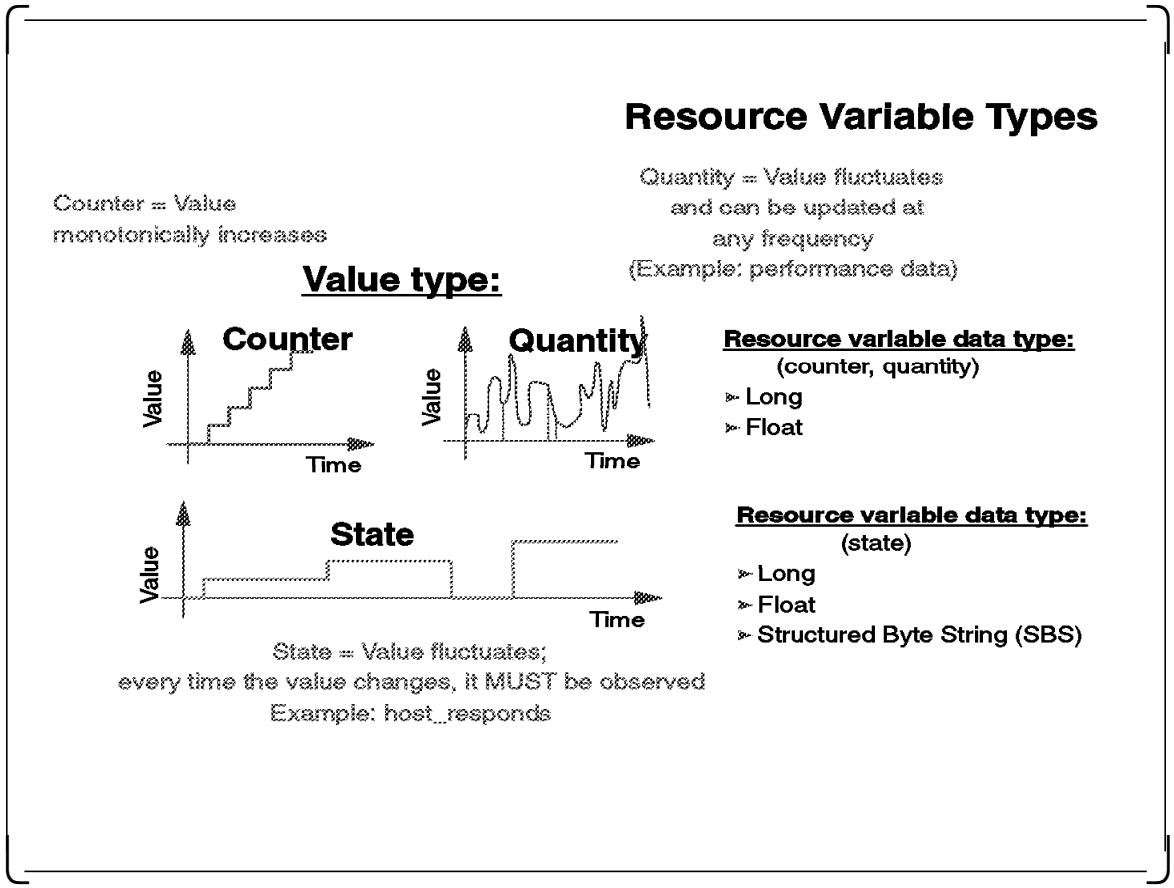
In order to uniquely identify each copy of a resource, each resource variable in the system has only one associated instance vector. An *instance vector* is a set of elements where each element is a *name-value* pair. The *name* is a description of the element and the *value* uniquely identifies the copy of a resource variable. If the resource variable can have only one copy in the entire system, the instance vector is NULL.

Examples of resource variables and instance vectors follow:

```
IBM.PSSP.aixos.Disk.busy (NodeNum=5, Name=hdisk10)
IBM.PSSP.PRCRS.procs_online (NodeNum=1)
IBM.PSSP.Prog.pcount (NodeNum=2, UserName=root, ProgName=dbserv)
IBM.PSSP.Response.Host.state (NodeNum=15)
IBM.PSSP.Response.Switch.state (NodeNum=2)
```

Currently, there is a limit of four name-value pairs per instance vector.

4.6 Resource Variable Types



A resource variable has one of three value types:

Counter This is a value type that monotonically increases.

Quantity This is a value type that fluctuates over time.

State This is a value type that fluctuates over time. Every time it changes value, it must be observed to avoid missing generating events.

The counter and quantity variables are taken from PTX/6000. The utility in these two types is that the resource variable can be updated at any frequency (preferably a high frequency), but it is only necessary to observe their values at a relatively low frequency (on the order of every 5 or 10 seconds or slower) to uphold the ability to generate useful events.

State variables, whose values also fluctuate over time, must be observed every time they change value to avoid missing generating events. While the time between any two updates of a state variable may be short, on average a state variable is expected to change at a relatively slow rate. The counter and quantity variables have data types of:

- Long
- Float

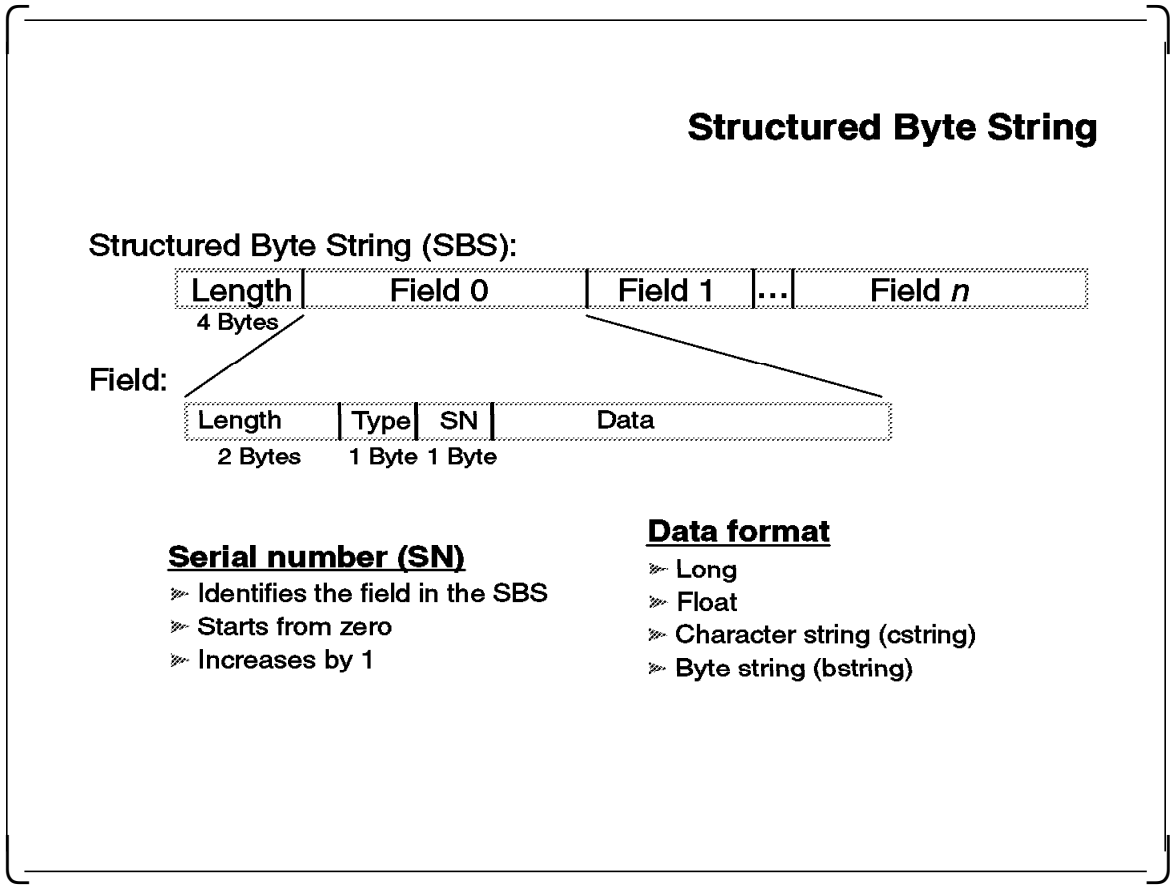
The resource variables of type state have data types of:

- Long
- Float
- Structured Byte String (SBS)

The long and float formats are identical to the C language types of the same names. The Structured Byte String (SBS) is explained in the next section.

The definitions for variables of value type counter and quantity are taken directly from the Performance Toolbox 1.2 and 2.1 for AIX (PTX/6000) product.

4.7 Structured Byte String



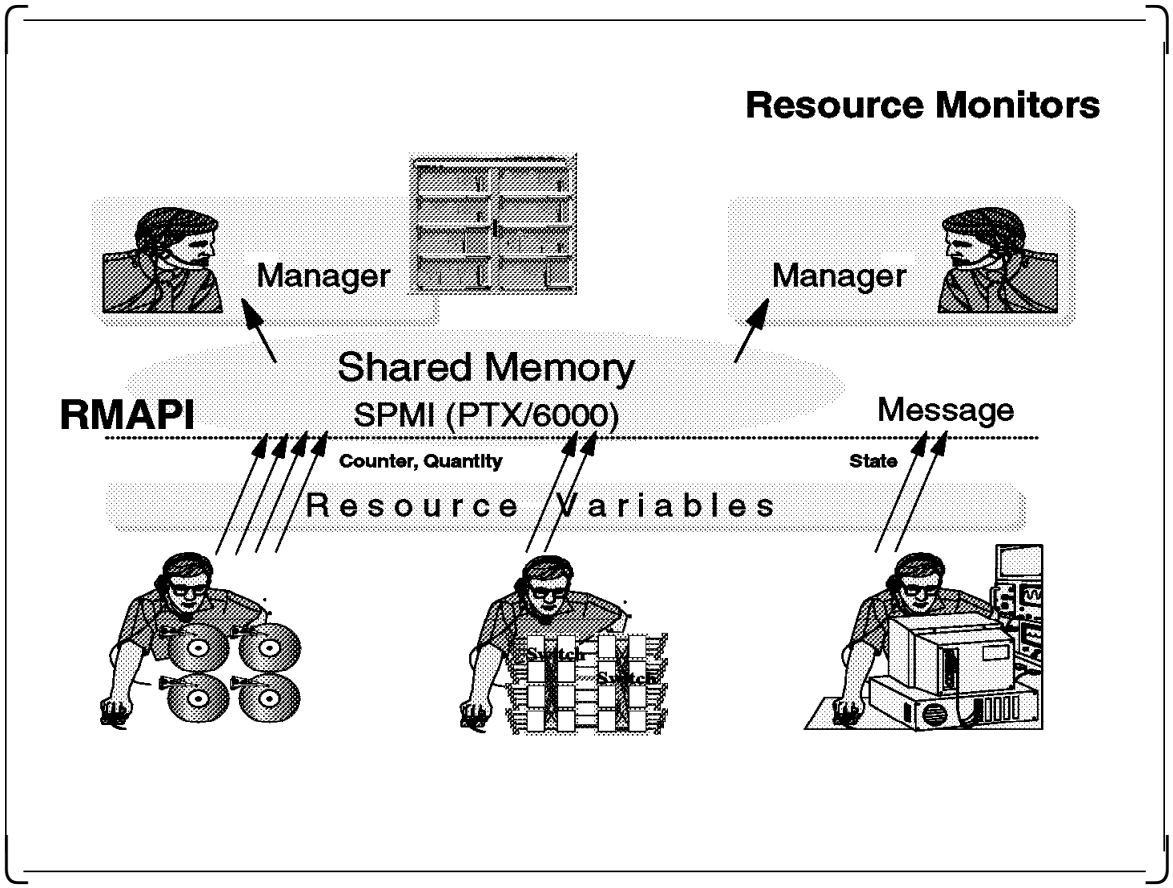
A *Structured Byte String (SBS)* is a string of bytes consisting of a four byte (32 bit) header followed by one or more structured fields. The SBS header specifies the total length of the structured fields that follow.

A structured field consists of a four byte header, followed by a value. The first two bytes of the header make up the length of the structured field value. The third byte is a structured field data type, and the fourth byte is an 8-bit serial number. The structured field type is one of the following:

- 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 value length. A byte string is some number of bytes, where each byte may have any value from 0 through 255. The serial number is a unique value that identifies the structured field. This serial number is defined by the Resource Monitor that supplies the SBS resource variable for each structured field. The set of serial numbers defined for the structured byte string starts with 0 and is contiguous.

4.8 Resource Monitors

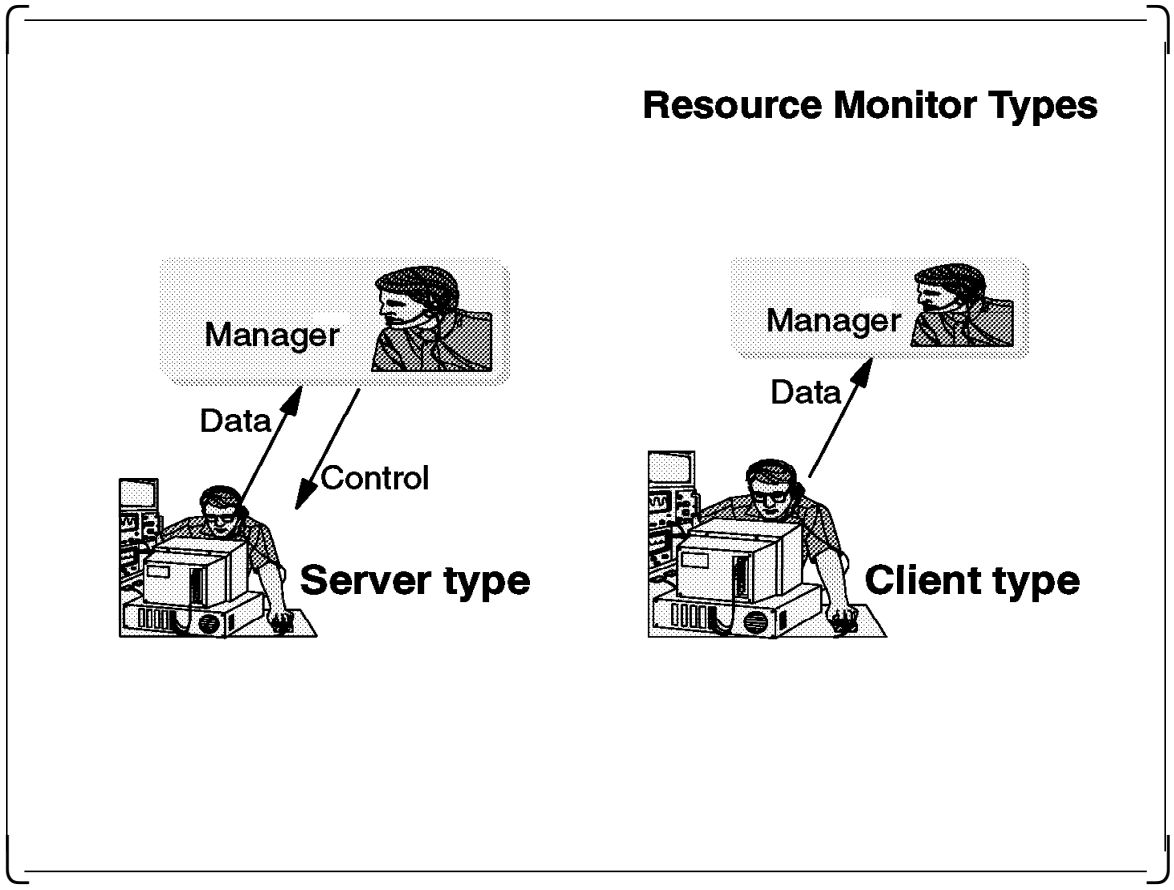


Resource Monitors are those software components that provide the actual resource variables to a higher level of manager software. The Resource Monitors are the system components that understand the structure of resources. They observe the state of specific system resources and transform that state into several resource variables. Then the Resource Monitors pass these variables to the manager software in a highly efficient manner.

The resource variables are passed to the manager software through a *Resource Monitor Application Programming Interface (RMAPI)*. This interface implements the System Performance Measurement Interface to transfer resource variables of type counter and quantity. Resource variables of type state are sent as a message directly to the manager software. The Performance Toolbox/6000 provides the System Performance Measurement Interface, which is a shared memory interface. This shared memory provides a mechanism for easily obtaining operating system resource variables.

Using the shared memory enables the system resource variables of type counter and quantity to be monitored by more manager softwares, like Performance Toolbox/6000 or the Event Management. There is a separate chapter dealing with the Event Management, Chapter 5, “Event Management” on page 135. Resource variables of type state are directly passed to the manager software.

4.9 Resource Monitor Types



Depending on the control that is applied in the Resource Monitor logic, there are two types of Resource Monitors:

Server type

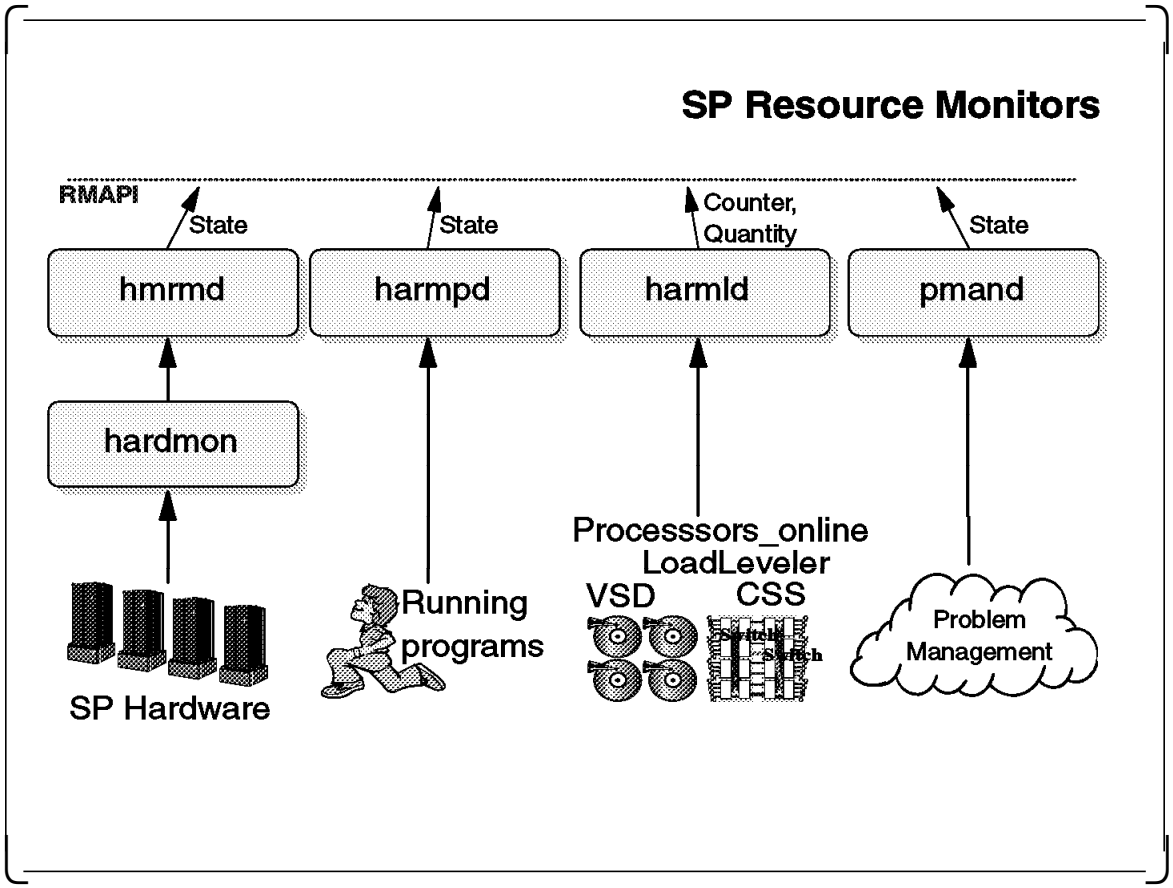
The server type of Resource Monitor expects the manager program to connect to the Resource Monitor. In this case, after the Resource Monitor program has started, the manager program sends control commands to the Resource Monitor. These commands control the flow of the resource variables. The Resource Monitor can be started by the manager program, or as a subsystem.

Client Type

The client type of Resource Monitor has the Resource Monitor logic implemented in a command. This type of Resource Monitor connects to the manager program to establish communication.

Note: The client type Resource Monitor does not expect control commands.

4.10 SP Resource Monitors



Shipped with the PSSP 2.2 are four external resource monitors. The Resource Monitors supply data to the Event Manager Daemon. The Event Manager Daemon is explained in section Chapter 5, “Event Management” on page 135.

IBM.PSSP.harmld

This monitor supplies resource variables from CSS, VSD, LoadLeveler, processor on-line information, and internal variables of the `harmld` daemon. The data is of type counter or quantity, and is transferred to the Event Manager Daemon through the SPMI shared memory interface. Therefore, the data is also furnished to the performance monitoring subsystem. This is a daemon server type.

IBM.PSSP.harmpd

This monitor supplies the resource variables that represent the number of processes executing a particular program. These variables can be used to determine if a particular system daemon is running or not. All variables are of type state and are sent directly to the Event Manager Daemon. All the variables are of SBS format. This is a daemon server type.

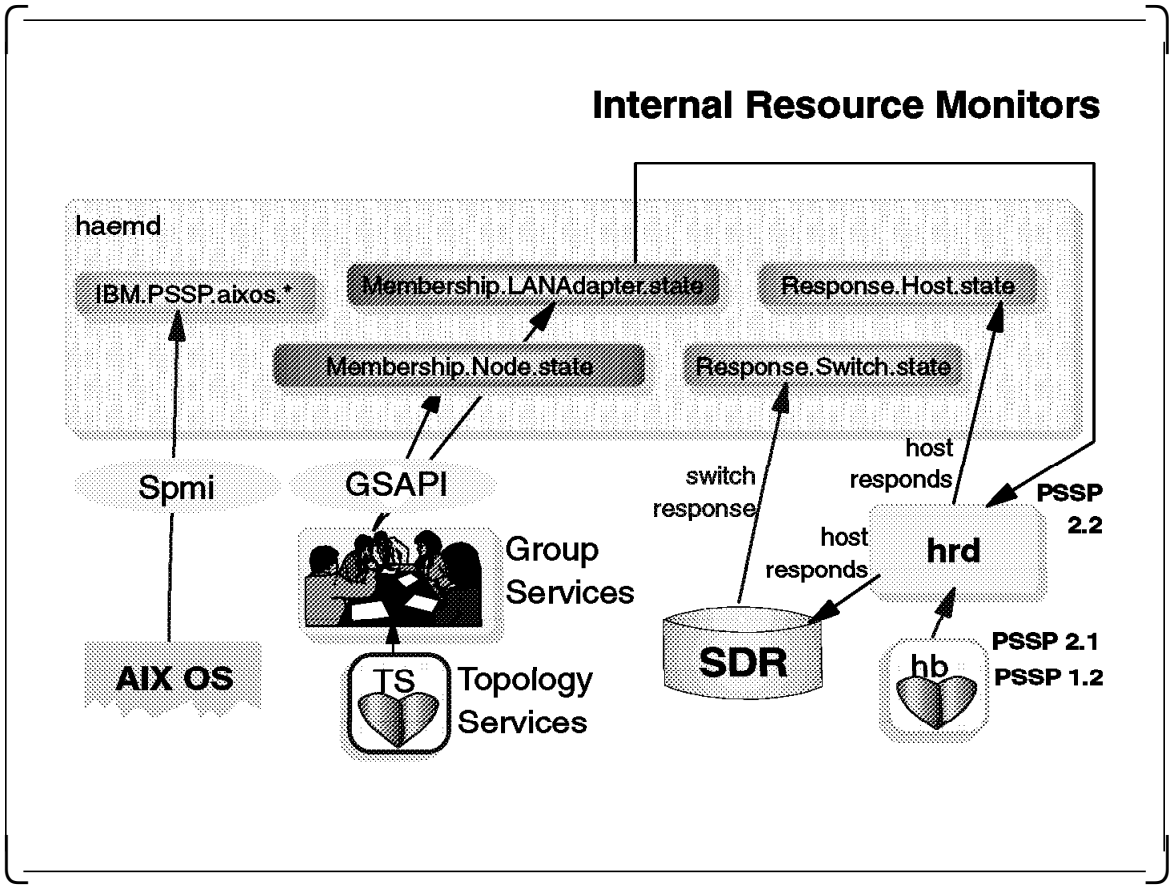
IBM.PSSP.hmrmd

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 Manager Daemon directly as a message. This is a daemon server type.

IBM.PSSP.pmanrmd

This monitor supplies the resource variables provided by the PSSP Problem Management subsystem. The variables are of type state and are transferred to the Event Manager Daemon directly as a message. This is a command-based client type.

4.11 Internal Resource Monitors



There are three internal types of Resource Monitors. These Resource Monitors are internal to the Event Management structure, which is explained in section Chapter 5, “Event Management” on page 135.

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.” This Resource Monitor supplies data into two resource variables:

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

The *IBM.PSSP.Membership.Node.state* resource variable represents the state of each node. This information is taken from the nodeMembership group from the Group Services subsystem. The Group Services receive the information about this group from the Topology Services subsystem. This resource variable has an instance vector of node number (NodeNum).

The *IBM.PSSP.Membership.LANAdapter.state* resource variable represents the state of communication adapters. This information is

taken from the enMembership group, and information about the cssMembership groups is taken from the Group Services subsystem. The Group Services receive the information about this group from the Topology Services subsystem. This resource variable has an instance vector of node number, adapter type, and adapter number (NodeNum,AdapterType,AdapterNum).

Response

This monitor supplies the resource variables that represent the information in the SDR classes host_responds and switch_responds. The resource variables are:

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

These resource variables are provided for compatibility with prior releases of the PSSP.

The *IBM.PSSP.Response.Host.state* resource variable that represents the *host response* information is obtained from the Host Response Daemon hrd. For the nodes running PSSP at level 2.2, hrd obtains information about the state of the nodes from the Event Manager Daemon. The hrd is using the client Event Management API and receives events from the IBM.PSSP.Membership.LANAdapter.state, with instance vector (NodeNum=*, AdapterType=en, AdapterNum=0). For nodes running at a lower level of PSSP than 2.2 (level 2.1 or level 1.2), the hrd obtains information about the state of nodes from the PSSP Heartbeat (hb) daemon. The information about the state of the nodes is then put in the IBM.PSSP.Response.Host.state resource variable. At the same time, the hrd daemon updates the SDR host_responds class.

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

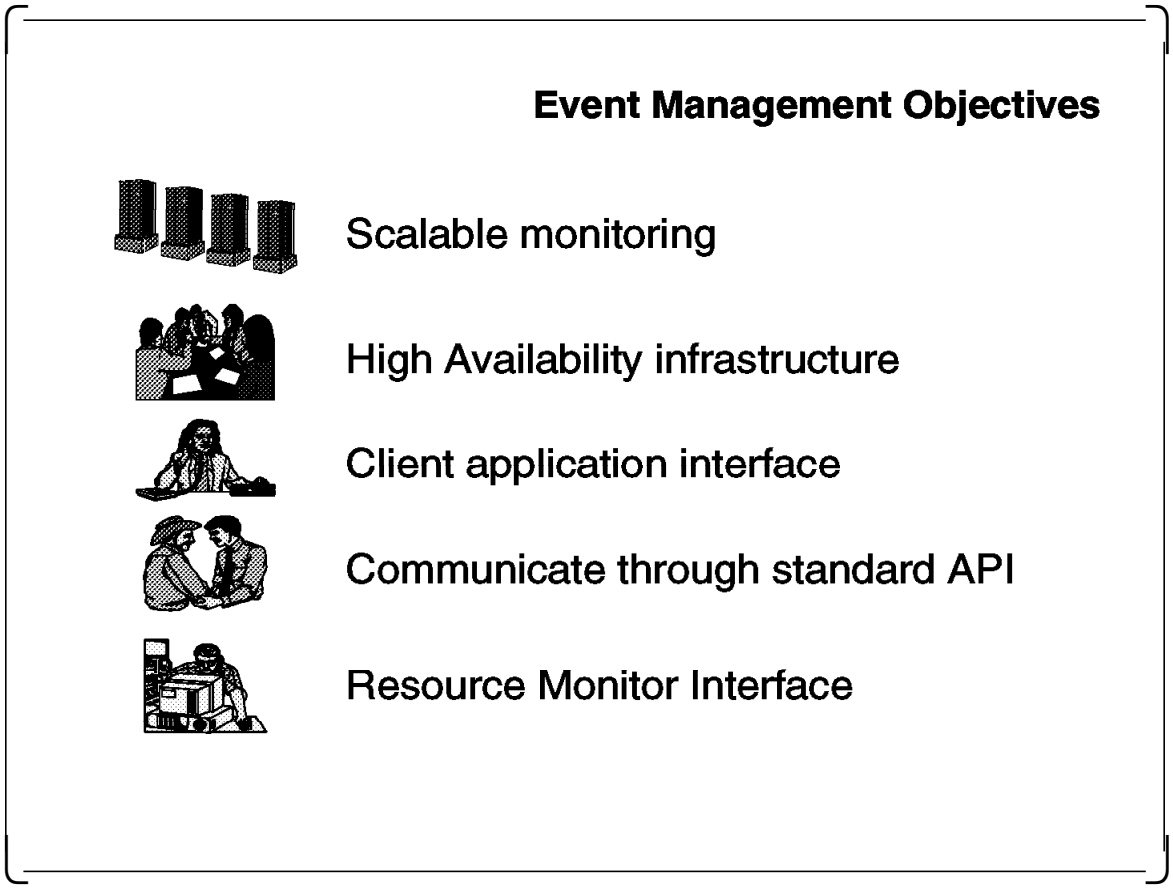
aixos

This monitor supplies the resource variables that represent AIX Operating System resources like CPU (idle, kern, user, wait), disks, file systems, LAN, memory, paging space, and processes (runque, swpque). The aixos Resource Monitor supplies resource variables IBM.PSSP.aixos.* and uses the System Performance Measurement Interface shared memory interface.

Chapter 5. Event Management

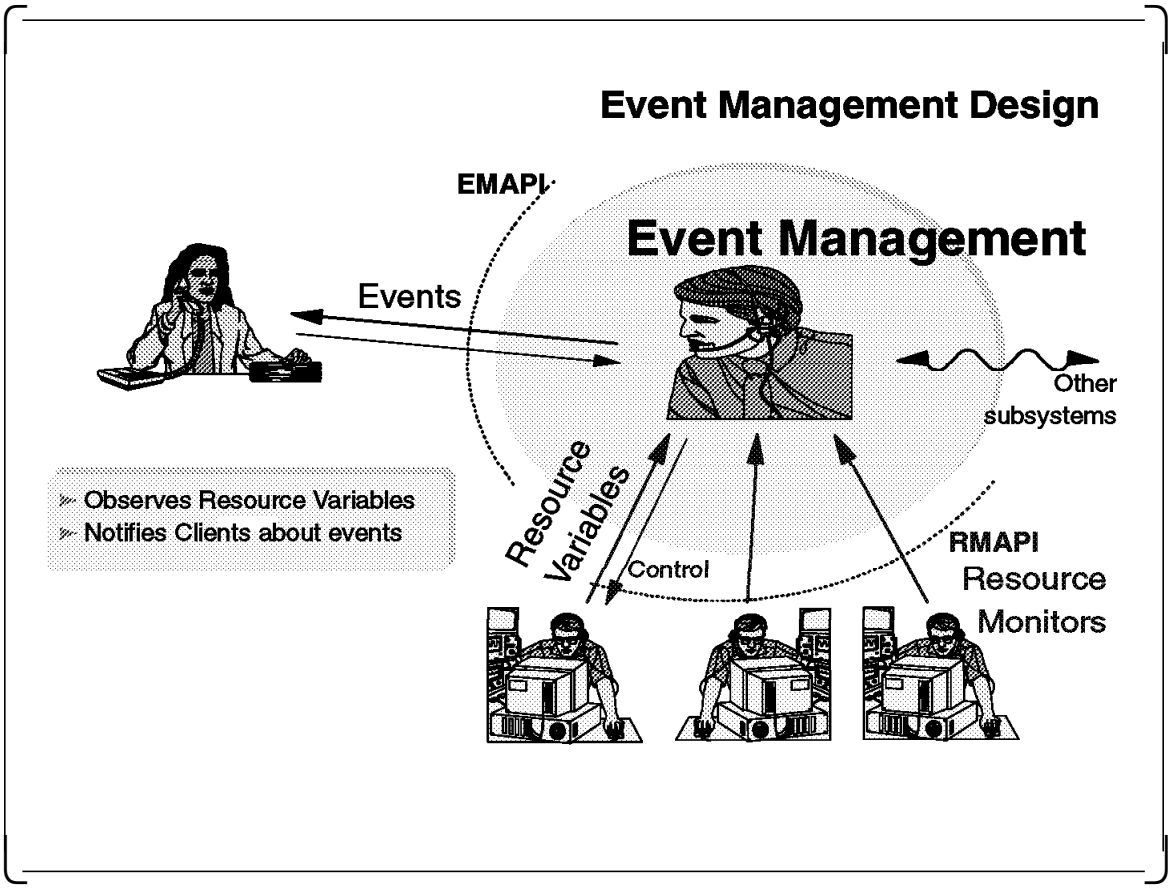
This chapter describes the Event Management associated with the RS/6000 SP High Availability Infrastructure.

5.1 Event Management Objectives



The Event Management must meet the requirements of scalable architecture and efficient notification structure. The Event Management is required to use a highly available infrastructure in order to coordinate its daemons. The monitoring subsystems use a standard API to transfer information. The information is processed and the generated events are passed to the client software that is interested in them.

5.2 Event Management Design



The *Event Management* subsystem provides comprehensive event detection by monitoring various hardware and software resources.

The Event Management subsystem consists of three types of components:

Event Manager Daemon

The Event Manager Daemon (EMD) is the central program that receives resource variables from the Resource Monitor, processes those resource variables and generates events that are transferred to the clients. The Event Manager Daemon notifies the client programs that have registered their interest in the events. Events are created by the Event Manager Daemon based on the state of the resources as reported from the Resource Monitor. The Event Manager Daemon uses two interfaces to communicate with client programs and the Resource Monitor.

Client programs

Client programs are applications or other subsystems that wish to receive event information when resources change state. Client programs use an Event Manager Application Programming Interface (EMAPI) for communication with the Event Manager Daemon. The clients define events by specifying predicates.

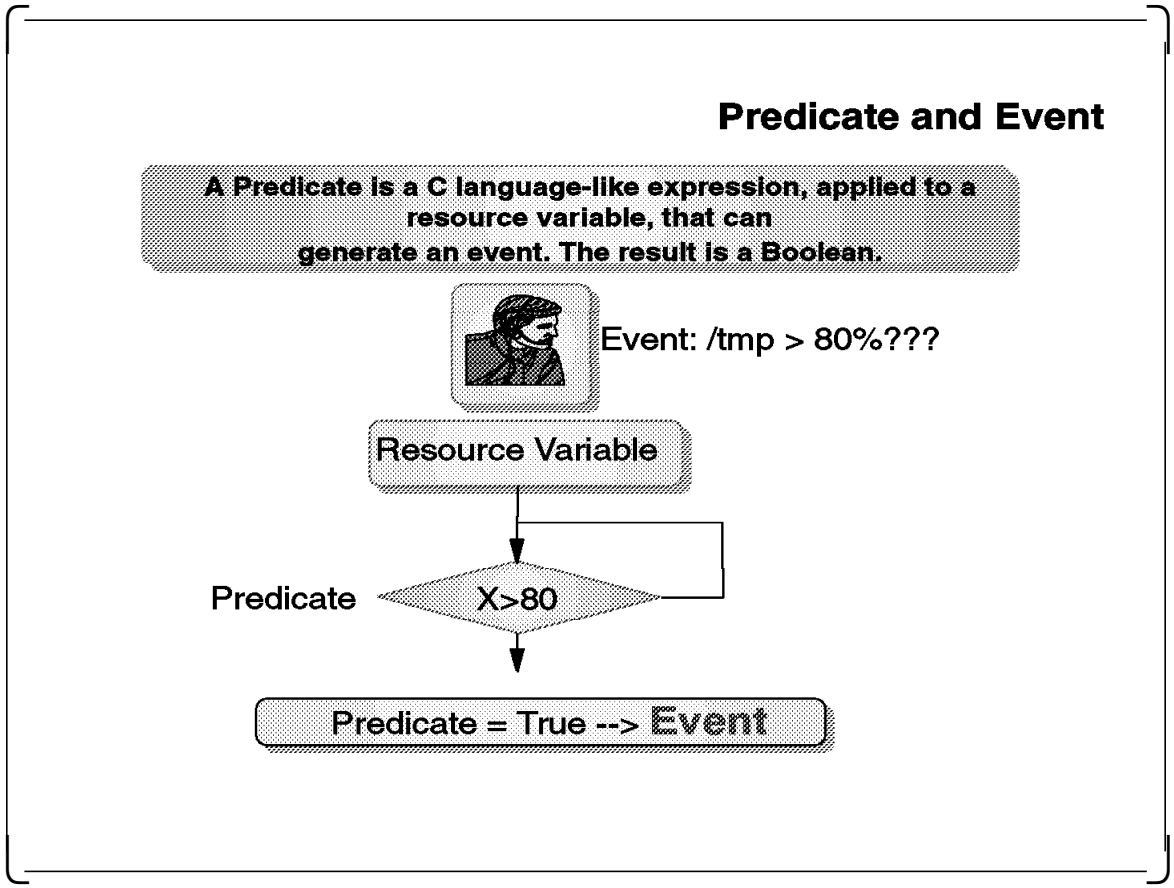
Resource Monitors

The *Resource Monitors* provide resource variables, which represent the states of the resources to the Event Manager Daemon. The Resource Monitors provide data to an Event Manager Daemon located on the same node. Resource Monitors provide their data to the Event Manager Daemon through the Resource Monitor Application Programming Interface (RMAPI). This interface is also the way the Event Manager Daemon controls the Resource Monitor.

There is an information flow from the lower level Resource Monitors to the Event Manager Daemon. This flow consists of a series of resource variables reported from the Resource Monitors. The Event Manager Daemon generates events and notifies client programs about them.

There are also relations from the Event Manager Daemon to other subsystems, see section 5.4, “Distributed Event Management” on page 141 for more details.

5.3 Predicate and Event



5.3.1 Predicate

A predicate is a C language-like expression containing a resource variable, in one or more modified forms. The evaluation result of a predicate is Boolean. This evaluation occurs each time a resource variable is observed. The form of a predicate is illustrated in the following example:

```
predicate::   expression    rel_op    constant
              expression    rel_op    expression
              predicate     log_op    predicate
```

In this example, the expression may be a variable name (var_name), or it may be in the following form:

```
expression::    var_name
                var_name      arith_op    constant
                var_name      arith_op    expression
```

The resource variable name (var_name) is represented by an uppercase X. This uppercase X may be followed by a modifier, as shown in the following example:

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

The variable name modifier is one of the following:

X@P This refers to the previous value of the resource variable.

X@R This refers to the raw value.

X@PR This refers to 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 this serial number.

X@Psbs_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 this serial number, which was returned from the previous observation.

In the examples, rel_op represents one of the following relational operators: ==, !=, <, >, <=, >=.

In the examples, arith_op represents one of the following arithmetic operators: *, /, %, +, -.

In the examples, log_op represents one of the following logical operators: &&, ||.

Operators have the same meaning and precedence as in the C language, and parentheses may be used for grouping, also as in C. Constants are integers or floating points. The variable modifier R is useful for specifying the raw value with a resource variable of type counter.

Following are examples of some predicates:

```
X>90                    The Resource Variable is greater than 90.
(X@0!=X@1)&&(X@2<20)    The first field (sbs_sn=0) of a structured
                          byte string is not equal to the second
                          field (sbs_sn=1) of the structured byte
                          string, and at the same time, the third
                          field (sbs_sn=2) of the structured byte
                          string is less then 20.
```

The clients specify the predicate.

5.3.2 Event

The predicate is the main input for event generation. An *event* is generated if the Boolean evaluation of a predicate results in the value TRUE.

5.3.3 Resource Variable Observation

The resource variables are observed in two ways:

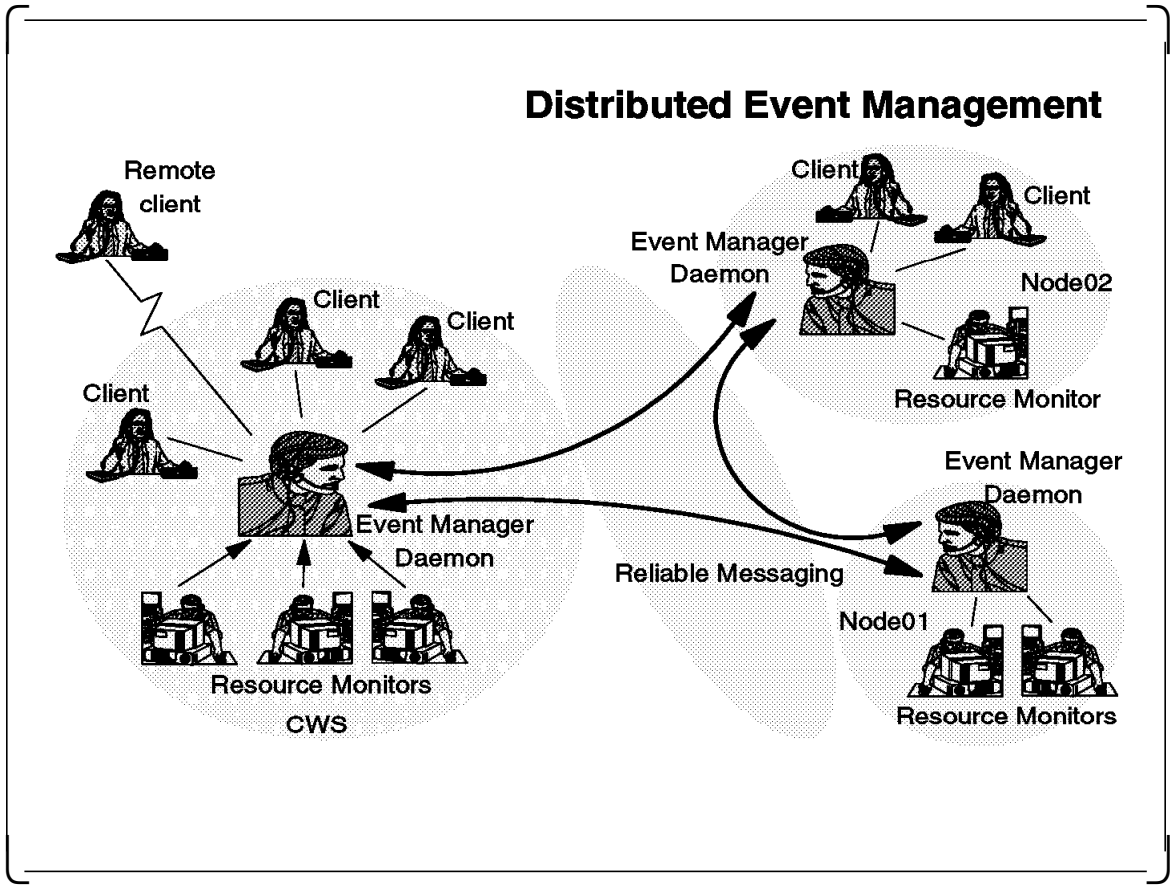
Counter, Quantity

These resource variables are observed at an interval that is configurable.

State

Resource variables of type state are observed when the value is received from a Resource Monitor.

5.4 Distributed Event Management



The Event Management is designed to cooperate with other Event Manager Daemons in the system. The Event Manager Daemon communicates with Event Manager Daemons on other nodes in order to provide data to its local clients. Therefore, a client may register for and receive events about any resource in the SP system. These communication links are called *peers*. The peer communication channels are internal to the Event Management subsystem. The communication path between Event Manager Daemons is based on the Reliable Messaging interface. The Reliable Messaging is part of the Topology Services described in Chapter 2, “Topology Services” on page 5.

The Resource Monitors always connect to the local Event Manager Daemon, because they use the shared memory interface and UNIX domain sockets to transfer resource variables to the Event Manager Daemon. If the Resource Monitors were remote, they would require a high communication load to the network, and from the communication needs, the CPU performance could be affected, too.

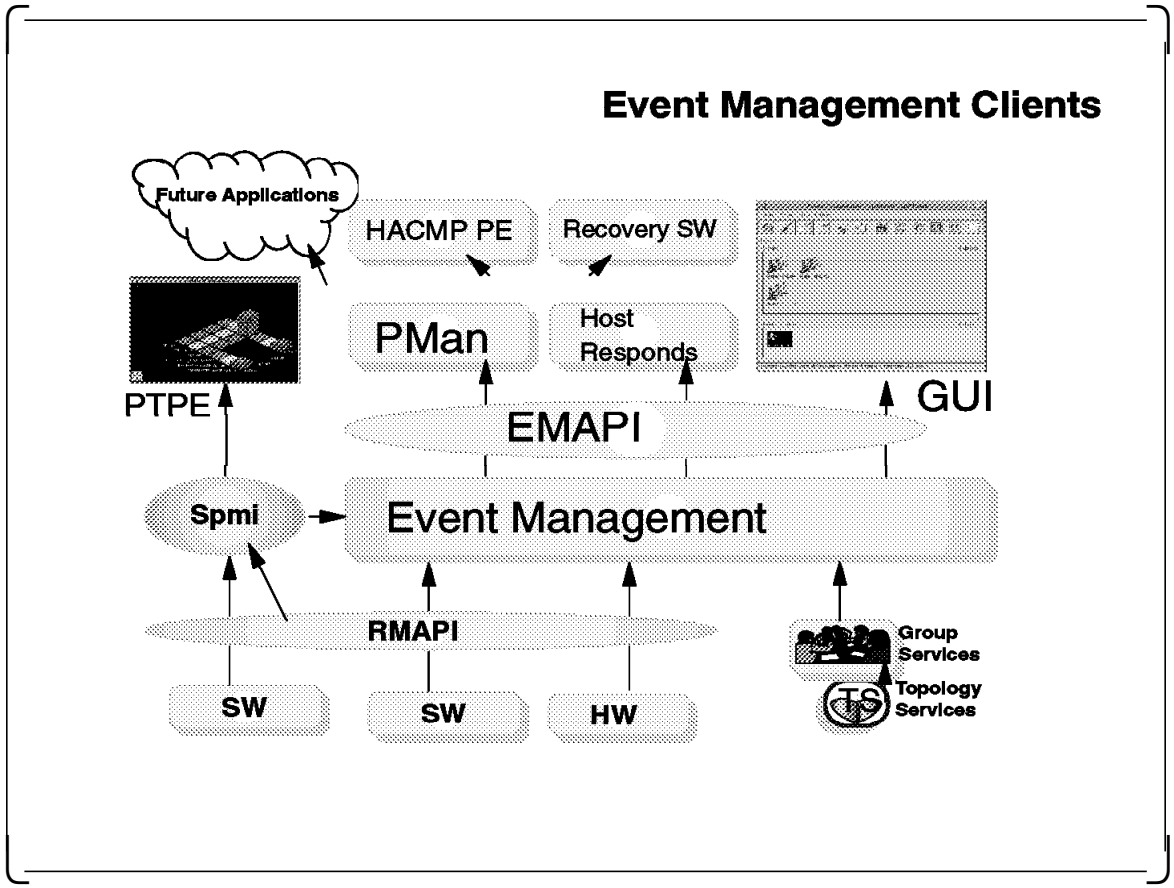
Note: The Resource Monitors are always local to the Event Manager Daemon.

The clients connect to the Event Manager Daemon. This connection can be local or remote. The communication between a client and the Event Manager Daemon is significantly low, because the Event Manager Daemon does not send every observed resource variable, but sends only the registered events. The local clients use UNIX domain sockets. The remote clients use TCP internet

sockets. The remote clients connect to the Event Manager Daemon only on the Control Workstation.

The Event Management in the SP system understands a system as a system partition. Every system partition is a separate operational domain for the Event Management subsystem. There is only one Event Manager Daemon on each node. On the Control Workstation, there is one Event Manager Daemon for each partition.

5.5 Event Management Clients

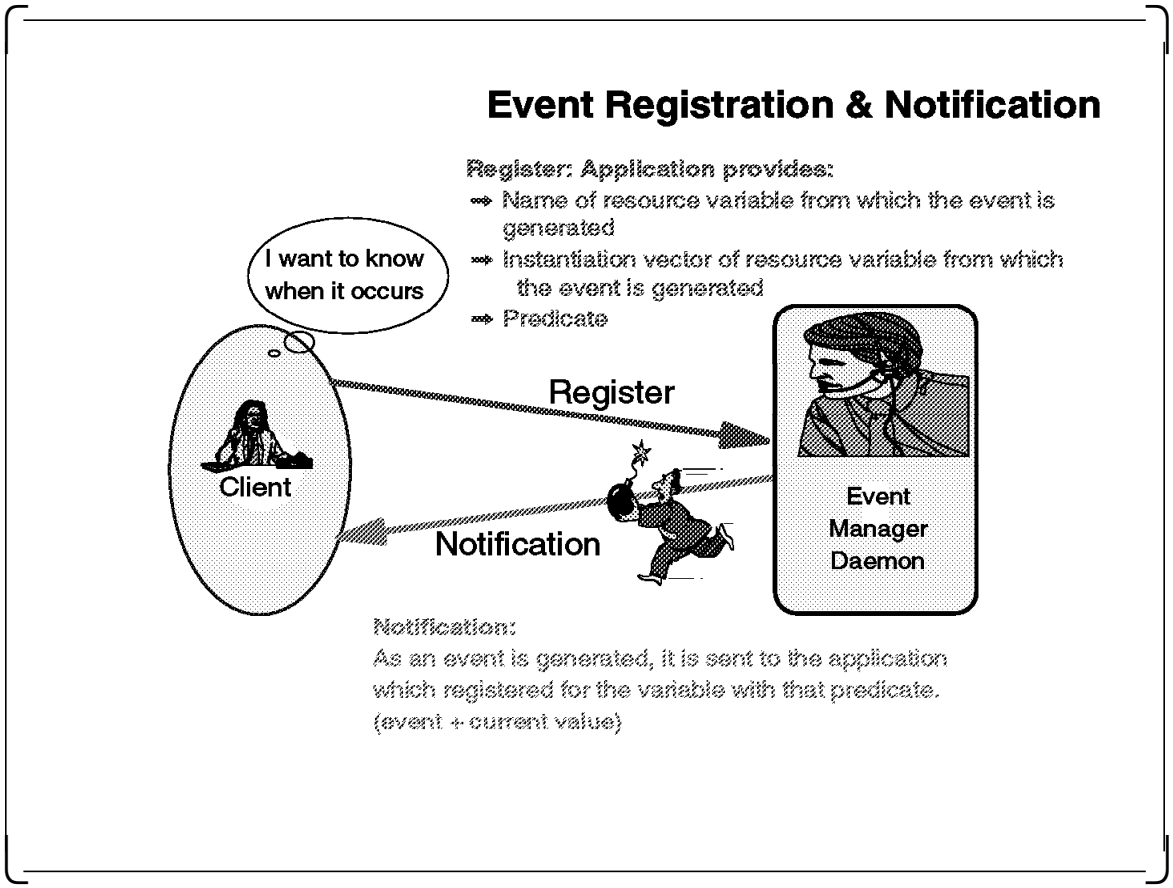


The Event Manager Daemon is the high availability structure component that interfaces with most RS/6000 SP subsystems. The Performance Toolbox/6000 System Performance Measurement Interface is used for transferring data from the Resource Monitors to the Event Manager Daemon. There is more than one Resource Monitor supplying resource variables to the Event Management. These resource variables are observed, and if an event is generated, it is sent to the registered clients through the EMAPI interface. The EMAPI interface has connected clients, like the Host Responds daemon. The Perspectives graphical user interface receives information about the RS/6000 SP system from the Event Manager Daemon through the EMAPI.

The Problem Management software of the PSSP Version 2 Release 2 uses both of the interfaces, EMAPI and RMAPI. This software is described in Chapter 4, “Resource Monitors” on page 119.

From the non-PSSP application, the HACMP PE takes advantage of receiving events from the Event Manager Daemon. Any recovery software that reacts as fast as possible on events that have occurred in the system will benefit from the Event Management architecture.

5.6 Event Registration and Notification



This section explains how to determine which client to notify when an event occurs.

Every client interested in an event has to inform the Event Management subsystem about its interest in that particular event. The process that an application or subsystem uses to declare its desire to be notified of these events is called *event registration*.

The application or client subsystem provides the following information to the Event Management subsystem:

- Name of the resource variable from which the event is generated
- Instance vector of the resource variable

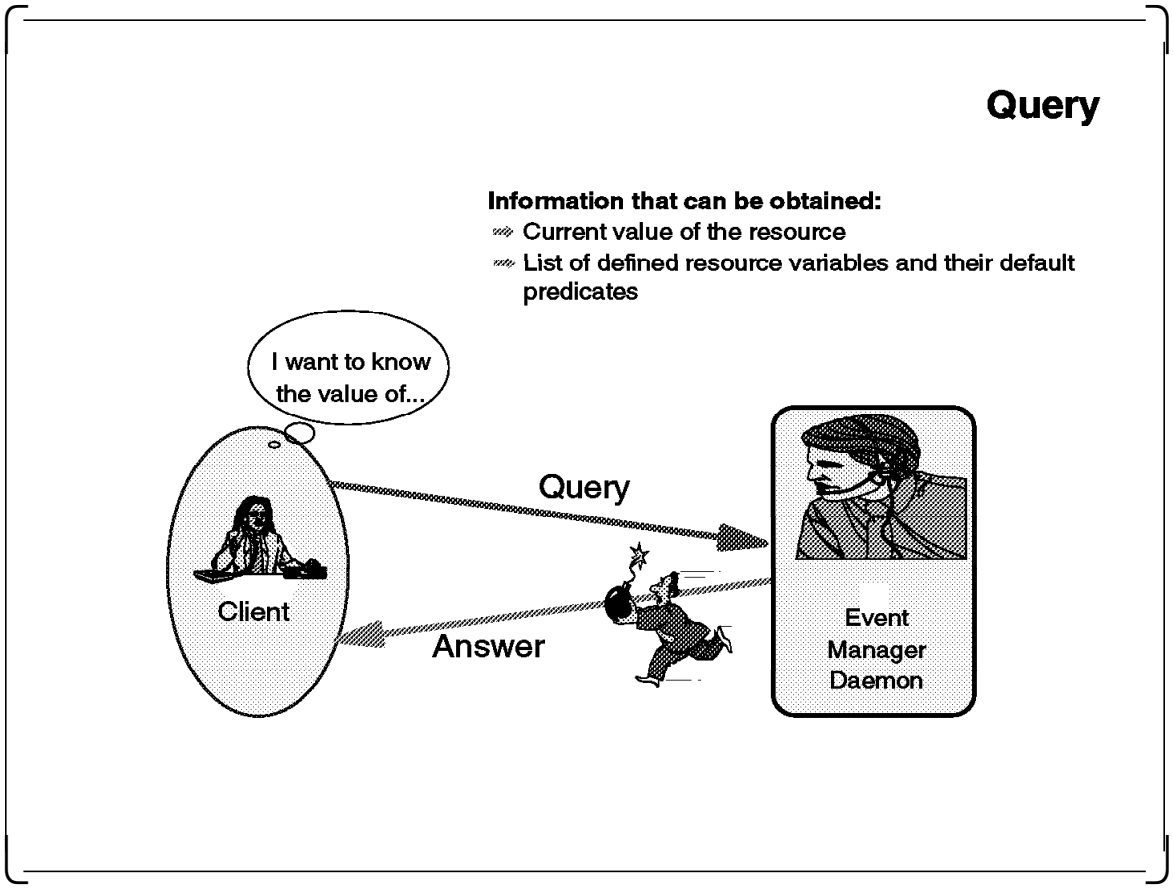
  Note: This instance vector may be wildcarded.

Each resource variable is optionally defined with a single predicate for event generation. The application or subsystem may provide predicates to be applied against the variable instead of the default predicate. These predicates are used only for the calling application (client).

When an event is generated by applying the predicate to the resource variable, this event is sent to the clients that registered for that variable with that predicate. In this way, the Event Manager Daemon *notifies* the client about an

event. Clients that registered for a resource variable without specifying predicates receive events generated by applying the default predicate. Included with the event information is the current value of the resource variable specified in the predicate.

5.7 Query



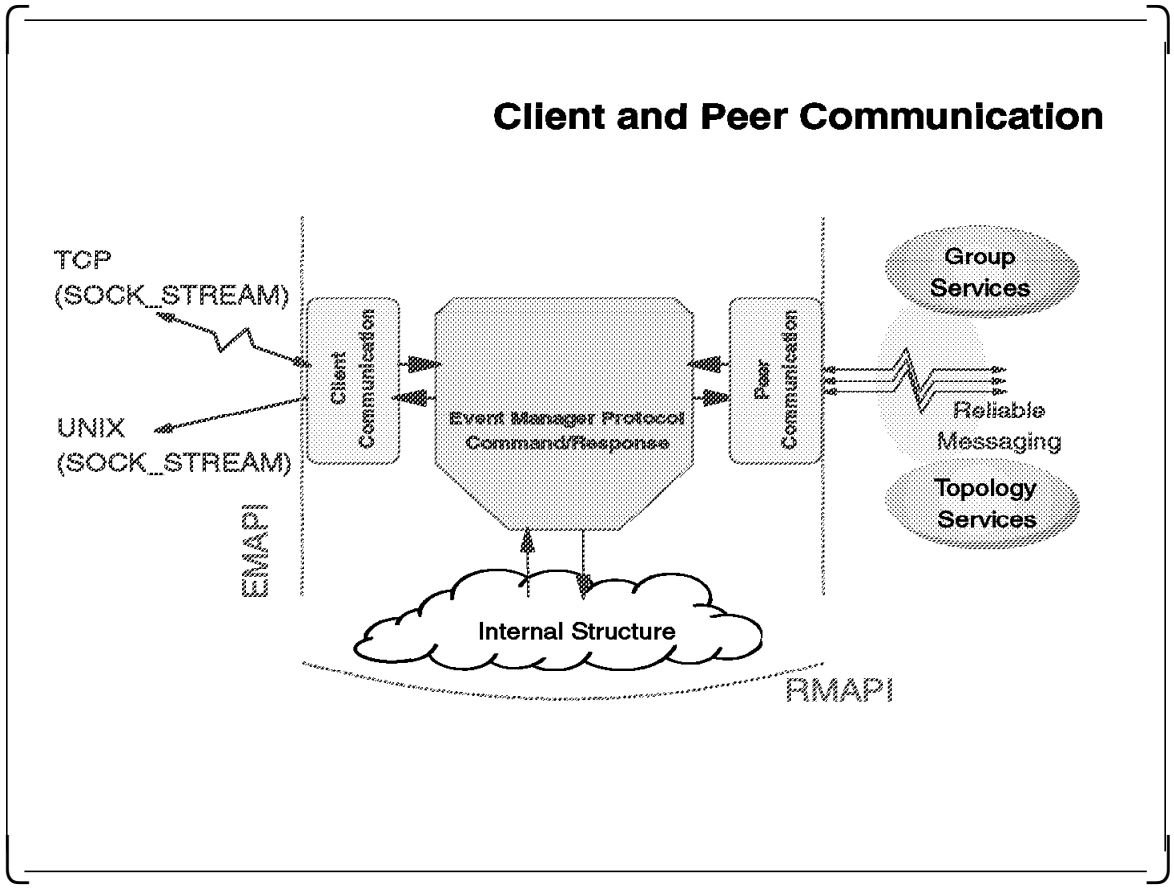
In addition to registering and receiving notification about events, the client subsystem or application may need to get information about the status of the resource variables, events, predicates, and registrations. The Event Management interface may query the following information:

- The current value of a resource variable
- A list of defined resource variables and their default predicates

The scope of each of these queries is defined by the specification of the resource variable name and instance vector.

Note: These specifications may be wildcarded.

5.8 Client and Peer Communication



The *Client Communication Interface* provides the logic necessary to send and receive messages with Event Management clients. The actual communication mechanism employs UNIX domain sockets of type SOCK_STREAM. This mechanism ensures reliable communication between clients and the Event Manager Daemon, including notification that one end of a connection has closed.

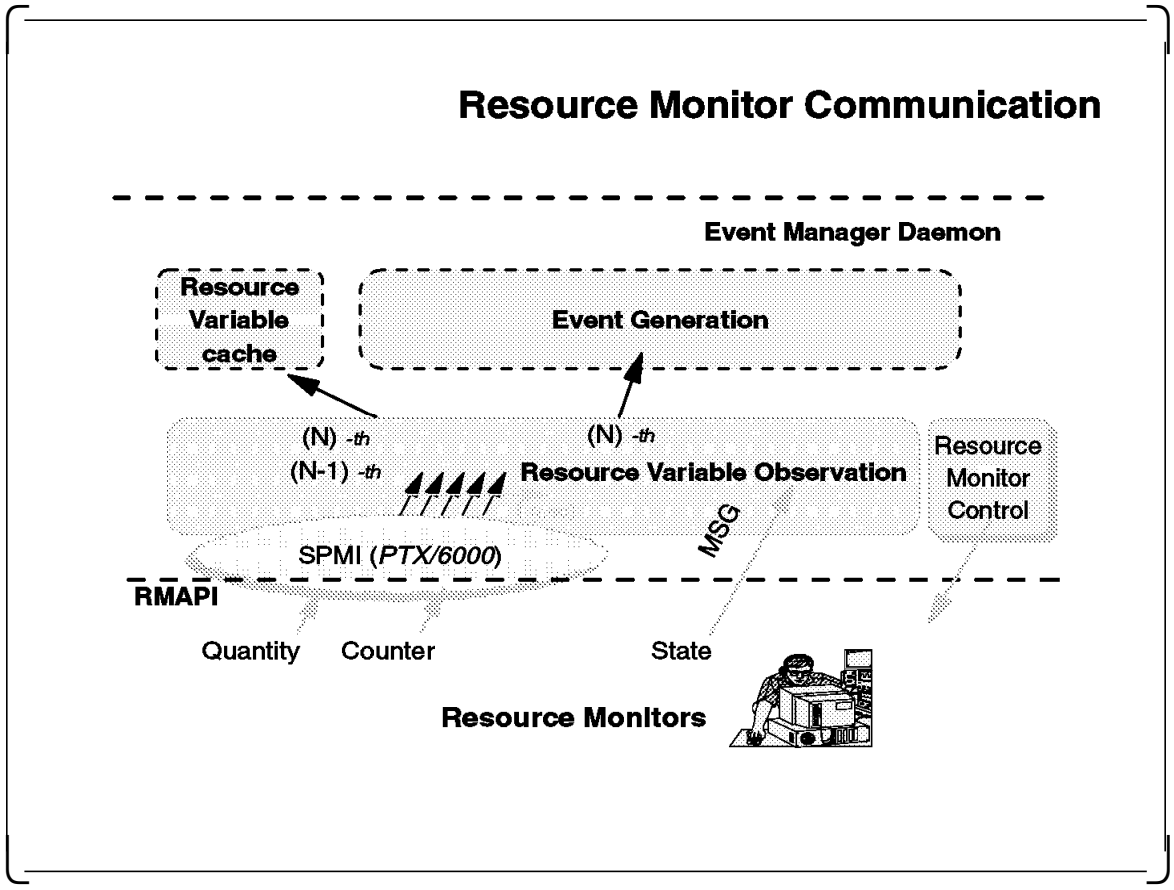
If a client is remote, then an internet domain socket of type SOCK_STREAM is used. The TCP protocol cannot always inform a program in a timely manner that the remote end of its session has crashed. Therefore, the reliability of remote connections between clients and the Event Manager Daemon is less than that of local connections.

During the initialization processing of the Event Manager Daemon, the Client Communication Interface module establishes a server socket in the UNIX domain, binding to a name known to clients. If the Event Manager daemon is on the Control Workstation, this module also establishes a server socket in the internet domain. The mainline logic of the daemon uses the select() system call to wait for connection requests from clients on these sockets. When select() indicates that data is available on any client socket session, this module is invoked to read the data or to process termination of the client connection. Data is passed to further Event Manager Daemon internal modules for parsing and forwarding to the appropriate logic module.

The Event Manager Daemon uses the Client Communication Interface module to send any data to the client.

The *Peer Communication Interface* provides the logic necessary to send and receive messages with other Event Manager Daemons in the system.

5.9 Resource Monitor Communication



The *Resource Variable Observation* provides the logic necessary to obtain resource variables from their respective Resource Monitors. The Resource Monitor sends resource variables to the Event Manager 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 using the Performance Toolbox/6000 System Performance Measurement Interface (SPMI).
- 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.

Resource variables that are located in shared memory are observed (read) by the Event Manager Daemon in every time period, where the time period is configurable. Resource variables are then queued to generate events, and they are also placed into the resource variable cache. The N-1th value is replaced by the Nth value, and then the new value is stored as the Nth value.

When a resource variable has been observed, a predicate is applied to this variable and an event may be generated.

The *Resource Monitor Control* module ensures that the Resource Monitor for each resource variable is running and sending resource variables to the Event Manager Daemon, unless the Resource Monitor is implemented as a command. A Resource Monitor that is a daemon can be started by the Event Manager Daemon. When the resource variables that are supplied by a Resource Monitor daemon are the target of a Register command, this module checks if the Resource Monitor is already running. If it is, and the Resource Monitor is not currently supplying the variables, this module sends an indication to the Resource Monitor to start sending the variables. If the Resource Monitor is not running, then it is started by this module.

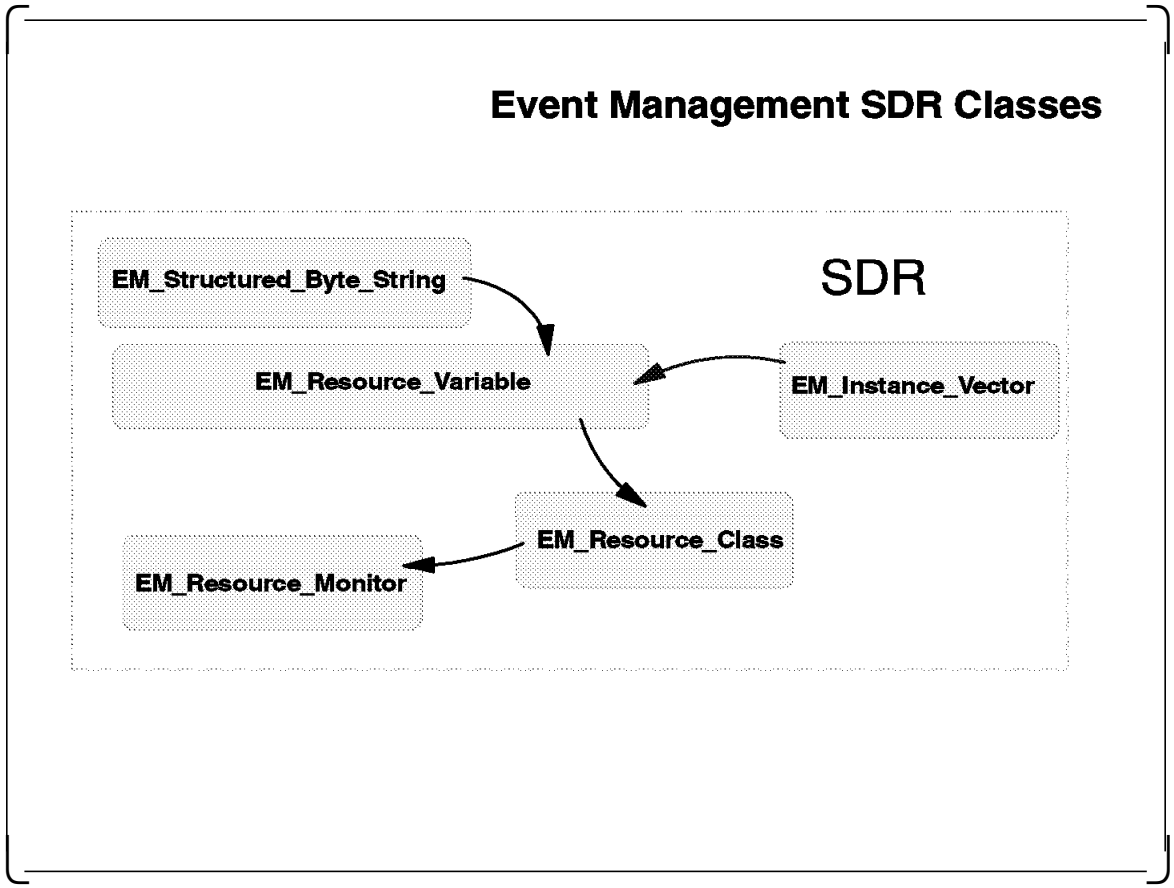
If the Resource Monitor is incorporated into a subsystem, then the Event Manager Daemon cannot start the Resource Monitor. If the Resource Monitor is running, the Resource Monitor Control module sends an indication to the Resource Monitor to start sending the variables. If the Resource Monitor is not running, this module periodically checks to see if it is running.

If the Resource Monitor is implemented as a command, it automatically sends its resource variables to the Event Manager Daemon. This is done so that the information may be available for later queries.

The Resource Monitor Control module establishes a communication path between the Event Manager Daemon and the Resource Monitor by using a UNIX domain socket of type SOCK_STREAM. This communication path is used to send control messages between the Event Manager Daemon and the Resource Monitor. It is also used to receive resource variables of type state from the Resource Monitor.

When there is no longer any interest in a resource variable by any client, the Resource Monitor Control module sends a control indication to the Resource Monitor to cease sending resource variables.

5.10 Event Management SDR Classes



The Event Management subsystem requires information that defines the resource variables and describes how to obtain them. This information is stored in the SDR database in the following object classes:

EM_Resource_Monitor

This SDR class contains information about every Resource Monitor program.

EM_Resource_Class

This class contains information about the common characteristics of resource variables. This class groups together resource variables with the same basic characteristics. It references the EM_Resource_Monitor for the Resource Monitor definitions.

EM_Resource_Variable

This class contains information about all resource variables. It references the EM_Resource_Class for common characteristics. It is coupled with EM_Instance_Vector and EM_Structured_Byte_String classes.

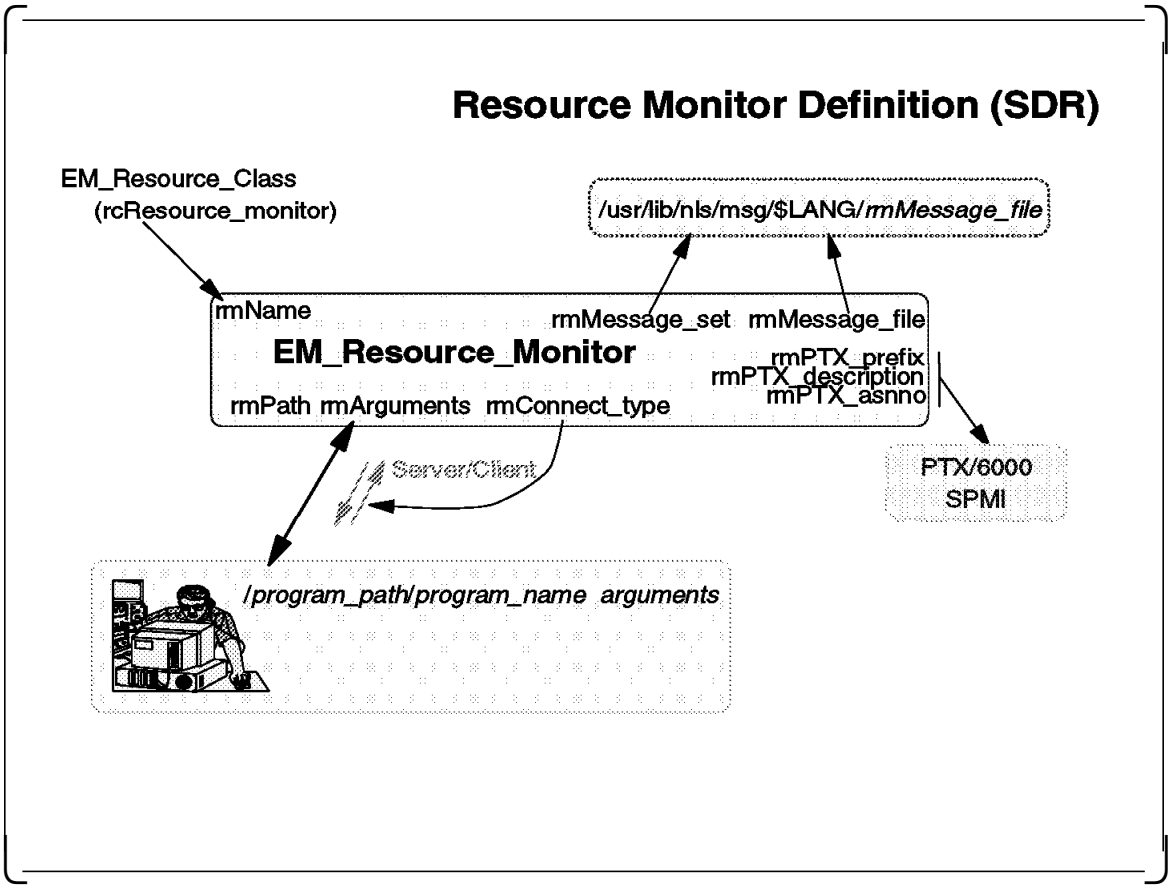
EM_Instance_Vector

This class describes the instance vector for particular resources.

EM_Structured_Byte_String

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

5.11 Resource Monitor Definition



The information about Resource Monitors that may connect to the Event Manager Daemon is stored in the EM_Resource_Monitor SDR class. Each Resource Monitor that may supply resource variables to the Event Manager Daemon has a Resource Monitor definition in the configuration database. Following are the SDR attributes included in a Resource Monitor definition:

rmName

This is the *name* of the Resource Monitor definition. This name must be unique in the database, and it uniquely identifies the SDR object in the EM_Resource_Monitor class. The rmName 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 Manager Daemon. If the Resource Monitor is not startable by the Event Manager Daemon, this entry is a NULL string.

rmArguments

This is a string of optional arguments that are passed to the executable when running the Resource Monitor daemon.

rmMessage_file

This specifies the name of the message catalog file for the Resource Monitor.

rmMessage_set

This specifies the message set within the message catalog.

rmConnect_type

This indicates whether the Event Manager Daemon connects to the Resource Monitor (“Server”) or the Resource Monitor connects to the Event Manager Daemon (“Client”).

rmPTX_prefix

This defines the PTX/6000 prefix used to supply data to the PTX/6000 shared memory interface (SPMI).

rmPTX_description

This is a string that contains a comma-separated list of message IDs. Each ID specifies a message that contains a short description (maximum 63 bytes) of a PTX context.

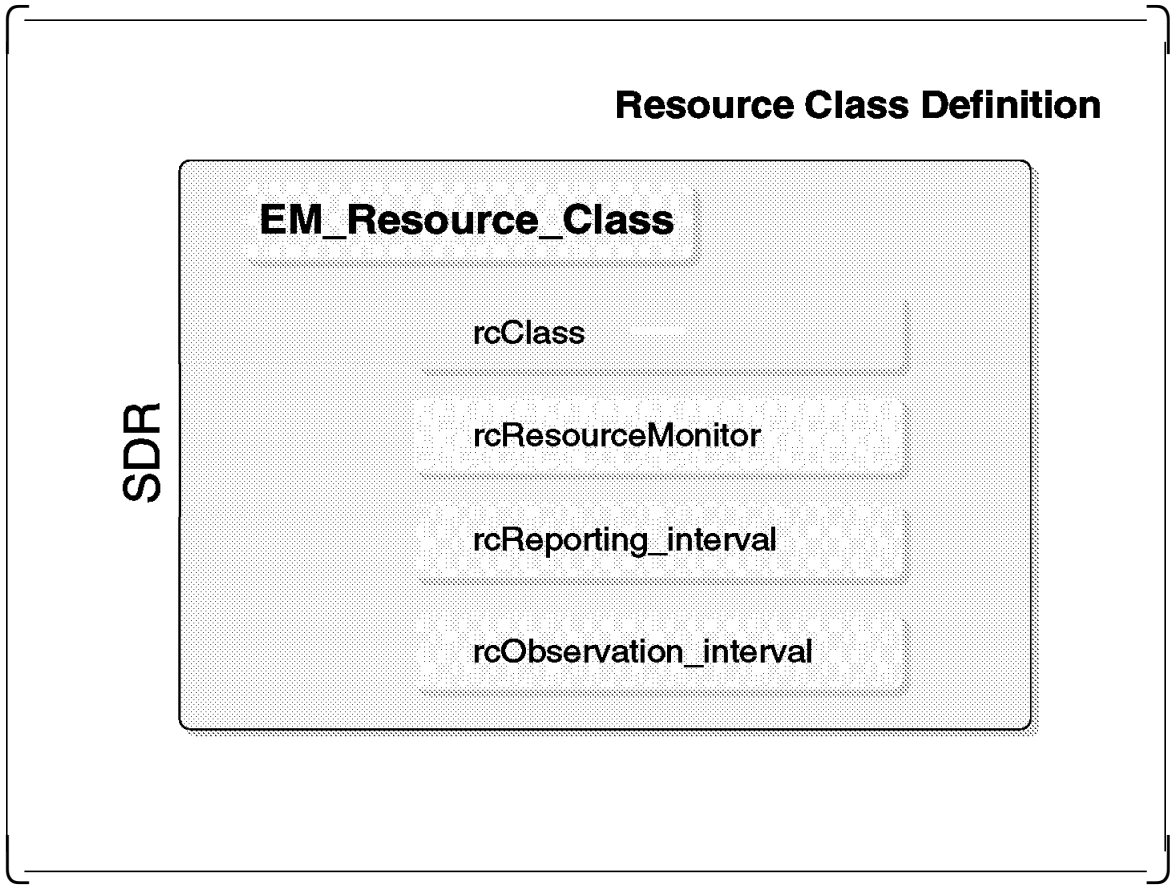
rmPTX_asnno

This is an integer that is the ASN.1 number equal to the SNMP Assigned Enterprise Number for the vendor that supplies the Resource Monitor. For IBM-supplied Resource Monitors, this value is 2.

The SDR objects for the Resource Monitor definition can be created by SDR commands (SDRCreateObjects, SDRListObjects, and so on). To create a sample Resource Monitor, use this example of the SDRCreateObjects command:

```
SDRCreateObjects EM_Resource_Monitor
        'rmName=IBM.PSSP.SampleDaeMon'                  \
        'rmPath='$rmpath                                \
        'rmMessage_file=rmapi_smp.cat'                  \
        'rmMessage_set=1'                               \
        'rmConnect_type=server'                         \
        'rmPTX_prefix=IBM/PSSP.SampleDaeMon'            \
        'rmPTX_description=1,2'                         \
        'rmPTX_asnno=2'
rc=$?; echo $rc
```

5.12 Resource Class Definition



A *class* of a resource variable is used to denote the subsystem which manages the associated resource. In other words, the *resource class* is used for grouping resource variables with the same resource characteristics, like supplying Resource Monitor, observation frequency, and reporting frequency. The actual name is to be unique within the resource class definition.

The following items are included in a resource class definition:

rcClass

This *class name* represents a name that uniquely defines the resource class. The rcClass element is referenced from the EM_Resource_Variable SDR class.

rcResource_monitor

A string that contains the name of the Resource Monitor definition for the Resource Monitor that supplies the resource variables in this class.

rcObservation_interval

The *observation interval* specifies the amount of time, in seconds, between each Event Manager Daemon observation of any counter or quantity resource variables in this class. This value must be greater than, or equal to, the Reporting Interval.

rcReporting_interval

The reporting interval is an integer number that is the amount of time, in seconds, between each update of any variable of value type counter or quantity in this class by the corresponding Resource Monitor.

The value of the Reporting interval may be zero (0) if the design of the Resource Monitor fixes the interval between variable updates. It may also be zero if the Resource Monitor is incorporated into a subsystem and the subsystem updates the variable as part of its normal execution.

The SDR objects for the Resource Class definition can be created by SDR commands (SDRCreateObjects, SDRGetObjects, and so on). Following is an example of the SDRCreateObjects command that creates a sample resource class:

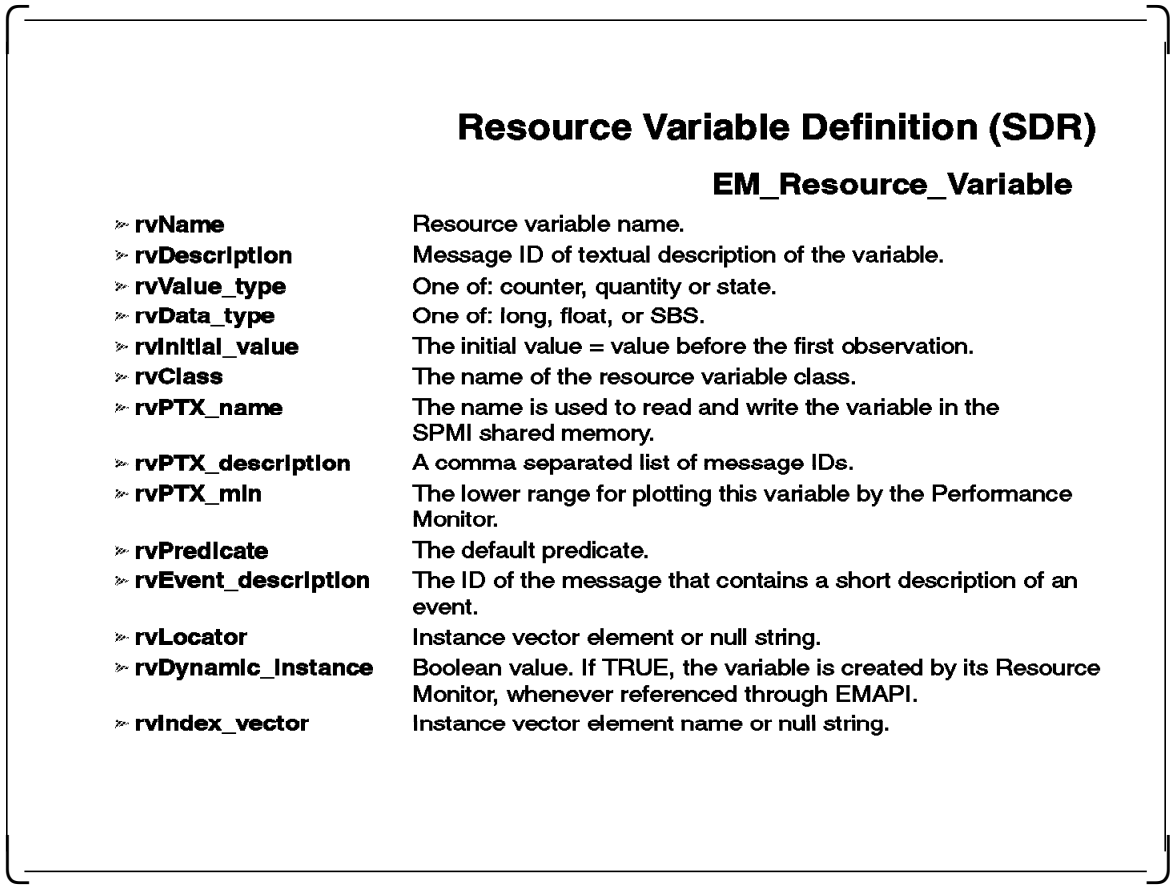
```
SDRCreateObjects EM_Resource_Class
        'rcClass=IBM.PSSP.SampleDaeClass'                  \
        'rcResource_monitor=IBM.PSSP.SampleDaeMon'         \
        'rcObservation_interval=10'                        \
        'rcReporting_interval=10'
rc=$? ; echo $rc
```

Following is an example of one of the seventeen resource classes that are shipped with the PSSP Version 2 Release 2 product:

```
IBM.PSSP.CSS is a class whose variables are supplied by the
     IBM.PSSP.harmld Resource Monitor with a reporting interval
     of 5 seconds, and the Event Manager Daemon observes the variables in the
     same frequency, that is, every 5 seconds.
     For the real resource variables names, see the
     EM_Resource_Variable SDR class definition.
```

Note: For details of the resources classes shipped with PSSP 2.2, see Appendix A, “Resource Class Definition” on page 289.

5.13 Resource Variable Definition (SDR)



The purpose of the resource variable definition is to inform the Event Management subsystem of the possible resource variables it is expected to manage. Any resource variable specified in the EMAPI call must first be defined in the configuration database. Items included in the Resource Variable Class follow:

rvName

This is a *resource variable name*, as described in section 4.4, “Resource Variable Name” on page 123.

rvDescription

This is a message ID of the textual description of the semantics of the variable.

rvValue_type

This is a *value type*, as described in section 4.6, “Resource Variable Types” on page 125. The type must be one of counter, quantity, or state.

rvData_type

This specifies the data type as either *long* or *float*. A variable of value type state may have a data type of *Structured Byte String*.

rvInitial_value

This is the resource variable initial value, which is the value of the resource variable before it is observed for the first time.

rvClass

This is a *resource variable class* that points to an EM_Resource_Class SDR object.

rvPTX_name

This name is used to read and write the variable in the Performance Toolbox/6000 shared memory (SPMI). This name is only required for variables of type counter and quantity.

rvPTX_description

This is a string that contains a comma-separated list (with no blanks) of message IDs. Each ID specifies a message that contains a short description of a PTX context.

rvLocator

This is a string that contains either an instance vector element name or a null string. If the instance vector for this resource implies the resource 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. Values of the vector element that is specified by the rvLocator attribute must be type of integer. If the value of this item is a null string, the Event Management subsystem must determine the location of the resource.

rvDynamic_instance

This is a Boolean value, that if TRUE, indicates that instances of this resource variable are created dynamically by the Resource Monitor whenever the instance is referenced through the EMAPI.

rvIndex_vector

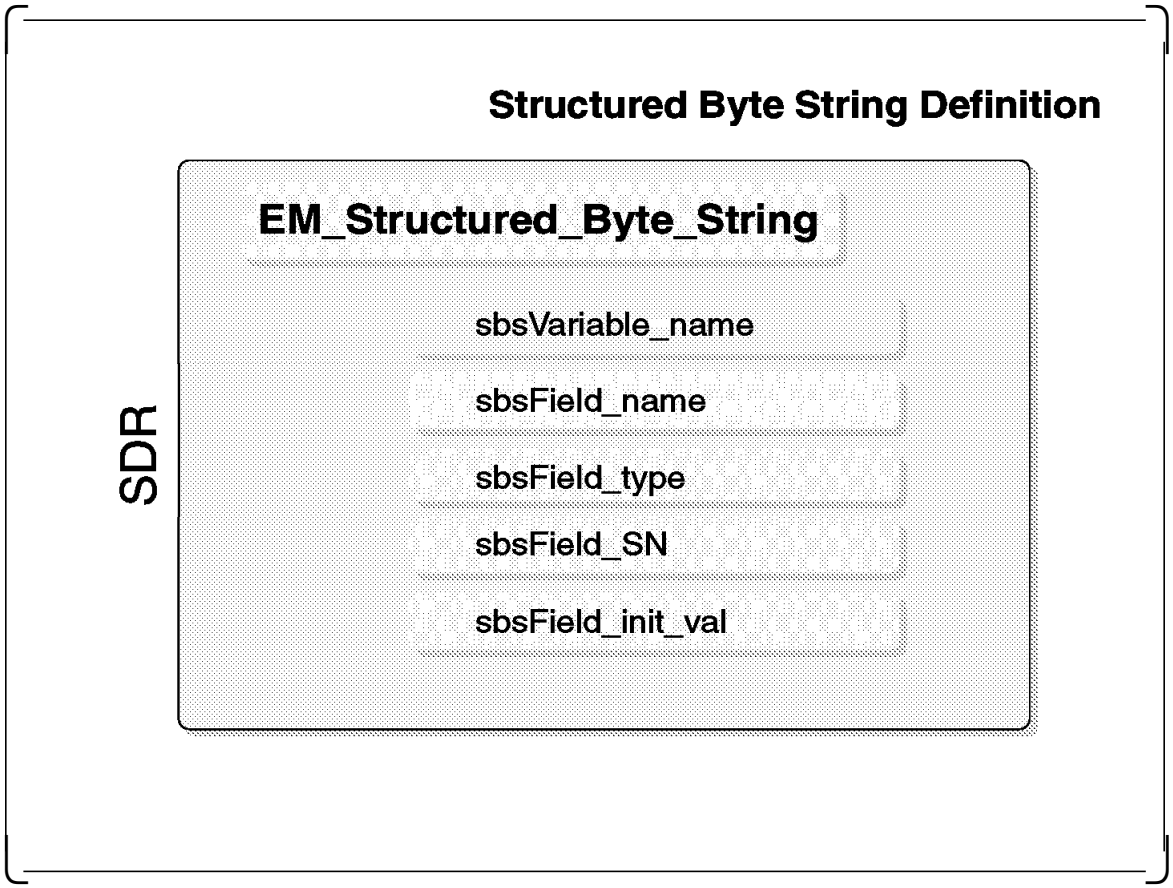
This is a string that contains either an instance vector element name or a null string. You cannot specify the same element for both the rvLocator and the Index_vector fields. Values of the vector element that is specified in the rvIndex_vector attribute must be of type integer. Specifying an index vector can improve the performance of the Event Management subsystem.

The SDR objects for the Resource Variable definition are to be created by SDR commands (SDRCreateObjects, SDRGetObjects, SDRDeleteObjects, and so on). The following is an example of creating a resource variable definition into the SDR:

```
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'
echo $?
```

5.14 Structured Byte String Definition



The *Structured Byte String (SBS)* is defined in the SDR EM_Structured_Byte_String class. Each SDR object in this class defines a field in the Structured Byte String. The Structured Byte String is explained in section 4.7, “Structured Byte String” on page 127. This class consists of five SDR variables:

sbsVariable_name

This SDR variable specifies the resource variable name, as defined in the EM_Resource_Variable SDR class.

sbsField_name

This represents the field name. It is a character string unique within one Structured Byte String.

sbsField_type

The field type is one of:

- long
- float
- cstring
- bstring

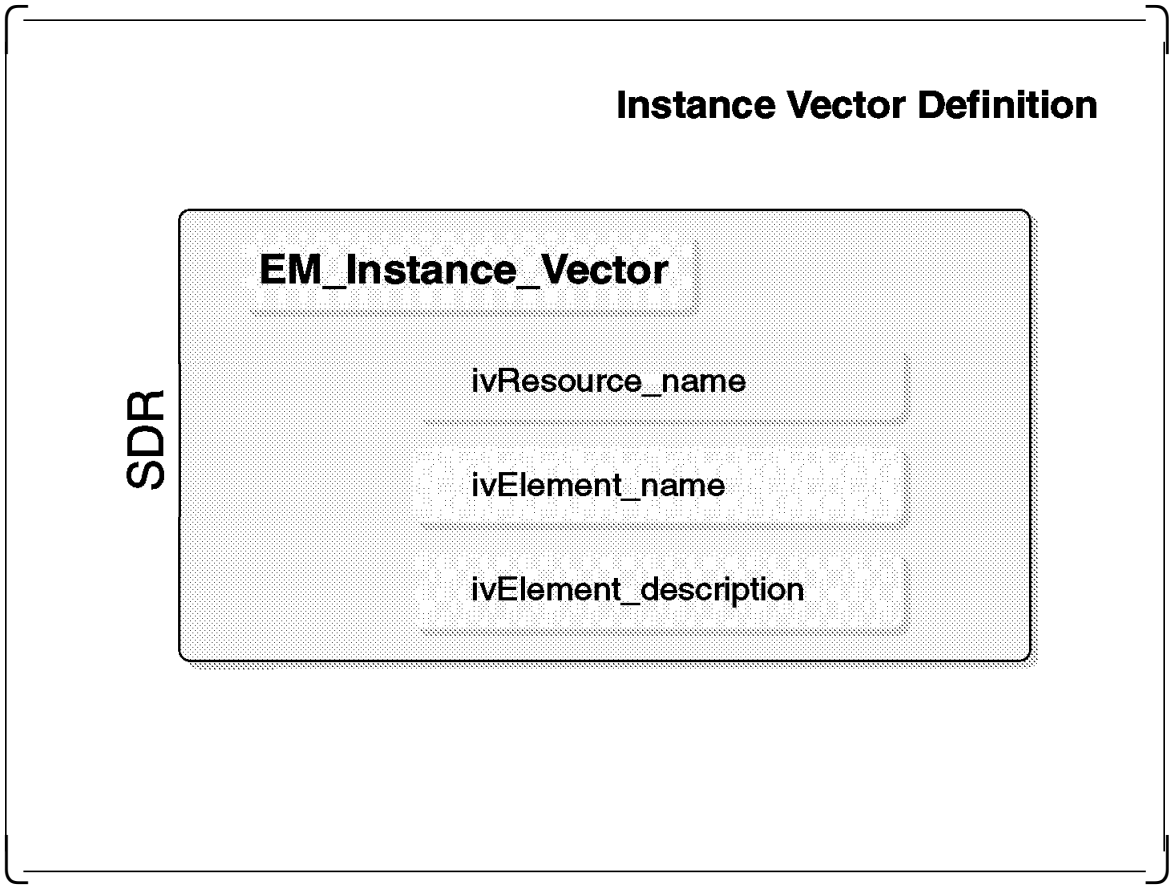
sbsField_SN

This identifies the serial number of the field within a Structured Byte String.

sbsField_init_value

This defines the initial value of the field. This value is placed in the resource variable before the first observation.

5.15 Instance Vector Definition



The Instance Vector structure, which can be applied to identify the copy of a resource, is defined in the *EM_Instance_Vector* SDR class. This class consists of the following three SDR variables, which identify the instance vector element structure:

ivResource_name

This is the resource name.

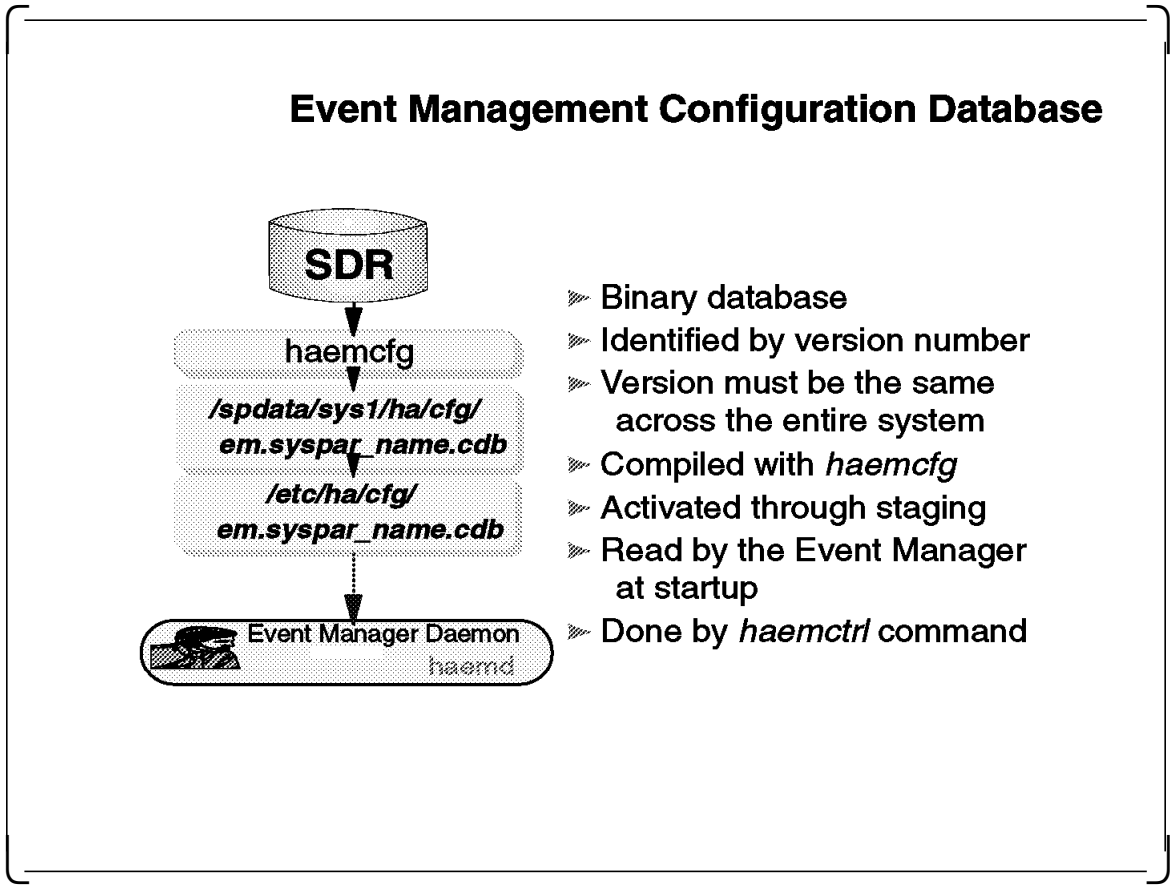
ivElement_name

This is the instance vector element name, like NodeNum, Adapter, CPU, LV, and so on.

ivElement_description

This is a message ID of a textual description of the instance vector.

5.16 Event Management Configuration Database



The information from the SDR has to be compiled into a form readable to the Event Manager Daemon. This is done by issuing the `haemcfg` command that resides in the /usr/lpp/ssp/bin directory. This utility compiles the database into a binary form placed in a file. This file is created in the /spdata/sys1/ha/cfg directory. This directory is used as a staging directory. The `haemcfg` utility also updates the SDR Syspar class with the new version of the configuration. During the compilation, the previous versions of the configuration files in the staging area and in the /etc/ha/cfg directory are backed up, and the name of the newly created file is the previous file name with a time stamp attached to it.

The `haemcfg` utility can also be run with one of two parameters:

-c

This parameter checks the configuration data in the SDR without building a database.

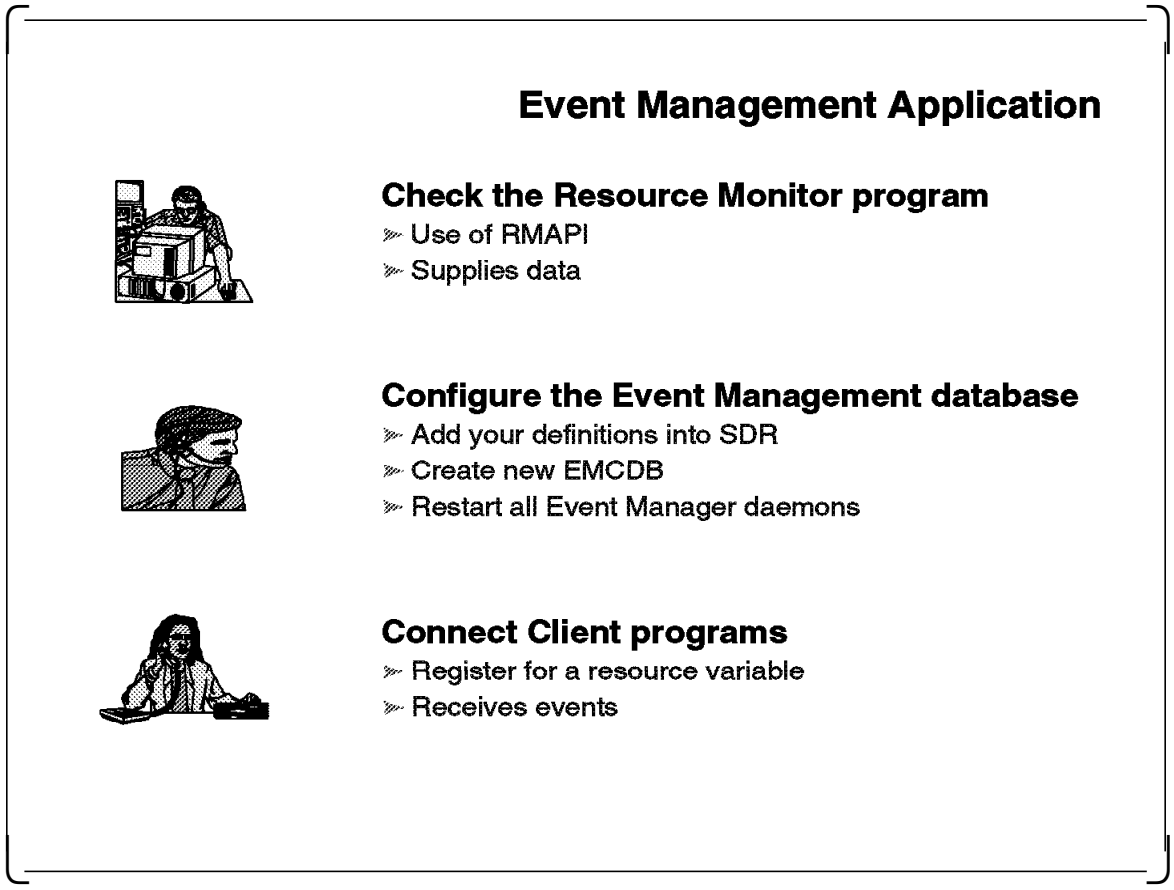
-n

This parameter creates a database in the current directory. (The SDR version string is not updated.)

When the Event Manager Daemon starts up, it checks the SDR Syspar class for the correct version of the configuration database (EMCDB) and copies the compiled configuration file from the staging area to its active directory,

/etc/ha/cfg. When an Event Manager Daemon starts up on a node, it uses the haemrcpcdb utility to transfer the file by using the RCP protocol.

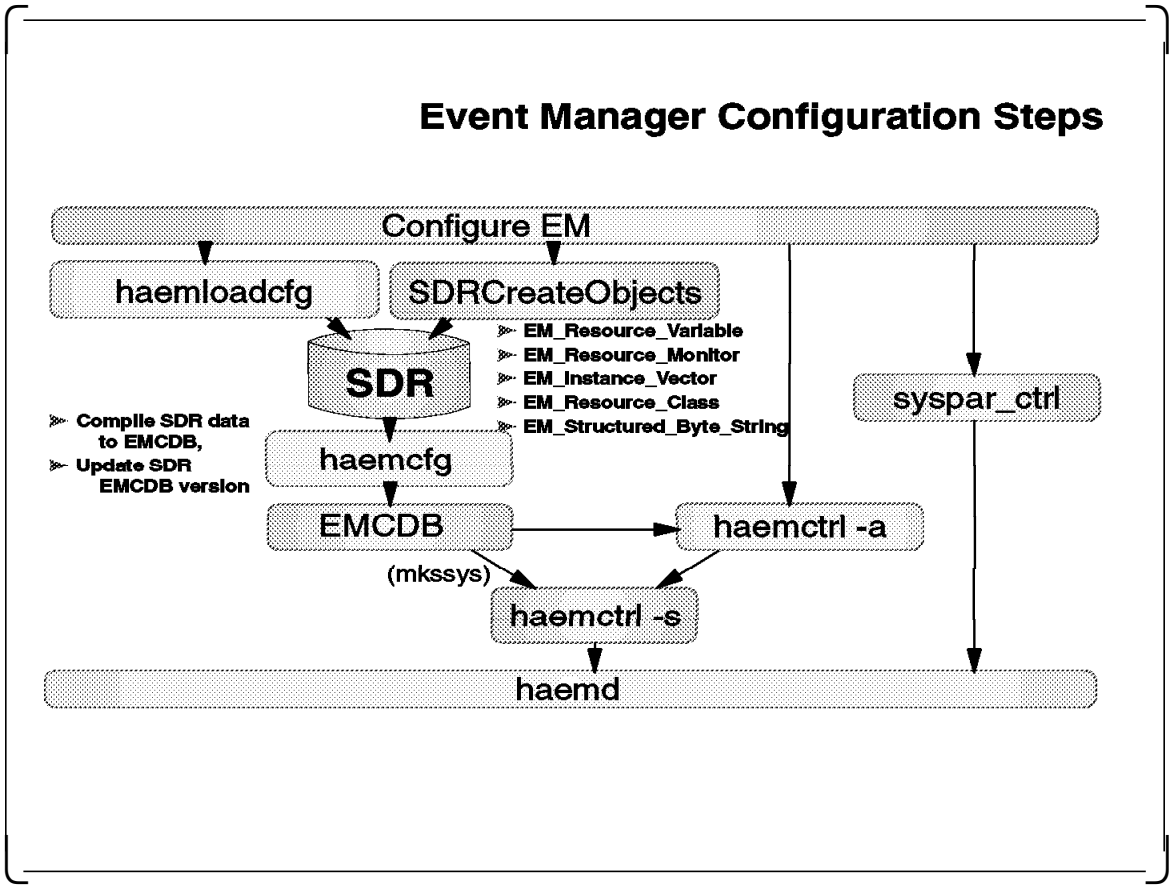
5.17 Event Management Application



When you intend to add your own application, you must perform these steps:

1. Check for the existence of a Resource Monitor that supplies the requested data. If there is no such Resource Monitor, you should write a program using the Resource Monitor API.
2. Check the configuration of the Event Management in the SDR. If the resource variable is not present, add your definition into the SDR. Every resource variable must have a corresponding Resource Class and Resource Monitor, and if required, the Structured Byte String and Instance Vector definition must be present, too. Whenever the SDR definition changes, the EMCDB must be re-created using the `haemcfg` command, and every Event Manager Daemon must be restarted in order to synchronize the configuration across the entire system.
3. Finally, you can use the Event Management API to register and receive events from the Event Manager Daemon. You may write your own application, or choose the Perspectives, Problem Management, or other existing client software.

5.18 Event Manager Configuration Steps



The configuration of the Event Management on an RS/6000 SP system consists of following steps:

1. Create SDR objects which uniquely identify the Resource Monitor, Resource Class, Instance Vector, and Resource Variable. The SDR structure is explained in section 5.10, “Event Management SDR Classes” on page 151.
2. Check the SDR for the newly created objects by using the `SDRGetObjects` command.
3. Run the `haemcfg` command to compile the SDR data to the Event Management Configuration Database (EMCDB).
4. Check if the new version of the EMCDB file has been created.
5. If the Event Management subsystem is not present, create it by issuing the `haemctrl -a` command. This command calls the `mkssys` command to define the Event Management subsystem to the AIX System Resource Controller.
6. Start the Event Manager Daemon by issuing the `haemctrl -s`, or `startsrc` command.
7. Check the Event Manager Daemon for correct functionality by issuing the commands `lssrc`, `ps -ef`, and so on.

If there is no need to add additional objects to the Event Management configuration, the following commands are recommended to use for initial configuration of the Event Management structure:

.

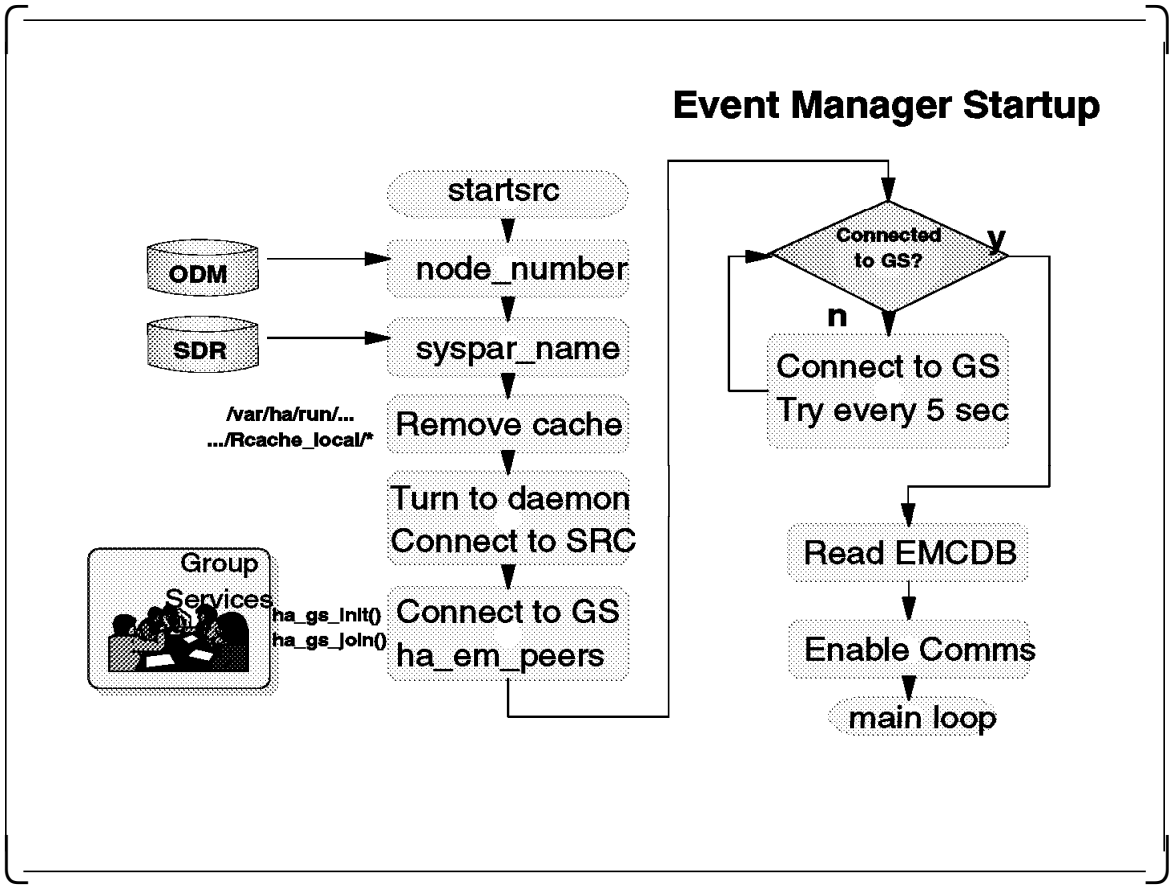
haemctrl -a

This command adds the Event Management subsystem to the system. The Event Management subsystems are added automatically when the nodes install. This command is performed by the syspar_ctrl script on the Control Workstation during the install process.

syspar_ctrl

The system partition configuration invokes the Event Management configuration steps.

5.19 Event Manager Startup



Normally, the Event Manager Daemon is started by an entry in the /etc/inittab after reboot:

```
/etc/inittab:
 haem:2:once:/usr/bin/startsrc -g haem
 > /dev/console 2>&1
```

If neccesary, the Event Manager Daemon can be started using the haemctrl command or the startsrc command directly. An example of this use follows:

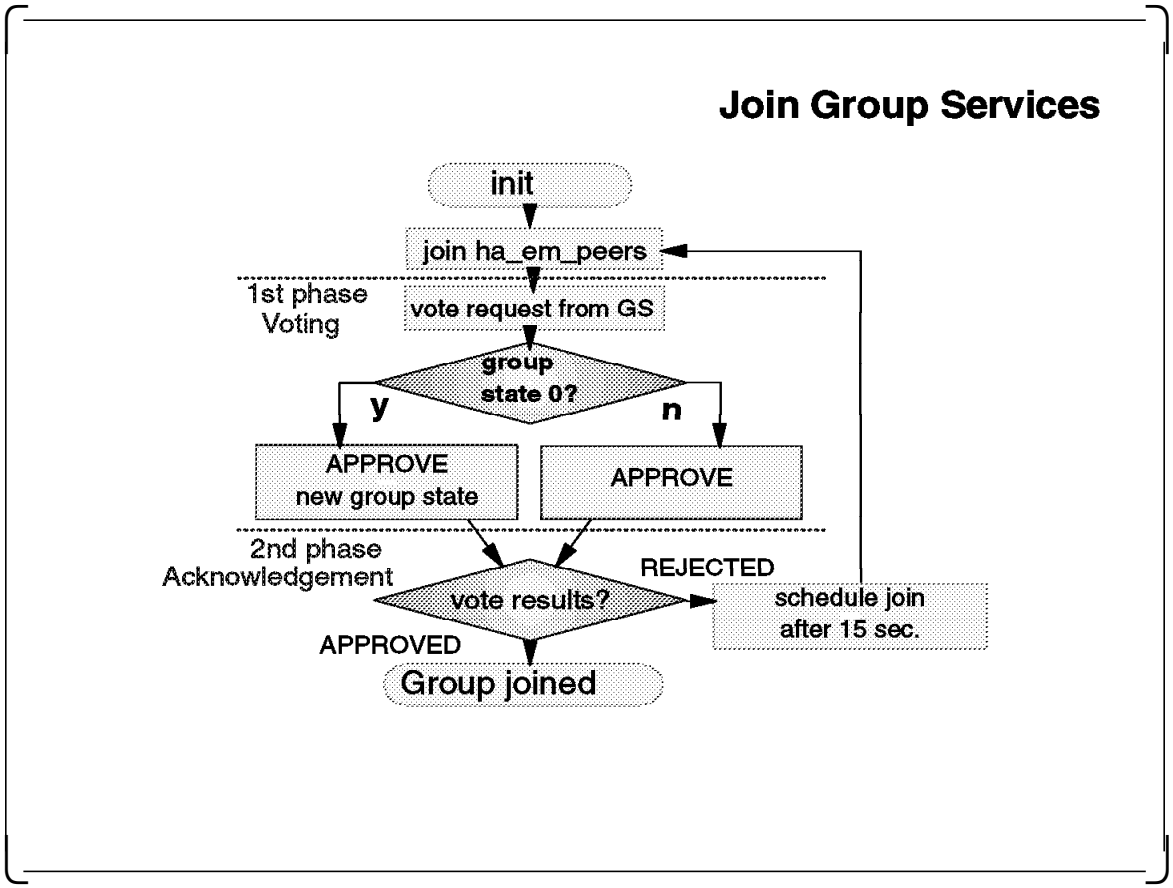
```
startsrc:
 /usr/bin/startsrc -g haem

Event Management Control Command (haemctrl):
 /usr/lpp/ssp/bin/haemctrl -s
```

The daemon executes the following steps during its initialization:

1. Gets the number of the node on which it is executing using the /usr/lpp/ssp/install/bin/node_number command. Node 0 is the Control Workstation.
2. Fetches the name of the system partition and the EMCDB version string from the Syspar SDR class.
3. Fetches the Event Manager Daemon remote client communications port number from the SP_ports SDR class.

4. Turns the program into a daemon. This includes establishing communications with the AIX System Resource Controller (SRC) subsystem in order to return status at the request of SRC commands.
5. Removes from the registration cache all subdirectories for local clients that no longer exist; this means that if the process ID specified in the subdirectory name (/var/ha/run/*) cannot be found, it removes the subdirectory. Subdirectories for remote clients cannot be removed automatically, since the daemon cannot determine if the remote processes still exist.
6. Connects to Group Services (hags). If the connection cannot be made because the Group Services subsystem is not running, then the connection attempt is scheduled to be made again in five seconds. This is repeated until the connection to Group Services is established. Meanwhile, the Event Manager Daemon initialization continues.
7. Enters the main control loop. This loop waits for requests from clients, messages from Resource Monitors and other Event Manager Daemons, messages from Group Services, and requests from the SRC for status. It also waits for internal signals that indicate that a function previously scheduled should now be executed; for example, make another connection attempt to the Group Services Subsystem. However, client requests and messages from Resource Monitors and other Event Manager Daemons (peers) are refused until the Event Manager Daemons successfully join the daemon peer group “ha_em_peers” and fetch the correct version of the EMCDB.

5.20 Join Group Services



Group Services provides another set of High Availability services. The Event Management subsystem primarily uses Group Services to monitor the state of each Event Manager Daemon in its operational domain. This is done by having each daemon in the domain join a group named ha_em_peers. Group Services then informs each Event Manager Daemon, in a synchronized fashion, when a daemon has joined or left the group.

Associated with the group is the group state. This state contains the version string of the EMCDB version string. From this information, the joining Event Manager Daemon can discover the version of the EMCDB that the remainder of the group is using.

After the Event Manager Daemon successfully establishes a connection with Group Services, it attempts to join the daemon peer group, ha_em_peers. The join procedure requests a 2-phase voting protocol.

The voting request that is received from Group Services contains the value of the group state variable. This value represents to the Event Manager Daemon the configuration database version that is being used in the operational domain (system partition). If this value is a null string, then the Event Manager Daemon is the first Event Manager Daemon in its operational domain.

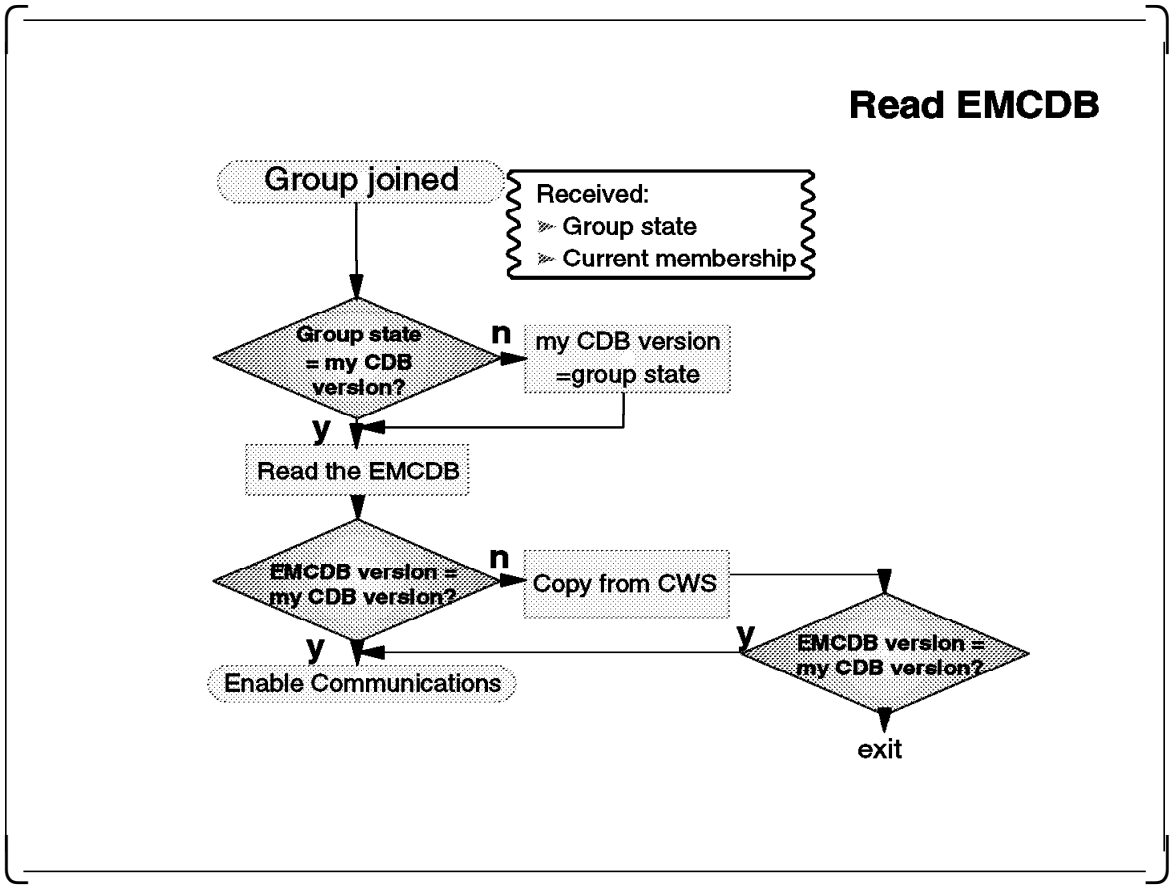
In the Group Services voting subroutine, the Event Manager Daemon votes APPROVE. If the group state received from the Group Services is zero, then the Event Manager Daemon votes APPROVE and proposes a new group state within the voting subroutine. This group state is always accepted because this Event Manager Daemon is the only one in the operational domain.

The second phase of the voting is used to acknowledge the membership of the Event Manager Daemon in the ha_em_peers group. In the voting protocol, the Event Manager Daemons vote APPROVE. A daemon join is rejected only if an existing peer group member is still recovering from a prior termination of the joining daemon.

If the join procedure is rejected, then the daemon schedules another attempt to join the group in 15 seconds. The daemon repeatedly attempts to join the group until it is accepted.

When the Event Manager Daemon has joined the group, it knows the version of the configuration that is used in the system (operational domain), and is ready to read the configuration database.

5.21 Read the EMCDB



After the Event Manager Daemon successfully joins the ha_em_peers group in Group Services, the Event Manager Daemon knows the group state variable of this group and the current membership of this group. The current membership is a list of all Event Manager Daemons in the operational domain (system partition). The group state represents the version of the configuration database that is used by already-running Event Manager Daemons in the operational domain (system partition).

The Event Manager Daemon compares the group state string received from the Group Services ha_em_peers group with the configuration database version string that was read from the Syspar SDR class at the Event Manager Daemon startup. If the version string in the group state variable is different from the string read from the SDR Syspar class, then the Event Manager Daemon will use the version string obtained from the group state variable of the Group Services. This occurs when the `haemcfg` utility is used and the utility recreates the EMCDB on the Control Workstation. In this case, the Event Manager Daemons in the operational domain still use the previous version of the configuration database, and they have not been restarted. Every Event Manager Daemon in an operational domain has to use the same configuration database, and this is the reason the version string from the group state has priority against the version string stored in the SDR.

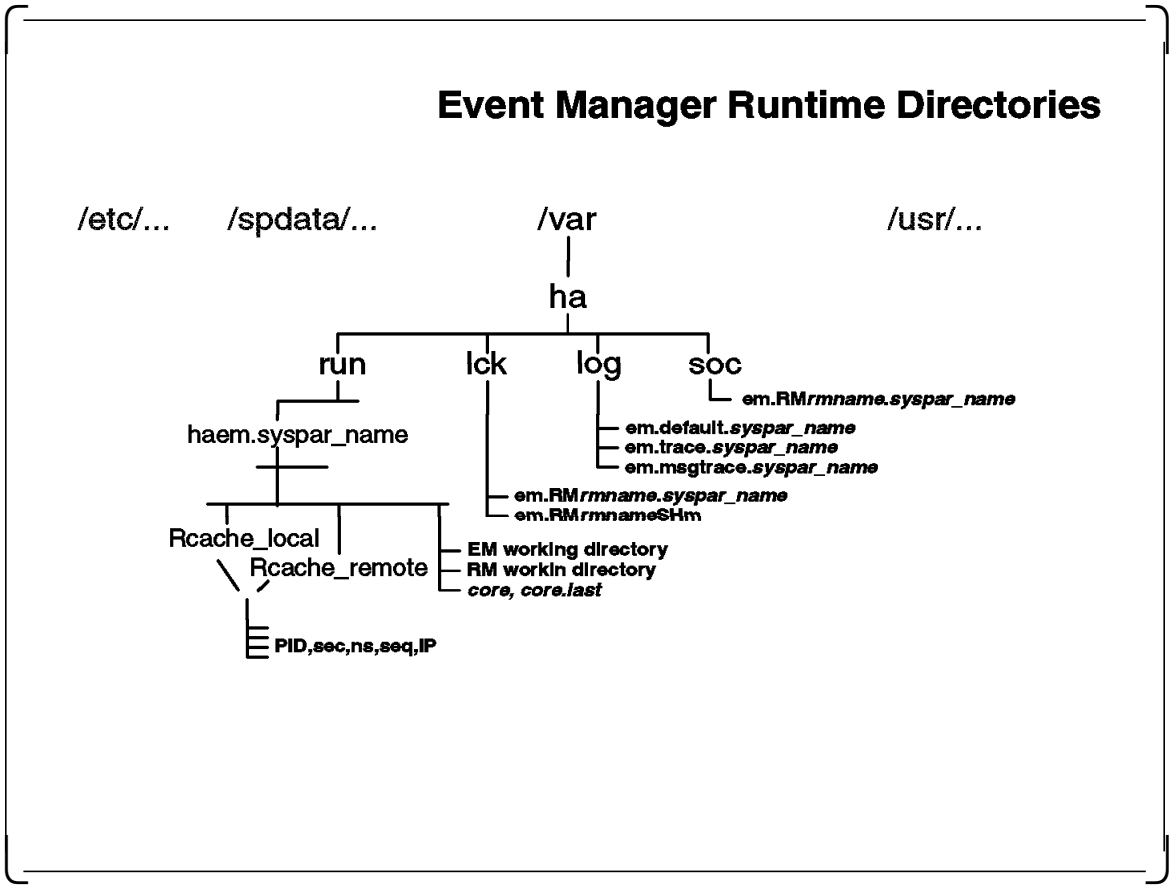
At this time, the Event Manager Daemon knows about the version of the configuration database that is used by other Event Manager Daemons in the operational domain, and attempts to load the same version of the database. The Event Manager Daemon reads the configuration database (EMCDB) (/etc/ha/em.syspar_name.cdb) and compares the version of this configuration database with the version of the configuration database used in the operational domain (which has been taken from the group state).

If the version of the configuration database corresponds to the version that is used by the running Event Manager Daemons in the operational domain, then the configuration database is loaded and the Event Manager Daemon is ready to accept clients.

If the configuration database is of a different version, then the Event Manager Daemon copies the configuration database from the staging area on the Control Workstation. To copy the configuration database, the Event Manager Daemon calls the `haemrcpcdb` script that is stored in the /usr/lpp/ssp/install/bin directory. This script obtains a Kerberos rcmdtgt ticket and uses the Kerberos authenticated `rcp` to receive the configuration database file.

After the Event Manager Daemon copies the new EMCDB file from the Control Workstation, the Event Manager Daemon compares the version of the copied configuration database with the version string that is used in the system. If the newly received configuration database is of the version that is used by the Event Manager Daemons in the system, then the Event Manager Daemon reads the configuration from this file and continues to enable communications. If the configuration database received from the Control Workstation is of a different version than the configuration database version in the running system, then the Event Manager Daemon will not be configured and clients will not be enabled to communicate with this Event Manager Daemon.

5.22 Event Manager Runtime Directories



The following is the runtime directory structure used by the Event Manager Daemon:

/var/ha/lck

This directory is where Event Manager Daemon lock files are stored. In this directory, the file *em.RMrmname.syspar_name* ensures a single running instance of a Resource Monitor. The *rmname* is the name of the Resource Monitor and *syspar_name* is the name of the system partition. On the Control Workstation, there may be several instances of the same Resource Monitor running, but they are in different operational domains (system partitions).

When the Resource Monitors attach through the RMAPI to the System Performance Measurement Interface, they use the file *em.RMrmnameSHm* to create a shared memory key.

The file *em.haemd.syspar_name* is used to ensure that there is a single running instance of an Event Manager daemon in an operational domain.

/var/ha/log

This is a log directory. The file *em.trace.syspar_name* contains the Event Manager Daemon trace output. The *syspar_name* is the system

partition name. The file *em.msgtrace.syspar_name* contains the Event Manager daemon message trace output.

The file *em.default.syspar_name* contains any Event Manager Daemon error messages that cannot be written to the standard AIX error log. (Normally all Event Manager Daemon errors are written to the AIX error log.)

/var/ha/run

In this directory, a *haem.syspar_name* directory is created, where *syspar_name* is the system partititon name. This directory is the current working directory for the Event Manager daemon. Any core file created when the Event Manager Daemon abnormally terminates is placed in this directory. Whenever the Event Manager Daemon starts, it renames the core file to core.last.

This directory also contains the working directories of any Resource Monitors started by the Event Manager Daemon. Such directories have the names of their Resource Monitors.

Finally, this directory contains two other directories, named *Rcache_local* and *Rcache_remote*. These directories contain the registration cache for local and remote client registration requests.

For each client that establishes communications with the Event Manager Daemon, a cache subdirectory is created in the appropriate registration cache directory. This subdirectory has a name of the form *p,s,n,q,i*, where *p* is the process ID of the client, *s* is the seconds part of the time when the client established its session with the Event Manager Daemon, *n* is the nanoseconds part, *q* is a session sequence number, and *i* is the IP address of the host where the client is executing. For local clients, the IP address is 0.0.0.0.

Within each subdirectory are several files with numeric names. Each file contains a registration request. The file name is the event command group ID of the events within the request.

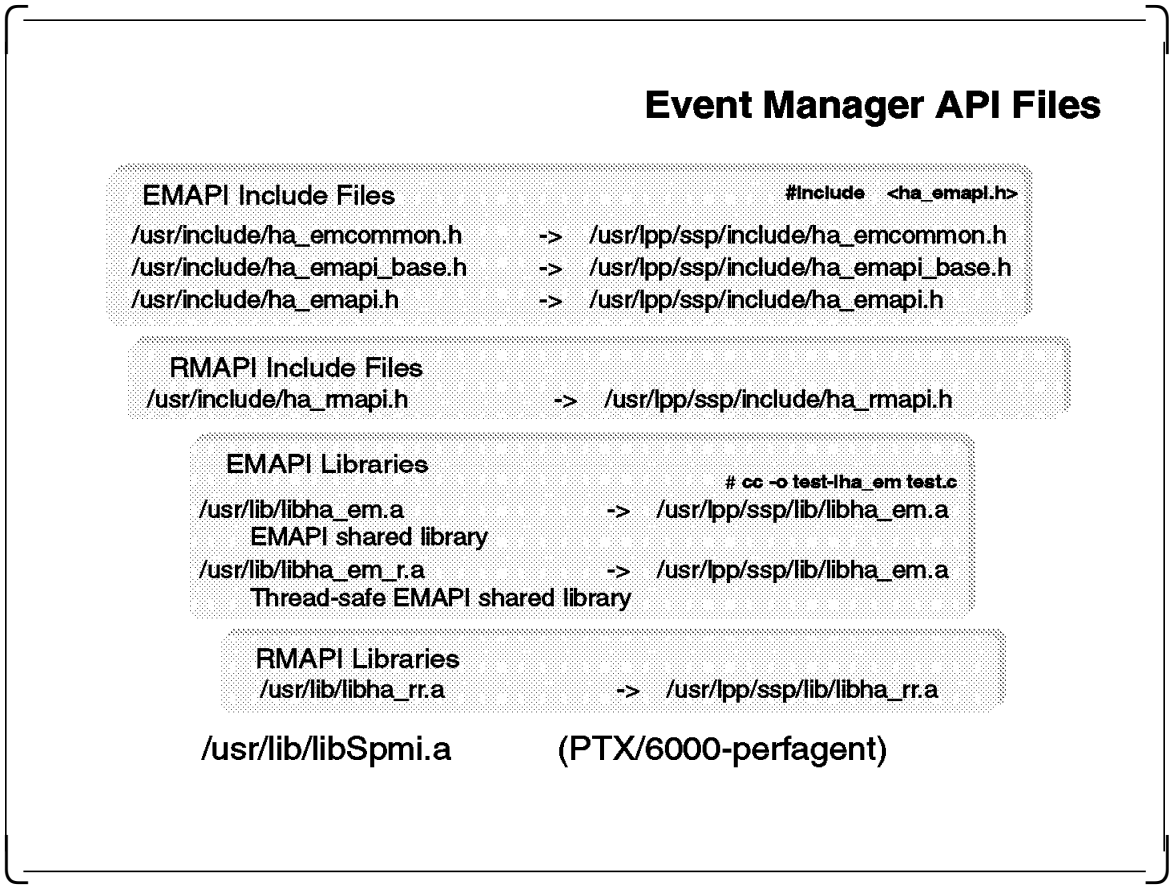
/var/ha/soc

This directory contains the following socket files:

em.clsrv.syspar_name	Used by EMAPI to connect to the Event Manager daemon.
em.rmsrv.syspar_name	Used by client type Resource Monitors to connect to the Event Manager daemon.
em.RMrmname.syspar_name	Used by the Event Manager daemon to to connect to the server type resource monitor specified by rmname.

In all three socket names, syspar_name is the system partition name.

5.23 Event Manager API Files



This foil shows include files and library files for the two Event Manager and Resource Monitor Application Programming Interfaces. The include file for the EMAPI is the ha_emapi.h file that, from its body, links the ha_emcommon.h and ha_emapi_base.h files. The include file for the RMAPI interface is ha_rmapi.h.

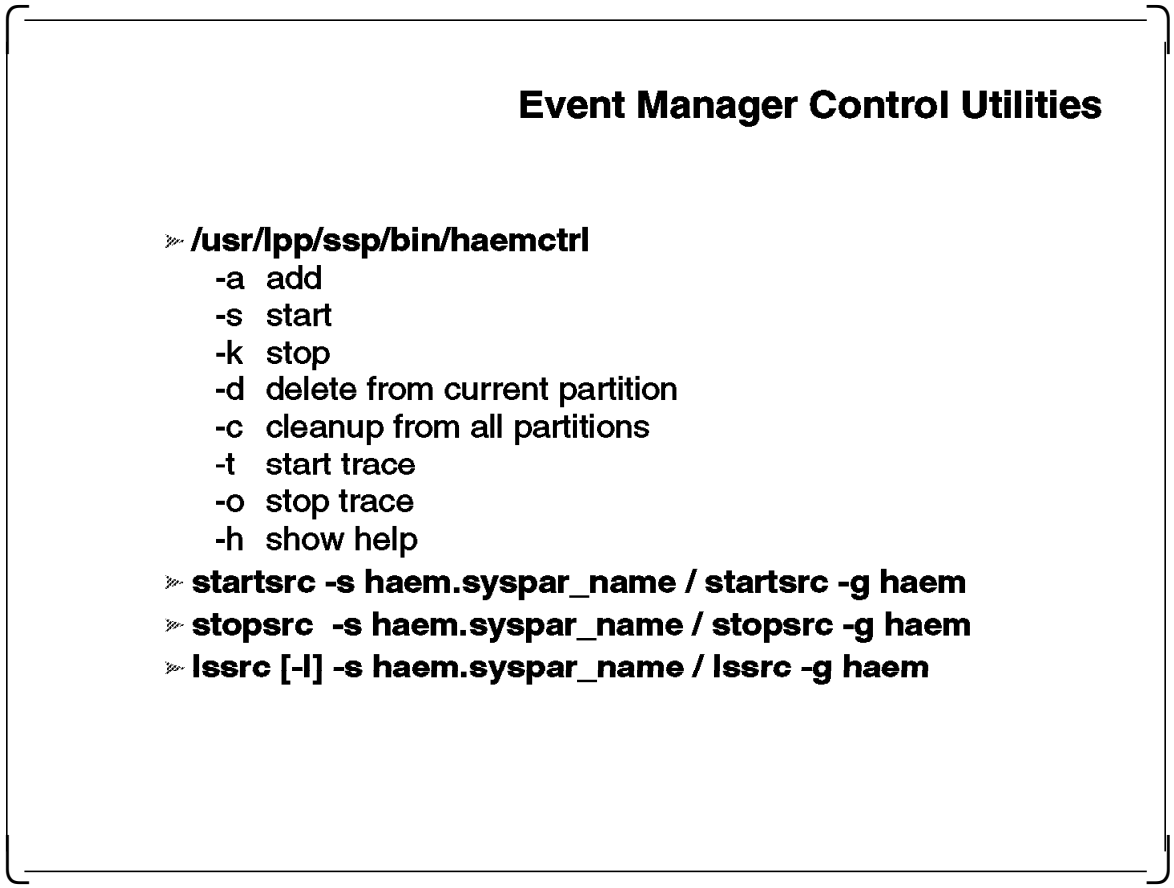
The EMAPI is a shared library used by a client program to obtain the services of the Event Management subsystem. This library is delivered in two versions:

- For thread-safe programs (libha_em_r.a)
- For non-thread-safe programs (libha_em.a)

All Event Management include files and library files are links to the files of the same names in directory /usr/lpp/ssp/lib and /usr/lpp/ssp/include.

Note: The */usr/lib/libSpmi.a* is a required file that is part of the PTX/6000 perfagent package. The *perfagent* must be installed prior to the Event Management.

5.24 Event Manager Control Utilities



5.24.1 Event Manager Control Program

The haemctrl (/usr/lpp/ssp/bin/haemctrl) is the Event Manager Daemon control program. This is a Korn shell executable file that manages the Event Manager Daemon as a subsystem. This script is normally invoked by the syspar_ctrl script, which is the interface to system partition-sensitive subsystems. If necessary, haemctrl can be invoked directly. When invoked directly from the command line, the SP_NAME environment variable must be set to the appropriate system partition name.

The following lists all of the haemctrl functions:

```
haemctrl { -a | -s | -k | -d | -c | -t | -o | -r | -h }
  -a  Add Event Management subsystem to this partition
  -s  Start Event Management subsystem in this partition
  -k  Stop Event Management subsystem in this partition
  -d  Delete Event Management subsystem from this partition
  -c  Remove Event Management subsystem from all partitions
  -t  Start Event Management subsystem trace in this partition
  -o  Stop Event Management subsystem trace in this partition
  -r  Refresh Event Management subsystem in this partition
  -h  Display Usage information
```

5.24.1.1 Add Function

The add function of the haemctrl script first sets the haem port in /etc/services where the default range of ports is 10000 - 10100 and writes the port number to the SDR (Syspar_ports), then checks for the existence of the Event Management subsystem in the AIX System Resource Controller (SRC). If the subsystem does not exist, the add function creates it with default parameters and parameters taken from the SDR (port number, partition), using the mkssys command, as follows:

```
SRCsubsys:
        subsysname = "haem.syspar_name"
        synonym = ""
        cmdargs = "9.12.1.138"              (CWS IP address/partition)
        path = "/usr/lpp/ssp/bin/haemd"
        uid = 0
        auditid = 0
        standin = "/dev/console"
        standout = "/dev/null"
        standerr = "/var/ha/log/em.default.syspar_name"
        action = 1
        multi = 0
        contact = 3
        svrkey = 0
        svrmtype = 0
        priority = 20
        signorm = 0
        sigforce = 0
        display = 1
        waittime = 30
        grpname = "haem"
```

Then an entry to /etc/inittab is created by the mkitab command, as follows:

```
mkitab "haem.syspar_name:2:once:
      /usr/bin/startsrc -g haem.syspar_name > /dev/console 2>&1"
```

In the following step, if the haemctrl -a command is run on the Control Workstation, the SDR database is loaded by haemloadcfg command. Note that the haemloadcfg command does not replace the already existing SDR objects, but only creates the objects if they are not present in the SDR. The haemctrl command does more checking on the validity of the objects created into SDR, SRC subsystems, inittab, /etc/services, and so on.

5.24.1.2 Start Function

The start function of the haemctrl only starts the haem.syspar_name subsystem as defined in the AIX System Resource Controller in the ODM database:

```
startsrc -s haem.syspar_name  (on the Control Workstation)
startsrc -s haem   (on the RS/6000 SP nodes)
```

5.24.1.3 Stop Function

The stop function of the haemctrl command only stops the haem.syspar_name subsystem as defined in the AIX System Resource Controller in the ODM database:

```
stopsrc -s haem.syspar_name  (on the Control Workstation)
stopsrc -s haem     (on the RS/6000 SP nodes)
```

5.24.1.4 Delete Function

The delete function of the haemctrl removes the subsystem from the AIX System Resource Controller (ODM), removes the /etc/services port number entry, and removes the /etc/initab entry for the Event Manager Daemon startup in the current partition. The delete function requires that the subsystem requested must already be stopped.

5.24.1.5 Clean function

The purpose of the clean function of the haemctrl command is similar to the delete one, but the deletion is done for *all* partitions in a system. This function does not remove objects from SDR.

5.24.1.6 Trace Functions

Running the haemctrl command with traceon (-t) makes the Event Manager Daemon record its steps into the log files *em.msgtrace.syspar_name* and *em.trace.syspar_name*, which are located in the /var/ha/log directory.

The traceoff function of the haemctrl command calls the haemtrcoff -s haem.syspar_name -a $Tracelist (Tracelist=all). Stopping of the trace mode is done by applying the traceoff function of the haemctrl command (parameter -o). The traceoff function of the haemctrl command calls the haemtrcoff -s haem.syspar_name -a $Tracelist (Tracelist=all).

The haemctrl command is the only command a system adminstrator is expected to use under normal conditions. The other commands mentioned are called from the haemctrl command, depending on the specified parameter.

5.24.2 SRC Commands

5.24.2.1 Subsystem Start Command

The startsrc command is called from the haemctrl -s command. This command is needed to start the Event Management subsystem for the partition specified in the subsystem name, or to start all Event Management subsystems in the System Resource Controller (SRC) group. If the system administrator needs to run this command, you have to check the partition name (syspar_name) and supply it into the subsystem name you intend to start.

startsrc -s haem.syspar_name

This starts the Event Management subsystem in the syspar_name partition. If the startsrc command is run on the Control Workstation, then the subsystem name consists of the *haem* name followed by a period and the system partition name. On the RS/6000 SP nodes the subsystem name is *haem*. Run the startsrc command on each node and the Control Workstation.

startsrc -g haem

This starts the Event Manager Daemon for every partition.

5.24.2.2 Subsystem Stop Command

Issuing the stopsrc command has similar results to issuing the haemctrl -k command, because the haemctrl -k command calls the stopsrc -s haem.syspar_name command. To stop the Event Management subsystem in all partitions, issue the stopsrc command with the -g parameter (group) and the SRC group name.

stopsrc -s haem.syspar_name

This stops the Event Management subsystem in the syspar_name partition. If the stopsrc command is run on the Control Workstation, then the subsystem name consists of the *haem* name followed by a period and the system partition name. On the RS/6000 SP nodes, the subsystem name is *haem*. Run the stopsrc command on each node and the Control Workstation.

stopsrc -g haem

This stops the Event Manager daemon for every partition.

5.24.2.3 List Subsystems Command

The lssrc command is used for diagnosing Event Manager status. The -l parameter specifies the long format output. This output gives detailed information on subsystem status, traceflags, EMCDB version, connection to group services, logical connections, Resource Monitor information, peer daemon status, and internal daemon counters. The -l parameter cannot be specified with the -g parameter.

lssrc -s haem.syspar_name

This lists the status of the Event Manager Daemon in the syspar_name partition. If the lssrc command is run on the Control Workstation, then the subsystem name consists of the *haem* name followed by a period and the system partition name. On the RS/6000 SP nodes, the subsystem name is *haem*.

lssrc -l -s haem.syspar_name

This lists the status of and detailed information about the Event Manager Daemon in the syspar_name partition.

lssrc -g haem

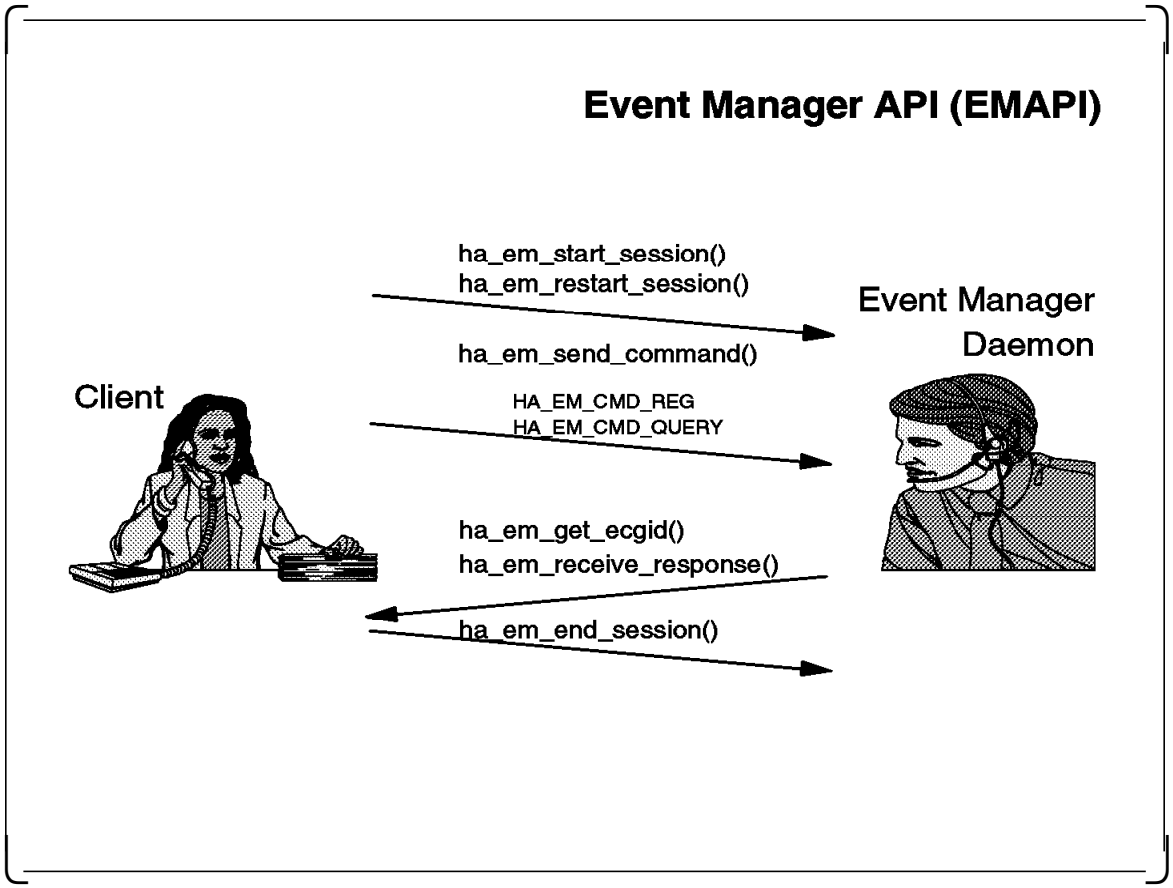
This lists the status of all Event Manager Daemons (SRC group haem).

Following is an example of the detailed output of the lssrc command:

```
root@sp21cw0 # lssrc -l -s haem.sp21cw0
Subsystem         Group            PID     Status
 haem.sp21cw0     haem             53572   active

Trace flags set:  None

 Configuration Data Base version: 838994663,151745024,0(SDR)

 Daemon started on 08/05/96 at 17:19:16.757872384
    running 3 days, 20 hours, 45 minutes and 49 seconds
 Daemon connected to group services: TRUE
 Daemon has joined peer group:       TRUE
 Daemon communications enabled :     TRUE
 Peer count:                         5
```

```
Logical Connection Information
 Type   LCID   FD  Node/PID  Start Time
local      0   13    37846   Mon Aug  5 17:20:08 1996
local      1   16    41058   Mon Aug  5 17:23:27 1996

Resource Monitor Information
  Resource Monitor Name      Type     FD    PID
IBM.PSSP.harmld            server     17      -1
IBM.PSSP.harmpd            server     20   19908
IBM.PSSP.hmrmd             server     18   57556
IBM.PSSP.pmanrmd           client     15      -1
Membership                 internal   -1      -1
Response                   internal   -1      -1
aixos                      internal   -1      -1

Highest file descriptor in use is 20

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

Internal Daemon Counters
       CCI connection rejects         = 0
       RMC connection rejects         = 0
       HR connection rejects          = 0
       Retry request message          = 0
       Retry response message         = 0
```

5.25 Event Manager API (EMAPI)



The Event Manager Application Interface (EMAPI) is the mechanism by which an application registers for and receives events. The EMAPI is also used to query the Event Management subsystem for information. The API is a command/response model: the client sends commands and then waits for asynchronous responses. One command may result in more than one response, as with events.

A client application must first establish a session with the Event Manager Daemon. The session validates that the client is permitted to use Event Management services, and then provides a communication path to the Event Manager Daemon. If this communication path is lost, it can be regained without starting another session. When the client no longer needs these services, the client terminates the session.

The ha_em_start_session() function establishes the session with the Event Manager Daemon, as follows:

```
int ha_em_start_session(char *part_name,
                        struct ha_em_err_blk *em_errb)
```

The part_name argument represents the partition in the RS/6000 SP environment. If this argument is NULL, the session is established with the Event Manager daemon in the current partition. If the client is executing on a node, then this daemon is the local Event Manager Daemon. If the client is executing

on the Control Workstation or on a workstation outside of the RS/6000 SP system, then the partition is determined from the SP_NAME variable.

The function returns a file descriptor that is used to reference the session. If the return code is -1, then the em_errb structure contains the failure reason and description.

The restart session subroutine reconnects an existing session to the Event Manager daemon to which the session was previously connected, as follows:

```
int ha_em_restart_session(int session_fd,
                          struct ha_em_err_blk *em_errb)
```

The session_fd contains the file descriptor returned from the ha_em_start_session() or ha_em_restart_session(). The subroutine returns the new file descriptor. The em_errb has the same meaning as before.

The end session subroutine terminates a session with the Event Manager daemon. This is the last call to the Event Manager Daemon, as follows:

```
int ha_em_end_session(int session_fd,
                      struct ha_em_err_blk *em_errb)
```

The representation of the subroutine inputs are the same as in the ha_em_restart_session().

All commands are sent to the Event Manager Daemon by using the send command subroutine, as follows:

```
int ha_em_send_command(int session_fd,
                       struct ha_em_cmd_blk *p,
                       struct ha_em_err_blk *em_errb)
```

The p argument is a pointer to a command block that specifies the command to be sent and any additional parameters needed for the daemon to execute the command. The session_fd and em_errb arguments have the same meaning as in previous subroutines.

The commands are of the type described in section 5.6, “Event Registration and Notification” on page 144 and in section 5.7, “Query” on page 146.

All events registered in a single call to the ha_em_send_command() subroutine can be identified by an *event command group ID*. This ID can be used to relate individual event responses back to the command with which they were registered. The event command group ID can be obtained by calling the get event command group ID subroutine, as follows:

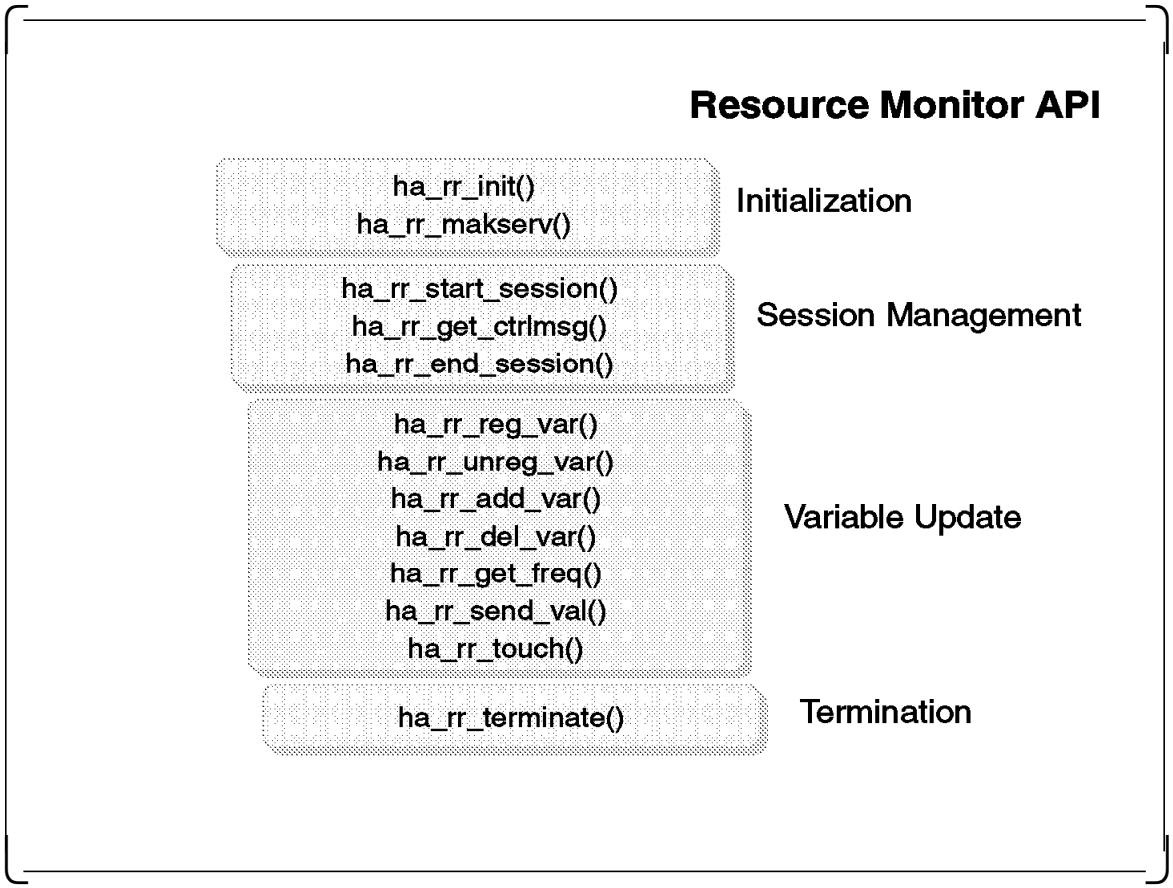
```
ha_em_ecgid_t ha_em_get_ecgid(ha_em_eid_t event_id)
```

The client uses the file descriptor returned by the ha_em_start_session() subroutine to determine when a response from the Event Manager Daemon has been received. Once a response has been received, the receive response subroutine is called, as follows:

```
int ha_em_receive_response(int session_fd,
                           struct ha_em_rsp_blk **em_rsp_blk,
                           struct ha_em_err_blk *em_errb);
```

If the return value is greater than 0, then a pointer to a buffer containing a response block is returned. If the return code is 0, then the response was already processed by the EMAPI. The em_rsp_blk argument points to an area where the buffer pointer is to be returned.

5.26 Resource Monitor API



The Resource Monitor Application Programming Interface (RMAPI) is used by Event Management Resource Monitors to send resource variables to the Event Manager Daemon. If the Resource Monitor is not implemented as a command, the RMAPI is also used for control functions between the Event Manager Daemon and the Resource Monitor. The RMAPI is designed in anticipation that a Resource Monitor provides data to these two masters:

- Event Manager Daemon
- Performance Monitoring subsystem

The RMAPI can be used in one of two ways, depending on whether the Resource Monitor is a command or not. A Resource Monitor implemented as a command connects to the Event Manager Daemon. Otherwise, the Event Manager Daemon (and the Performance Monitoring Subsystem) connects to the Resource Monitor.

The RMAPI has the following four categories of operation:

- Initialization
- Session Management
- Variable Update
- Termination

The RMAPI has two initialization subroutines. The initialize API subroutine is called by the Resource Monitor to inform the API of its identity and to verify that this is the only copy of the Resource Monitor that is executing:

```
int ha_rr_init(char *name, struct ha_em_err_blk *rr_errb)
```

The make server subroutine creates a server session such that Resource Monitor managers can connect to the monitor. Resource Monitor managers are the Event Manager Daemon and the Performance Monitor subsystem. If the Resource Monitor is not command-based, then this subroutine should be called after the call to ha_rr_init():

```
int ha_rr_makserv(int rr_notify_proto, struct ha_em_err_blk *rr_errb)
```

If the ha_rr_makserv() subroutine is not called, that is, the Resource Monitor is command-based, the start session subroutine is to be called immediately upon successful completion of the call to initialize the API subroutine, ha_rr_init():

```
int ha_rr_start_session(int rr_notify_proto,
                        struct ha_em_err_blk *rr_errb)
```

When a file descriptor is returned by the ha_rr_start_session() subroutine, the socket is ready for reading. The Resource Monitor has to fetch the control message from the socket by using the get control message subroutine:

```
int ha_rr_get_ctrlmsg(int session_fd,
                      struct ha_rr_ctrl_msg **rr_ctrl_msg,
                      struct ha_em_err_blk *rr_errb)
```

When a Resource Monitor manager has disconnected or a command-based Resource Monitor wants to terminate, the Resource Monitor calls the end session subroutine. This subroutine has the following syntax:

```
int ha_rr_end_session(int session_fd,
                      struct ha_em_err_blk *rr_errb)
```

Each instance of a resource variable known to the Resource Monitor must be registered. The following two subroutines are used to register and unregister a resource variable:

```
int ha_rr_reg_var(struct ha_rr_variable *pv,
                  int numv,
                  struct ha_em_err_blk *rr_errb)

int ha_rr_unreg_var(struct ha_rr_variable *pv,
                    int numv,
                    struct ha_em_err_blk *rr_errb)
```

Two commands are used to add or delete resource variables in the RMAPI interface. The command-based Resource Monitor adds a resource variable to the RMAPI interface immediately after registering its variable. The daemon- or subsystem-based Resource Monitor adds its resource variables to the RMAPI interface after it receives the appropriate control command from the Event Manager Daemon. The syntax of these subroutines is as follows:

```
int ha_rr_add_var(int session_fd,
                  struct ha_rr_variable *pv,
                  int numv, int add_complete,
                  struct ha_em_err_blk *rr_errb)
```

```
int ha_rr_del_var(int session_fd,
                  struct ha_rr_variable *pv,
                  int numv,
                  struct ha_em_err_blk *rr_errb)
```

If a resource is designed to have a configurable reporting frequency for a class of resource variables, then the get reporting frequency subroutine is used by the Resource Monitor to obtain the frequency value. The subroutine has the following syntax:

```
int ha_rr_get_freq(char *cname, struct ha_em_err_blk *rr_errb)
```

Whenever it is time to send variable values to the RMAPI interface, the Resource Monitor calls the send value subroutine. This subroutine has the following syntax:

```
int ha_rr_send_val(struct ha_rr_val *pv,
                   int numv, int refresh,
                   struct ha_em_err_blk *rr_errb);
```

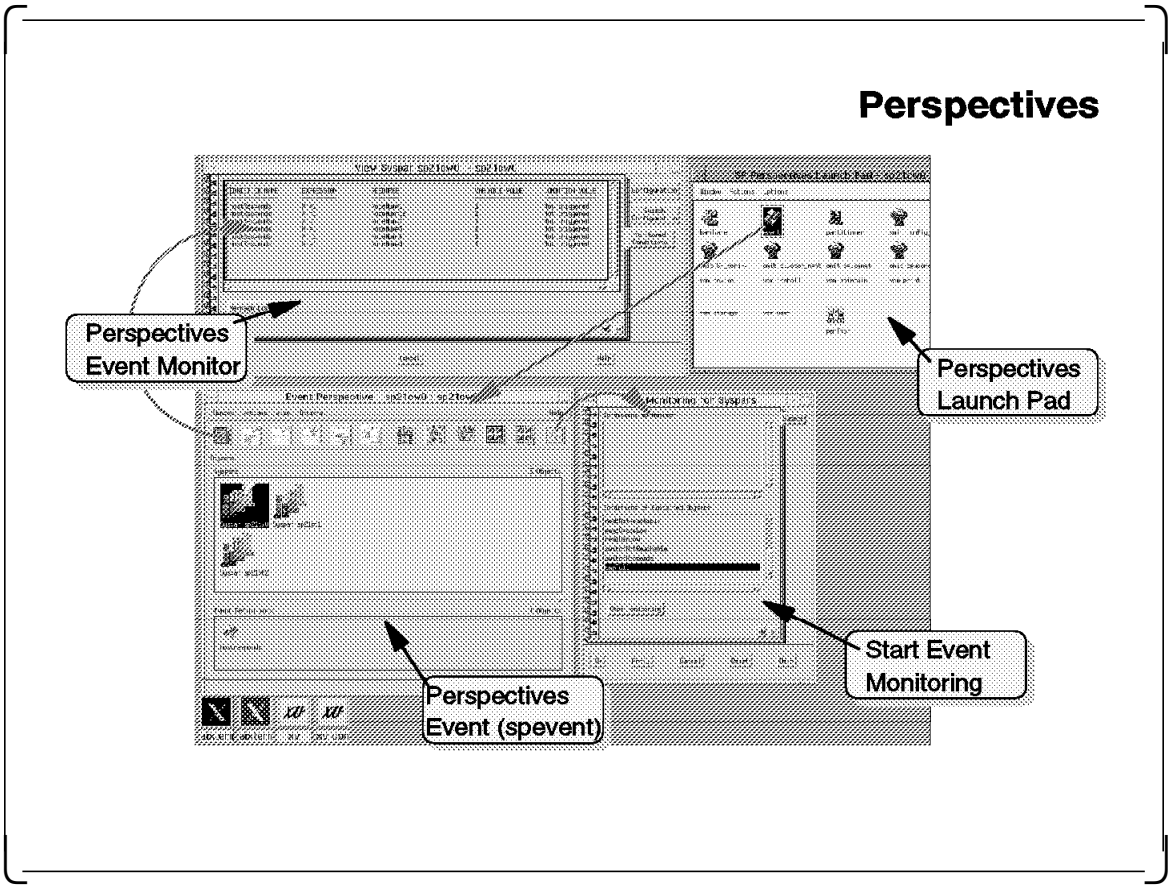
If the Resource Monitor has no values to send within a defined time period, and the variables are of type counter or quantity, then the touch subroutine must be called. This subroutine has the following syntax:

```
int ha_rr_touch(struct ha_em_err_blk *rr_errb)
```

Termination of a Resource Monitor with its managers is done by the terminate subroutine. The terminate subroutine has the following syntax:

```
int ha_rr_terminate(struct ha_em_err_blk *rr_errb)
```

5.27 Perspectives

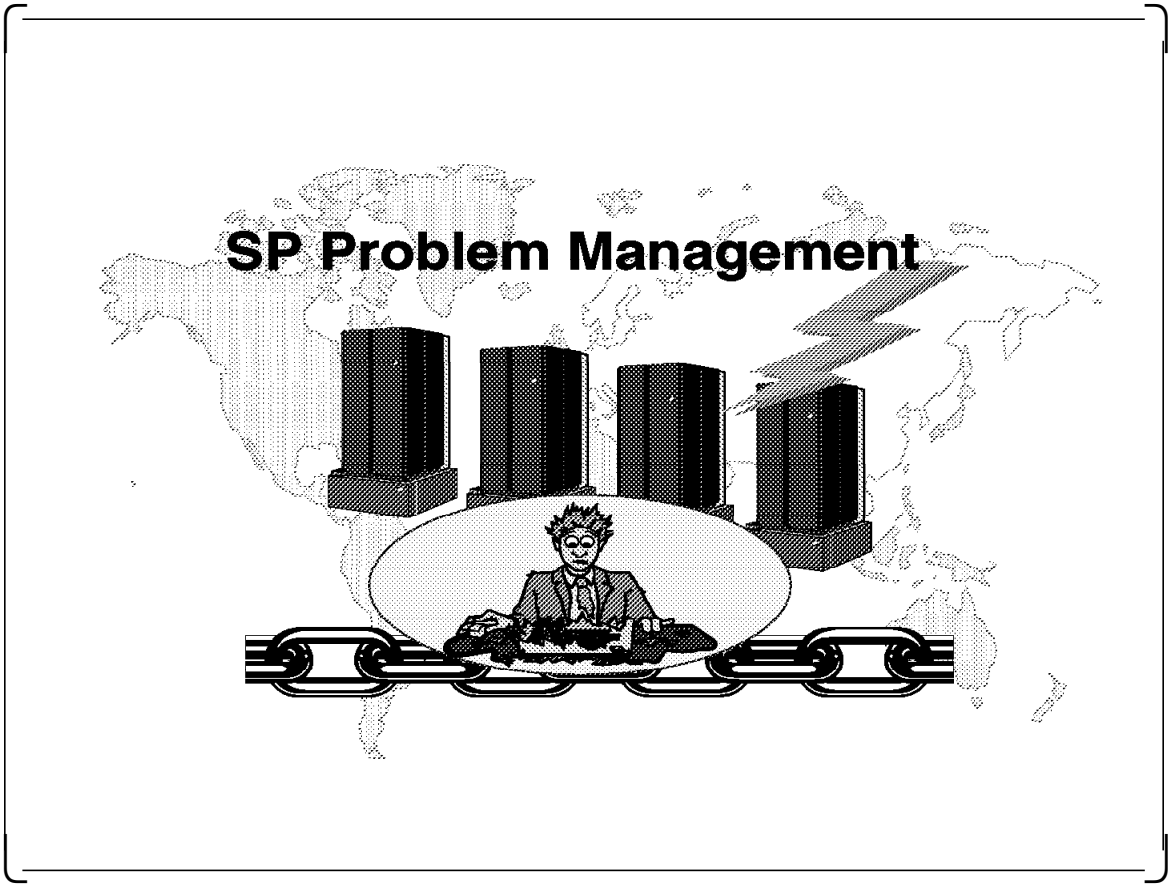


Perspectives is the most frequently used PSSP application that uses the Event Management EMAPI interface. Perspectives can be started with a launch pad by the command `perspectives`, or the Event Management window of the Perspectives can be started by the command `spevent`.

The program behind the graphical interface registers for events and receives notification about occurrence of registered events. The Perspectives environment also enables the user to monitor resource variables and query them (Perspectives Event Monitor).

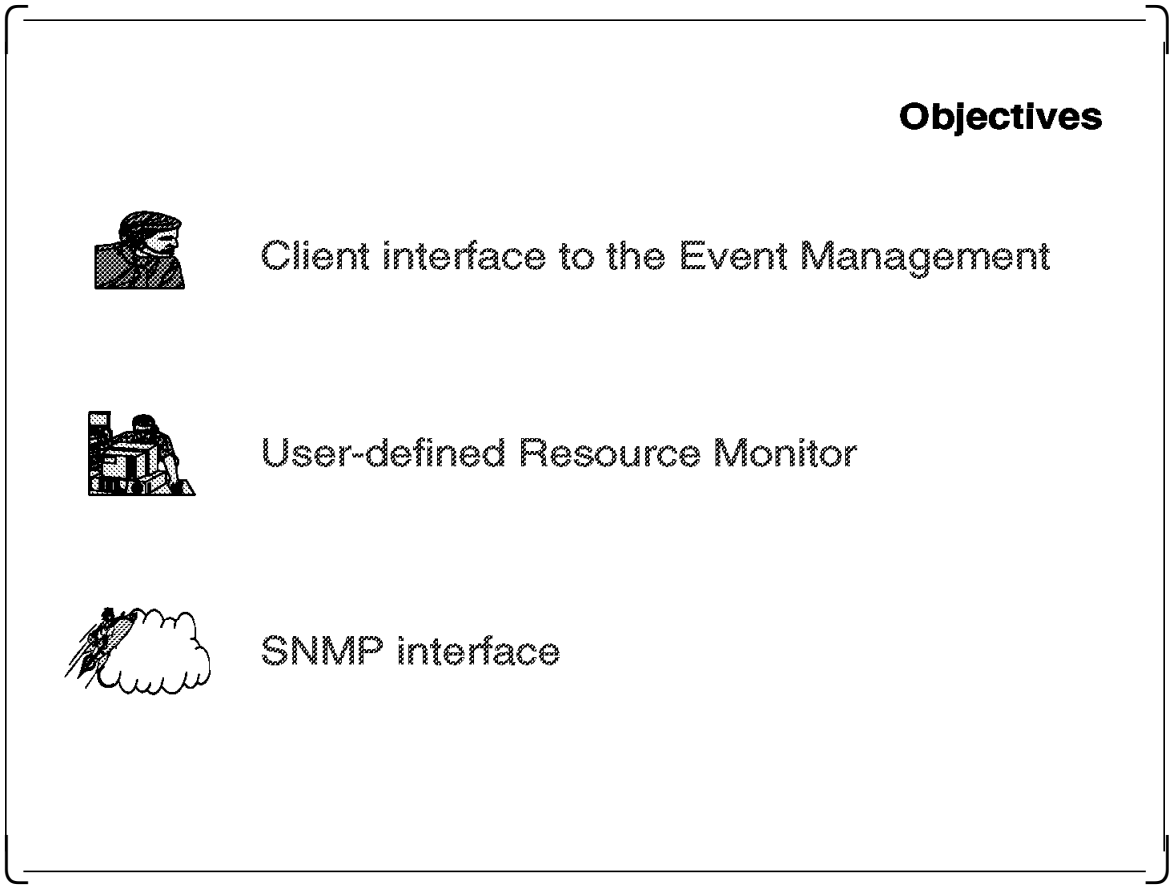
For more information about Perspectives, refer to *RS/6000 SP PSSP 2.2 Technical Presentation*, SG24-4868.

Chapter 6. Problem Management



This section describes the Problem Management subsystem included in the PSSP 2.2 package.

6.1 Problem Management Objectives

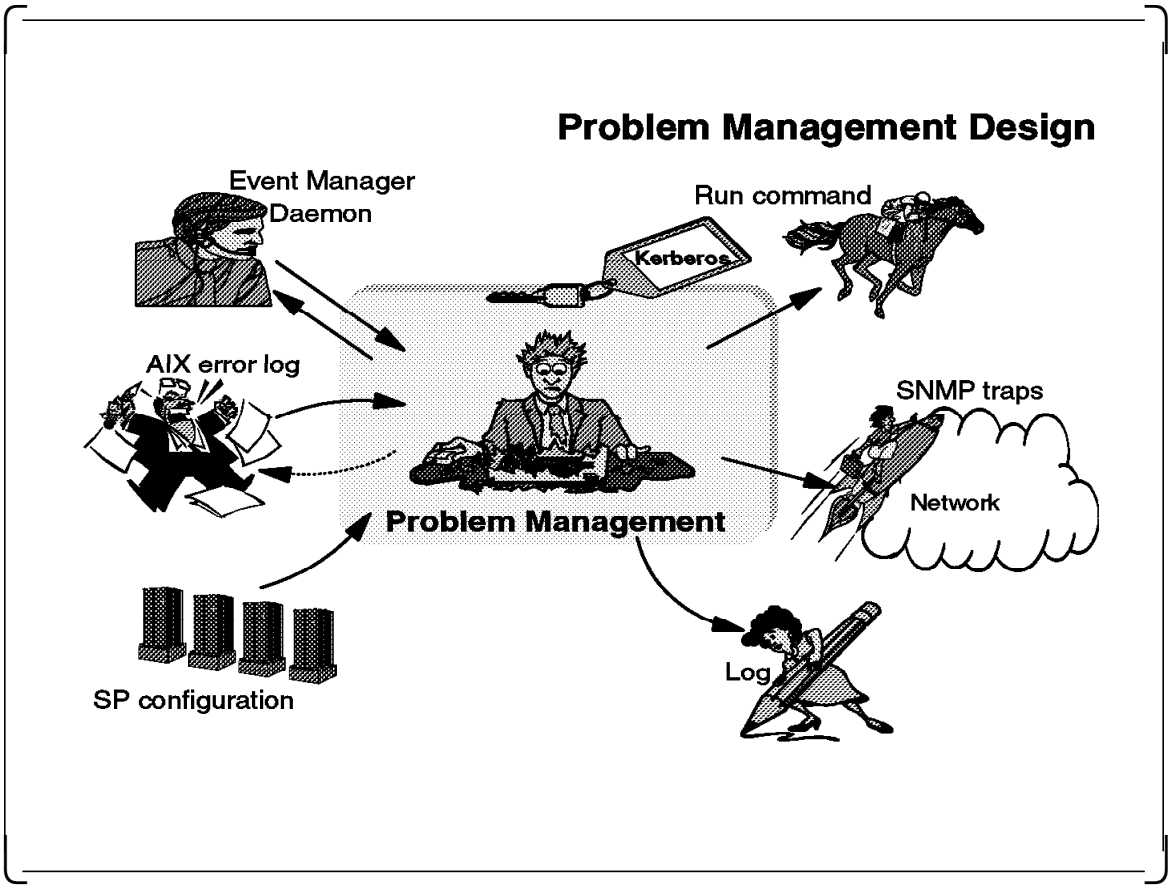


Problem Management provides an infrastructure for recognizing and acting on problem events within the RS/6000 SP system. This infrastructure is based on an Event Management application that provides an Event Management client interface (EMAPI) and Resource Monitor interface (RMAPI) without the necessity of writing C programs that use the Event Management APIs.

The ability to issue an SNMP trap in response to an event allows you to report problem events occurring in your RS/6000 SP system to a TCP/IP-based Internet network manager existing on a remote node. The network manager application is not supplied with the PSSP Version 2 Release 2.

An ibmSP MIB provides a network manager with access to RS/6000 SP configuration information as well as access to Event Management variables.

6.2 Problem Management Design



This section describes the Problem Management design and activities performed by the Problem Management subsystem.

The Problem Management subsystem provides the following functions:

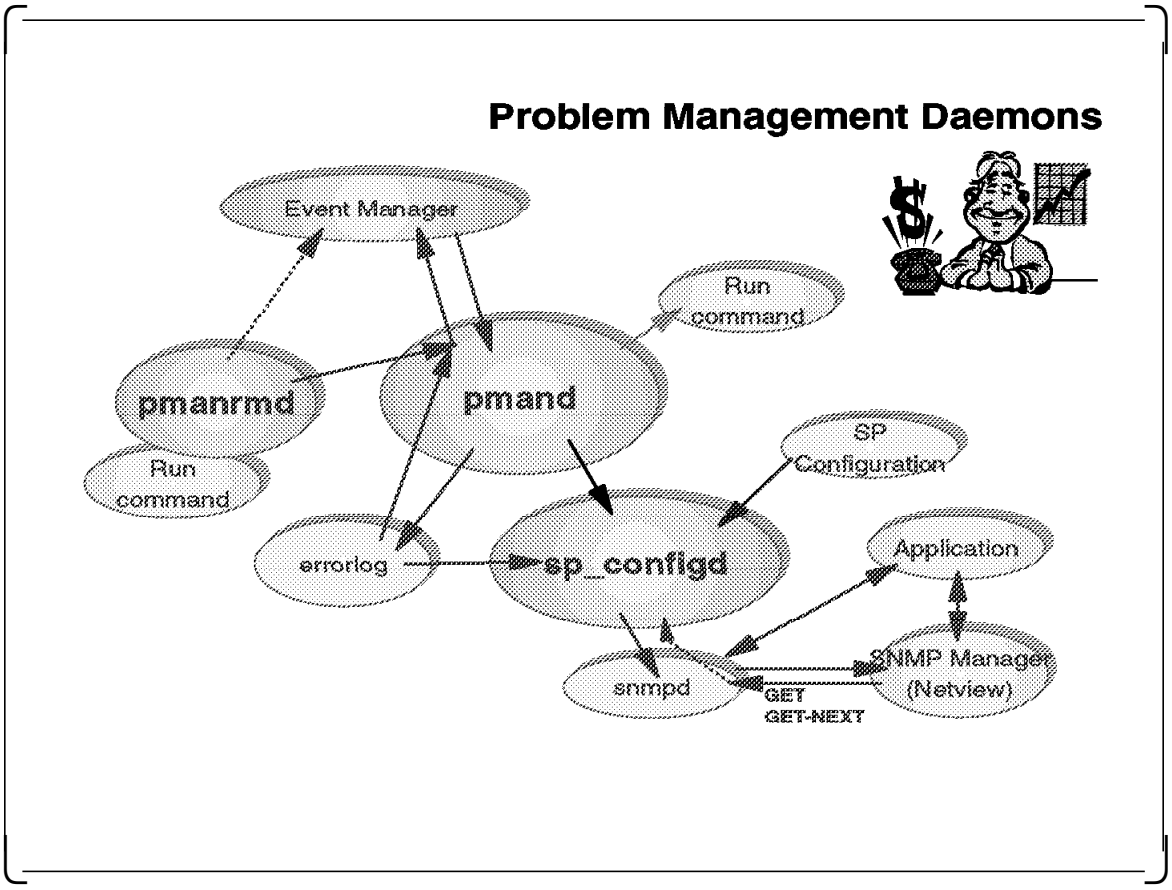
- Providing resource variables to the Event Management subsystem through the resource monitor API (RMAPI)
- Providing an AIX error log resource variable for AIX error log writes
- Providing client access to the Event Management subsystem on all nodes and receiving events from the predefined resources
- Running a user-specified command or script for generated events
- Writing to the AIX error log and BSD syslog for received events from the Event Management
- Generating SNMP traps to SNMP managers from:
  - Event Management events
  - AIX error log writes for entries marked “Alert=true”
- Providing SNMP access to the last SNMP trap issued from the AIX error log on each node
- Providing SNMP read access to the RS/6000 SP configuration information

- Providing SNMP access to the last generated event from the Event Management subsystem for defined subscriptions on the local node
- Providing SNMP access to information about Event Management variables, and the current values of Event Management variables defined on local the node

The Problem Management subsystem registers to the Event Management subsystem for events and is notified by the Event Manager Daemon when an event occurs. The system administrator can specify an action to be taken when an event is reported by the Event Management. Examples of events monitored by default configuration are /tmp 95% full, /var 95% full, and daemons dying.

The Problem Management subsystem is designed to provide a problem reporting methodology to the enterprise. The enterprise, specifically Netview for AIX or another SNMP manager, understands the SNMP protocol. Notifying the enterprise of problems can be achieved by issuing SNMP traps to the SNMP managers. The SNMP trap management is provided by an SNMP manager, like Netview for AIX. This design relies on the AIX `snmpd` daemon for SNMP agent support. All SNMP agent functions in the RS/6000 SP Problem Management are implemented as a SMUX peer subagent, *sp_configd*, of the `snmpd` daemon.

6.3 Problem Management Daemons



The Problem Management consists of the following three daemon processes:

pmand

This daemon is the direct interface to the Event Manager Daemon. The pmand registers for events, receives them, and initiates appropriate actions. Pmand writes the event response supplied by the event manager subsystem to a FIFO file. The FIFO is akin to a pipe.

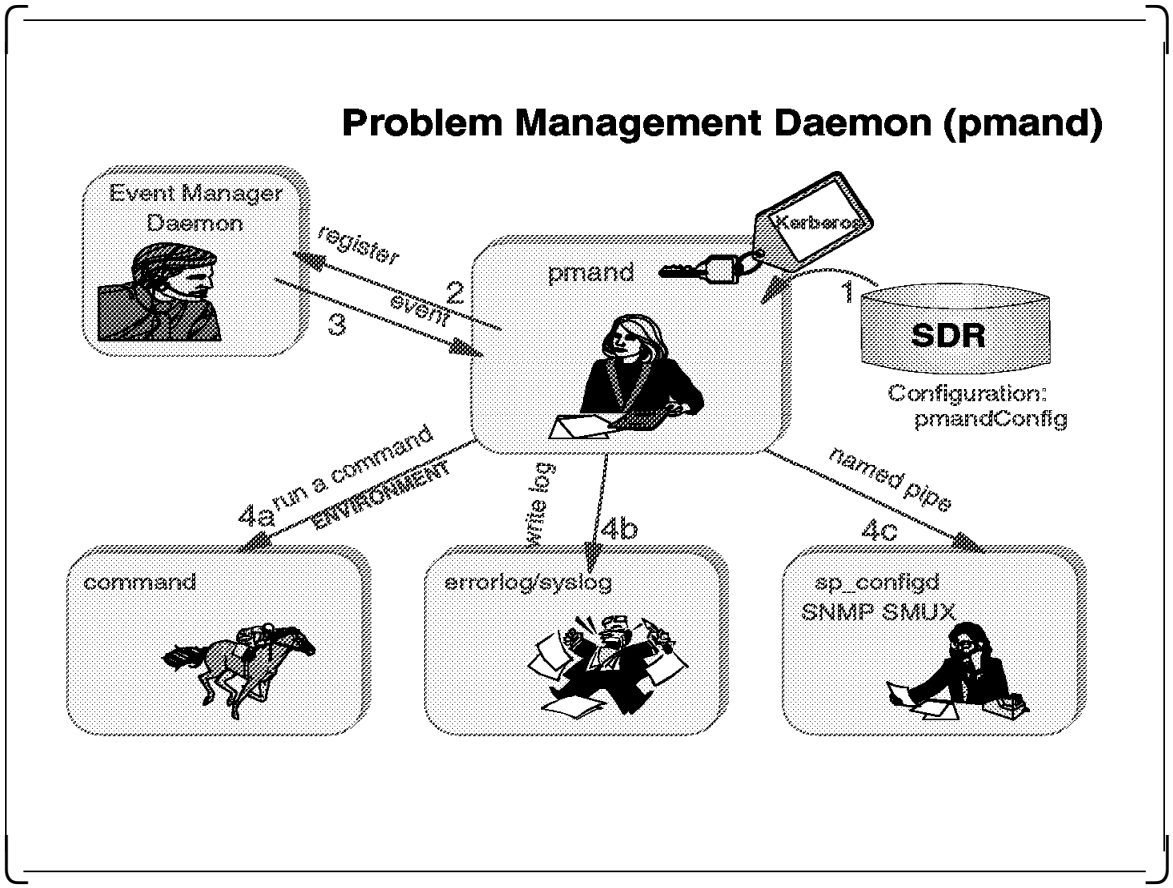
pmanrmd

This daemon is a Resource Monitor subsystem that supplies data to the RMAPI Event Management interface. This daemon also cooperates with the pmand daemon.

sp_configd

This is the third Problem Management daemon. This daemon is generating SNMP traps to the SNMP manager application, like Netview for AIX. This daemon reads the data from the FIFO file (named pipe) and creates an SNMP trap from the data. It also sends SNMP traps when an alterable entry is posted to the AIX error log. It replies to the SNMP GET or GET-NEXT requests, and sends the values of requested variable instances to the requester.

6.4 Problem Management Daemon (pmand)



This section describes the Problem Management Daemon (pmand) functions.

The Problem Management Daemon represents the interface to the Event Management subsystem to which it connects as a client. Every pmand connects to the local Event Manager Daemon. The Problem Management Daemon registers for events. After the pmand has successfully connected to the Event Manager Daemon and registered for events, it waits for a registered event to occur. When the event has occurred and the notification is received, the pmand starts the actions defined in the configuration information.

The actions defined to the Problem Management Daemon can be:

- Run a command.
- Issue an SNMP trap.
- Write to the AIX error log or BSD syslog.

The Problem Management Daemon (pmand) can be configured to run a recovery or notification command when an event occurs. This command is run on the node where the pmand is running. The commands can be run concurrently, and any number of recovery commands can be run simultaneously. Commands are initiated when the daemon receives a corresponding event from the Event Manager Daemon, regardless of whether recovery commands are already running for that or any other event. The system administrator can specify any

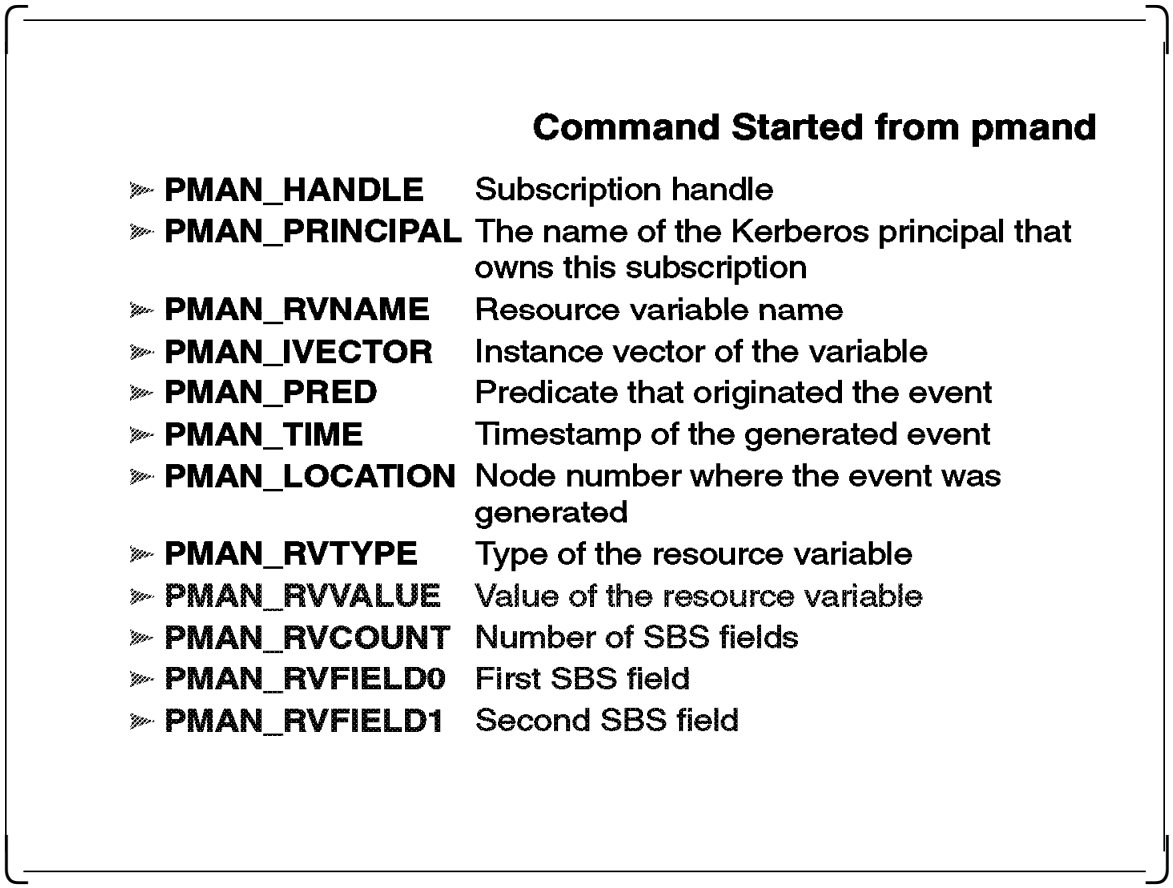
executable program or script. The *notify_event* sample script may be specified to send mail to the user running the command on the local node, or the *log_event* sample script may be specified to log event information into a wraparound file.

Each time the pmand starts a process as a reaction to an event, it passes environment variables to that process.

If an SNMP trap must be generated, the pmand daemon passes the request to the sp_configd to format and issue an SNMP trap. To identify the event that occurred, the pmand passes to the sp_configd the trap ID specified in the pmand configuration. The pmand also passes information about the event. For detailed information about the SNMP trap generation, see section 6.17, “SNMP Traps from Events (ibmSPEMEvent)” on page 227.

The system administrator may specify text that gets written to the AIX error log and BSD syslog when a specific event occurs.

6.5 Command Started from pmand



This section describes the environment that is passed to a command that is started from the pmand as a reaction to an event.

Each time the pmand starts a process, as a reaction to an event, it passes environment variables to that process. These environment variables are:

PMAN_HANDLE

This is a handle that uniquely identifies the subscription.

PMAN_PRINCIPAL

This is the name of the Kerberos principal that owns this subscription.

PMAN_RVNAME

This is the resource variable name.

PMAN_IVECTOR

This is the instance vector of the resource variable.

PMAN_PRED

This is the predicate that originated the event.

PMAN_TIME

This is the time stamp indicating the time the Event Manager Daemon generated the event.

PMAN_LOCATION

This is the node where the event was generated.

PMAN_RVTYPE

This is long, float, or SBS, depending on whether the type of the resource variable value is a long integer, a floating point value, or a Structured Byte String.

PMAN_RVVALUE

This is the value of the resource variable that triggered the event. This variable exists only if the resource variable is of type long or float. If the resource variable is of type SBS, the PMAN_RVCOUNT and PMAN_RVFIELDn are set.

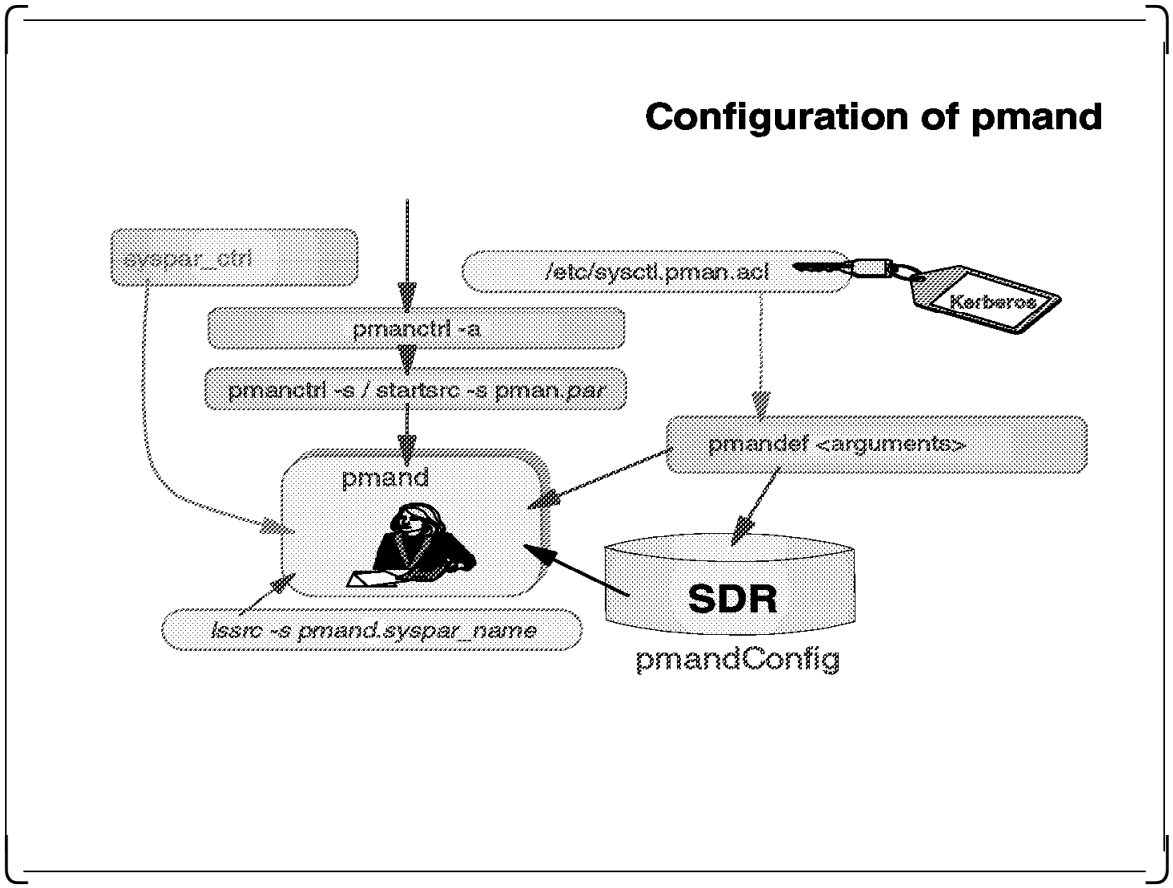
PMAN_RVCOUNT

If the PMAN_RVTYPE is of type Structured Byte String (SBS), then the resource variable value is composed of one or more structured fields. The PMAN_RVCOUNT defines the number of fields within the Structured Byte String.

PMAN_RVFIELDn

If the PMAN_RVCOUNT variable is set (the resource variable is an SBS), then there are as many PMAN_RVCOUNTn variables as fields in the Structured Byte String resource variable. The n character in the PMAN_RVFIELDn represents the field serial number. For example, if there are three fields in an SBS resource variable, then the PMAN_RVCOUNT is set to three and there are three variables: PMAN_RVFIELD0, PMAN_RVFIELD1, and PMAN_RVFIELD2.

6.6 Configuration Steps of pmand



The Problem Management pmand subsystem is an AIX SRC controlled subsystem. The subsystem must be present in the AIX ODM database. To add the subsystem, you have to run the `pmanctrl -a` command. This command calls the `mkssys` command to create the subsystem under the AIX SRC and to enter the subsystem definition into an ODM object. This command also makes an entry into the /etc/inittab file. This entry will start the Problem Management subsystem every time the system is restarted.

The Problem Management subsystem is under the control of AIX SRC. To start the subsystem, issue one of the following commands:

- `pmanctrl -s`
- `startsrc -s | -g  <subsystem | group>`

The `startsrc` command, if called with the -s parameter, only starts one subsystem for the partition specified by the subsystem name. If the `startsrc` command is called with -g parameter, every Problem Management daemon is started. The Problem Management subsystem AIX SRC group name is *pman*.

If you want to allow non-root users to perform a Problem Management configuration, you must allow them to have access. To do this, make an entry in the Kerberos ACL file, /etc/sysctl.pman.acl. If the subscription is defined to write into the AIX error log, or generate SNMP traps, then the Kerberos principal that owns the subscription must be listed in the root user's *$HOME/.klogin* file.

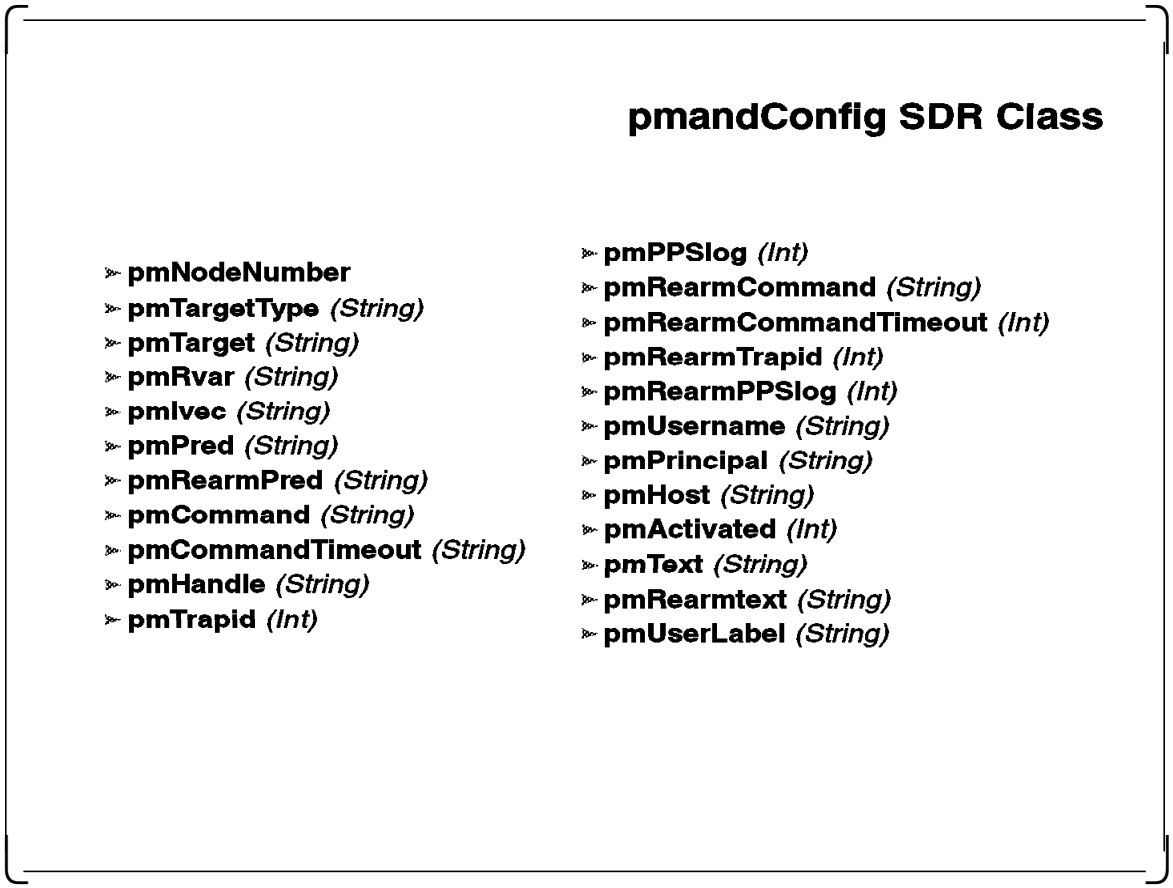
When the pmand is started, it reads its configuration (defined subscriptions) from the SDR database. The pmand configuration is stored in the pmandConfig SDR class.

To add new subscriptions, you have to issue the `pmandef` command, which creates new SDR objects in the pmandConfig SDR class and updates the pmand daemon with the new subscription. The `pmandef` command is described in section 6.8.2, “pmandef” on page 207.

The pmand subsystem is restarted each time the system is repartitioned. The syspar_ctrl script invokes the `pmanctrl` command in order to recreate the subsystem.

To check the correct state of the pmand subsystem, issue the `lssrc` command. This command is described in section 6.8.3, “pmanquery” on page 210.

6.7 pmandConfig SDR Class



The configuration information of the pmand subsystem is stored in the *pmandConfig* SDR class. This class contains the following information:

pmNodenumber

This is the node number where the subscription has to be activated.

pmTargetType

This defines the type of the pmTarget information.

pmTarget

This defines the target nodes, or Control Workstation, where the pmand starts a subscription to the Event Manager Daemon.

pmRvar

This is the registered resource variable.

pmIvec

This specifies the instance vector for the resource variable.

pmPred

This is the predicate used for the registration to the Event Manager Daemon.

pmRearmPred

This specifies the rearm predicate used to register for an event.

pmCommand

This is the full path of a program that is started when the event is generated.

pmCommandTimeout

This is the time limit, in seconds, that the command, started as a reaction to an event, is expected to run. After this time, the command is killed by SIGTERM, followed by SIGKILL.

pmHandle

This is a handle that the subscription can be referenced by.

pmPPSlog

If this is different from zero, for every event received, an AIX error log entry and a BSD syslog entry is written.

pmRearmCommand

This is to define the command to run when rearm events occur.

pmRearmCommandTimeout

This is the rearm command timeout, like the pmCommandTimeout.

pmRearmTrapid

This is the specification of whether an SNMP trap is to be generated when a rearm command occurs.

pmRearmPPSlog

This specifies whether an AIX error log and BSD syslog entry are to be generated when a rearm event occurs.

pmUsername

This is the user name that the commands started from pmand use.

pmPrincipal

This is the Kerberos principal of the user.

pmHost

This is the hostname of the system node, where the subscription is done.

pmActivated

If this is zero, then this subscription is not activated automatically.

pmText

This is text that is written to the AIX error log and BSD syslog when an event occurs.

pmRearmText

This is text that is written to the AIX error log and BSD syslog when a rearm event occurs.

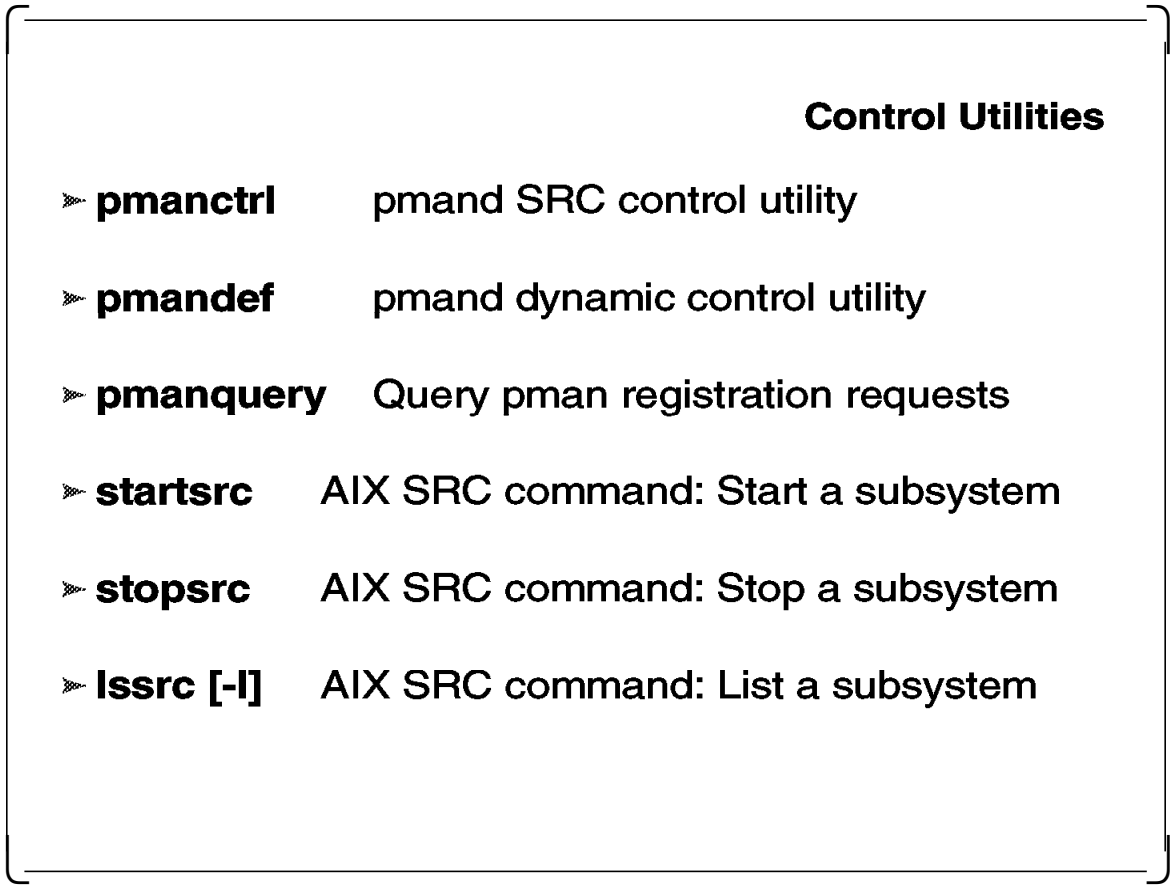
pmUserLabel

This is a label text that can be used by a user.

There are also other variables of the pmandConfig class that are present in the SDR, but they are not configurable:

- pmEventid
- pmThrottle

6.8 pmand Control Utilities



This is a list of the Problem Management control commands:

pmanctrl

This command is used to control the pmand subsystem as an AIX SRC subsystem. This command is explained in section 6.8.1, “pmanctrl” on page 205.

pmandef

This command is used to handle event subscription. A more detailed explanation is provided in section 6.8.2, “pmandef” on page 207.

pmanquery

This command is used to query the subscriptions initiated by a user. The command returns an output based on the information in the SDR.

startsrc

The startsrc command is used to start the Problem Management subsystem. Following are examples of its use:

```
/usr/bin/startsrc -s pman         (on SP nodes)
/usr/bin/startsrc -s pman.syspar_name  (on the Control Workstation)
/usr/bin/startsrc -g pman   (Starts every Problem Management daemon)
```

stopsrc

The stopsrc command is a standard AIX SRC command that stops the subsystem or a group of subsystems. Following are some examples:

```
/usr/bin/stopsrc -s pman         (on SP nodes)
/usr/bin/stopsrc -s pman.syspar_name  (on the Control Workstation)
/usr/bin/stopsrc -g pman   (Stops every Problem Management daemon)
```

lssrc

The lssrc command is used to list subsystems and their state. The lssrc command accepts the -l flag to display long format output. The following example shows how to use the command:

```
/usr/bin/lssrc -a
/usr/bin/lssrc -s pman    (on the SP nodes)
/usr/bin/lssrc -s pman.syspar_name  (on the Control Workstation)
/usr/bin/lssrc -l -s pman.syspar_name (on the Control Workstation)
```

There are additional commands that are present in the system, but they are used internally by the Problem Management subsystem:

pmaneventon

This command is used to activate a subscription. This command is called from the pmandef command; users are not expected to use this command directly.

pmaneventoff

This command is used to deactivate a subscription. This command is called from the pmandef command; users are not expected to use this command directly.

pmanq

This command queries the pmand daemon for the status of a subscription. This command is called from the pmandef command; users are not expected to use this command directly.

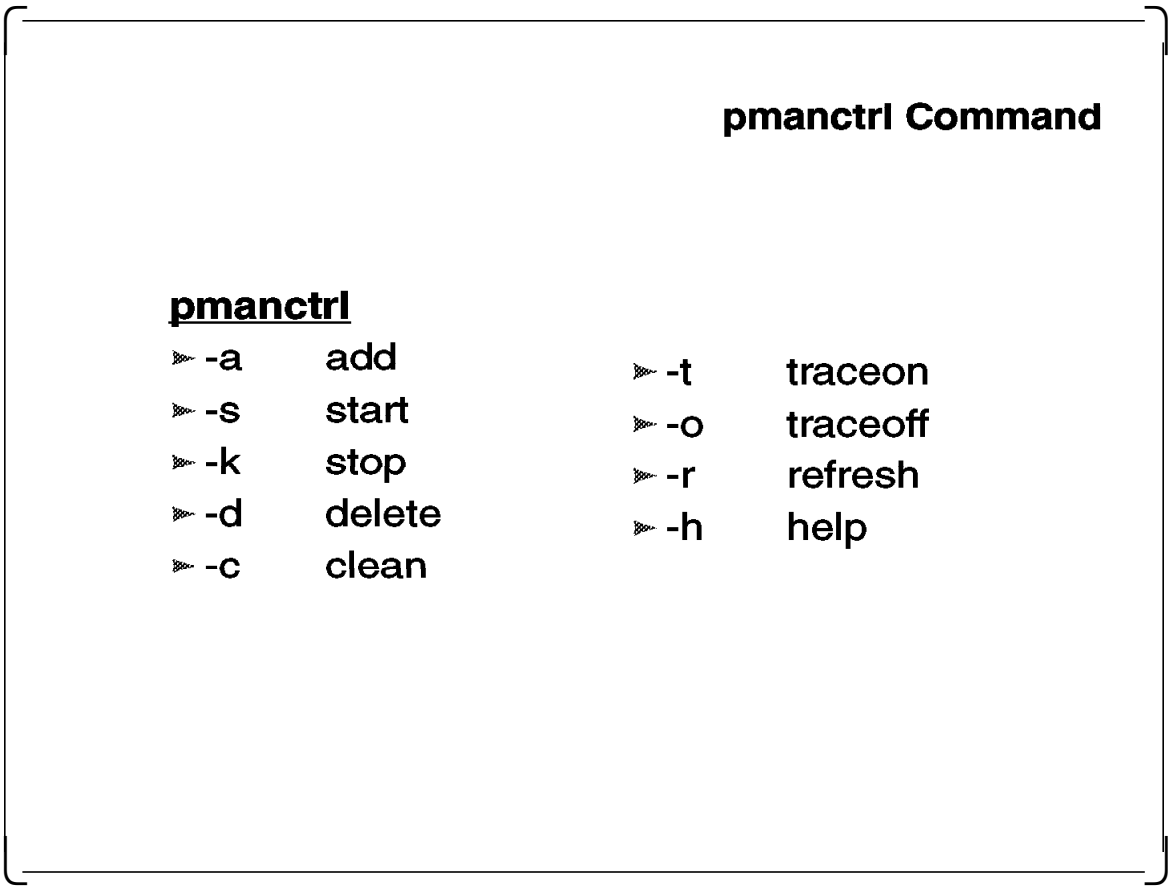
pmansubscribe

This command creates a subscription to an event. This command is called from the pmandef command; users are not expected to use this command directly.

pmanunsubscribe

This command deletes the specified subscription. This command is called from the pmandef command; users are not expected to use this command directly.

6.8.1 pmanctrl



The `pmanctrl` command is used to control the Problem Management subsystem as an AIX SRC subsystem. The parameters that specify the type of action to be performed by this command are the following:

-a/add

This flag specifies the request to create a pmand subsystem for the current partition.

-s/start

This flag stands for starting the Problem Management subsystem in the current partition. The subsystem is then started by the `startsrc` command.

-k/stop

This flag is the opposite flag of -s/start. The command causes the Problem Management subsystem to stop.

-d/delete

The delete function causes the Problem Management subsystem to be deleted from the current partition. The SDR objects are not deleted.

-c/clean

The clean function causes the Problem Management subsystem to be deleted from all partitions. The SDR objects are not deleted.

-t/trace

This function of the `pmanctrl` command turns on tracing for the Problem Management subsystem. The trace output is in the /var/adm/SPlogs/pman directory.

-o/traceoff

This turns the trace off.

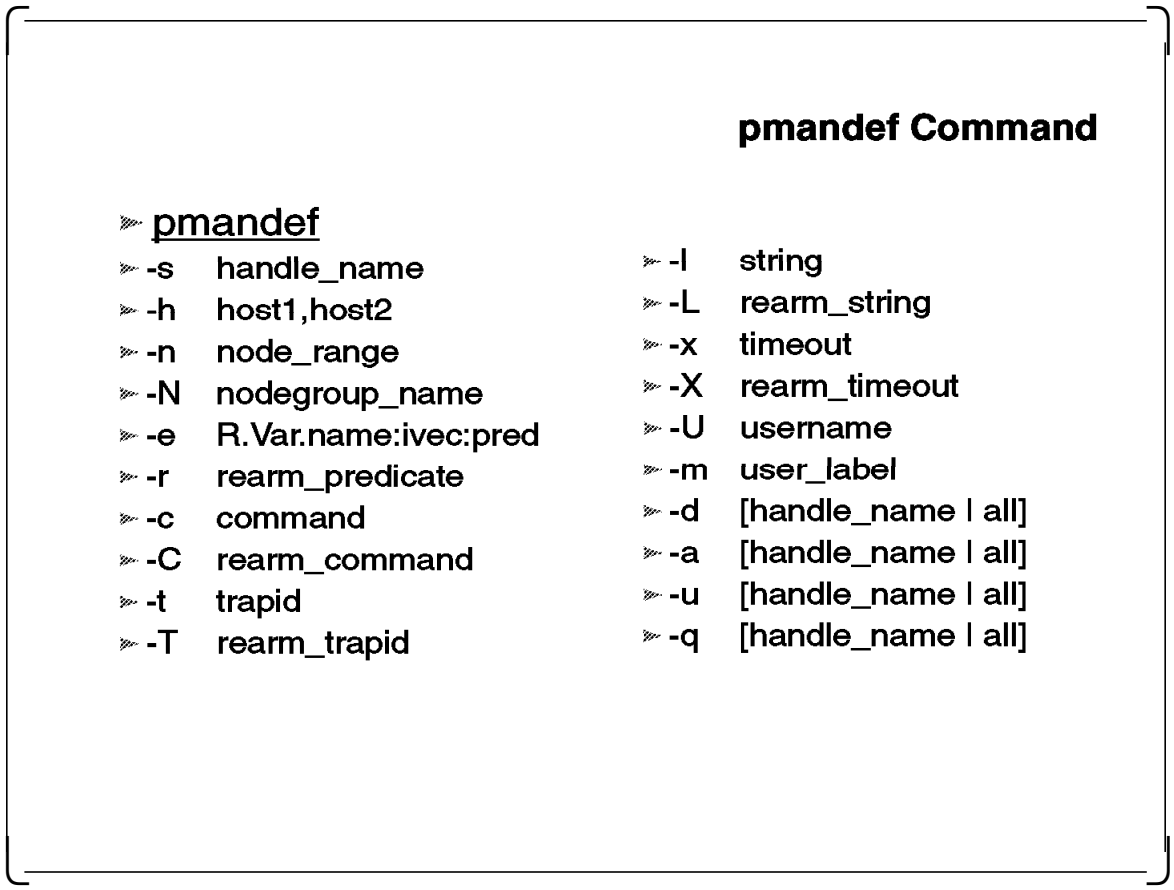
-r/refresh

The refresh function does nothing at this time.

-h/help

This displays the usage information.

6.8.2 pmandef



The pmandef command is the mechanism provided for creating Problem Management subscriptions.

The pmandef command defines:

- What Event Management events to register for
- What actions to take when those events occur

There are three actions the pmand daemon can initiate after the event notification is received:

- Run a command.
- Issue an SNMP trap.
- Write to the AIX error log or BSD syslog facilities.

The pmandef command is based on Sysctl, which uses Kerberos for user authentication. All users of the pmandef command must have valid Kerberos authentication. In addition, the user's Kerberos principal must be listed in the /etc/sysctl.pman.acl file on the local node, to store the subscription in the SDR and on all the nodes that are affected by the new subscription. This enables the affected Problem Management daemons to be notified of the new subscription.

The information, specified by parameters that have to be supplied with the command, has the following format: pmandef <arguments>, where the argument can be the following:

-s handle_name

This specifies that the command is a subscribe request, and the remaining flags are to define the Problem Management subscription. The handle_name is an identifier of the subscription.

-d handle_name

This specifies that the registrations identified by the handle_name should be deactivated.

-a handle_name

This specifies that the subscription specified by the handle_name should be activated.

-u handle_name

This specifies that the subscription identified by the handle_name should be removed (unsubscribed).

-q handle_name

This is a query request for the subscription identified by the handle_name.

Note: The -d, -a, -u, and -q arguments can be followed by the *all* keyword. You can also specify the following attributes when creating a new subscription:

-h host1,host2,...

This specifies the hosts that belong to the subscription. Instead of this parameter, the -N or -n parameter can be specified. These represent a node group (-N) or a node range (-n).

-e resource_variable:instance_vector:predicate

With this parameter and the attributes that follow it, you can specify the resource variable, instance vector, and the predicate. These are explained in section Chapter 4, “Resource Monitors” on page 119.

-r rearm_predicate

This specifies the Event Management rearm predicate.

-c command

This specifies a command to be executed when the event defined by the -e flag occurs.

-C rearm_command

The rearm_command is the name of the command that is to be run when the rearm event occurs.

-t trapid

When this flag is specified, an SNMP trap is to be generated when an event occurs. The “trapid” is the specific trap ID to be used.

-T rearm_trapid

This flag specifies that an SNMP trap is to be generated when a rearm event occurs, and the rearm_trapid is the specific trap ID to be used.

-l log_text

If this flag is specified, an entry is written to the error log and syslog when the event occurs. The log_text is put into the description of the log entry.

-L rearm_log_text

If this flag is specified, an entry is written to the error log and syslog when the rearm event occurs. The rearm_log_text is put into the description of the log entry.

-x timeout

This flag specifies the time limit, in seconds, for the command initiated by the event. If the command does not complete within the specified timeout, a SIGTERM signal is sent to this command. If the command does not complete within an additional five seconds, a SIGKILL signal follows.

-X rearm_timeout

This flag specifies the time limit, in seconds, for the command initiated by the rearm event. If the command does not complete within the specified timeout, a SIGTERM signal is sent to this command. If the command does not complete within additional five seconds, a SIGKILL signal follows.

-U username

This specifies the username that the command and rearm command use when they run.

-m user_label

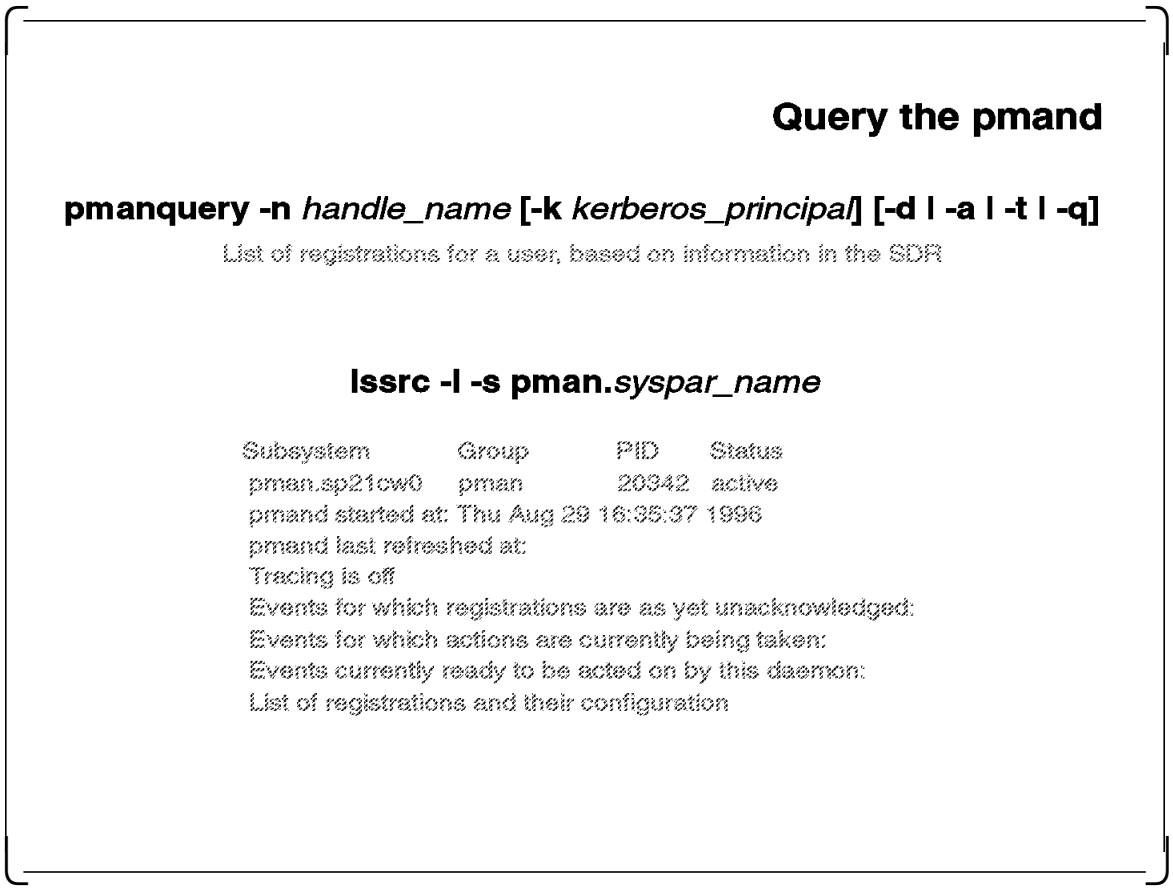
This defines a label that may be used by user. This label can be retrieved by the pmanquery command.

An example of the pmandef command follows:

```
pmandef -s program_monitor \
-e 'IBM.PSSP.Prog.pcount:NodeNum=1;ProgName=mycmd;UserName=bob:X@0==0' \
-r "X@0==1" -c "echo program has stopped > /tmp/myevent.out"\
-C "echo program has restarted >/tmp/myrearm.out"
```

This example causes a write in the /tmp/myevent.out when the mycmd program ends, and causes a write to the /tmp/myrearm.out when the program restarts.

6.8.3 pmanquery



The pmanquery command is used to query the SDR for a description of a Problem Management subscription. After a Problem Management subscription definition has been stored in the SDR by the pmandef command, the pmanquery command may be used to retrieve the subscription definition. The pmanquery command prints the details of the subscription definition in a raw format which is intended to be parsed by other applications. The -q, -a, -d, and -t flags are used to format the output of the command. The -n and -k flags control the scope of the search for subscriptions in the SDR.

-n handle_name

The -n flag followed by the handle name or the “all” keyword searches for Problem Management subscriptions with the specified handle name that was previously given as the argument to the -s flag of the pmandef command. The “all” keyword allows any handle name to be selected by the search of the SDR.

-k Kerberos_name

The -k flag followed by the Kerberos principal name searches for Problem Management subscriptions owned by the specified Kerberos principal. If the -k flag is omitted, the caller′s Kerberos principal is used. The “all” keyword allows any Kerberos principal to be selected by the search of the SDR.

Following are examples of the pmanquery command:

```
pmanquery -n all
pmanquery -n my_handle -k all
pmanquery -n all -k all
```

Following is an example of the output created by the pmanquery command:

```
pmActivated:pmHandle:pmRvar:pmIvec:pmPred:
pmCommand:pmCommandTimeout:

pmTrapid:pmPPSlog:pmText:pmRearmPred:
pmRearmCommand:pmRearmCommandTimeout:pmRearmTrapid:
pmRearmPPSlog:pmRearmText:
pmUsername:pmPrincipal:pmHost:
pmTargetType:pmTarget:pmUserLabel

1:Ad_test:IBM.PSSP.pm.User_state1:
NodeNum=*:X@0!="":/test/hello2.ad:10:
-1:1:Somebody touched my file!:X@0=="":
/test/bye2.ad:10:-1:1:
Somebody deleted my file!:adrian:
adrian.@MSC.ITSO.IBM.COM:sp21cw0:NODE_RANGE:0:

1:Tmp_filling_on_nodes:IBM.PSSP.aixos.FS.%totused:
NodeNum=*;VG=rootvg;LV=hd3:X>75:/test/hello.ad:0
:-1:0: :X<60:
/test/bye.ad:0:-1:0: :
root:root.admin@MSC.ITSO.IBM.COM:
sp21cw0:NODE_RANGE:0:1
```

6.8.4 lssrc Command

The lssrc command is a standard AIX SRC command that gets the status of a subsystem, a group of subsystems, or a subserver. When the -l flag is specified, the lssrc command sends a subsystem request to the AIX SRC daemon to be forwarded to the subsystem (pmand). The subsystem returns information in a long format.

The lssrc -l command provides the following status information:

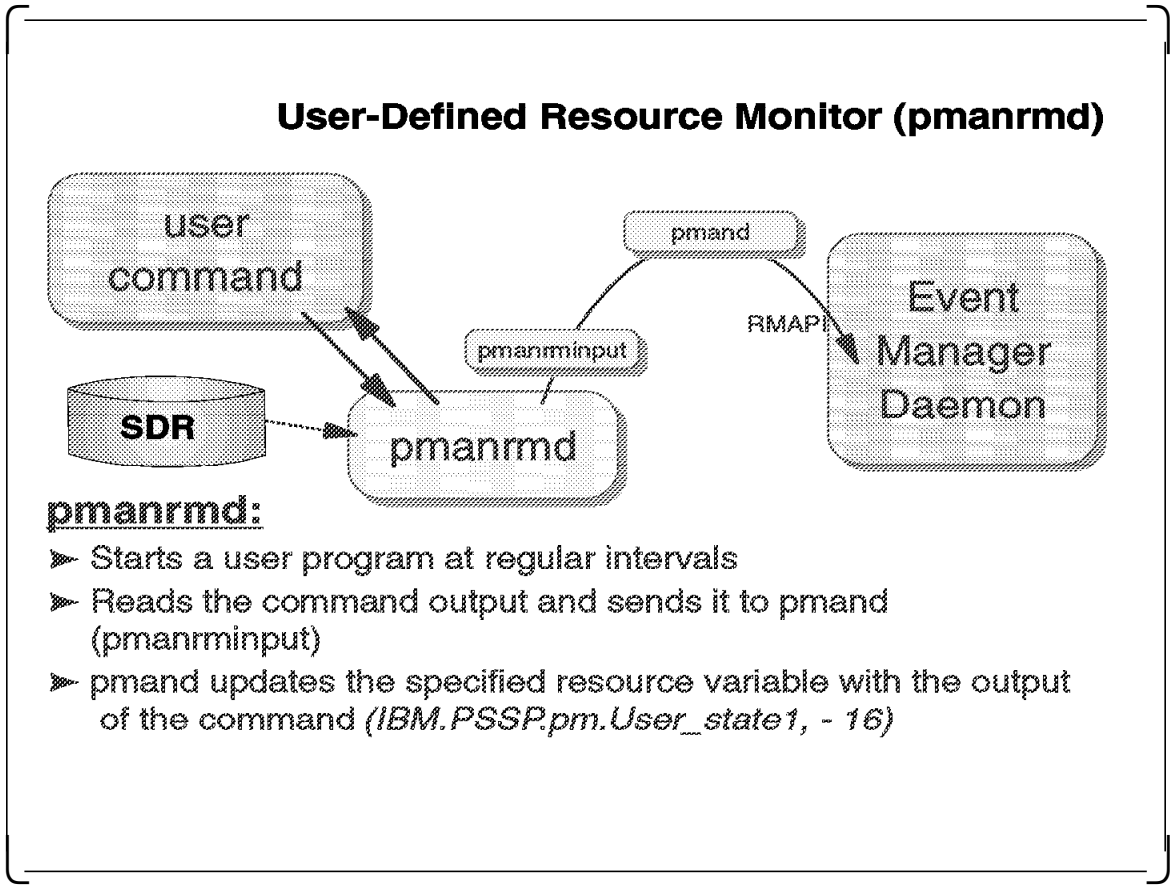
- When pmand was started.
- When pmand was last refreshed.
- Whether tracing is turned on or off. When debug mode is on, all SRC requests and all events are logged to the */var/adm/SPlogs/pman* directory.
- Events for which registrations are as yet unacknowledged.
- Events for which actions are currently being taken.
- Events currently ready to be acted on by this daemon.

An example of the lssrc -l output follows:

```
# lssrc -ls pman.sp21cw0

Subsystem         Group            PID     Status
 pman.sp21cw0     pman             35900   active

 pmand started at: Sat Aug 31 17:56:10 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:
==========================================================
------------------ Adrian_test ----------------
Currently ACTIVE
Client adrian adrian.@MSC.ITSO.IBM.COM at sp21cw0
Resource Variable: IBM.PSSP.SP_HW.Node.keyModeSwitch
Instance: NodeNum=*
Predicate: X!=0
Command to run: /test/hello.ad
Has run 0 times
Logging on with text: Keymode switch changed FROM normal.
Rearm predicate: X==0
Command to run on rearm event: /test/bye.ad
Has run 0 times
Logging on for rearm event with text: Keymode switch changed TO
normal.
------------------ Tmp_filling_on_nodes ----------------
Currently ACTIVE
Client root root.admin@MSC.ITSO.IBM.COM at sp21cw0
Resource Variable: IBM.PSSP.aixos.FS.%totused
Instance: NodeNum=*;VG=rootvg;LV=hd3
Predicate: X>75
Command to run: /test/hello.ad
Has run 0 times
Rearm predicate: X<60
Command to run on rearm event: /test/bye.ad
Has run 0 times
```

6.9 User-Defined Resource Monitor



The Event Management Resource Monitor interface is an API that requires users to write C programs in order to access it. The Problem Management product offers an interface to create user-defined script-based Resource Monitors as Perl or shell (Korn-shell, C-shell or Bourne-shell) scripts.

The user-defined programs, or scripts, are any program, script, or UNIX command. The output of the program has to be a character string.

There are sixteen resource variables defined in the Event Management subsystem that are dedicated to the user-defined resource monitors. Their names are IBM.PSSP.pm.User_state*nn*, where the *nn* is a number from 1 to 16. These resource variables are of type Structured Byte String, and consist of a single character string (cstring). The instance vector for each is NodeNum.

The *pmanrmd* is a daemon that runs commands at set intervals and causes the standard output (stdout) of these commands to be provided as the user state resource variables. One instance of the daemon runs on each node, supplying variables for that node. On the Control Workstation, there is one instance of pmanrmd for each partition. This structure corresponds to the instances of Event Management operational domains.

The pmanrmd updates the configured resource variables by issuing the `pmanrminput` command. This command accepts an argument that defines the

resource variable name and the value the resource variable has to be updated with. The command has the following format:

```
pmanrminput -a argument -s pman.syspar_name
```

The argument represents the specification of the resource variable and the value that has to be inserted in the resource variable. The format of the argument is a string that consists of the following:

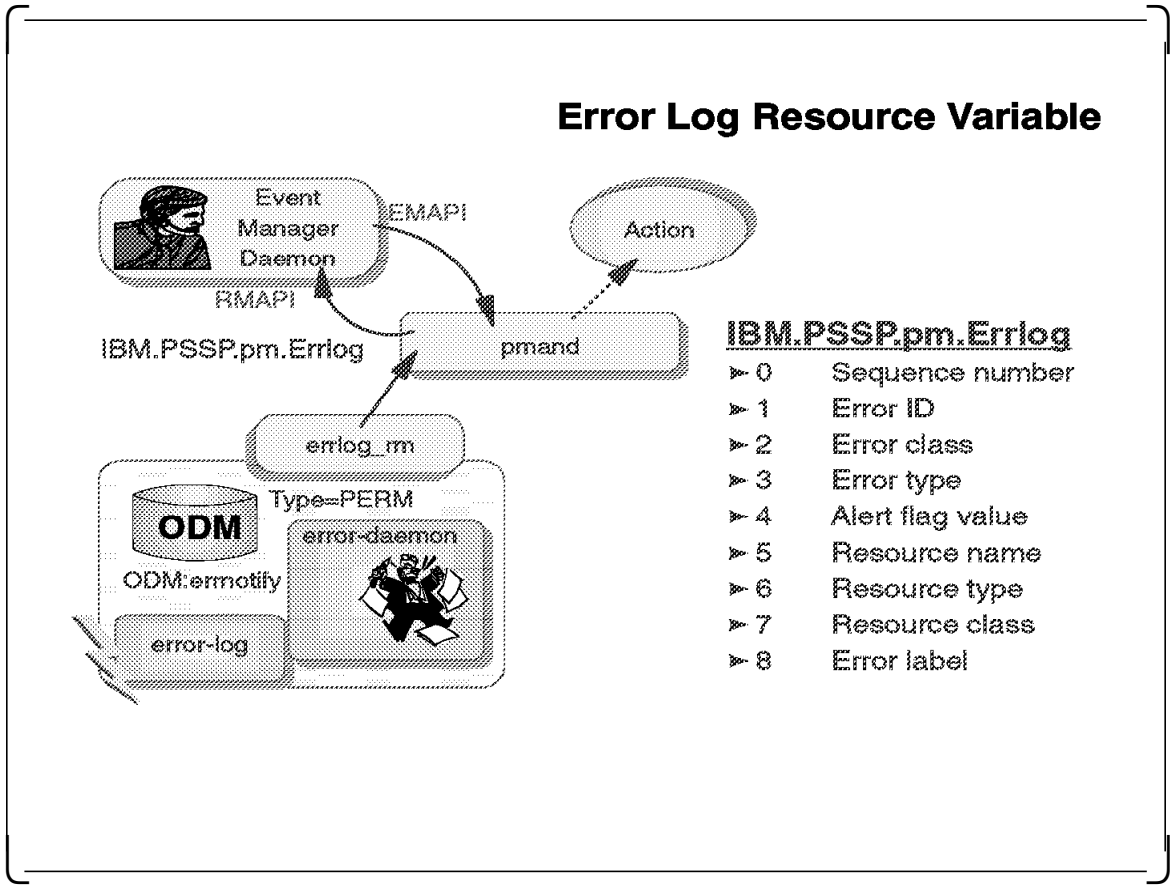
1. The resource variable name
2. A delimiter (+)
3. The resource variable value (character string)
4. Second delimiter (+)

Following is an example of the pmanrminput command:

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

Actually, the pmanrminput command calls the pmand subsystem. The pmand subsystem is directly connected to the RMAPI interface of the Event Manager Daemon.

6.10 Error Log Resource Variable



There is an AIX error log variable defined in the Event Management. This variable is updated by the pmand daemon, which is a client to the Event Manager Daemon. The error notification is initiated by the AIX error log notification method, which is configured in the ODM *errnotify* class. Use the odmget command to see the ODM error notification objects, for example:

```
odmget errnotify
odmget -q en_name="errlog_rm" errnotify
```

This errnotify object sets the error daemon to run the errlog_rm script every time there is an error entry that has specified error type *permanent* (PERM). The /usr/lpp/ssp/bin/errlog_rm program updates the *IBM.PSSP.pm.Errlog* Event Management resource variable. This resource variable is of Structured Byte String type, where the fields are of character string (cstring) type. This is a list of the defined errlog fields:

Serial Number Description

0 Sequence number

1 Error ID

2 Error class

3 Error type

4 Alert flag value

5 Resource name

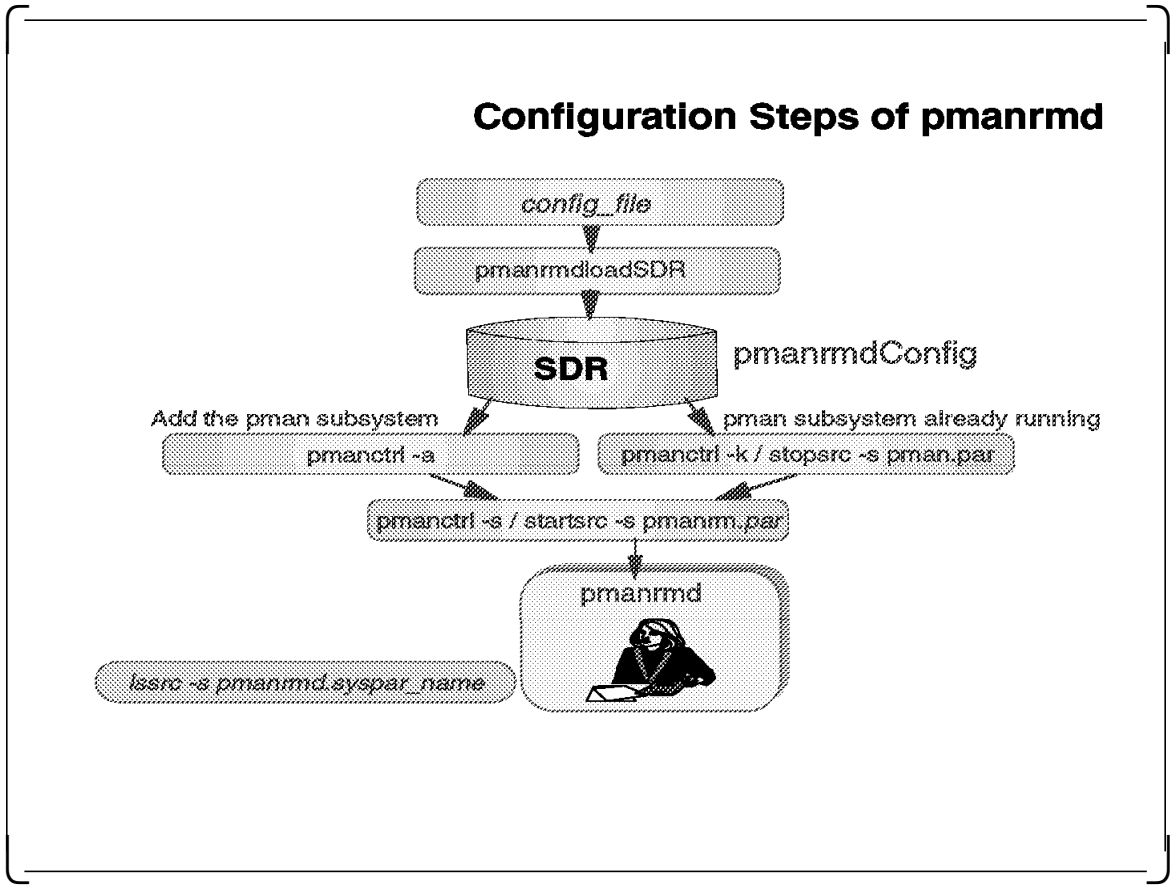
6 Resource type

7 Resource class

8 Error label

These errors do not generate SNMP traps by default. These errors only update an Event Management resource variable. Event Management clients may register for an event based on this resource variable.

6.11 Configuration Steps of pmanrmd



To configure the *pmanrmd* subsystem, follow these steps:

1. Create a *configuration file*, where you specify each Resource Monitor program that will supply data to the Event Manager Daemon. The format of this file is described in section 6.12, “Configuration File and SDR” on page 219.
2. The pmanrmd configuration file must be loaded into the SDR by using the command `pmanrmdloadSDR` (/usr/lpp/ssp/bin/pmanrmdloadSDR). The configuration file name is passed to this command as an argument. An example of this command follows. The configuration file is a user-created file:

   ```
   /usr/lpp/ssp/bin/pmanrmdloadSDR /spdata/sys1/pman/user_rm.conf
   ```

3. The SDR is now loaded with the configuration information, and this data is stored in the *pmanrmdConfig* SDR class. This command has to be run on the Control Workstation only. To check the SDR pmanrmdConfig class objects, issue the `SDRGetObjects` command as follows:

   ```
   SDRGetObjects pmanrmdConfig
   ```

4. If the pmanrmd subsystem is not present, use the `pmanctrl -a` command to add the subsystem.

   Note: This command is used also to create the pmand subsystem. An example follows:

```
pmanctrl -a
```

If the pmanrmd is already running, stop the subsystem by issuing the `pmanctrl -k` or the `stopsrc -s pmanrm.syspar_name` command. The syspar_name is the system partition name.

5. At this point, the pmanrmd subsystem is ready to start up. To start it, issue one of the following commands: `pmanctrl -s` or `startsrc -s pmanrm.syspar_name`. The `pmanctrl -s` command calls the `startsrc` command. The `startsrc -s` starts the pmanrmd subsystem for the system partition. Its name is specified in the subsystem name (syspar_name). To start pmand and pmanrmd subsystems for all partitions, issue the `startsrc -g pman` command. The `pmanrmd` is a Perl command. An example of how to use these commands follows:

   ```
   pmanctrl -s
   ```

   or

   ```
   startsrc -s pmanrm.syspar_name
   ```

   or

   ```
   startsrc -g pman
   ```

6. After completing the previous step, the pmanrmd subsystem should be running. Check the subsystem by issuing the `lssrc` or the `ps -ef` command:

   ```
   lssrc -g pman
   ```

   or

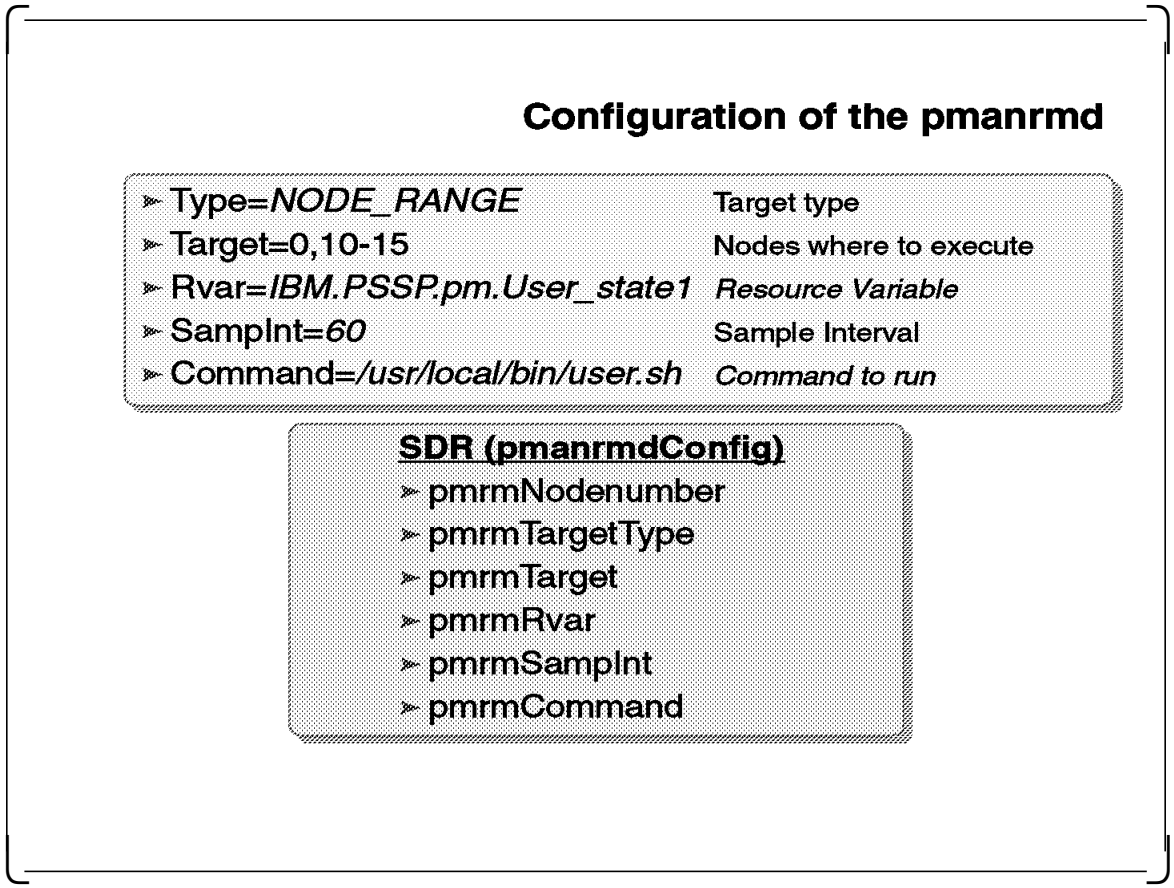
   ```
   lssrc -s pmanrm.syspar_name
   ```

   or

   ```
   ps -ef | grep pmanrmd
   ```

These steps are performed when the system administrator defines new partitions to the RS/6000 SP system. The pmanrmd subsystem is configured with an empty configuration and the pmanrmd process is in a sleeping state. To activate a configuration, the system administrator must define the Resource Monitor configuration as it is defined in the next section.

6.12 Configuration File and SDR



The basic configuration information of the pmanrmd has to be defined in a file that the user creates following the required format. The format of the configuration file is similar to the pmand configuration file format. The file contains stanzas that have the following structure:

TargetType=

This specifies the Target item format. It is one of the following:

- NODE_LIST
- NODE_RANGE
- NODE_GROUP

Target=

This specifies the set of nodes on which the daemons are to be configured. This can be a list of hostnames, a node range as specified in the hostlist command, or the name of the node group.

Rvar=IBM.PSSP.pm.User_state*nn*

This specifies the fully qualified resource variable name as it is defined in the Event Management configuration. The resource variables may be one of the sixteen IBM.PSSP.pm.User_state*nn* resource variables, where the *nn* means a number from 1 to 16. These resource variables are of type character string (cstring).

SampInt=

This specifies the interval, in seconds, between each time a resource variable is sent to the Event Manager Daemon. This is referred as the *Sampling Interval*. The user command is to be run in the intervals specified here, to send the data to the Event Manager Daemon.

Command=

This specifies the user command that provides the resource variables. The resource variable is then created from the stdout of this command.

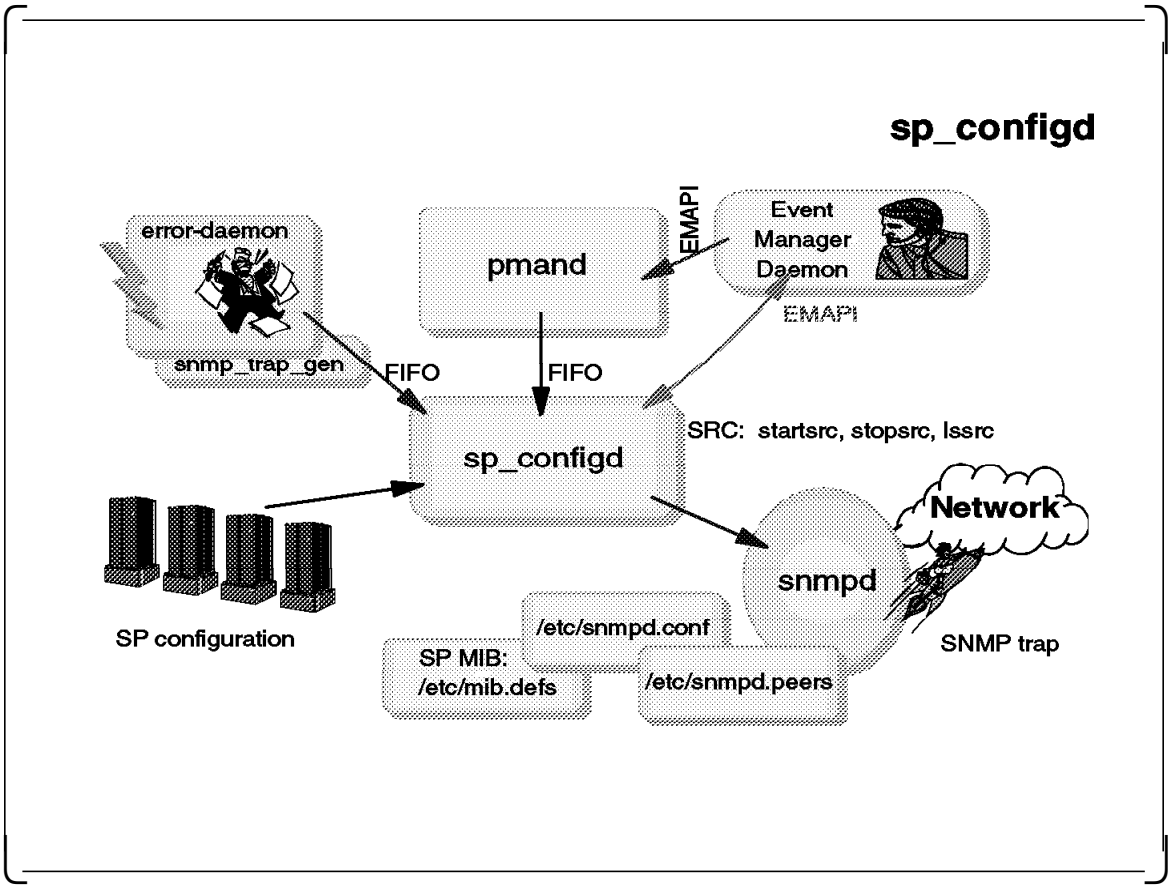
There is no default configuration file. By default, the pmanrmd comes up, but it "sleeps" until it is configured. There is a sample configuration file, pmanrmd.conf.

An example of configuration file follows:

```
* This will provide a Resource Monitor that will run every 10 seconds
* and will run a user command /test/adrian.pm; its output is a
* character string. This string is supplied every 10 seconds
* to the Event Manager daemon IBM.PSSP.pm.User_state1 resource
* variable. This command runs only on the Control Workstation.
*
TargetType=NODE_RANGE
Target=0
Rvar=IBM.PSSP.pm.User_state1
SampInt=10
Command=/test/adrian.pm
*
* This will provide a Resource Monitor that will run every 10 minutes
* and will provide the most recently changed file in the
* etc filesystem as a string state variable named
* IBM.PSSP.pm.User_state2.
* It runs on all the nodes in the partition.
*
TargetType=NODE_RANGE
Target=0,5-16
Rvar=IBM.PSSP.pm.User_state2
SampInt=600
Command="/bin/ls -tl /etc | /bin/head -2 | /bin/grep -v total"
*
* ------------------------------------------------------------ *
```

The configuration file has to be loaded into the SDR. This is done by the *pmanrmdloadSDR* command. The SDR object format corresponds to the configuration file stanzas. The *pmrmNodenumber* item is the node number of the host on which the pmanrmd has to instantiate the resource variable.

6.13 sp_configd



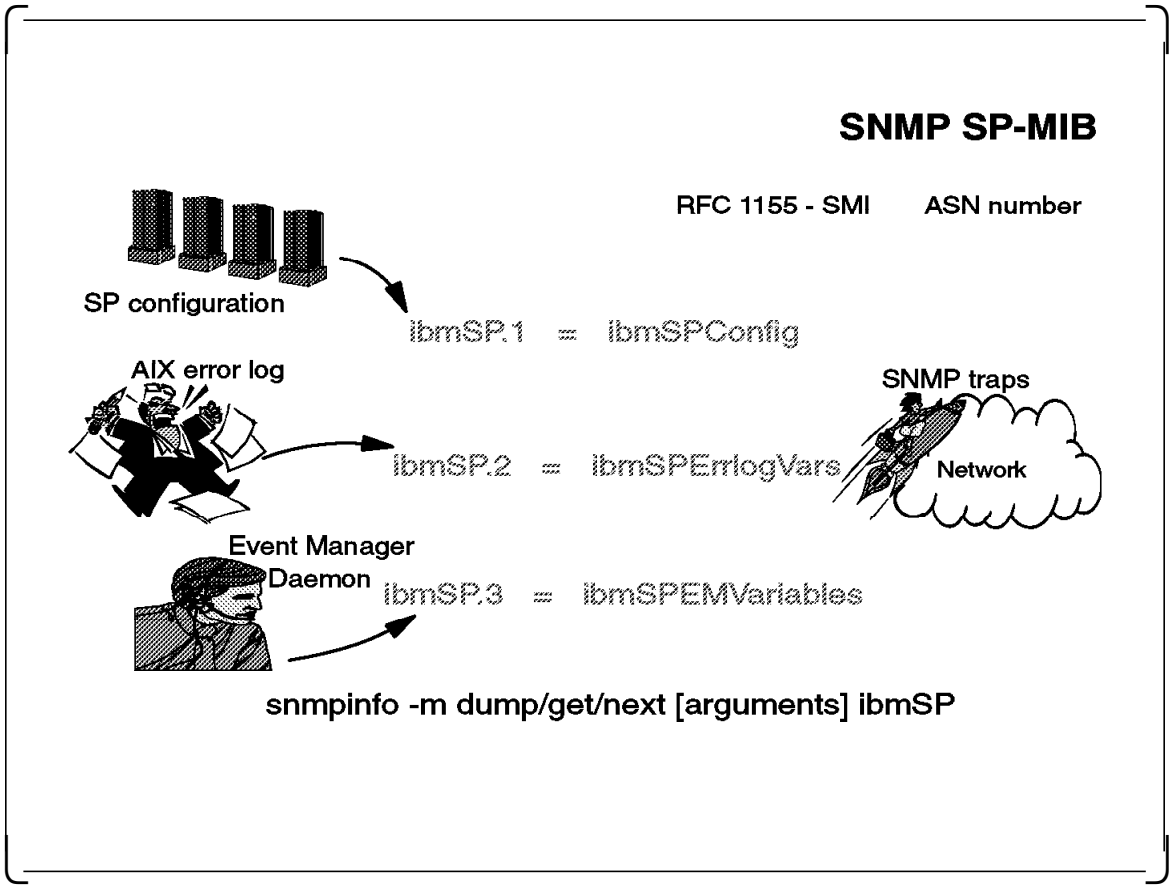
The *sp_configd* is an SP SNMP proxy agent (sometimes referred to as subagent) that runs on the Control Workstation and every RS/6000 SP node. The SP proxy agent provides the following functions:

- A Management Information Base (MIB), *ibmSP*
- SNMP GET and GET NEXT command support that allows data in the ibmSP MIB to be accessed by a network manager application
- The creation and transmission of SNMP traps to an installation-defined network manager application

An SNMP trap is generated when the following events occur on the node while the sp_configd is running:

- A cold start trap is issued when the agent 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 from the Event Management.

6.14 SNMP SP MIB



An ASN.1 MIB (Management Information Base) named ibmSP is provided to SNMP users. An ASN number consists of integer numbers and is defined in RFC-1155. The ASN number assigned to the ibmSP MIB can be determined by looking in the ibmSP MIB.

Every SNMP agent supports a MIB, which is a set of variables (sometimes referred to as objects) that represent the physical and logical resources of the managed systems or agent. The MIB is not a database, in the sense of a monolithic collection of data, but rather it represents dynamic information. The values of the variable instances are maintained by different system functions such as the kernel, device drivers, or subsystems running on the node.

The SNMP Manager can read variable instance values through these SNMP requests:

GET Requests the SNMP agent to retrieve the value of the specified variable instance and return it to the requester.

GET NEXT Requests the SNMP Agent to retrieve the value of the next variable instance, after the one specified in the request, and return it to the manager. This is useful for retrieving tabular information and multiple variables.

Note: The SNMP *SET* request is not implemented in the sp_configd design. The SP MIB consists of the following three groups:

1. *ibmSPConfig* (1.3.6.1.4.1.2.6.117.1)
2. *ibmSPErrlogVars* (1.3.6.1.4.1.2.6.117.2)
3. *ibmSPEMVariables* (1.3.6.1.4.1.2.6.117.3)

ibmSPConfig

This group defines objects containing SP system configuration information.

ibmSPErrlogVars

This group consists of a sequence of objects containing information about the latest error log write that caused an SNMP trap.

ibmSPEMVariables

This group consists of a sequence of objects containing information about the last Event Management event that caused an SNMP trap.

To view the ibmSP MIB, use the `snmpinfo` command. This command may be used in place of a manager client, such as Netview, to view the contents of the ibmSP MIB. The use of this command requires root authentication. The following are examples of using the `snmpinfo` command:

- To dump the contents of the ibmSP MIB located on the host_name in verbose mode, enter the command:

  ```
  snmpinfo -m dump -v -h host_name ibmSP
  ```

- To dump the contents of the ibmSPConfig group, enter the command:

  ```
  snmpinfo -m dump -v -h host_name ibmSPconfig
  ```

- To get values for the MIB variable instance SPhostnodenumber.0 located on the local host, enter:

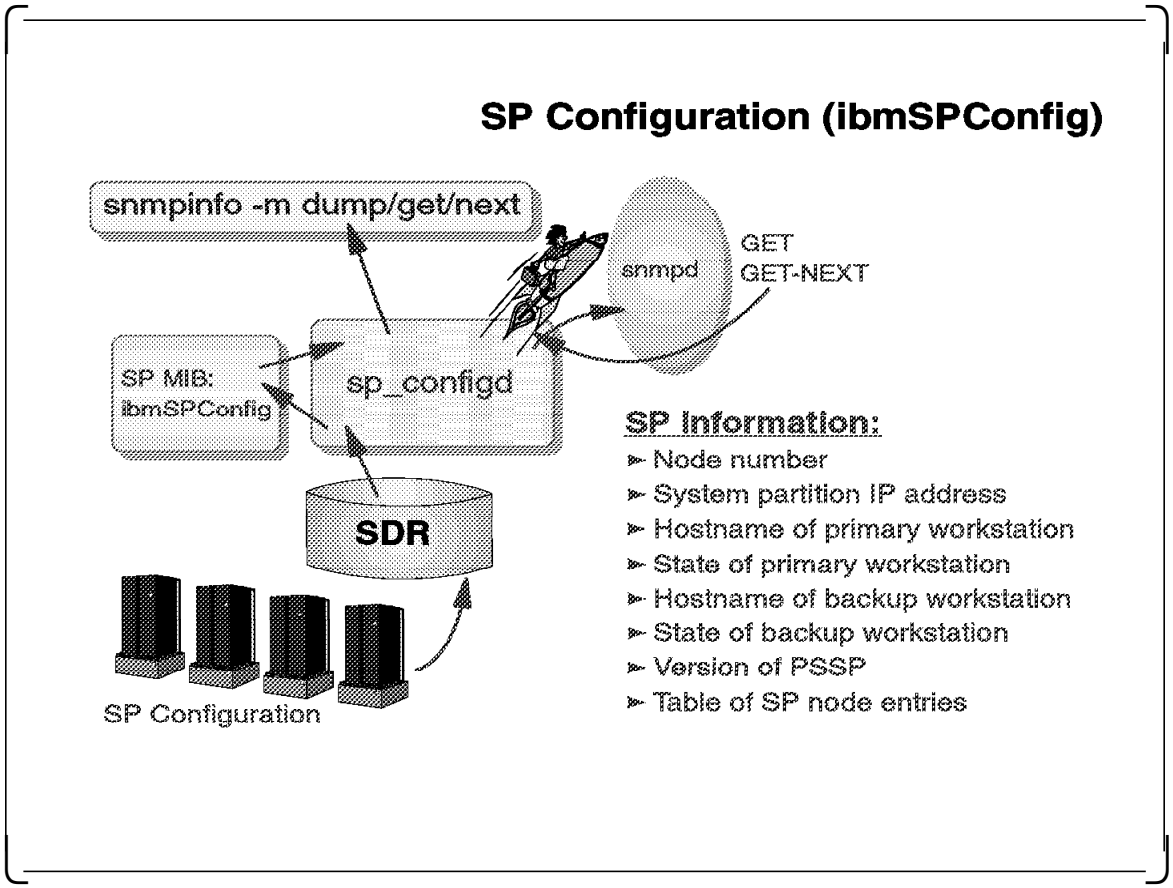
  ```
  snmpinfo -m get ibmSPhostnodenumber.0
  ```

- To get the value for the MIB variable instance following the ibmSPhostnodenumber.0 variable instance, enter:

  ```
  snmpinfo -m next ibmSPhostnodenumber.0
  ```

Note: A copy of the ibmSP MIB is supplied in Appendix B, “ibmSP MIB” on page 291.

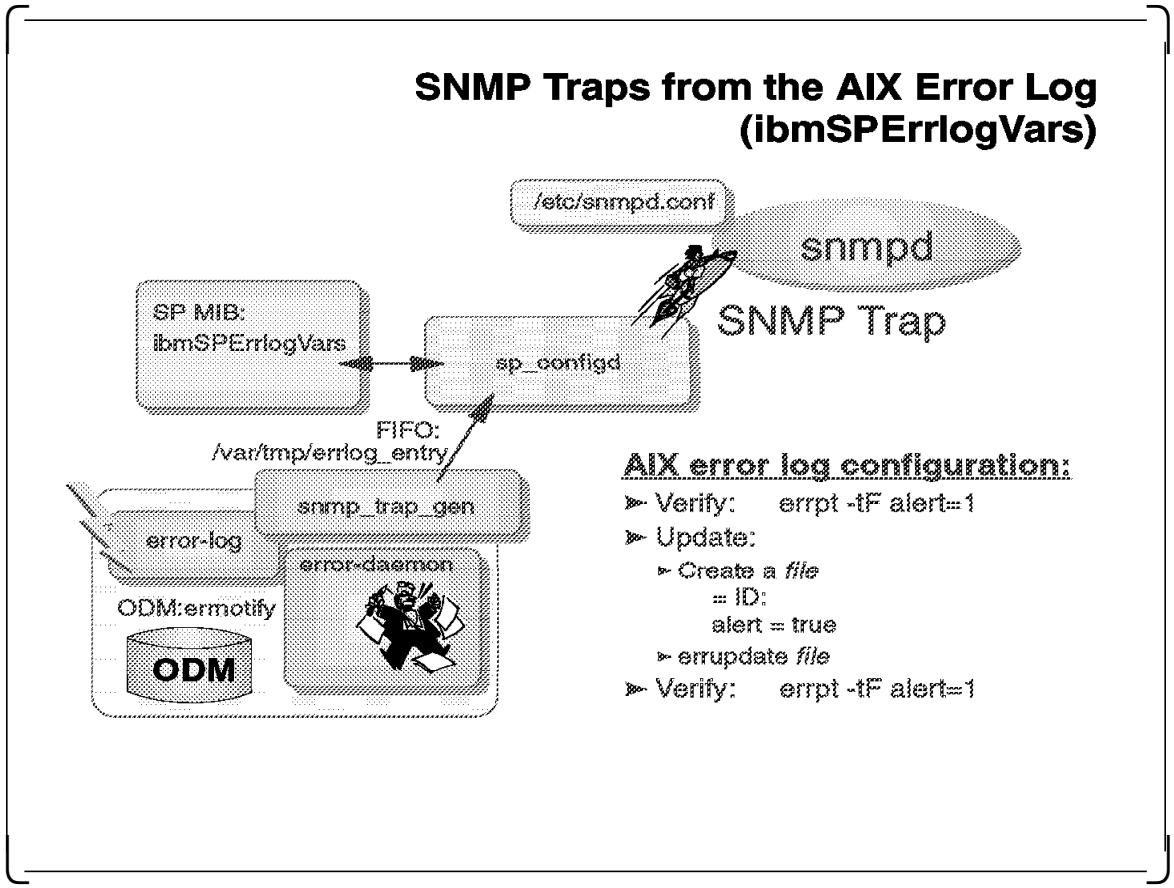
6.15 SP Configuration (ibmSPConfig)



The RS/6000 SP information that can be retrieved by the SNMP GET and GET-NEXT requests is stored in the SDR. The RS/6000 SP configuration group contains the following information:

- The node number of the node on which the queried subagent is running
- The IP address of the system partition in which the queried subagent is running
- The hostname of the primary control workstation
- The operational state of the primary control workstation
- The hostname of the backup control workstation
- The operational state of the backup control workstation
- The version of the PSSP software
- A table of RS/6000 SP node entries. Each row in the table has an instantiation index consisting of the IP address of the partition followed by the number of the node. Each row in the table contains variables identifying the IP address of the partition, the node number, the frame number, the lowest slot in the frame occupied by the node, the number of slots occupied by the node, the reliable hostname of the node, the initial hostname, the system partition name, and the version of PSSP on the node.

6.16 SNMP Traps from the AIX Error Log



The snmp_trap_gen method monitors the AIX error log and notifies the sp_configd daemon to generate SNMP traps for errors that contain the *ALERT=true* attribute. This error notification uses the AIX error notification that is defined in the AIX ODM database by the *errnotify* class. This error notification starts a program, which is defined in the *errnotify* ODM class, whenever the error log entry contains flags corresponding to the *errnotify* object specification. The `sp_configdctrl -a` command creates the following entry in the *errnotify* ODM class:

```
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 following steps describe the flow of events for creating an SNMP trap for an AIX error log event:

1. An application writes to the AIX error log.
2. If the error log entry contains an ALERT=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. snmp_trap_gen places the event information in a FIFO file called /var/tmp/errlog_entry.
5. sp_configd 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. sp_configd sends the trap to the AIX agent, snmpd, which then sends the trap to network managers specified in the /etc/snmpd.conf file existing on the node.

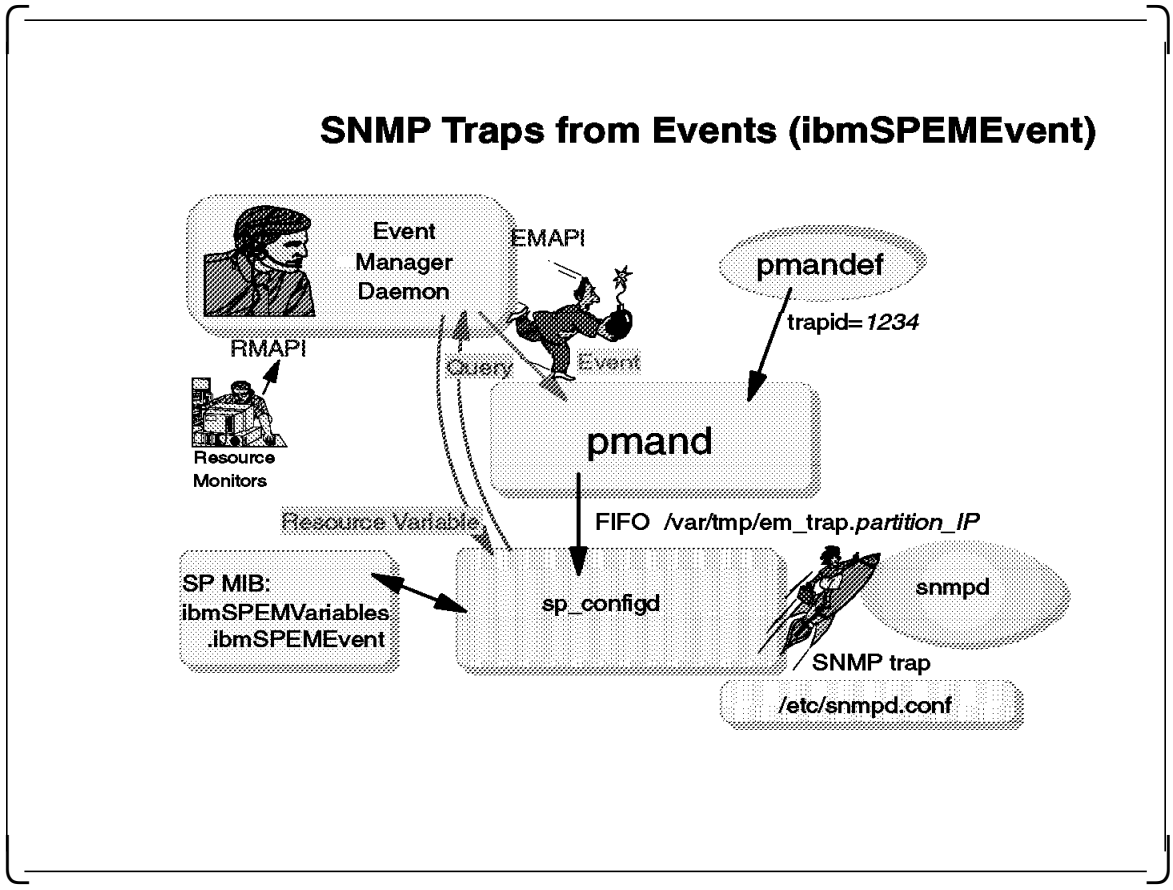
The following information is provided in SNMP traps from error logs:

- The enterprise field contains the Object ID (OID) of the sp_configd subagent. This is the OID assigned to the ibmSP MIB.
- The specific-trap field contains the error ID from the error log entry. This is the convention expected by Netview when configuring events.
- The variable bindings field.

The variable bindings field contains the following object values paired with their OIDs:

- Error label
- Error ID
- Error log entry time stamp
- Unique sequence number
- Machine ID parameter
- Node ID parameter
- Error class
- Error type
- Resource name
- Resource class
- Resource type
- Location code of the device
- Vital product data
- Error description
- Probable causes
- User causes
- User actions
- Install causes
- Install actions
- Failure causes
- Failure actions
- Detail data

6.17 SNMP Traps from Events (ibmSPEMEvent)



The following list is the flow of actions that are necessary to generate an SNMP trap for an Event Management event:

1. Use the `pmandef` command to subscribe to an Event Management event. Specify the trap ID in the `pmandef` command.
2. pmand subscribes to the event as defined in step 1. The event will occur on each node where an action is defined to take place.
3. pmand writes the contents of the Event Management subsystem-supplied event response and the user-specified event configuration information into a FIFO file.
4. sp_configd reads the data and creates an SNMP trap from it.
5. sp_configd sends the trap to the SNMP managers specified in the /etc/snmpd.conf file on the node.
6. The specific trap ID field in the SNMP trap is set to the trap ID specified in the `pmandef` command subscription.

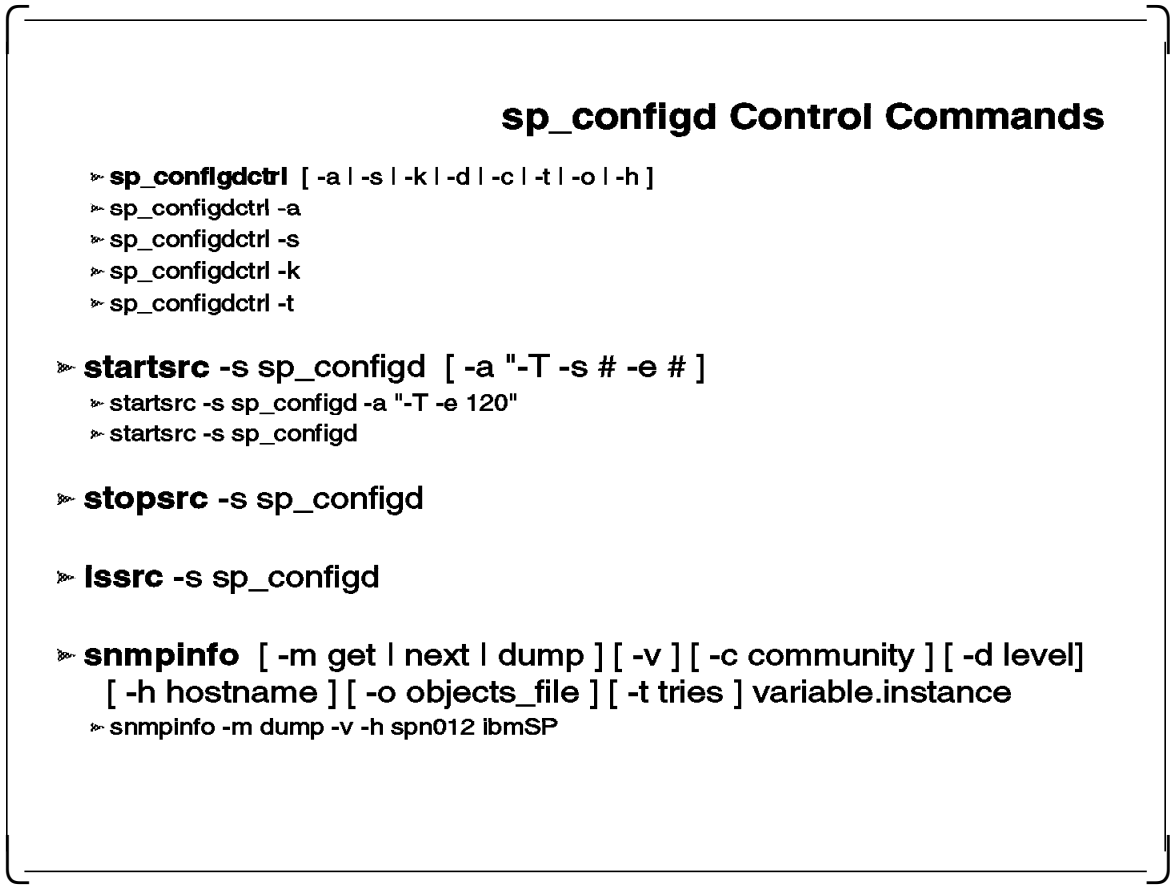
The contents of SNMP traps containing Event Management events are the following:

- The enterprise field contains the OID of the ibmSP MIB.
- The specific trap field contains the trap ID field from the pmand configuration for this event.

- The variable bindings field, which contains the following object values paired with their OIDs:
  - Event ID
  - Event flags
  - Time stamp indicating the time the Event Manager Daemon generated the event
  - Node number and the partition where the event occurred
  - The resource variable name
  - MIB table and instance information, providing further information about the variable involved with the event

    Note: This information is provided only when the Event Management variable associated with the event is defined in the node from which the SNMP trap is issued.
  - The value of the resource variable
  - The predicate for which the event was triggered

6.18 sp_configd Control Commands



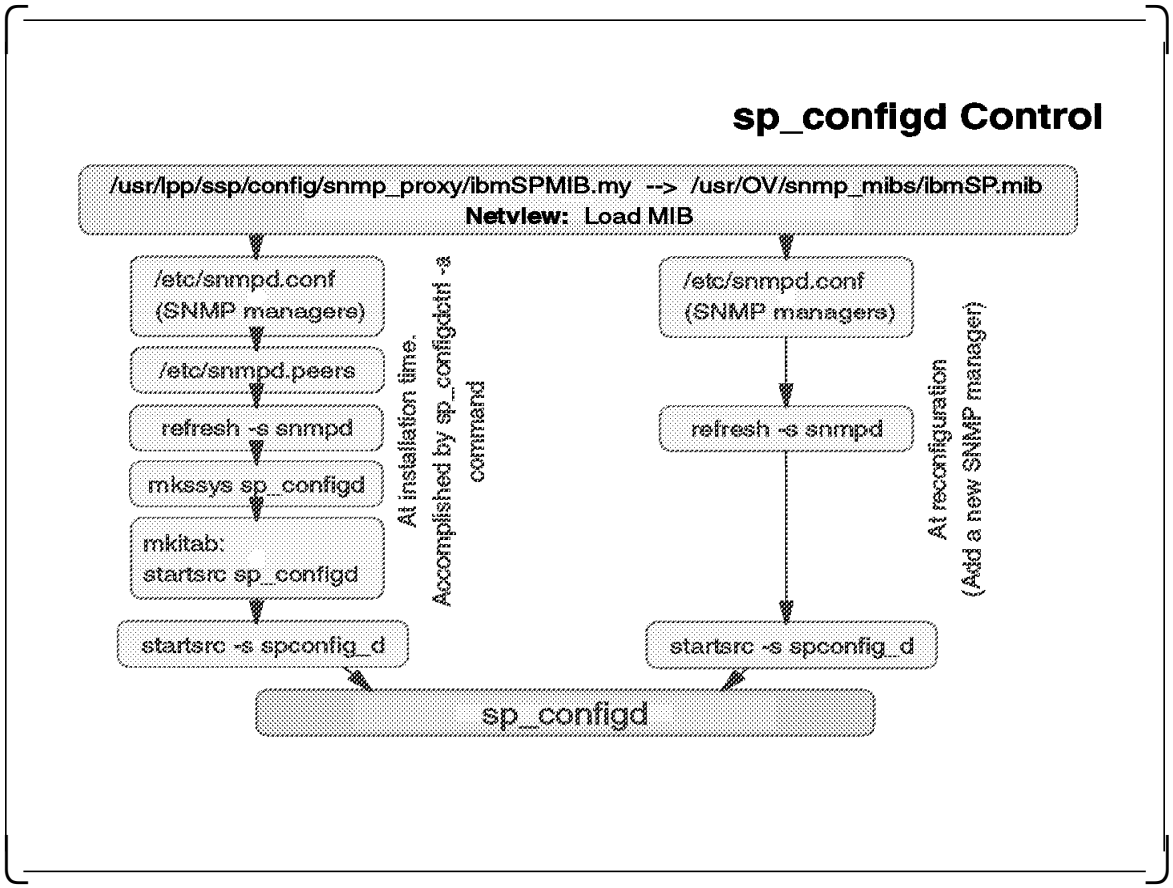
The *sp_configd* runs under the AIX System Resource Controller (SRC) control. The AIX SRC control commands are used to start, stop, and query the state of the daemon. The `startsrc`, `stopsrc`, and `lssrc` AIX SRC control commands are available to start, stop, and check the subsystem.

The `sp_configdctrl` command provides support for the sp_configd daemon which is similar to that of `pmanctrl` (see 6.8.1, “pmanctrl” on page 205 and the `sp_configdctrl` man page). The `sp_configdctrl` command provides parameters for performance-related processing and a trace option. These parameters can be specified at the spconfigd start with the `startsrc` command, using the `-a` option. To supply attributes to the `startsrc` command, use the following example:

```
startsrc -s sp_configd -a"-T -e120"
```

The `snmpinfo` command provides access to the ibmSP MIB, and can be used in place of a manager, such as Netview for AIX. For detailed information, see the standard AIX documentation (`info`) or the snmpinfo manpage.

6.19 sp_configd Control



The configuration of the sp_configd consists of two configuration sections:

- SNMP manager (Netview for AIX) configuration
- sp_configd configuration

6.19.1 Configuring Netview for AIX

If you are using Netview for AIX as the network manager for your RS/6000 SP system, you must configure it to recognize trap and configuration information.

Performing the following steps configures the SP MIB to be used by Netview for AIX:

1. Copy the file */usr/lpp/ssp/config/snmp_proxy/ibmSPMIB.my* from an RS/6000 SP node into */usr/OV/snmp_mibs/ibmSP.mib* on the node running Netview for AIX.
2. Load the MIB using the Netview for AIX menu option *Load/Unload MIBs*. You may have to change the file attributes to match other MIB files in this directory.
3. You may tailor the information in the traps by using the *Trap Customization* selection under the *Event/Configuration* menu option.

6.19.2 Configure sp_configd

The SP proxy agents (sp_configd) reside on each managed RS/6000 SP node and Control Workstation. The sp_configd daemons are configured to be started automatically whenever AIX is started. The configuration changes necessary to accomplish this are performed during the installation of the nodes and the Control Workstation by the `sp_configdctrl -a` command.

The sp_configd is configured at installation, or at partition configuration. The configuration is performed by the syspar_ctrl script on every node and the Control Workstation. The entire configuration of the sp_configd is described in the following steps:

1. If you have a management station that listens for traps, place the information for the trap destination in the */etc/snmpd.conf* file on each RS/6000 SP node and Control Workstation. This step is to be done by the system administrator.
2. The `sp_configdctrl` script modifies the */etc/snmpd.conf* file and adds the RS/6000 SP subagent (sp_configd) to the set of SMUX peers with a *smux* record.
3. The SP MIB file */usr/lpp/ssp/config/snmp_proxy/ibmSPMIB.defs* is appended to the /etc/mibs.def file. (The /usr/lpp/ssp/config/snmp_proxy/ibmSPMIB.defs file is compiled from ibmSPMIB.my.)
4. Information for the new sp_configd subagent is added to the /etc/snmpd.peers file. The information is the Object ID (OID) of the agent and the community name (password). The /etc/snmpd.peers file update is performed by the `sp_configdctrl` script.
5. At this point, the *snmpd* daemon has to be refreshed. This step is performed by the `sp_configdctrl` script.
6. The `sp_configdctrl` script creates an AIX SRC subsystem, and the sp_configd is added to the AIX ODM with default parameters. The default parameters are the following:

```
SRCsubsys:
        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 = ""
```

7. The sp_configdctrl creates an /etc/inittab entry to start the daemon at every system restart. This entry calls the startsrc command as follows:

   ```
   sp_configd:2:once:/usr/bin/startsrc -s sp_configd
   ```

8. The sp_configd can be started by issuing the startsrc command, as follows:

   ```
   /usr/bin/startsrc -s sp_configd
   ```

Chapter 7. VSD/RVSD

IBM Virtual Shared Disk (VSD) is a distributed subsystem that allows application programs executing on different RS/6000 SP nodes to access a raw logical volume (LV) as if it were local to each of the nodes.

VSD also provides a device driver that allows application programs to stripe data across the physical disks in multiple VSDs, thus reducing I/O bottlenecks and hot spots.

Recoverable Virtual Shared Disk (RVSD) provides high availability to VSD and VSD applications by detecting and executing the failover of the VSD server node failure.

Table of Contents

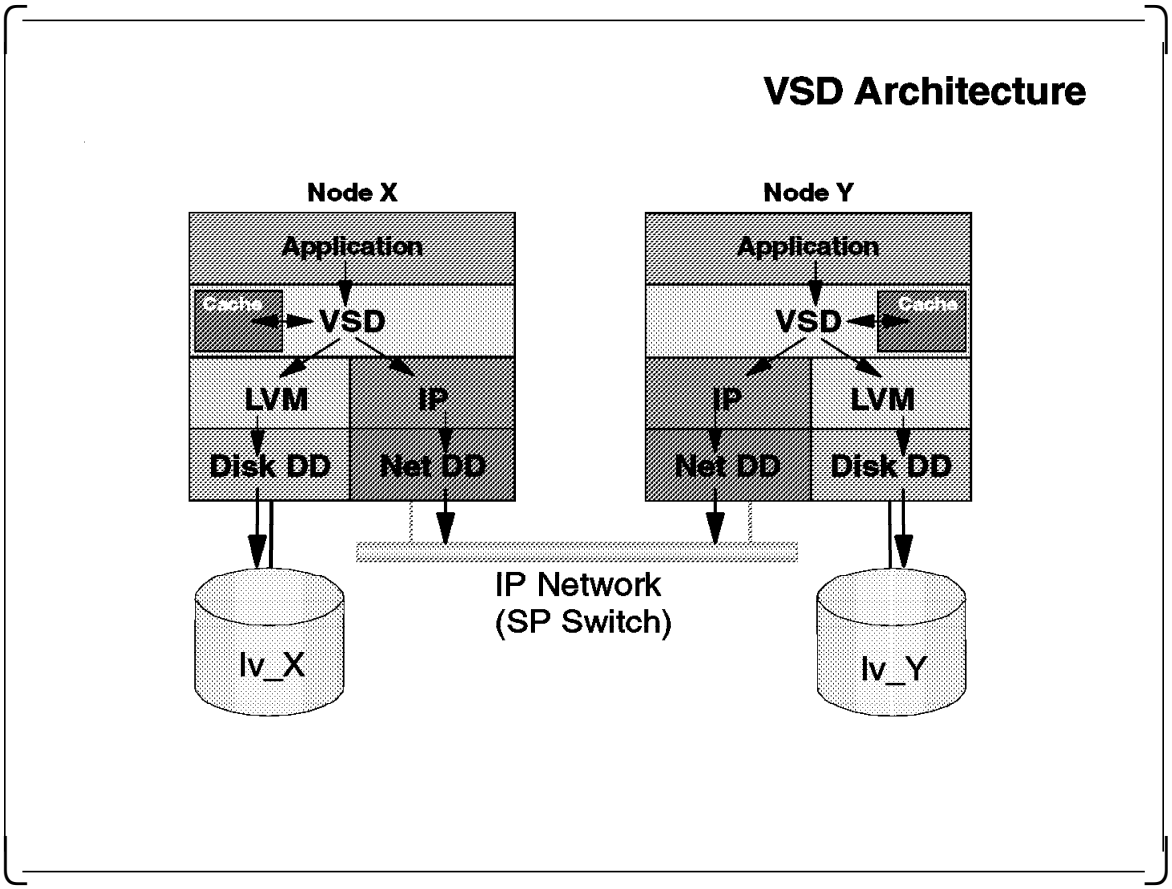
This chapter describes the new release of VSD, which is part of PSSP Version 2 Release 2, and RVSD Release 1.2, which is a separately orderable licensed program.

The first section introduces the functional overview of VSD, data striping device (HSD), and RVSD. The second section covers the new features in this release of Parallel System Support Programs.

This chapter does not include details for planning and operating VSD, HSD, and RVSD. Before you actually install VSD or VSD with RVSD, refer to *Managing Shared Disks: IBM VSD, HSD, and IBM RVSD Licensed Program Release 1.2*, GC23-3849.

7.1 Functional Overview

7.1.1 VSD Architecture



This figure illustrates a simplified IBM Virtual Shared Disk (VSD) implementation. The logical volume is local at each of the nodes, called *direct client* nodes. VSD software routes I/O requests from the other nodes, called *client* nodes, to the direct client node (that is, the VSD server node), and returns results back to the client nodes. In the figure, node X is the direct client node for the VSD named "lv_X," and node Y is the client node for VSD lv_X. For VSD lv_Y, node Y is the direct client node, and node X is the client node.

The I/O routing is done by the VSD device driver layer that sits on top of the Logical Volume Manager (LVM) and IP subsystem. The device driver is loaded as a kernel extension on each node. In this way, logical volumes can be made globally accessible. Any IP-network defined in SDR can be used for VSD communications. However, either the High Performance Switch (HPS) or the Scalable Parallel Switch (SP Switch) (both are defined as *css0*) is recommended for the best performance.

The application program interface to a VSD is the character (raw) special device.

Attention

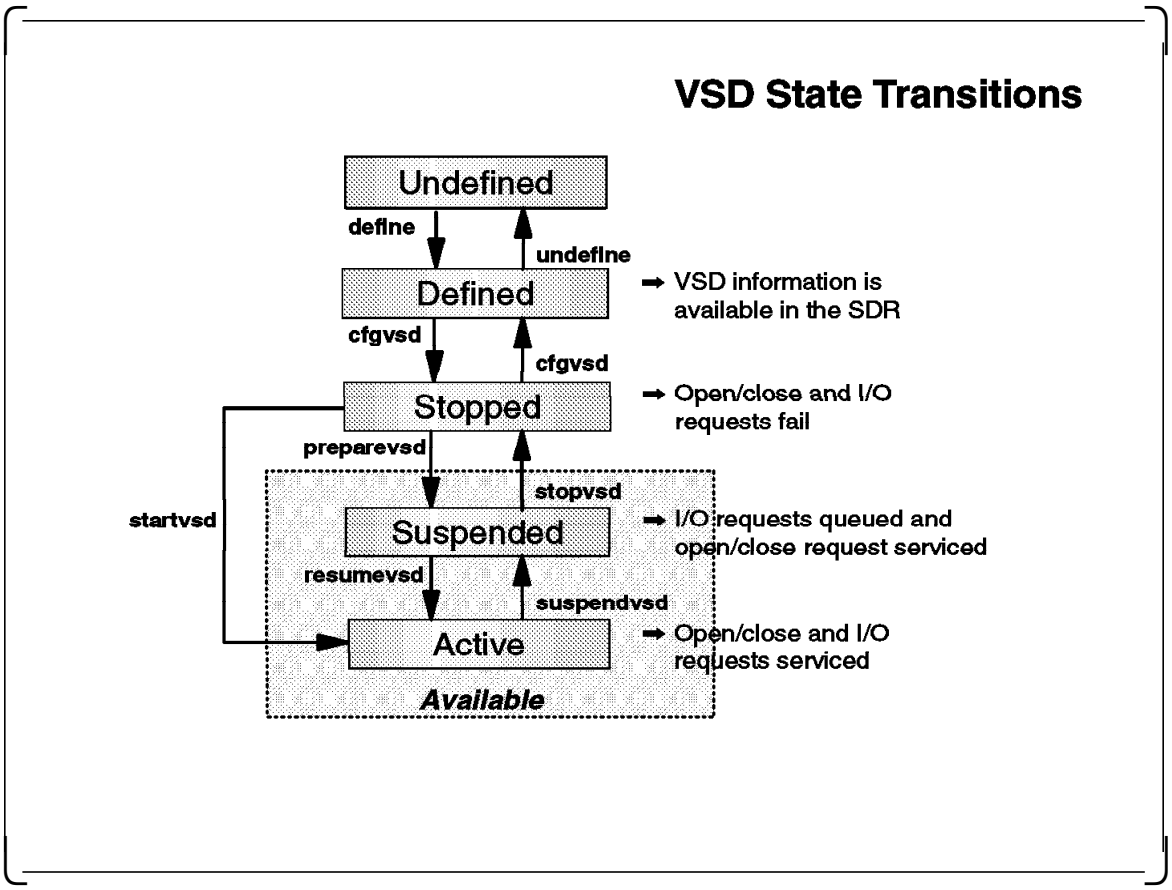
AIX Journal File System (JFS) on a VSD to be mounted and accessed from other nodes is not supported in this release.

Each VSD device driver on each node has a single cache buffer, called the LRU (Least Recently Used) cache, shared by all cacheable VSDs configured on and served by the node. The cache is used to store the most recently-accessed data from the cached VSD devices on the server node. The objective is to benefit from classic cache effect, that is, to minimize physical disk I/O activity. If the requested data is found in the cache, it is read from the cache, rather than the corresponding logical volume.

Data in the cache is stored in 4 Kbyte blocks. The content of the cache is a replica of the corresponding data blocks on the physical disks. The cache contains only valid data (no dirty blocks). Write-through cache semantics apply: the write operation is not complete until the data is on the disk.

The LRU cache is not “free” in the sense of performance consideration. There are overheads that must be paid for each and every access, since the VSD has no way of knowing which 4 Kbyte accesses are due to bouncing data. For more details on the performance consideration with the LRU cache, refer to Chapter 5, “Performance and Tuning Considerations for IBM Virtual Shared Disks and Data Striping Devices” in *Managing Shared Disks: IBM VSD, HSD, and IBM RVSD Licensed Program Release 1.2*, GC23-3849.

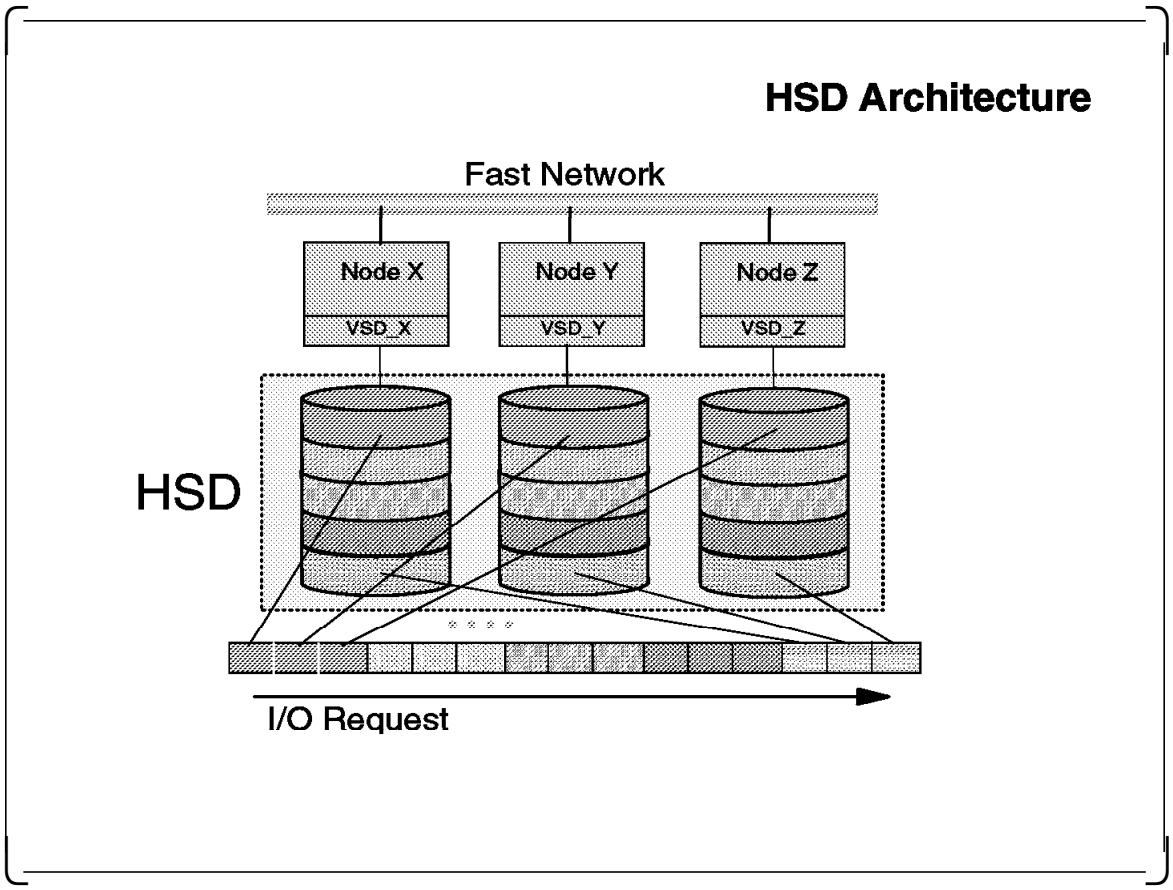
7.1.2 VSD State Transitions



Once VSDs have been defined in the SDR, they can be configured to the system, moved between states, and monitored with system commands through SMIT panels or by using the VSD Perspective (see 7.2.2, “GUI Interface” on page 248).

This figure shows the possible states of a VSD and the names of the commands that move VSDs from one state to another. You need to move VSDs between states when you change the VSD configuration on a system partition or, if you do not have the IBM Recoverable Virtual Shared Disk licensed product installed, when you perform manual recovery after system or network problems.

7.1.3 HSD Architecture



Instead of writing all the data from one I/O request on one VSD at a specific location, data striping writes or reads blocks of the data on several separate VSDs. Thus, HSD (Hashed Shared Disk) provides automatic distribution or partitioning of data across physical disks and nodes, while requiring minimal administrator intervention. When the data is well-partitioned among the disk drives, the application can get better performance from the underlying disk drives.

The HSD driver is a pseudo device driver layer located above the VSD layer. It is used only as a raw device driver interface with the VSD devices.

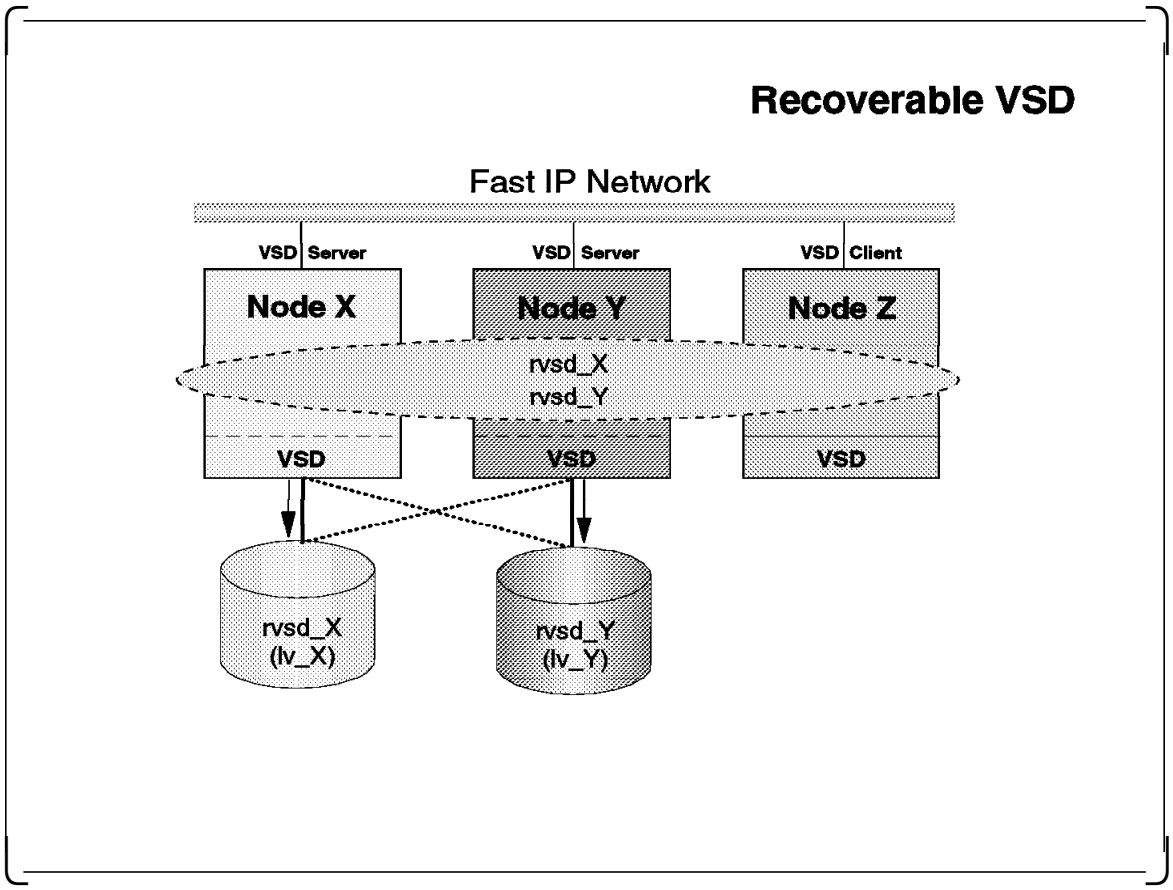
Each HSD is defined with a list of VSDs and a stripe size. The stripe size indicates the maximum size of a block of data to be stored on a single VSD in one I/O operation. The HSD uses a hash function based on the following parameters to determine which VSD to place data on:

- The offset in the HSD file
- The number of VSDs defined in the HSD
- The stripe size

Whether or not to use the HSD depends on your configuration of VSDs and the I/O characteristics of the application programs. If the I/O load to a specific VSD

is too heavy (a hot spot), you can use HSD to distribute the load to other VSDs and nodes.

7.1.4 Recoverable VSD (RVSD)



The IBM Recoverable Virtual Shared Disk (RVSD) software lets you use twin-tailed disks and configure nodes as *primary* and *secondary* VSD server nodes, and provides transparent switch-over to a secondary server if the primary server node for a set of VSDs fails.

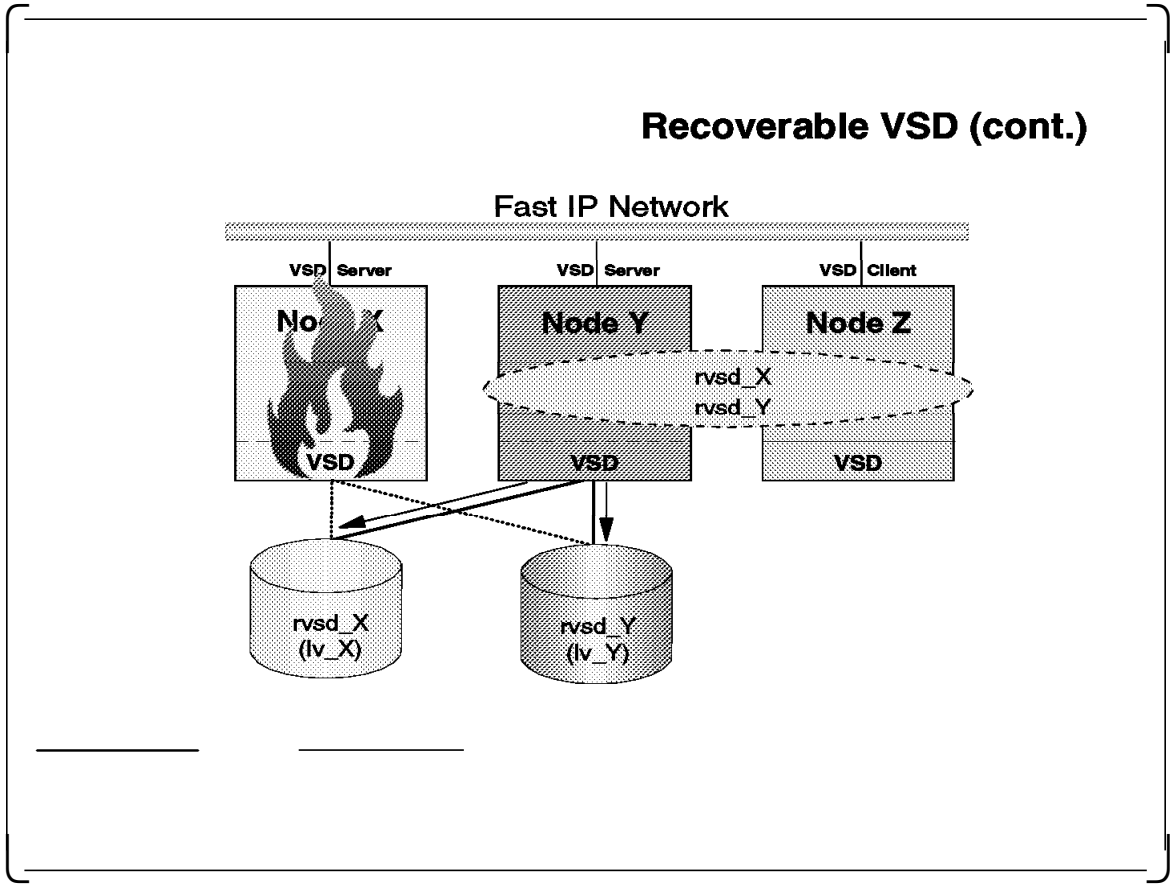
A *twin-tailed* disk is a disk or group of disks that are attached to two nodes of an RS/6000 SP. For RVSD, only one of these nodes serves the disks at any time. The secondary or backup node serves the disks only if the primary node fails or is powered off.

Note: In this release of RVSD, any twin-tailed disks supported by the Logical Volume Manager running on AIX V4.1 can be used.

This figure shows a mutual-takeover configuration with a three-node RVSD cluster. Table 2 shows the functional role of the nodes for each VSD:

Table 2. Functional Role of RS/6000 SP Nodes in the RVSD Configuration

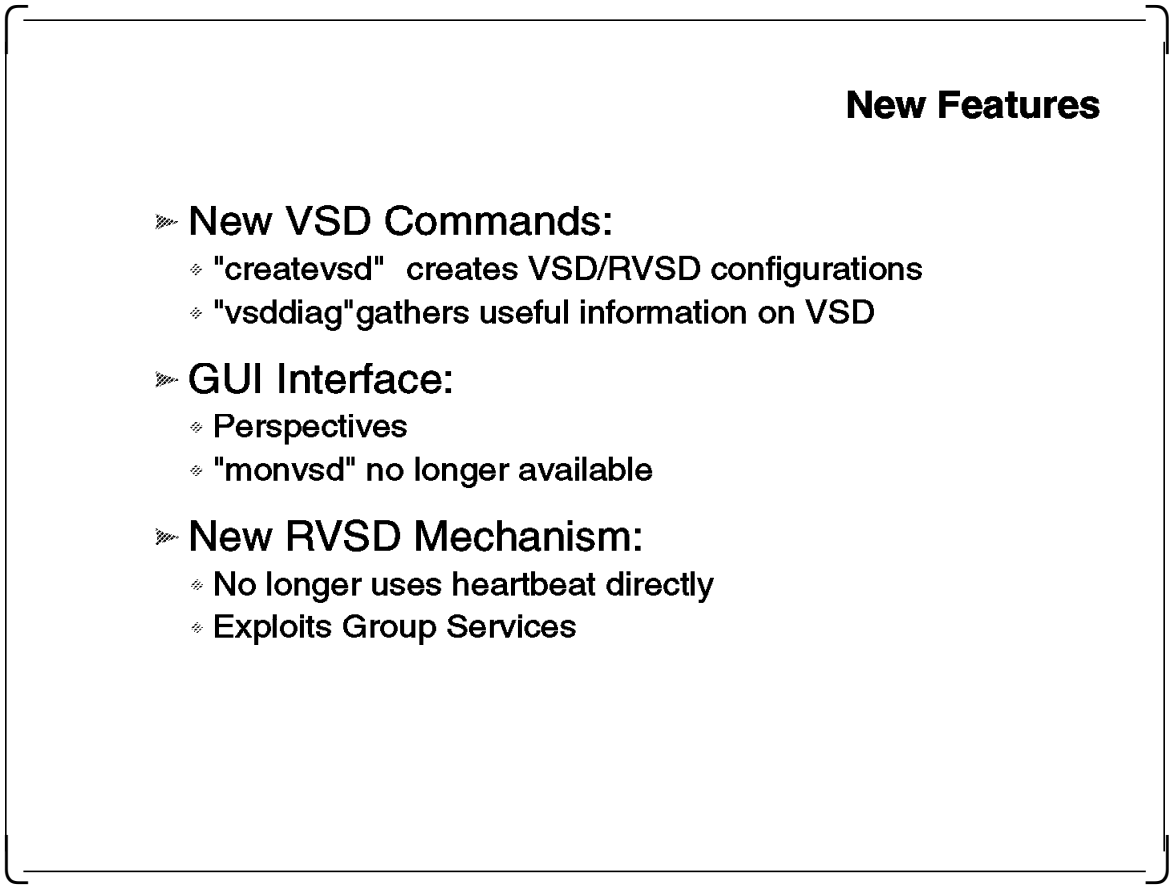
Node	**The Role for rvsd_X**	**The Role for rvsd_Y**
Node_X	Primary Direct Client	Secondary Direct Client
Node_Y	Secondary Direct Client	Primary Direct Client
Node_Z	Client	Client



If node X fails, node Y will takeover the VSD server for rvsd_X. The recovery is transparent to an application running on a client node; there is no disruption of service, only a slight delay while takeover occurs. Now node Y is the VSD server for both rvsd_X and rvsd_Y.

When node X is rebooted, the RVSD software switches the I/O load back to the primary server node. RVSD software provides interfaces that enable your application to be recoverable. For more information on RVSD application programming interface, see Chapter 4, “Application Programming Considerations ” in *Managing Shared Disks: IBM VSD, HSD, and IBM RVSD Licensed Program Release 1.2*, GC23-3849.

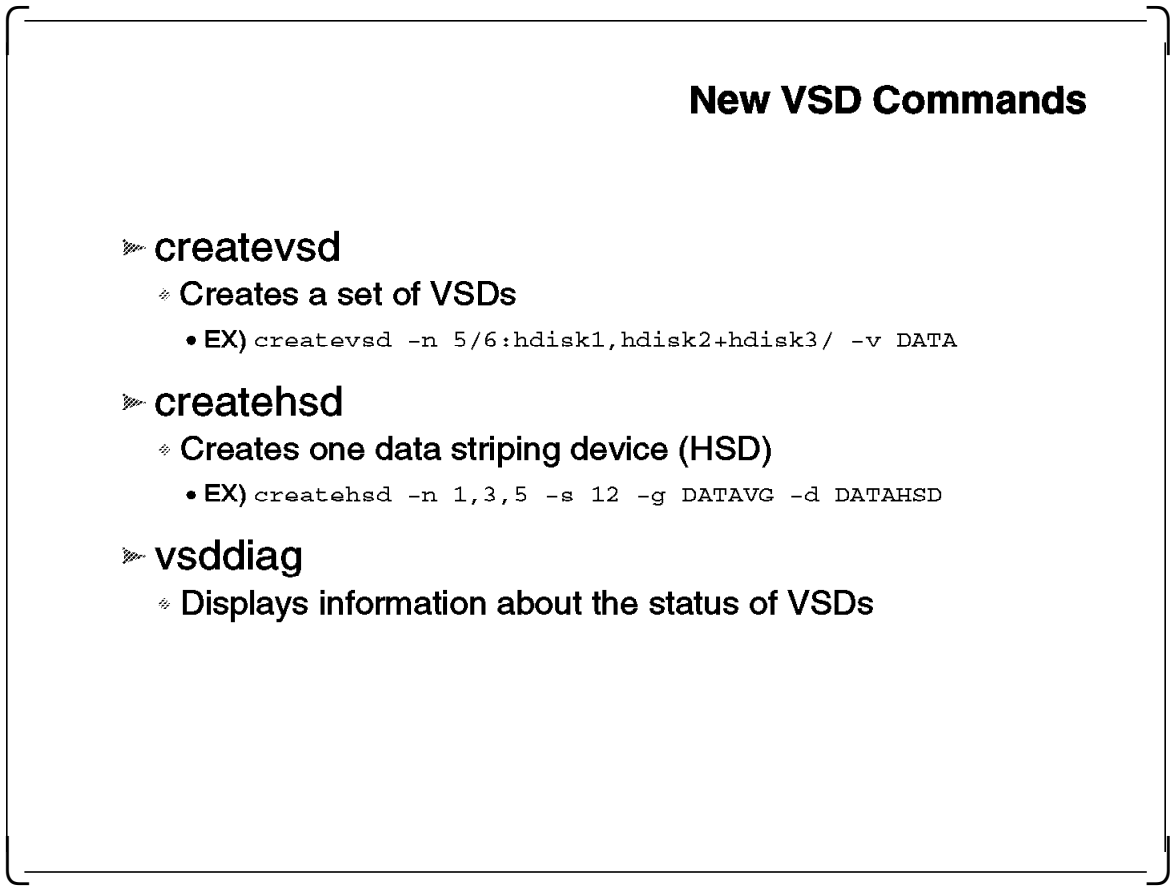
7.2 New Features



This section describes the major new features of VSD (with PSSP Version 2 Release 2) and RVSD Release 1.2, which are:

1. New commands that allow you to define VSD and HSD configurations, including identifying secondary direct clients if you have the RVSD licensed product installed. A new diagnostic tool, `vsddiag`, which gathers useful information before calling IBM service with a VSD problem, is also introduced.
2. A graphical user interface, *Perspectives*, has been developed to help you create and manage VSD and HSD easily.
3. VSD no longer uses heartbeat directly. Instead, Group Services determines which nodes are up and operational and manages the recovery process.

7.2.1 New VSD Commands



New commands introduced in PSSP Version 2 Release 2 are as follows:

- createvsd

  You can create a whole configuration of VSDs with a single command, createvsd, including a secondary direct client node if you have the RVSD licensed program installed. The syntax of the createvsd command is:

  createvsd **-n** {*node_list* | **ALL**}
  -s *size_in_MB* **-g** *volume_group_name*
  [{**-c** *number_of_vsds_per_node* | *-L*}]
  [**-o cache | nocache**] [**-m** *mirror_count*]
  [**-p** *lvm_stripe_size*] [**-v** *vsd_name_prefix*]
  [**-l** *lv_name_prefix*] [**-T** *lp_size*]

  The following list describes the parameters of the createvsd command:

  - Primary direct client node for the volume group (**-n**)

    This is the node number of the primary direct client node. If you do not have the RVSD product installed, all direct client nodes are primary nodes.

    If you want to create identical VSDs on all nodes in the system partition, you can use the **ALL** parameter.

    You can also specify the physical disks you want the volume group and logical volume to span on each node.

- Secondary direct client node for the volume group (**-n**)

  This is the node number of the secondary direct client node. The secondary direct client node is only intended for use with the RVSD product.

- Size in megabytes (**-s**)

  You must specify the size of the VSD in megabytes.

- Volume group name (**-g**)

  VSDs have a *local volume group name* and a *global volume group name*. The local volume group name is the volume group name on the direct client (or direct clients if a secondary direct client is being defined) and the global volume group name is the name across the system or system partition. The createvsd command uses this parameter for the local volume group name, and from it creates a global volume group name with the node numbers of the primary and secondary direct client nodes.

  The length of the name must be less than or equal to 14 characters.

- The number of VSDs to be created on each node (**-c**)

  This is an optional parameter. If you do not specify it, one VSD definition will be created on each node in the node list or on each node in the system partition (if you specified **ALL** instead of a *node_list*).

- The *cache* or *nocache* option (**-o**)

  Use the *cache* option only if your application does I/O in 4 Kbyte blocks aligned on 4 Kbyte disk boundaries, and issues a read immediately following a write.

  Nocache is the default for new VSDs created with the createvsd command.

- Logical volume manager mirroring count (**-m**)

  The mirroring count sets the number of physical partitions allocated to each logical partition on the disks. The range is from one to three, and the default is one.

- Logical volume manager stripe size (**-p**)

  If the node on which the VSD is defined (the direct client node) has more than one physical disk, you can have the data striped across the disks for better performance. Specify the stripe size in kilobytes.

- VSD name prefix (**-v**)

  This prefix will be concatenated with the VSD number, primary direct client node number, and secondary direct client node number (if you have the RVSD licensed program installed), to form a VSD name that is unique across the system partition. The last character of this prefix cannot be a digit.

- Logical volume name prefix (**-l**)

  If you supply this operand, you specify a prefix that overrides the naming convention applied by the createvsd command.

- Size of the physical and logical partitions in the LVM logical volume group (**-T**)

  If you specify this option, it must be a power of 2 in the range of 2 through 256. The default is 4.

To create a volume group that spans hdisk1, hdisk2, and hdisk3 on node 5, with a backup on node 6, type:

```
createvsd -n 5/6:hdisk1,hdisk2+hdisk3/ -v DATA
```

This creates the VSD definition, DATA1n5b6, with logical volume lvDATA1n6b6 defined on a volume group with the local volume group name DATA on node 5, imported to node 6. The global volume group name is DATAn5b6.

- createhsd

  Also, you can create a whole configuration of HSD with a single command, including secondary direct client node for the underlying VSDs if you have the RVSD licensed program installed. The syntax of the createhsd command is:

  createhsd **-n** {*node_list* | **ALL***}*
  -s *size_in_MB* **-g** *volume_group_name*
  -t *stripe_size_in_KB* [*{***-c** *number_of_vsds_per_node* | *-A}*]
  [**-o cache | nocache**] [**-m** *mirror_count*]
  [**-d** *hsd_name*]
  [**-l** *lv_name_prefix*] [**-S**]
  [**-T** *lp_size*]

  The following list describes the parameters of the createhsd command:

  – Nodes on which this HSD is to be created (**-n**)

    This is the node number of the underlying VSDs, including the primary direct client nodes and the secondary direct client nodes.

    If you do not have the RVSD product installed, all direct client nodes are primary nodes.

    If you want an HSD that includes identical VSDs on all nodes in the system partition, you can use the **ALL** parameter.

    You can also specify the physical disks you want the volume group and logical volume to span on each node.

  – Size in megabytes (**-s**)

    This is the usable size, in megabytes, of this HSD. Unless **-s** is specified, createhsd adds at least one stripe size to each VSD definition′s size for each HSD. This protects the first LVM control block (LVCB).

  – Volume group name (**-g**)

    VSDs have a *local volume group name* of the underlying VSDs. The local volume group name is the volume group name on the direct client (or direct clients if a secondary direct client is being defined), and the global volume group name is the name across the system or system partition. The createhsd command uses this parameter for the local volume group name, and from it creates a global volume group name with the node numbers of the primary and secondary direct client nodes.

    The length of the name must be less than or equal to 14 characters.

  – Stripe size (**-t**)

    The stripe size in kilobytes that this HSD will use. The stripe size must be a multiple of 4096 bytes and less than or equal to 1 Gbyte.

- The number of VSDs to be created on each node (**-c**)

  This is an optional parameter. If you do not specify it, one VSD definition will be created on each node in the node list or on each node in the system partition (if you specified **ALL** instead of a *node_list*.)

- The *cache* or *nocache* option (**-o**)

  Use the *cache* option only if your application does I/O in 4 Kbyte blocks aligned on 4 Kbyte disk boundaries, and issues a read immediately following a write.

  Nocache is the default for new VSDs created with the `createvsd` command.

- Logical volume manager mirroring count (**-m**)

  The mirroring count sets the number of physical partitions allocated to each logical partition on the disks. The range is from one to three, and the default is one.

- HSD name (**-d**)

  This is the name of the HSD. It will also be used as the name prefix for the underlying VSDs. The prefix will be concatenated with the VSD number, primary direct client node number, and secondary direct client node number (if you have the RVSD licensed program installed) to form a VSD name that is unique across the system partition. The last character of this prefix cannot be a digit.

- Logical volume name prefix (**-l**)

  If you supply this operand, you specify a prefix that overrides the naming convention applied by the `createhsd` command.

- Skip option (**-S**)

  Specifying **-S** overrides the default skip option and *does not* skip the first stripe to protect the first LVM control block (LVCB).

- Size of the physical and logical partitions in the LVM logical volume group (**-T**)

  If you specify this option, it must be a power of 2 in the range of 2 through 256. The default is 4.

To create an HSD that stripes data across three identical VSD definitions on each of three disks in a system partition, type:

```
createhsd -n 1,3,5 -s 12 -g DATAVG -d DATAHSD
```

This creates the HSD definition DATAHSD and its underlying VSD definitions:

- DATAHSD1n1 on node 1. The local volume group name on node 1 is DATAVG. The global volume group name is DATAVGn1. The logical volume is lvDATAHSD1n1.
- DATAHSD2n3 on node 3. The local volume group name on node 3 is DATAVG. The global volume group name is DATAVGn3. The logical volume is lvDATAHSD2n3.
- DATAHSD3n5 on node 5. The local volume group name on node 5 is DATAVG. The global volume group name is DATAVGn5. The logical volume is lvDATAHSD3n5.

The usable HSD size is 12 Mbyte.

- *vsddiag*

  This command displays information about VSDs that can help you determine their status and collect information that helps IBM service representatives diagnose system problems.

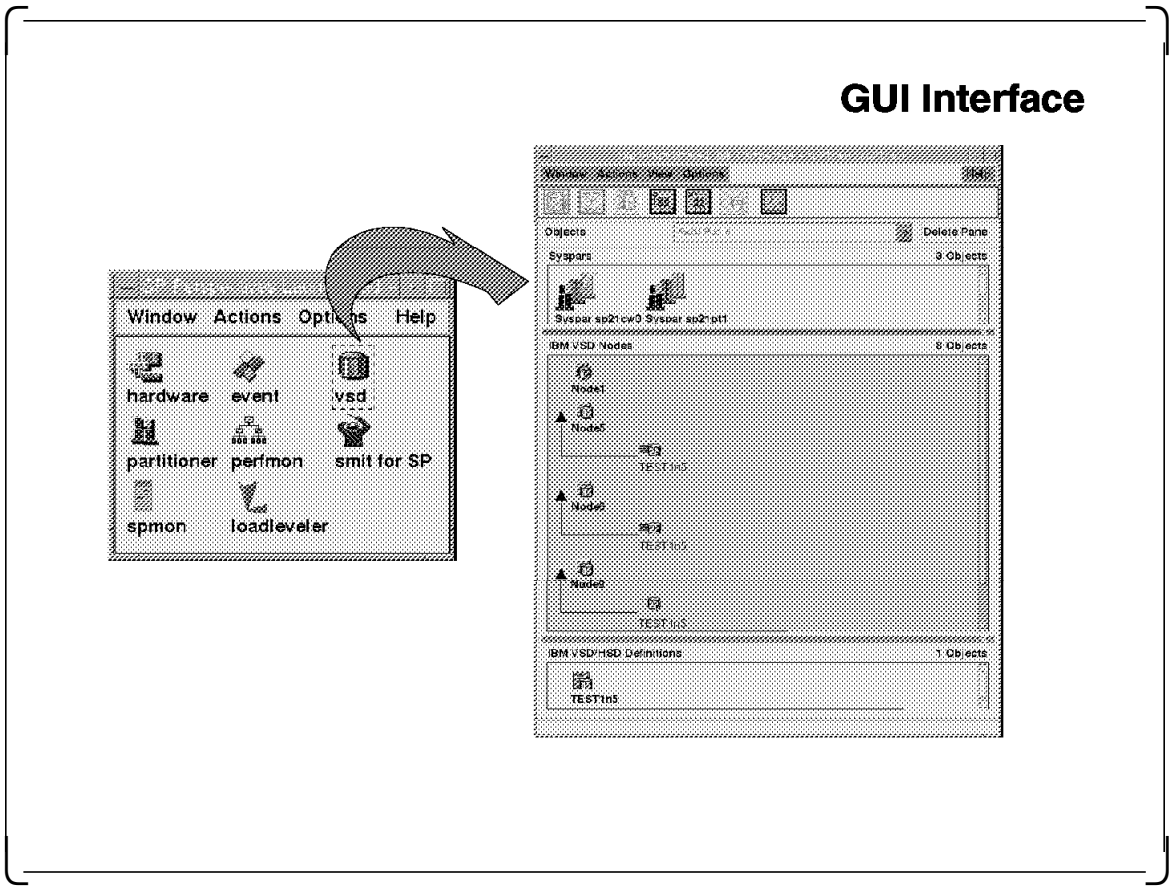
  Note: The `vsddiag` command can only be used when no VSD I/O is in progress.

Attention

In order to issue commands such as `createvsd` that operate on multiple nodes, you must have Kerberos authority and sysctl authorization.

For a complete description of these commands, refer to *Command and Technical Reference*, GC23-3900.

7.2.2 GUI Interface



Once VSDs have been defined in the SDR, they can be managed easily with the *Perspectives* graphical user interface, a part of the PSSP Version 2 Release 2.

To bring up the VSD Perspective initial panel, type spvsd or select the VSD icon from the Perspectives main panel.

With VSD Perspective, you can:

- Display or register the setup information in the SDR required for each node before you create VSDs or HSDs, such as:
  - VSD Adapter Name
  - Initial cache buffer count
  - Maximum cache buffer count
  - Number of outstanding logical volume read/write requests
  - Minimum buddy buffer size
  - Number of buddy buffers
  - Request block count
  - Maximum IP message size
  - Level of parallelism

  See *Managing Shared Disks: IBM VSD, HSD, and IBM RVSD Licensed Program Release 1.2*, GC23-3849, for the description of each parameter.

- Run verification write tests on VSDs.
- Move the state of VSDs.
- Monitor the status of the VSD nodes.
- Add a client to an existing VSD configuration.
- Display information about VSD clients, such as:
  - Name of the VSD
  - Node number on which this client is configured
  - Minor number of this client
  - State of the VSD
  - VSD client reads
  - VSD client writes
  - Owner

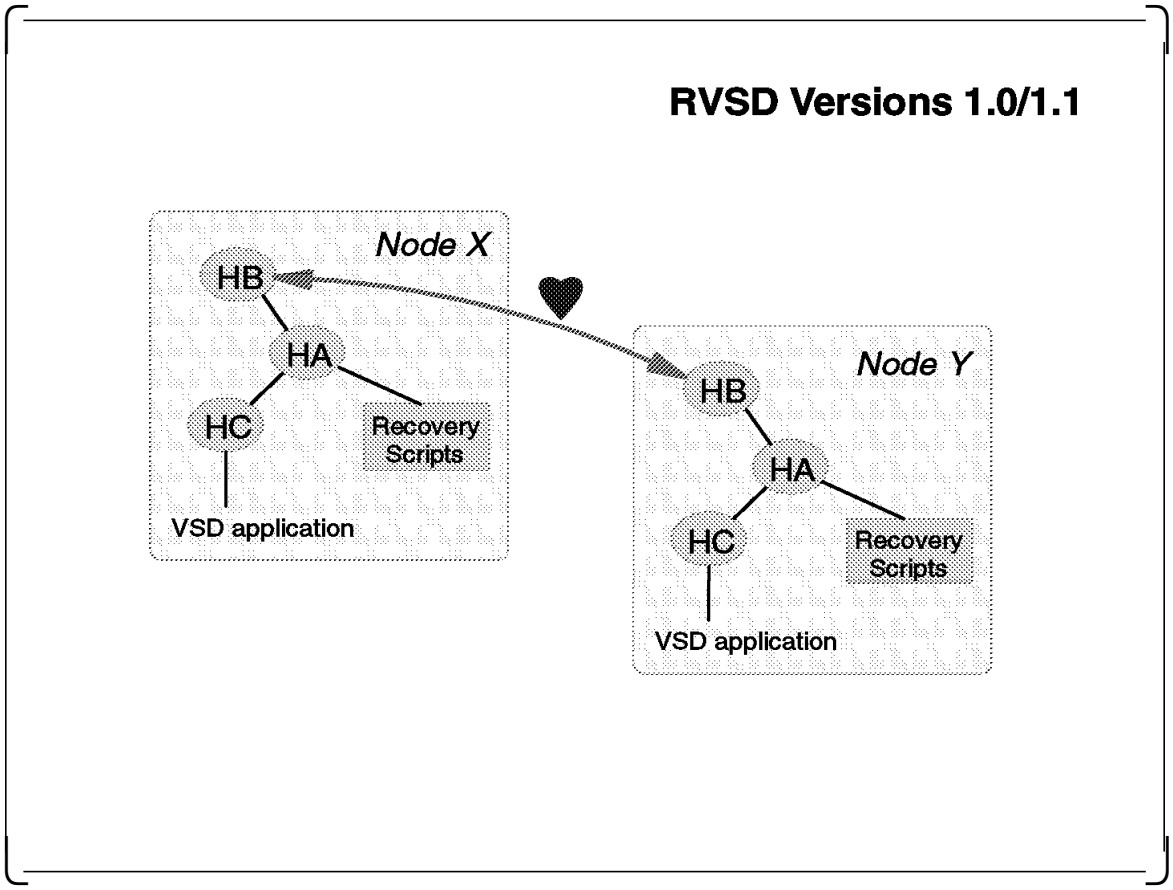
  The information is displayed on the client attributes notebook page.
- Display information about HSD clients, such as:
  - Name of the HSD
  - Node number on which this client is configured
  - Minor number of this client
  - Stripe size
  - VSD client reads
  - VSD client writes
  - Owner
- Display VSD and HSD statistics, such as :
  - Number of local logical read and write operations
  - Number of remote logical read and write operations
  - Number of client logical read and write operations
  - Number of physical reads and writes
  - Number of cache hits for read
  - Number of 512 Kbyte blocks read and written
- Reset VSD and HSD statistics.
- Run VSD diagnostics.

For the details on operating VSD Perspectives, refer to *RS/6000 SP PSSP 2.2 Technical Presentation*, SG24-4868 or *Managing Shared Disks: IBM VSD, HSD, and IBM RVSD Licensed Program Release 1.2*, GC23-3849.

Note: The *monvsd* tool is no longer available.

7.2.3 New RVSD Mechanism

7.2.3.1 RVSD Versions 1.0/1.1



This figure shows the components of RVSD Release 1.0 and 1.1. RVSD spawns three user level daemons on each node: HB, HA, and HC.

The heartbeat subsystem is comprised of all heartbeat daemons across a partition. It is responsible for detecting node status: up or down. When the status of nodes changes, the heartbeat daemon (HB) performs a two-phase commit protocol for the new node membership to guarantee that all nodes will follow an identical sequence of events. When all HB daemons agree on the new membership, each will notify its local HA daemon of the of the new node group.

Once notified by HB, the HA daemon will synchronize the running of recovery scripts that will flip VSD servers (if needed) or move VSDs to the stop state (in the case where a server is not available). Note that while this synchronization occurs, HB may detect more changes in the group. The HA daemon is designed to handle such scenarios and synchronize compound recovery.

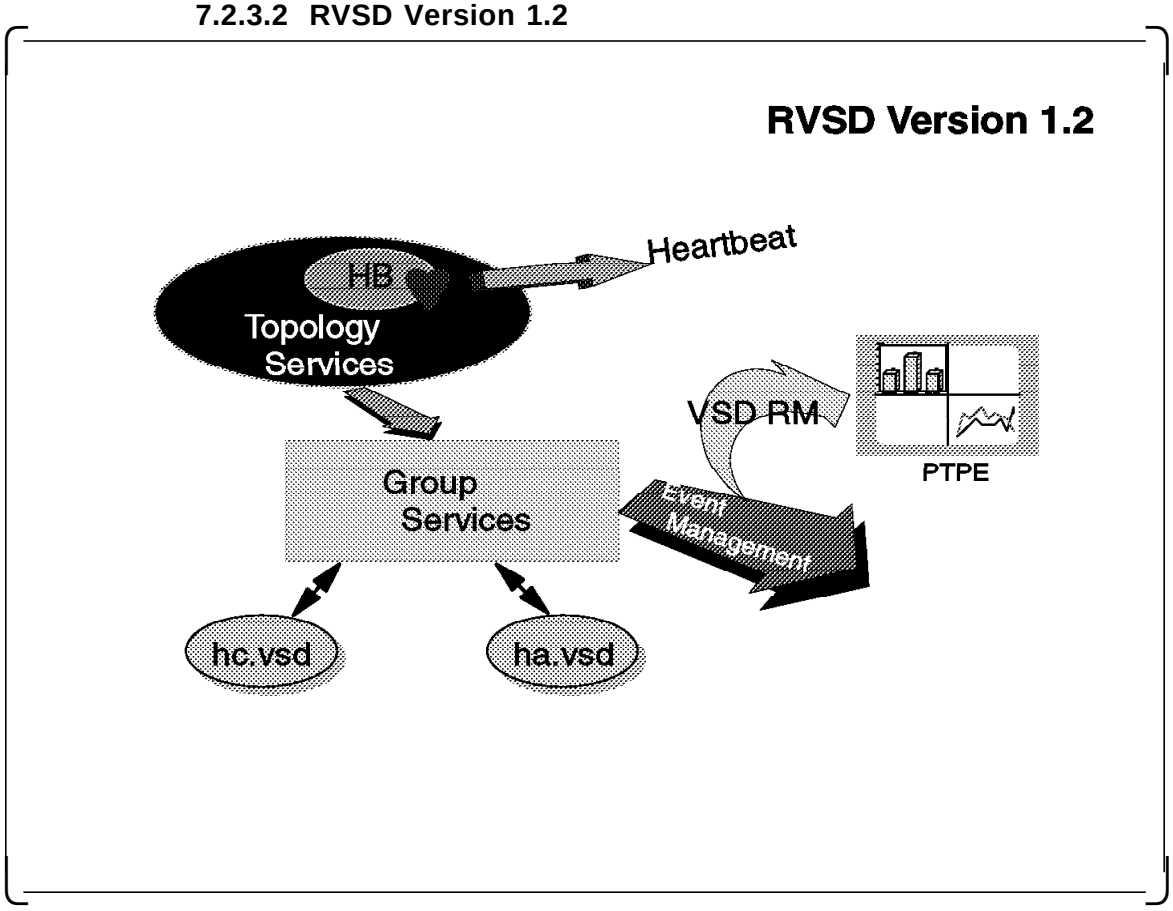
Once HA completes its recovery for the VSD subsystem, the HC daemon is notified that the group membership has changed. HC in turn, will notify its client that the group has changed, if needed (the HC daemon may or may not have a client connected to it on a given node).

Note that the heartbeat subsystem is also responsible for determining if quorum is reached. Quorum is computed by the following equation:

(number_of_nodes/2)+1

If the number of nodes that are considered by HB to be up is greater than or equal to quorum, then the group is considered active. Within an active group, the HA daemon runs scripts to bring to the active (ACT) state, to every VSD that has a server. If at any time quorum is lost, the HA daemon will run scripts to bring all VSDs into the stopped (STP) state. This is to eliminate the possibility that two nodes will try to access the same disk in the case of network partitions.

7.2.3.2 RVSD Version 1.2



RVSD Release 1.2 no longer includes the heartbeat subsystem as its component. Instead, RVSD uses the Topology Services deamon, heartbeatd, which is the component of the PSSP Version 2 Release 2 that keeps track of which nodes are available.

The Group Services daemon hagsd runs on every VSD node and receives availability information from the heartbeatd daemon. It uses this information to keep track of changes in process groups. Group Services daemon also manages group subscriptions, notification of membership list changes, and other message broadcasts.

The two major subsystems of RVSD, the VSD recovery driver (also called *ha*) and Connection Manager (also called *hc*), are both clients of Group Services, and exploit the Group Services functions.

Notes:

1. The *ha.vsd* shown in this figure is the name of the group consisting of the *ha* subsystems. It is also the name of the script that starts had daemon, which implements the *ha* function.
2. The *hc.vsd* shown in this figure is the name of the script that starts the hcd daemon, which implements the *hc* function. The name of the group specifically used by *hc* subsystems is *hc*.

The *ha* depends on Group Services to:

- Determine group membership.

- Store group state (active versus inactive as determined by *ha*).
- Queue pending membership changes.
- Expose membership and state changes consistently.
- Order proposed membership changes to expose failures first.
- Expose all known “simultaneous” departures together.
- Expose all known “simultaneous” joins together.
- Provide serialization primitives (group *voting*), and detection of failures during voting.

The *ha* itself will:

- Maintain the *membership* and *local node* environment variables.
- Invoke the UP/DOWN scripts during voting.
- Interface to the AIX System Resource Controller (SRC) as a reliable daemon.
- Perform error logging.
- Handle *group activation* with quorum.

The *hc* depends on Group Services to:

- Determine and expose group membership consistently.
- Manage group state.
- “Source” *ha* membership and state changes.

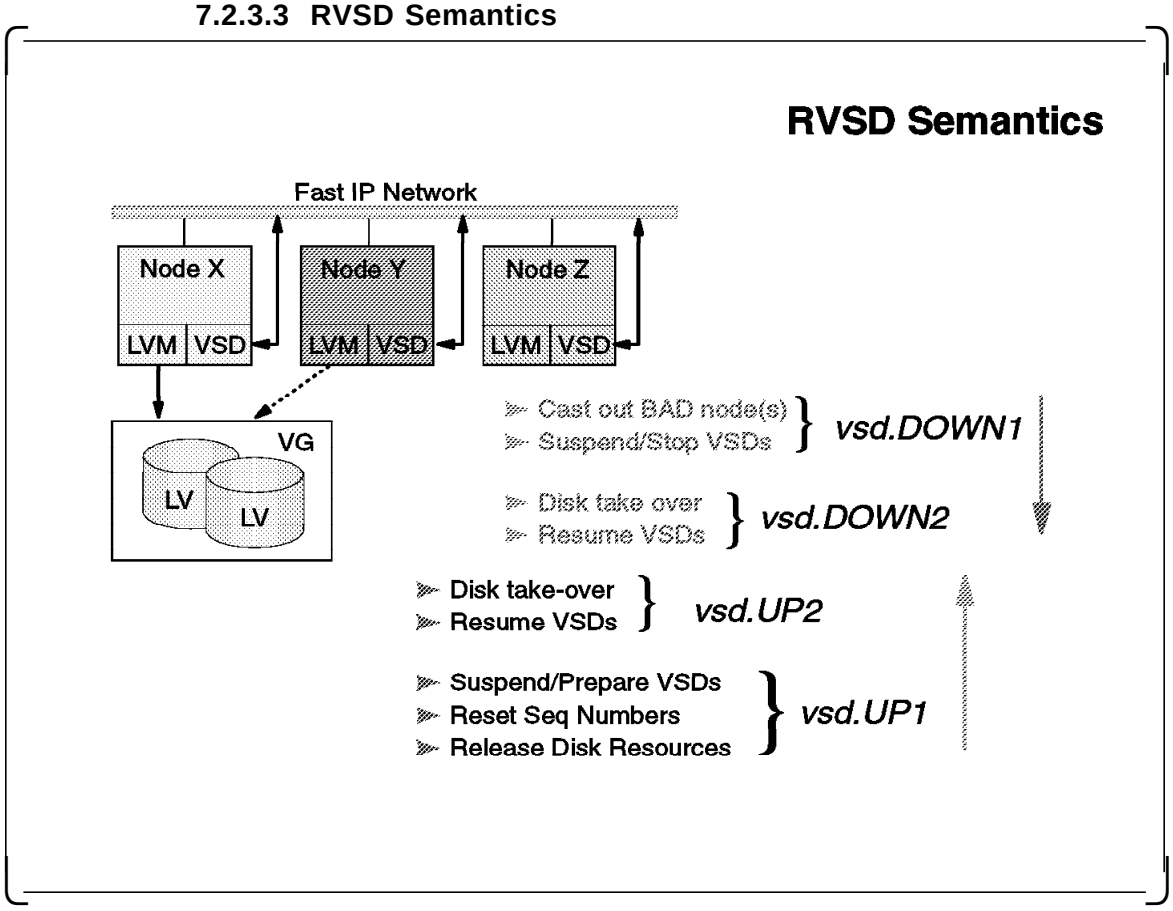
The *hc* itself will:

- Handle group activation based on the *ha*group state.
- Report current membership to its client.
- Perform “liveness” test (ping) on its client.
- Run the *activate* and *deactivate* scripts to start and kill its client as necessary.
- Interface to the SRC as a reliable daemon.
- Perform error logging.

For the details on Topology Services and Group Services, see Chapter 2, “Topology Services” on page 5 and Chapter 3, “Group Services” on page 25.

In addition, you can use Performance Toolbox Parallel Extensions (PTPE) to monitor the status of VSD, for example, I/O statistics, queued I/O requests waiting for a cache block, or rejected I/O requests. For details on how to use PTPE, see Chapter 8, “Performance Toolbox Parallel Extensions” on page 257.

7.2.3.3 RVSD Semantics



RVSD provides the following four script files that are executed by had daemon when the status of any VSD nodes changes:

- vsd.UP1

  This script is called first on the VSD primary direct client node after it is booted.

- vsd.UP2

  This script is called after completion of *vsd.UP1* script on the VSD primary direct client node.

- vsd.DOWN1

  This script is called on the VSD secondary direct client node (that is, backup VSD server) after the VSD primary direct client fails, or on the VSD primary direct client node on which the VSD server function is stopped.

- vsd.DOWN2

  This script is called after the completion of the *vsd.DOWN1* script.

The following table shows a sample scripts execution flow:

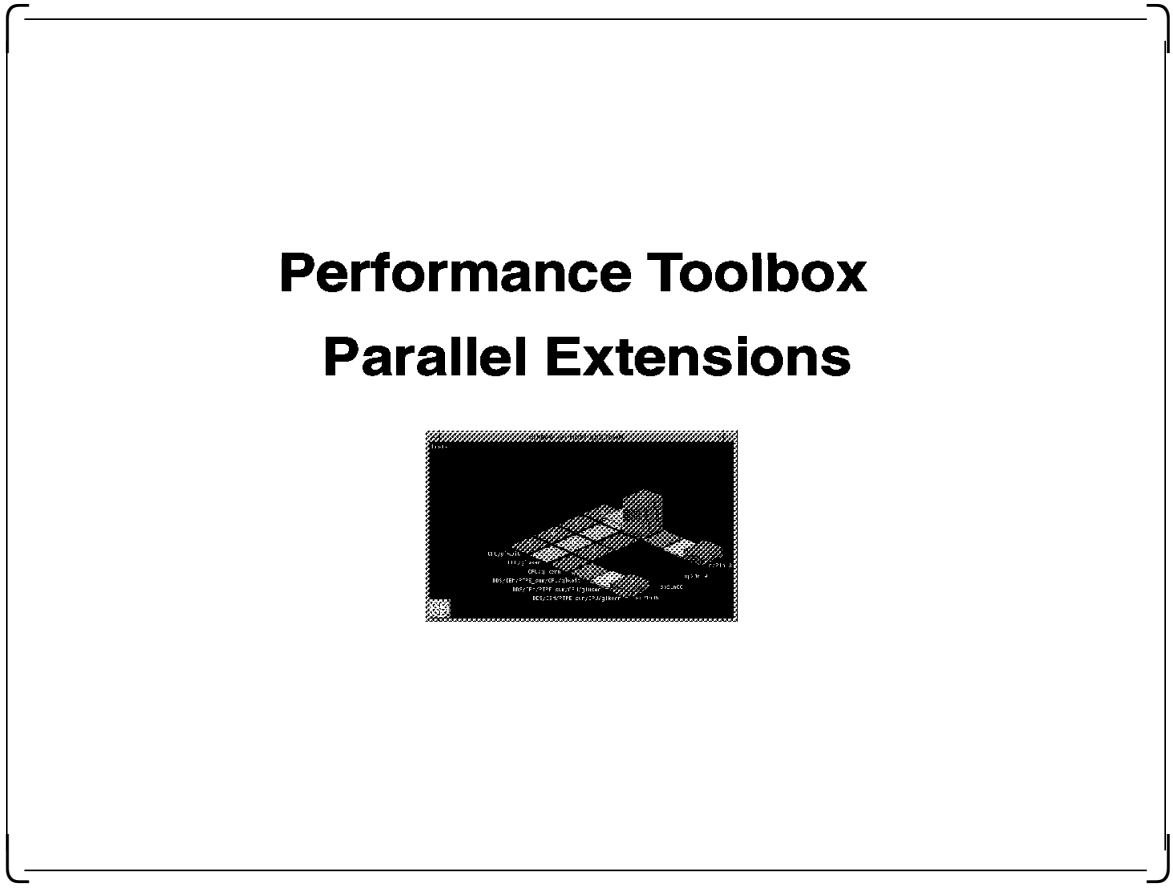
Table 3. RVSD Recovery Scripts Execution Sequence

VSD Nodes Status	Scripts Executed on P	Scripts Executed on S	VSD State
1. P and S stopped			Stopped
2. P starts	vsd.UP1 P vsd.UP2 P		Suspended Active
2. S starts	vsd.UP1 S vsd.UP2 S	vsd.UP1 S vsd.UP2 S	Suspended Active
3. P fails		vsd.DOWN1 P vsd.DOWN2 P	Suspended Active
4. P starts	vsd.UP1 P vsd.UP2 P	vsd.UP1 P vsd.UP2 P	Suspended Active

Note:

In this table, “P” stands for *Primary direct client node* (that is, primary VSD server), and “S” stands for *Secondary direct client node* (that is, backup VSD server).

Chapter 8. Performance Toolbox Parallel Extensions



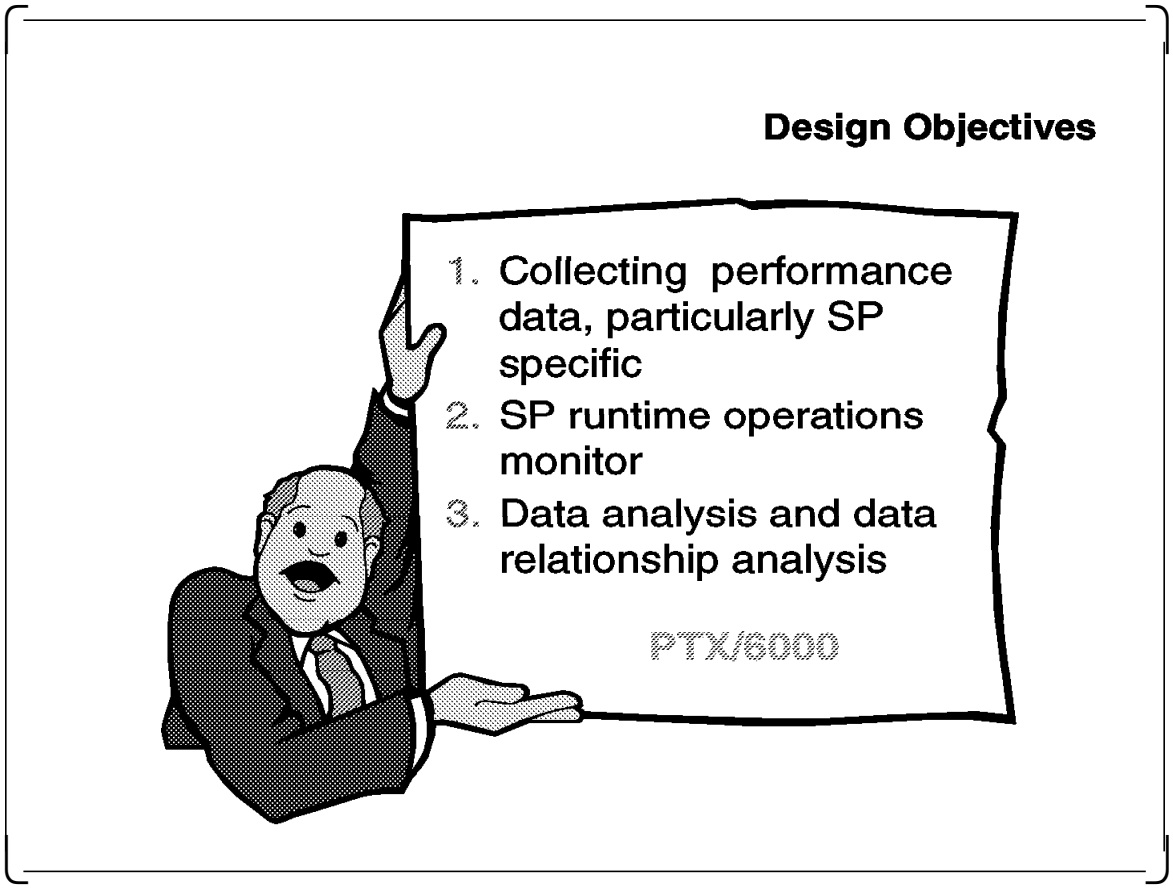
This chapter describes the Performance Toolbox Parallel Extensions. Performance Toolbox Parallel Extensions (PTPE) is a performance monitor for RS/6000 SP systems. PTPE builds on the capabilities of the Performance Toolbox for AIX, adding new monitoring functions that are specifically for the RS/6000 SP system.

PTPE provides four functions:

1. **Summary performance statistics**. These are the performance statistics available through the Performance Toolbox, summarized per group of nodes, and eventually, summarized for the entire SP.
2. **Subsystem statistics**. SP-specific subsystems are monitored through this interface for LoadLeveler, Virtual Shared Disk, and the Switch (SP or HiPS).
3. **Archiving**. The process where statistics are stored on the local node for later use in a spreadsheet, for analysis, or for storage in a database.
4. **PTPE Application Programming Interface**. This allows the statistics collected by PTPE to be accessed through customer applications using the PTPE API subroutines.

This chapter will cover these topics, as well as the installation and configuration of PTPE. Collection examples are provided at the end of this chapter.

8.1 Design Objectives



The design objectives behind PTPE correlate with the design objectives of the High Availability Infrastructure: Topology Services, Group Services, and Event Management. The High Availability Infrastructure is dependent on the existence of a pool of information reflecting the status of hardware and subsystems, and will eventually reflect the status of software and processes. Rather than inventing a completely new information provider of that kind, the advantages and infrastructure of Performance Toolbox for AIX were selected to provide the information necessary to enable a High Availability Infrastructure.

The three main objectives for the development of PTPE are:

1. Collecting SP-specific performance data
2. Providing an RS/6000 SP runtime operations monitor
3. Providing tools for data analysis and data relationship analysis

PAIDE/6000, the agent component of Performance Toolbox, is used as the statistics database for the High Availability Infrastructure. When availability needs to be implemented, statistics data of the involved subsystems must be available in order to take appropriate actions on failures. When the Event Manager client requests the status of the switch subsystem, the request is directed to the Event Manager cache on the local node. The Event Manager cache is regularly updated from the shared memory controlled by PAIDE/6000.

The PAIDE/6000 shared memory is fed by the Resource Monitor. PAIDE/6000 plays a fundamental part in the whole design of PSSP V2.2.

Performance Toolbox Parallel Extensions is built on top of the functionality of PAIDE/6000. Looking at the design objectives again, PTPE enables a system administrator to:

- Collect SP-specific performance data. This criteria is satisfied by implementing two different monitoring schemes:

  1. Collecting summary statistics. By grouping nodes together logically, a whole group of nodes collect performance statistics, and one group manager collects all that data. This creates a new set of performance statistics called the summary statistics.

  2. Collecting subsystem statistics. The High Availability Infrastructure software already collects and uses subsystem statistics. However, the way in which PTPE uses this information is slightly different in nature. When monitoring subsystem information, a user wants to see regularly refreshed statistics data. The Event Manager requests the Resource Monitor to give regular updates about subsystems, but requests updates on VSD only every 10 seconds. A user investigating the VSD subsystem may request information on a more frequent basis, for instance, every 5 seconds. Through the `xmperf` program, PTPE allows a user to monitor with a monitoring frequency as small as 1 second. The High Availability Infrastructure is based on event-driven information retrieval. Information retrieval happens based on monitoring intervals, defined in the SDR class EM_Resource_Class. This means that the information in the shared memory does not always reflect the actual status, that is, statistic value.

- Act as the RS/6000 SP runtime operations monitor. When trying to display performance statistics through the Performance Manager, larger RS/6000 SP systems (greater than 24 nodes) cannot be displayed in the same 3dmon window. By collecting summary statistics of groups of nodes, the display window can be decreased dramatically, without losing important information about the performance status of the group. A monitored group will develop a performance pattern. When this pattern is different from the previous moments, the group can be examined on an individual basis.

- Perform data analysis and data relationship analysis. PTPE implements a method to archive data and to manipulate that archived data, so that relational analysis can be performed. In `xmperf`, the `azizo` program allows a system administrator to do analysis only on data that is recorded. If performance statistics are influenced by data other than the recorded statistics, the relationship is lost. PTPE provides an API which allows a programmer to write applications to analyze the archived data. The archive can contain all available statistics, but may also be restricted to a selected number of statistics.

Because the High Availability Infrastructure depends on the PAIDE/6000 functionally, PAIDE/6000 is shipped automatically as a component of PSSP V2.2.

An additional advantage of this enhanced packaging structure is the changed pricing related to the Performance Toolbox. Previously, PAIDE/6000 had to be purchased per node individually, and for the Performance Toolbox Manager there was a one-time charge for the managing machine (usually the Control Workstation). Now, PAIDE/6000 is shipped at no cost with PSSP V2.2, so only the

managing applications are required to be purchased to harvest the advantages of the Performance Toolbox infrastructure. The two Licensed Program Products that explore the data collected by PAIDE/6000 are:

1. Performance Manager/6000 for AIX
2. Performance Toolbox Parallel Extensions for AIX. PTPE is a feature of PSSP Version 2.2.

Both products are available for a one-time charge, and there is a charge for each RS/6000 SP system PTPE is used on.

Also, when using Performance Toolbox for AIX, displaying individual performance statistics for more than 16 to 24 nodes can prove difficult, certainly when you use the 3D monitor. PTPE is designed so that the system administrator can configure groups of nodes that logically belong together. Each of these groups of nodes represent a particular function and particular behavior pattern in their usage.

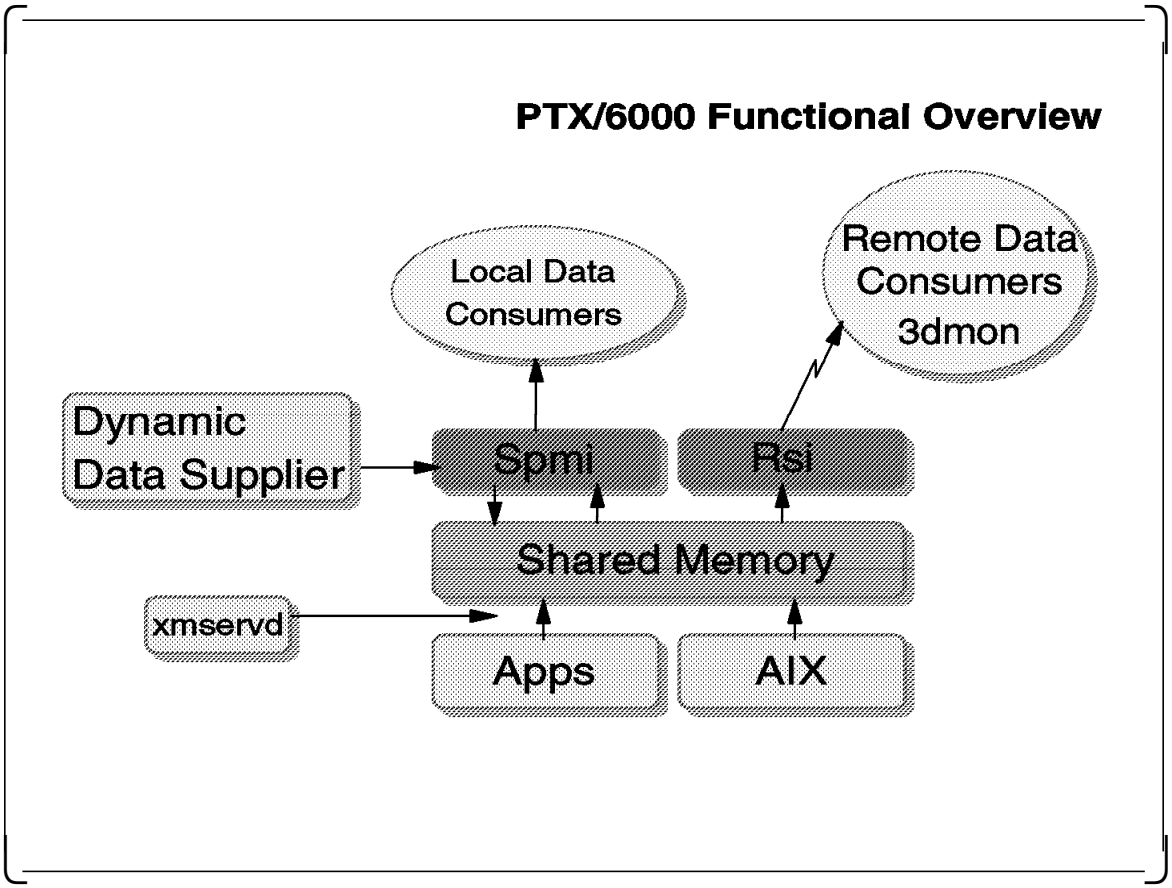
Note: Grouping in the PTPE sense is not related to the other RS/6000 SP grouping functions.

By grouping functionally similar nodes together, a system administrator can now concentrate on monitoring groups of nodes rather than monitoring nodes individually. If a group's behavior changes over time, the monitoring can then concentrate on monitoring the members of the group, since the normal Performance Toolbox functionality will be available as well.

This functionality of PTPE and PAIDE/6000 and Performance Manager/6000 enables a system administrator to:

1. Collect performance statistics.
2. Collect summary performance statistics.
3. Collect RS/6000 SP specific subsystem statistics.
4. Monitor RS/6000 SP operations.
5. Store and archive statistics for data analysis and relationship analysis.
6. Write customized applications using the PTPE API.

8.2 PTX/6000 Functional Overview



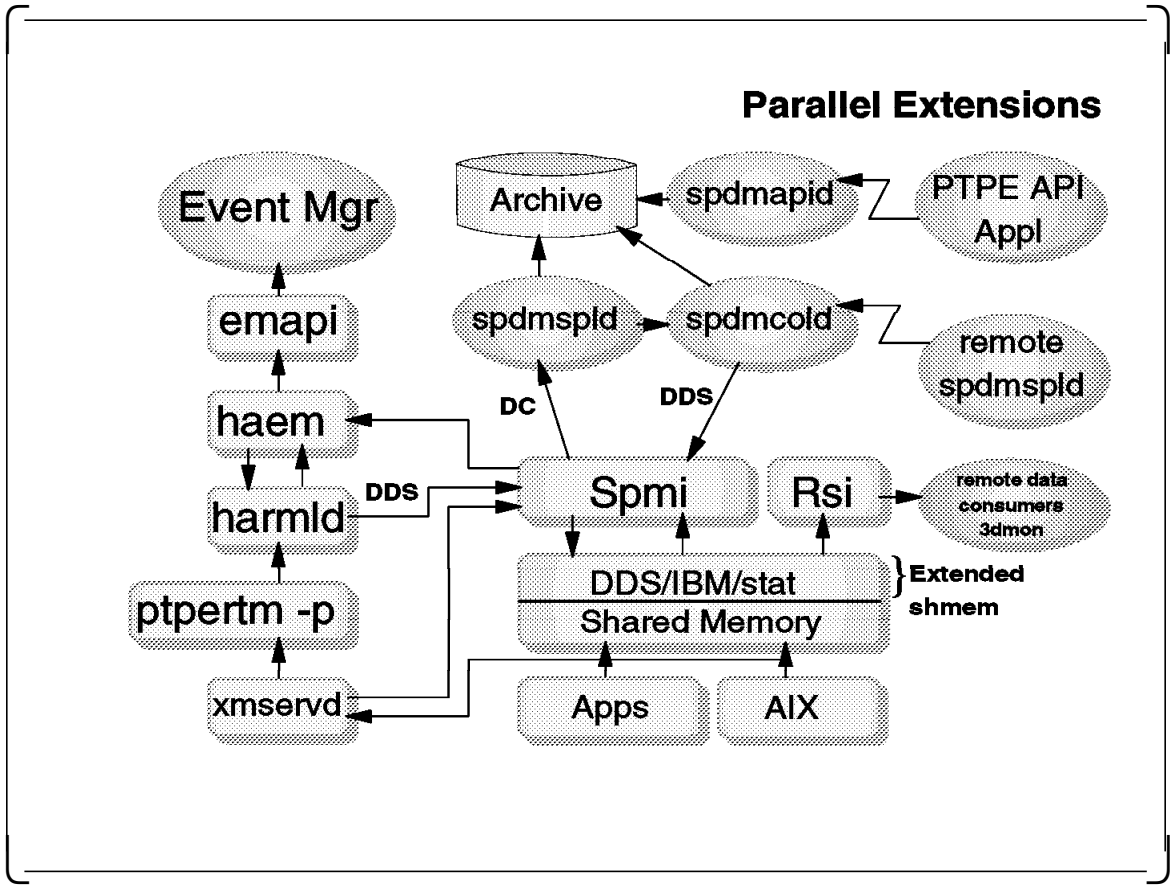
The Performance Toolbox for AIX consists of the Performance Manager/6000 and PAIDE/6000; they are also called the manager and the agent, respectively.

Usually, the PAIDE/6000 component (the agent) is installed on all machines or nodes that require monitoring. The agent software monitors local application and AIX performance statistics. These statistics are only collected when requested, that is, when a local or remote data consumer requests performance statistics from a particular machine. It is at this time that the agent software starts collecting data and allocates a piece of shared memory in which the statistics are stored. After a time out period, the shared memory segment is cleared and freed.

Performance statistics are provided locally by the `xmservd` program, which is started by `inetd` on the request of a data consumer. The data is stored in shared memory through the System Performance Measurement Interface (SPMI). The SPMI is an API which can also be used by other applications that provide monitoring information. Such an application is called a Dynamic Data Supplier (DDS). A Dynamic Data Supplier is functionally similar to the function supplied by the `xmservd` application, but it monitors application-related monitoring statistics. In the High Availability Infrastructure, both the Resource Monitor and the Event Manager interface with the SPMI.

Local and remote data consumers can access the shared memory, that is, the stored statistics, by using the Remote Statistics Interface (RSI). The RSI API allows you to access more than one `xmservd` daemon, and thus allows you to monitor a cluster of machines at the same time.

8.3 PTX Parallel Extensions Functional Overview



As previously mentioned, the Parallel Extensions to Performance Toolbox performs the following functions:

1. **Collecting SP-specific performance data**. SP-specific performance data is currently implemented for the following subsystems:

   - The Switch
   - LoadLeveler
   - VSD

   One of the tasks in the High Availability Infrastructure in PSSP V2.2 is to monitor subsystems and hardware. This task is performed by the Resource Monitor component. Particularly, the `harmld` daemon provides this information and feeds the subsystems performance statistics through the Dynamic Data Supplier and SPMI interface into shared memory.

   This information can be used to monitor through the Performance Manager software, but it is specifically used for the High Availability Infrastructure. The Event Manager (haem) reads this information from the shared memory segment to feed the subsystems responsible for providing information about the hardware and software status of the RS/6000 SP system.

2. **SP runtime monitoring**. The second part of the Parallel Extensions is built from the idea that a system administrator should have a global view of the RS/6000 SP performance behavior. Also, given the large number of nodes in

bigger RS/6000 SP systems, it is humanly impossible to view all nodes at the same time from, for instance, the 3dmon application. Runtime monitoring can be accomplished by summarizing performance statistics of individual nodes and displaying the summaries of groups of nodes rather than individual nodes. The summary statistics are collected by Data Manager nodes from individual collector nodes by the spdmcold daemon (the RS/6000 SP Data Manager COLlector Daemon). The spdmcold daemon communicates with its collectors, the spdmspld daemons (the RS/6000 SP Data Manager SamPLer Daemon) by way of a private protocol.

On top of the Data Manager nodes (summarizing collected statistics for groups of nodes), the Data Manager nodes are also summarized. The summary statistics of the Data Managers are collected and summarized at a higher level by the Central Coordinator. This will be discussed in more detail later in this chapter.

Both spdmcold and spdmspld respectively feed and read from the shared memory segment by way of the SPMI interface. Contrary to other remote data consumers, the spdmcold daemon does not use the RSI interface; however, xmperf and 3dmon, as part of the Performance Manager software, do use the RSI interface.

3. **Data Analysis and Data Relationship Analysis**. Performance Manager provides the ability to analyze performance data with azizo. However, azizo can only analyze data that was collected from the recordings of a monitoring console. This means that if a statistic is influenced by a statistic that was not recorded, the relationship will never be found. Also, analysis is restricted to derivatives like maximum, minimum, and average statistics.

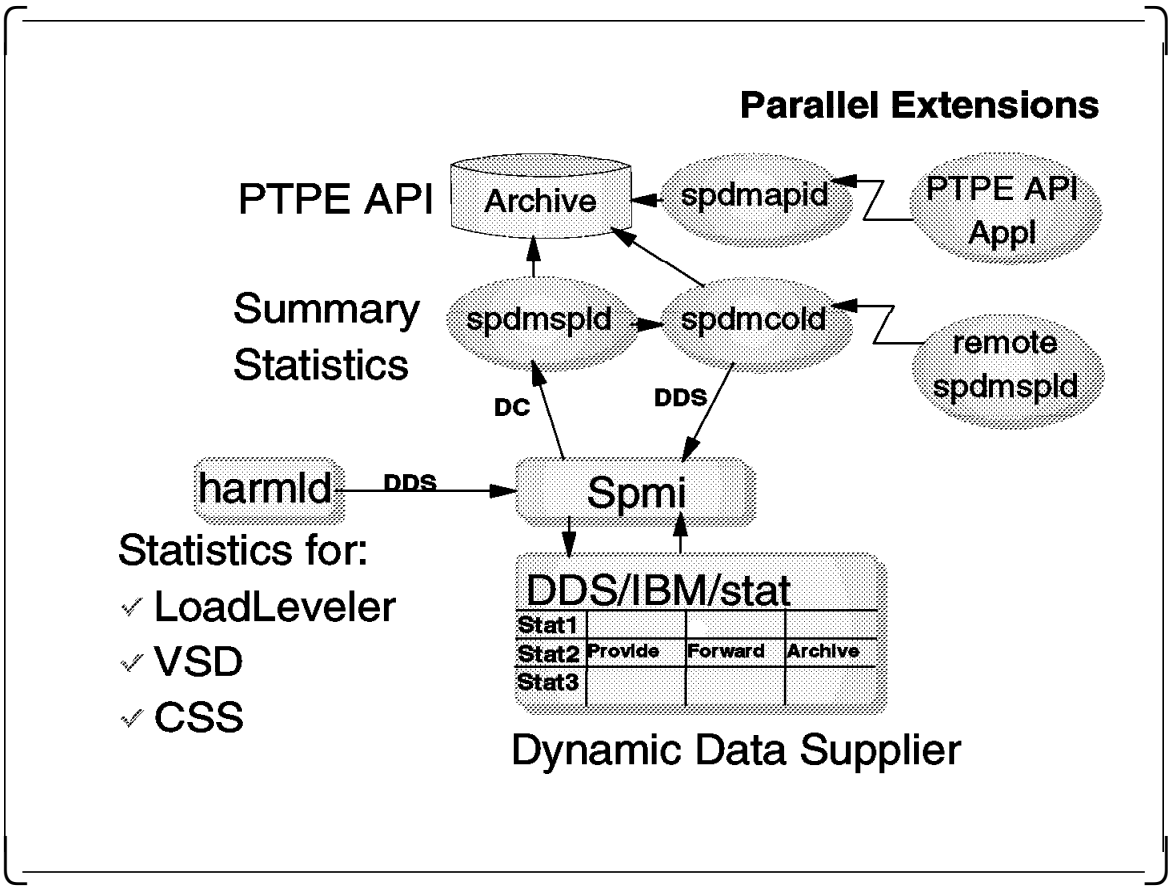
   In order to be able to find statistic relationships, all collected data should be recorded and archived. The archive created by PTPE is created on every node. The analysis tools are provided through the PTPE API interface. So, using this API, relational statistics can be retrieved from the archive by writing applications.

The shared memory segment that is allocated by the Performance Agent software will be extended for the RS/6000 SP specific statistics. If no other Data Supplier, like the harmld daemon, is running at that moment, then the extended shared memory segment is created by the ptpertm program (PTPE RunTime Manager). This program is invoked during the startup of xmservd from inetd, by specifying this program as a supplier program in the */etc/perf/xmservd.res* file. The entry that appears in the /etc/xmservd.res file should look like this:

```
supplier:   /usr/lpp/ptpe/ptpertm -p
```

Also, this program makes sure that if the harmld daemon is not running, it is started. The harmld daemon is part of the Resource Monitor. All statistics provided by PTPE and the harmld daemon will be fed into this extended shared memory segment and are readable by local data consumers through the SPMI interface and by remote data consumers through the RSI interface.

8.3.1 Parallel Extensions



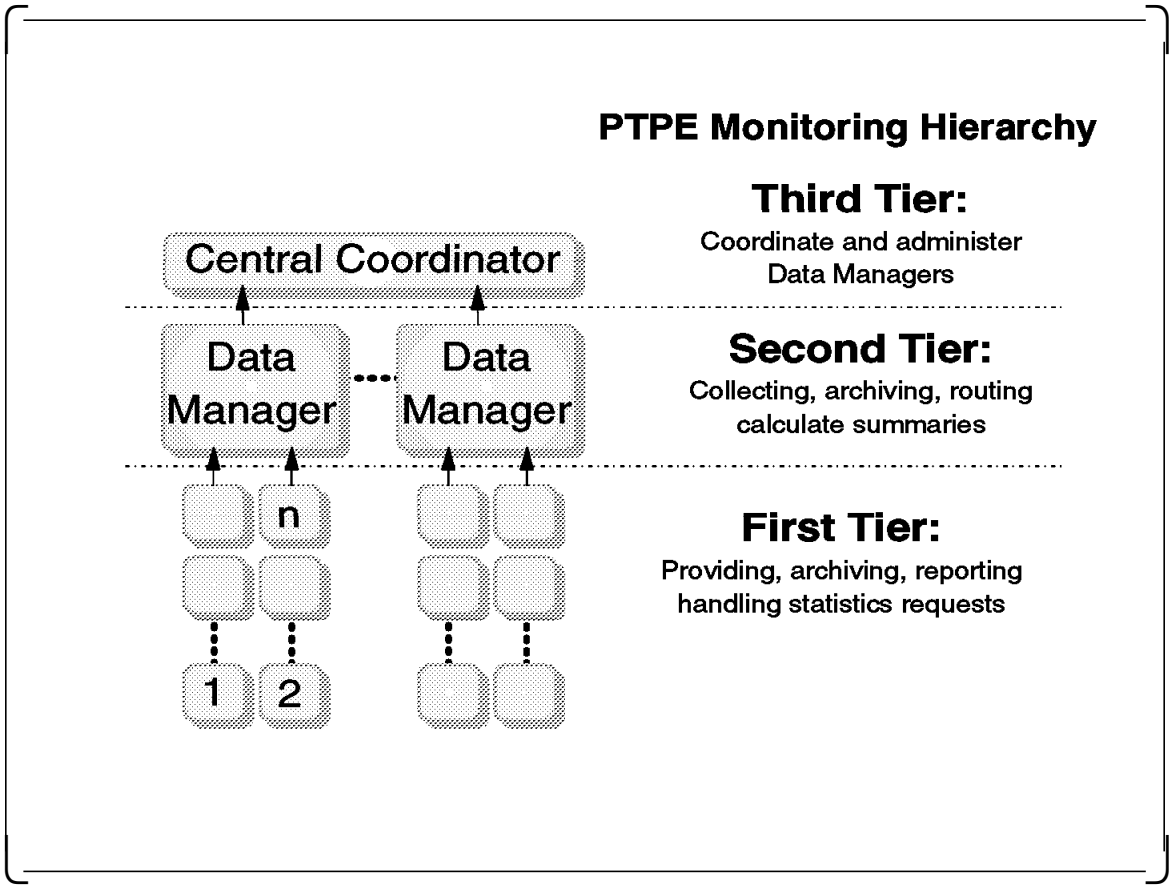
Performance statistics provided by the performance agent are stored in a shared memory segment. These statistics are either a quantity value or a ratio value. For instance, a value like TCP/sndtotal represents the number of packets sent in the last sampling interval, and thus this value is a quantity. The value CPU/glwait represents a ratio, or a percentage, of the time the CPU is waiting for I/O.

All performance statistics provided by the extended shared memory segment also represent values for specific actions. These actions are:

1. **provide**, this value represents whether this statistic is available for monitoring on the node. The PTPE subsystem knows about all statistics, and thus all statistics are represented in the shared memory segment. If a statistic cannot be found on the node, the provide flag is set to off or 0. This flag will be referenced by subroutines from the PTPE API, when the statistic value is referenced by an application. If the provide flag is 0, the requesting subroutine knows not to expect data of this statistic.
2. **forward**, this value tells the SPDM sampler daemon to forward this statistic to its Data Manager. Forward values can be 0 or 1.
3. **archive**, this value is set when this statistic needs to be archived. Forward values can be 0 or 1.

The system administrator can configure this behavior in the file: */etc/perf/ptpe.cf*.

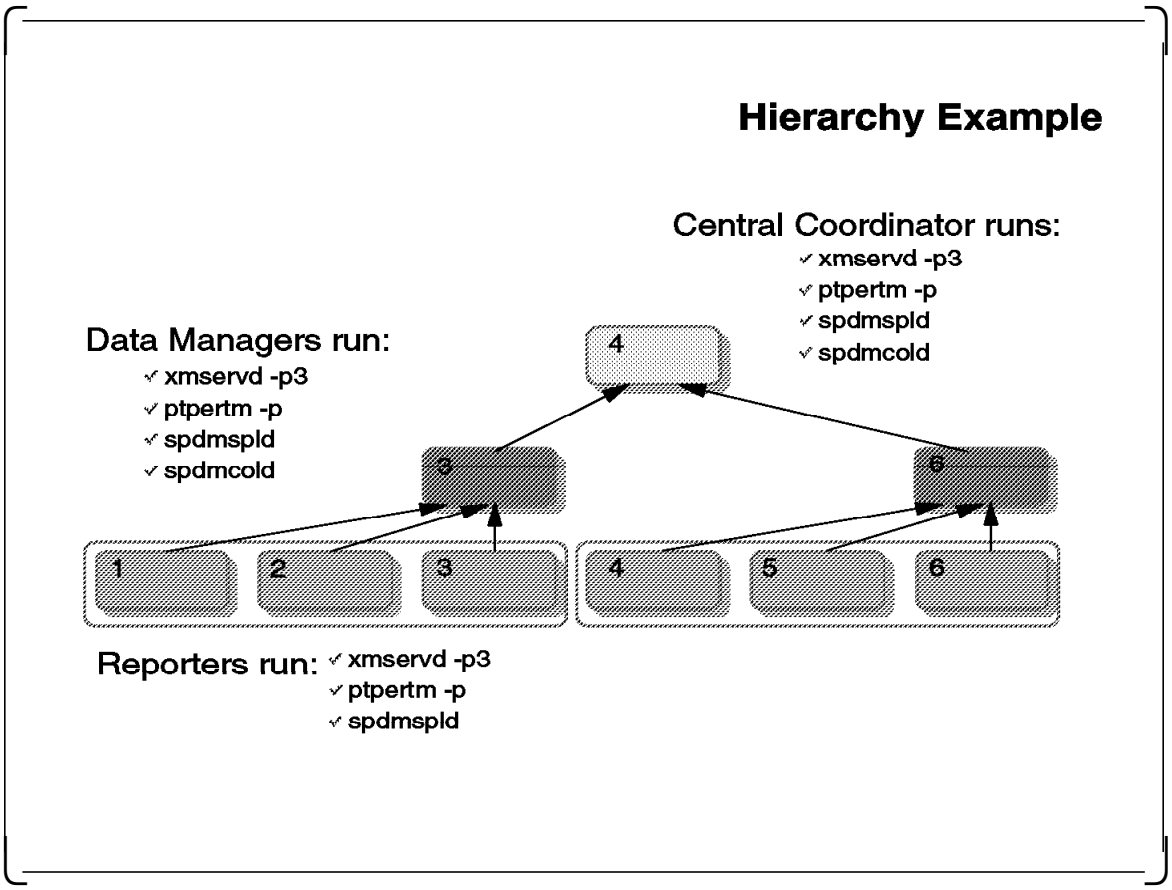
8.4 PTPE Monitoring Hierarchy



In order to set up an environment that allows you to collect summary statistics, PTPE must be configured with a monitoring hierarchy. The first tier consists of the Data Collector nodes. The collector nodes provide the performance statistics for the local subsystems and pass on performance statistics to the next tier: the Data Manager tier. The Data Manager can collect performance statistics for as many nodes as you wish; however, generally Data Managers collect performance statistics for up to 16 nodes. Of course, there may be situations where an RS/6000 SP system will have more than 16 nodes. Grouping considerations are discussed later in this chapter.

The Data Manager tier reports to the highest level in the hierarchy, the Central Coordinator tier. The Central Coordinator collects the summary data of the Data Managers and summarizes the Data Managers' performance statistics of the Data Managers. The summarized performance statistics that are provided by the Central Coordinator represent the total performance overview of one RS/6000 SP system. The performance statistics values that come from the Central Coordinator do not make sense for individual components of the RS/6000 SP, but provide a global impression. This information could be further investigated by sub-analyzing the underlying layers, such as the Data Managers.

8.4.1 Hierarchy Example

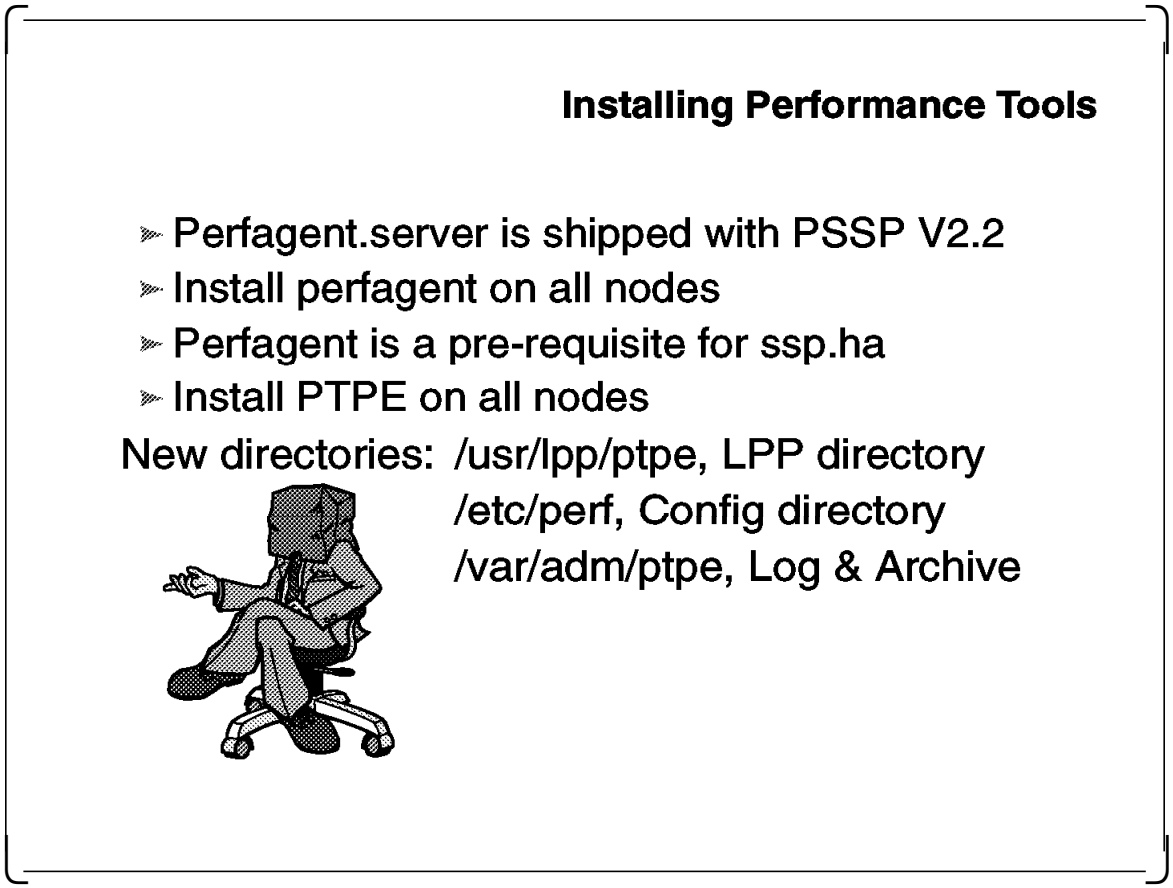


Creating a sampling hierarchy will strongly depend on the way your RS/6000 SP works and how it is configured. Scientific usage of a system differs completely from the way a LAN Consolidation RS/6000 SP works. Later in this chapter, the specific hierarchy configuration methods are discussed in more detail.

In the example shown above, the hierarchy is configured in a six-node RS/6000 SP system. The system is logically divided into two hierarchy groups, both reporting to their own Data Managers. The Data Managers are also Data Collectors. When looking at the first group (nodes 1, 2, and 3), node 3 is assigned as the Data Manager of that group. All three `spdmspld` daemons on the node communicate to the `spdmcold` daemon on node 3. The `spdmspld` daemon on nodes 1 and 2 communicate through the network while the `spdmspld` daemon on node 3 communicates locally.

Both Data Managers, nodes 3 and 6, report their summary statistics to the Central Coordinator, node 4. The basic rule in assigning a Central Coordinator is that this node cannot also be a Data Manager. The Central Coordinator communicates to its Data Managers exclusively through the network.

8.5 Installing PTPE



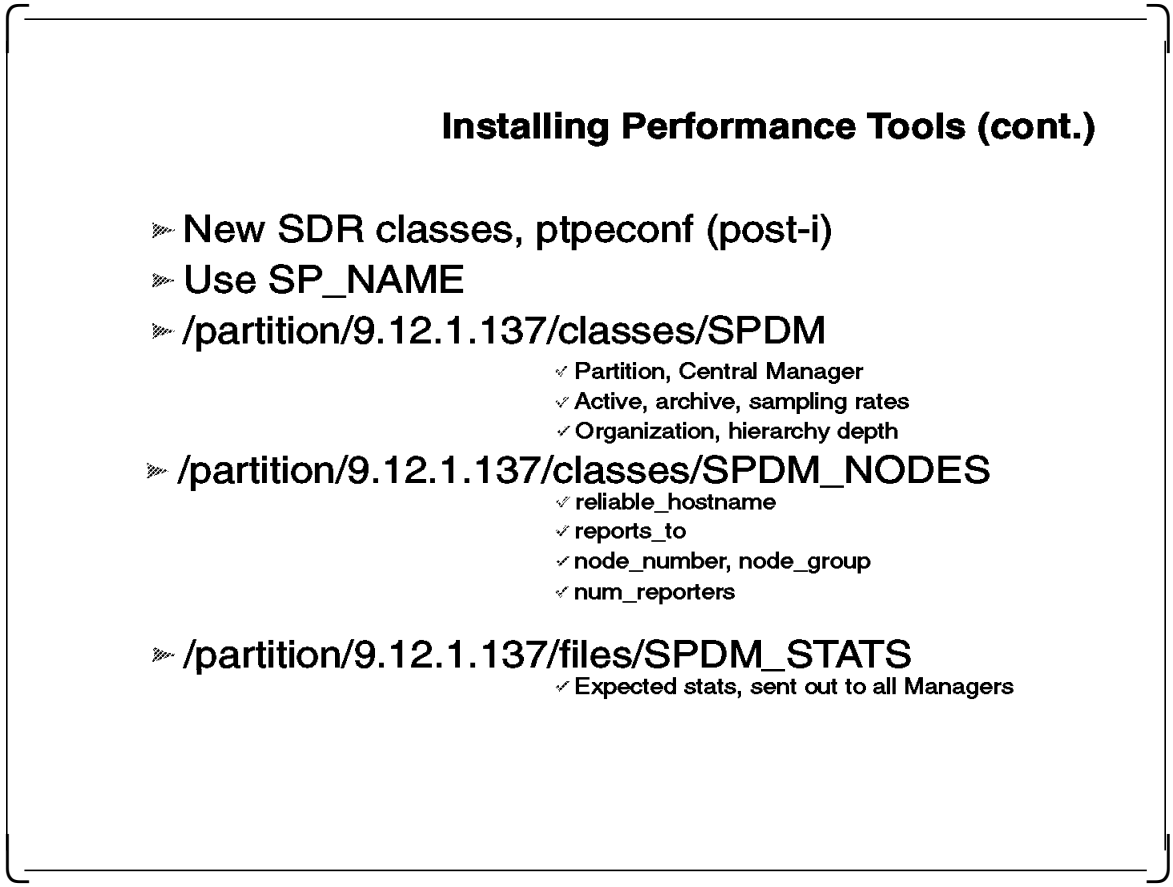
Now, in PSSP V2.2, the PAIDE/6000 is a prerequisite of a component in the PSSP software: ssp.ha, the High Availability Infrastructure software. The PAIDE/6000 software will be shipped at the same time the PSSP software is ordered. Also, for PSSP V2.2, the software is of level PAIDE/6000 V2.2. Normally, this level of software would be configured for AIX V4.2 configurations. However, this level is also supported for AIX 4.1.4 systems and up.

This new way of packaging also changes the pricing structure of the PAIDE/6000 product. Previously, the PAIDE/6000 software had a one-time charge per node. Now PSSP V2.2 comes with PAIDE/6000 at no charge, independent from the number of nodes.

To fully harvest the PAIDE/6000 functionality, two products can be purchased on a one-time charge basis:

1. **Performance Manager**, which provides the applications `xmperf`, `3dmon`, `exmon` and `azizo`. There is a charge for each machine the Performance Manager is running on.
2. **Performance Toolbox Parallel Extensions**, which is charged per RS/6000 SP system, and is a feature of the PSSP V2.2 software.

In order to use the PTPE functionality on the entire system, PTPE needs to be installed on all nodes. This requires approximately 4 MByte of storage on each node.

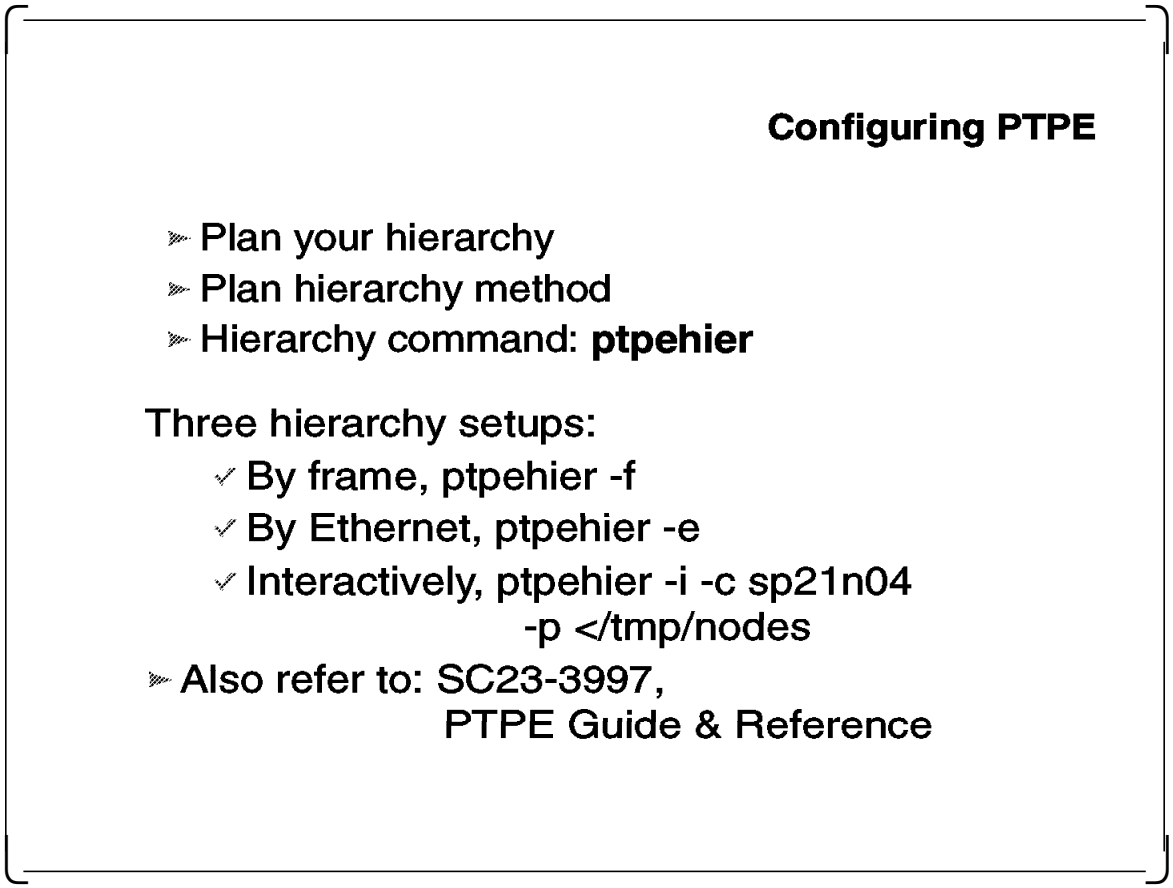


After completing the installation of PTPE on the Control Workstation, the installation process created two new SDR classes in each configured and active partition. The new SDR classes are:

1. **SPDM**. This class stores the information related to this partition and the characteristics of the Central Coordinator. Also, the sampling rate of collecting and archiving is stored in this file.
2. **SPDM_NODES**. All reporting Data Managers and Reporters belonging to this partition are listed in this class.

Also, a file is created in each partition's file directory, containing all the performance statistics that a Data Manager can expect from its reporters. This file is called SPDM_STATS.

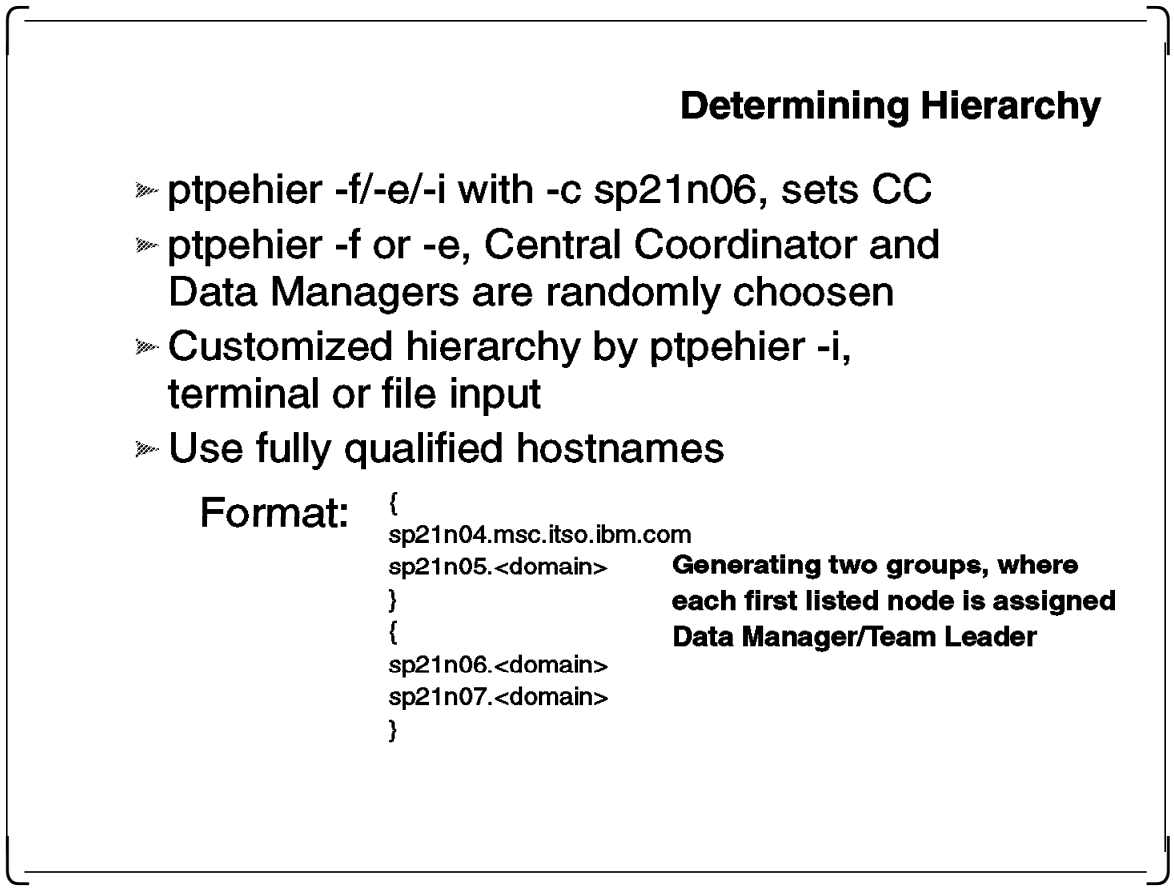
8.6 Configuring PTPE



As discussed before, the hierarchy of a PTPE configuration strongly depends on the usage of the RS/6000 SP system. Therefore, the hierarchy configuration command understands three methods of configuring a hierarchy:

1. **Frame** `ptpehier -f` configures a hierarchy based on the frame layout of the RS/6000 SP system. Each frame contains a group of nodes reporting to a Data Manager.
2. **Ethernet** `ptpehier -e` configures a hierarchy that reflects the Ethernet subnetting in the RS/6000 SP. Each Ethernet subnet is configured as one group of nodes reporting to a Data Manager. Each subnet Data Manager reports to a Central Coordinator.
3. **Interactively** `ptpehier -i -c nodexyz < /tmp/input` configures a hierarchy according to the input file /tmp/input and with a Central Coordinator node *nodexyz*.

8.6.1 Determining Hierarchy

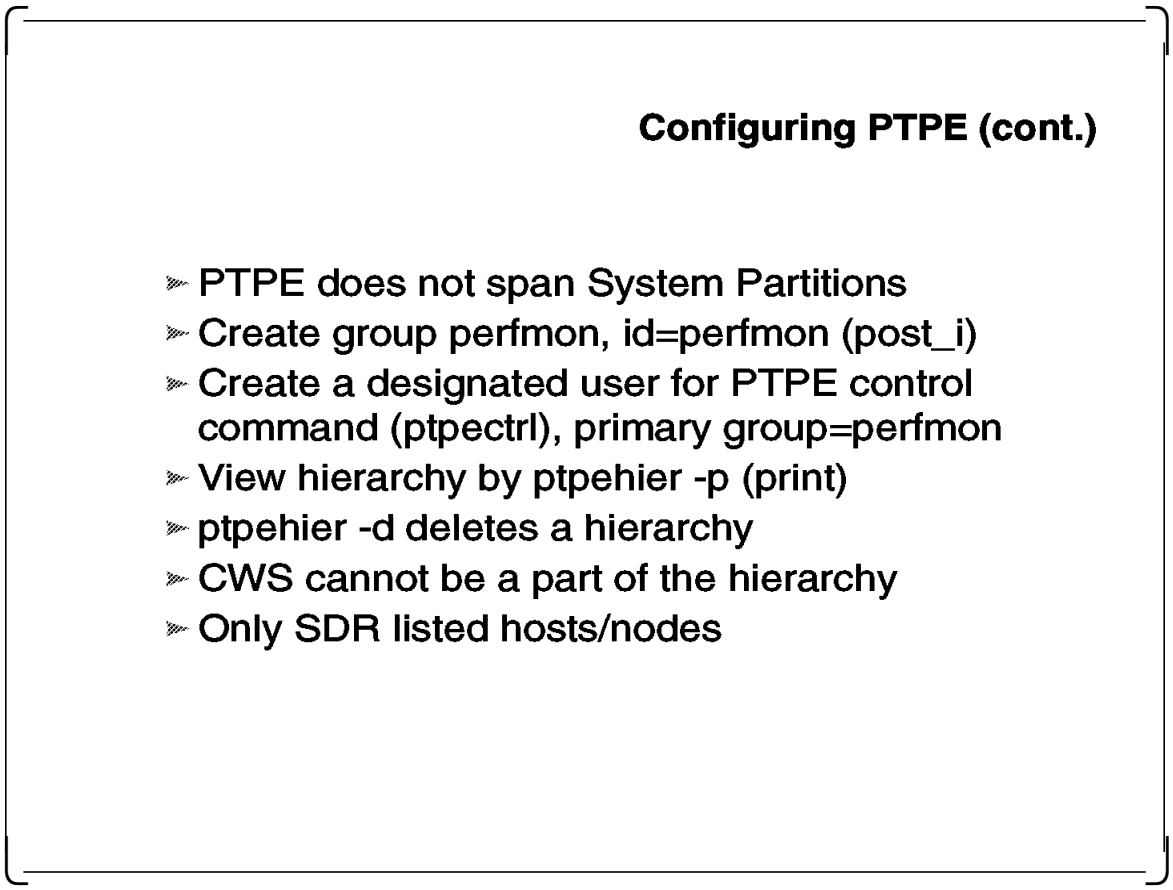


When configuring the hierarchy with flags `-e` or `-f`, the Data Manager and the Central Coordinator are randomly selected. When configuring a hierarchy with flags `-e`, `-f`, or `-i`, the Central Coordinator can be set specifically with the `-c` flag. When configuring the hierarchy with `ptpehier -i`, the nodes listed in the input file are assigned to their respective roles. The format should look as follows:

```
{
sp21n04
sp21n05
}
{
sp21n06
sp21n07
}
```

In this example, the hierarchy consists of two groups of nodes. The Data Manager of each of the groups is the node listed at the top of each group. In this example, nodes sp21n04 and sp21n06 would become Data Managers. If the `-c` flag is provided, the Central Coordinator will be the node listed with the flag. Otherwise, one of the remaining nodes will become the Central Coordinator. A Data Manager cannot be a Central Coordinator.

8.6.2 Configuring PTPE, Miscellaneous



PTPE does not cross partition boundaries. This means that setting up a hierarchy is determined by the way the system is partitioned. There are scenarios that a system administrator should be aware of when the RS/6000 SP system is (or will be) partitioned.

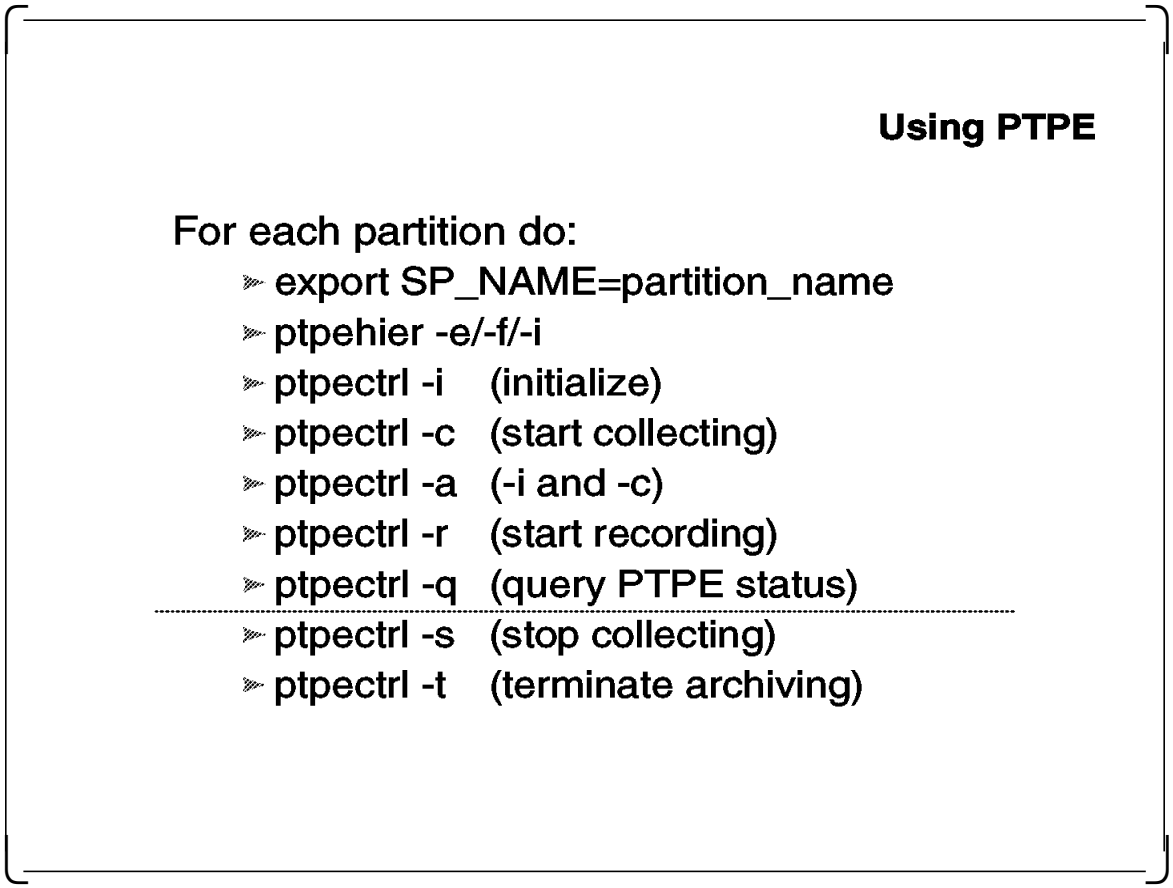
The scenarios are based on newly-delivered systems. The default partition, when configuring the SDR for the first time, will be a total system; that is, all the nodes will be in one partition. When the PTPE hierarchy is configured, this partition is referenced and accordingly reflected in the PTPE hierarchy.

When the system administrator decides to change the partitioning of the RS/6000 SP system, the PTPE hierarchy will not be updated. It will work as if nothing happened to the system. But when the PTPE hierarchy needs to be re-established, the `ptpehier` command only references the partitions. This means that the old PTPE hierarchy cannot be reconfigured. So when a system undergoes partitioning, the PTPE hierarchy should be reconfigured accordingly.

In order to manage the PTPE configuration and to issue control commands, such as to start or stop collections, the user ID should be configured with a primary group named perfmon. This group is automatically created when installing PTPE. If no propagation mechanism is used for user administrative configuration files, the nodes may have different group IDs for the perfmon group. Make sure that each node will eventually get updated with similar group IDs.

Finally, the supported machines in a PTPE hierarchy can be only nodes listed in the SDR. So the Control Workstation cannot be a part of the monitoring hierarchy.

8.7 Using PTPE

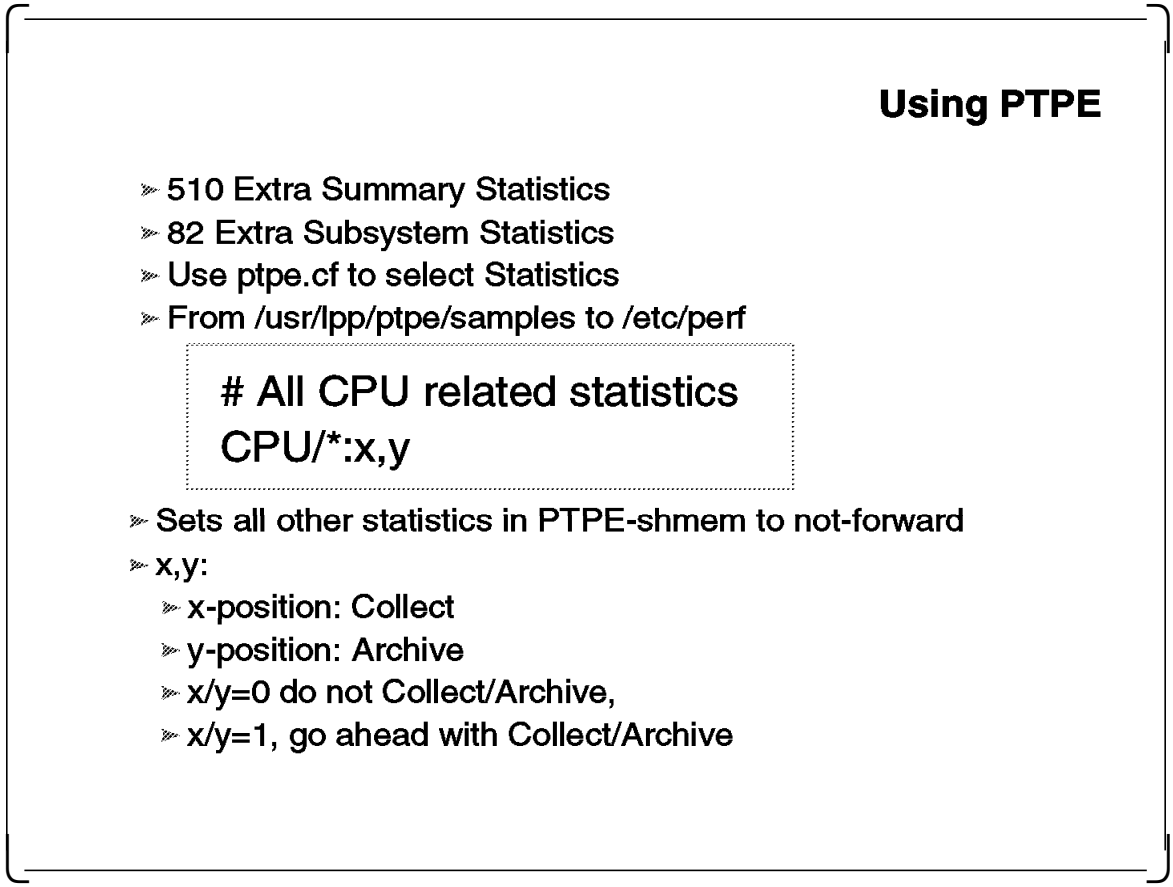


When the RS/6000 SP system is partitioned, each partition should be configured with its own hierarchy, with each partition represented by a designated Central Coordinator. There are two ways to configure and control the PTPE monitoring hierarchy:

1. Using the command line interface, with commands like `ptpehier` and `ptpectrl`.
2. With the Performance Toolbox Perspectives tool. This tool is to be used when the *ssp.perfmon.gui* fileset is installed on the Control Workstation. The Perspectives launch pad will show a *perfmon* icon after installation of ssp.perfmon.gui. Later in this chapter, the Perspectives interface for Performance Toolbox will be discussed in detail.

With the command line interface, the monitoring hierarchy in each partition can be managed and controlled by:

- Setting the SP_NAME environment variable
- Using the `ptpectrl` command



PTPE will cause extra overhead on the total RS/6000 SP system. Through the Performance Toolbox Parallel Extensions, approximately 500 statistics are collected and manipulated, on top of the statistics already monitored through Performance Toolbox. Also, the Resource Monitor subsystem in the High Availability Infrastructure monitors 82 extra subsystem statistics. In order to minimize the impact of this overhead, PTPE allows a system administrator to select the statistics for monitoring. The file /etc/perf/ptpe.cf can be used to set up a selected number of monitored statistics. This file originates in the LPP directory of PTPE: /usr/lpp/ptpe/samples. The general format of this configuration file is:

```
Stat/substat:x,y
```

When PTPE feeds summary statistics into the shared memory segment, the full annotation of the statistic is:

```
DDS/IBM/PSSP.harmld/CSS/nobufs
```

or

```
DDS/IBM/PTPE_sum/DDS/IBM/CPU/glwait
```

In /etc/perf/ptpe.cf, the statistics do not need to be listed in the long format. It is sufficient to provide the subsystem's real statistic, like CPU/glwait, or a wildcard, as in the foils example of CPU/*. Or, if a system administrator is only interested in the CPU summary statistics, the ptpe.cf can contain only the line:

```
DDS/IBM/PTPE_sum/DDS/IBM/CPU/*:1,0
```

The configuration part following the colon tells the PTPE software how to react to the statistic just listed:

- The x-position represents the collection action.
- The y-position represents the archive action.

When x=0, then collection is false. When x=1, then collection is true. When y=0, then archiving is false. When y=1, then archiving is true.

The following chart depicts the relationship between the ptpe.cf file configuration and the values of provide, forward, and archive in the shared memory segment.

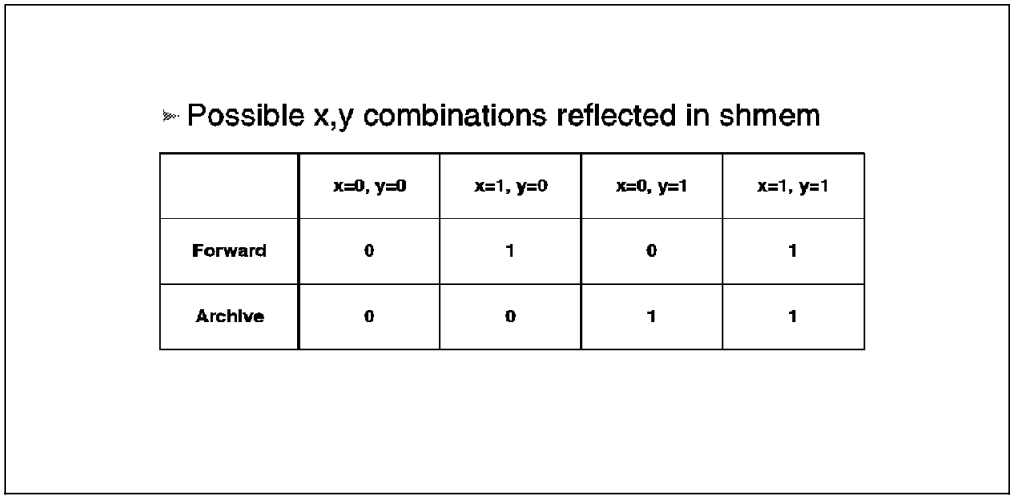
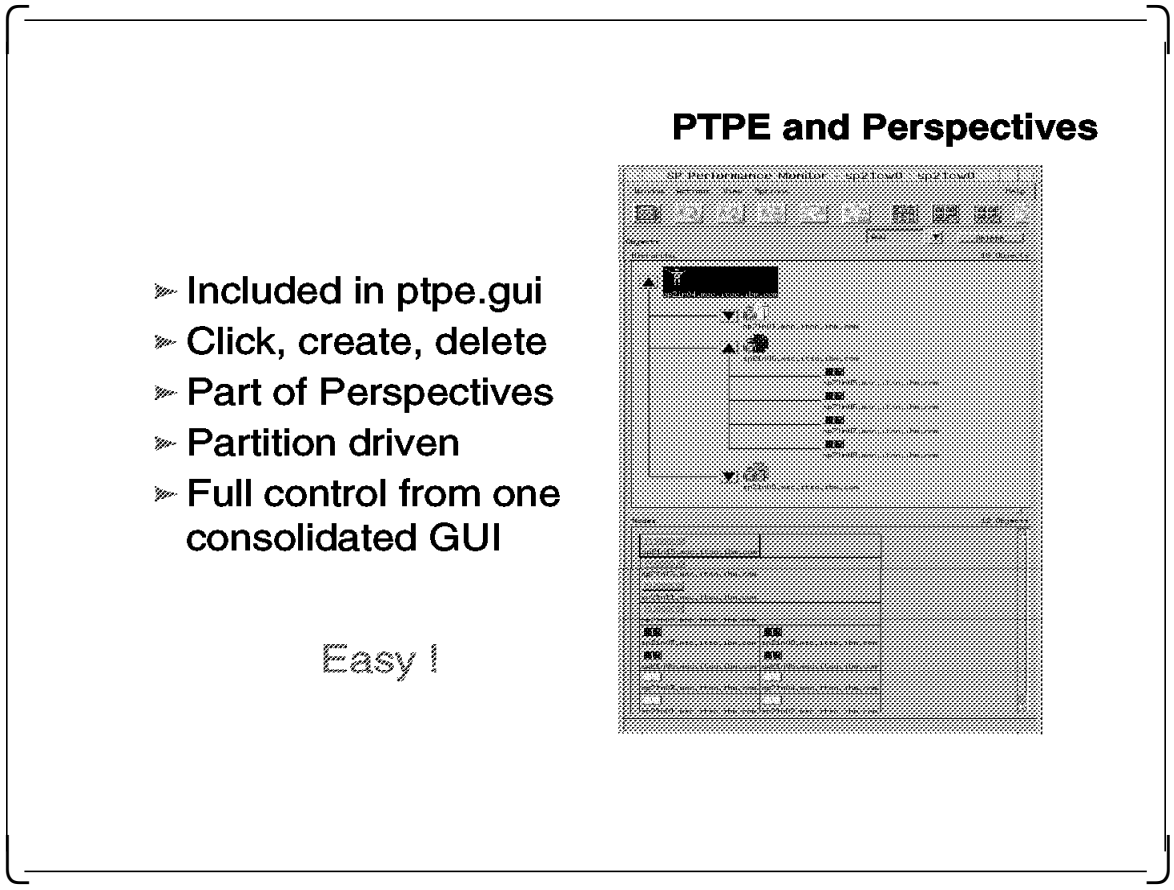


Figure 1. Shared Memory

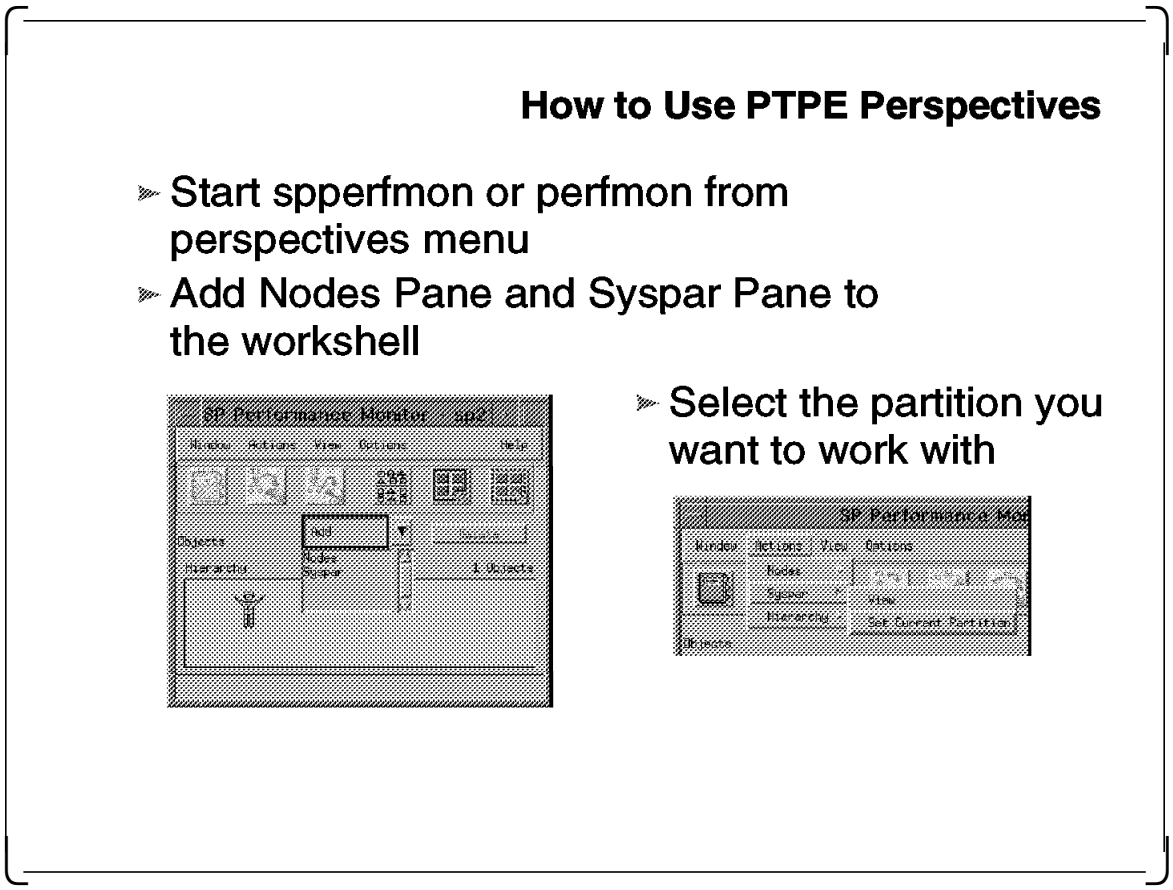
8.8 PTPE and Perspectives



Configuration and the control of PTPE can also be performed through the *Perspectives* interface. Perspectives is a new consolidated Graphical User Interface that was developed for PSSP V2.2. The Perspectives application for PTPE can be launched from the Perspectives menu (invoked through the `perspectives` command), but it can also be launched directly from the command line with `spperfmon`. The PTPE GUI is shipped with the *ssp.perfmon.gui* fileset.

In the following section, the configuration and control of PTPE will be covered step by step, similar to what already has been covered with command line programs.

8.8.1 How to Use PTPE Perspectives



When starting the spperfmon application, the window that shows is empty, provided no hierarchy is stored in the SDR. The performance monitor Perspectives interface has three window areas:

1. The **action buttons** in the top of the window: Window, Action, View, Options and Help.
2. The **action icons** in the icon bar of the window. Depending on which pane is active in the Perspectives window, the number of action icons will be different. Always present are the following icons:
   - The Sort Object icon which allows objects in a certain pane to be displayed according to a certain sorting rule, for instance by node number.
   - The Select All icon.
   - The Deselect All icon.
   - The Notebook icon which provides detailed information on selected objects, either a node, a system partition, or a hierarchy member.
3. The **pane windows area**, which can contain three views:
   - The hierarchy
   - The nodes
   - The system partitions

The way in which the perfmon Perspectives application works is to select a partition in which to work (compare this with setting the SP_NAME variable to a partition name), then select a node, and finally select the role the selected node is going to play in the hierarchy.

The number of icons in the icon area depend on the active panes. The following list shows the panes and their associated icons:

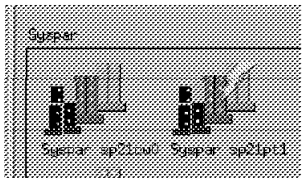
a. Hierarchy pane
   - Remove Data Manager
   - Remove Central Coordinator

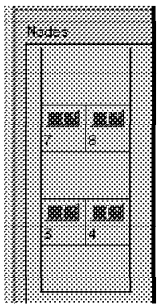
b. Nodes pane
   - Filter Objects
   - Remove Active Filter

c. Partition pane
   - Select Central Coordinator
   - Select Data Manager
   - Select Data Reporter

How to Use PTPE Perspectives

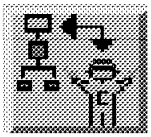
- The selected System Partition





- From the Nodes Pane, select a node to become the Central Coordinator

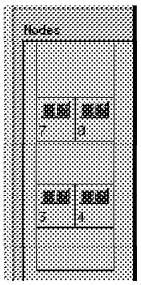
- Click on the Central Coordinator icon, highlighted through selecting a node

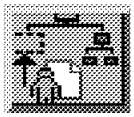


When the System Partition pane and the Nodes pane are added to the workshell, one node must take the role as Central Coordinator. Select a node and click on the Central Coordinator icon to set its role.

How to Use PTPE Perspectives

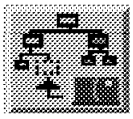
- From the Nodes Pane, select a node to become a Data Manager



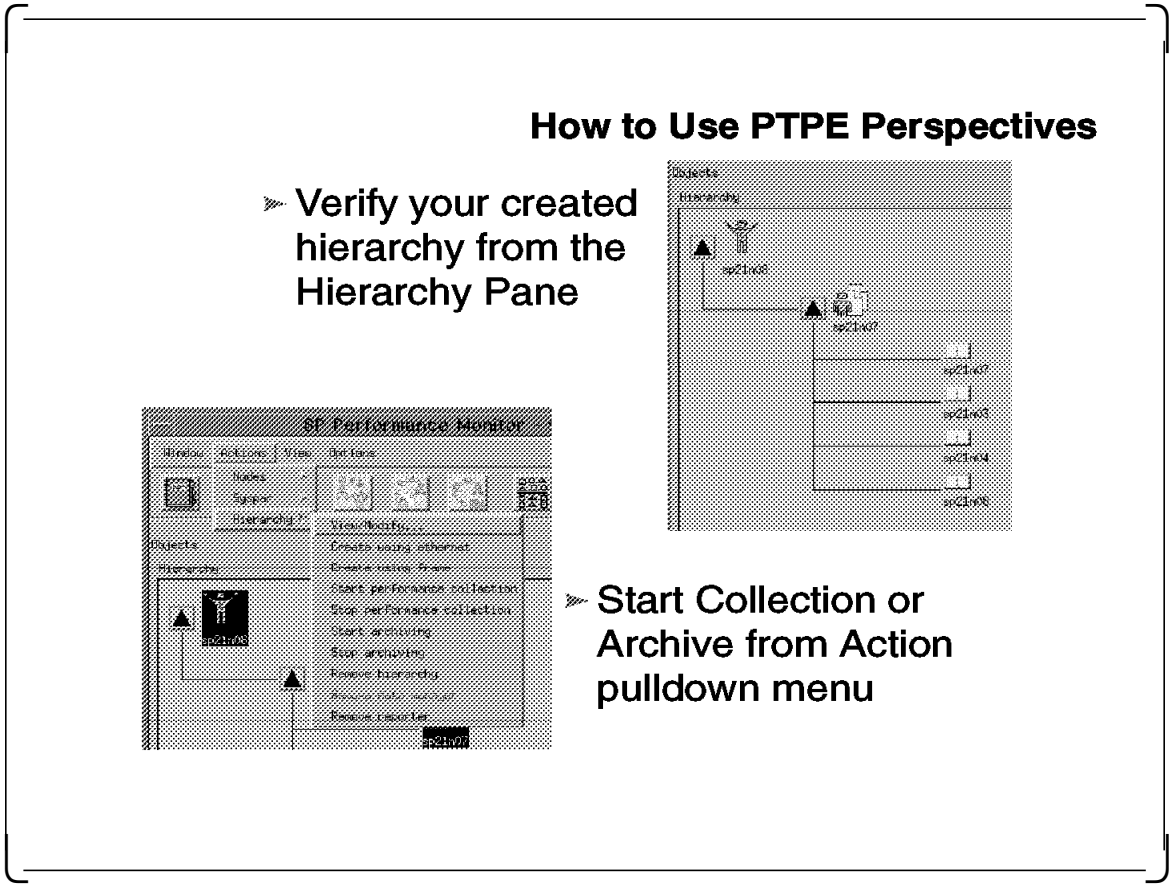


- Click on the Data Manager icon

- Select the rest of the nodes to become Data Reporters
- Click on the Data Reporter icon



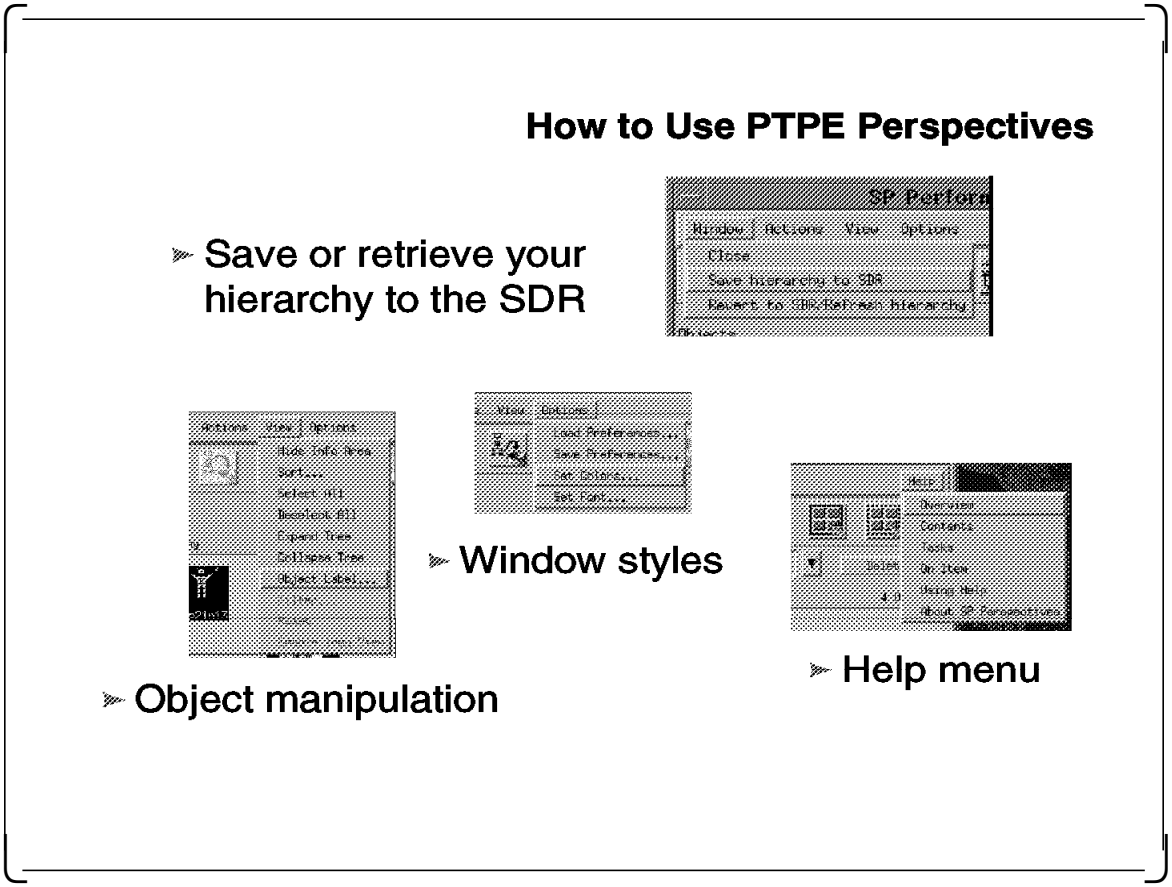
If the Central Coordinator is selected, the next tier can now be defined by again selecting a node in the nodes pane, and assigning a Data Managers role to the selected node. When the Data Manager tier is defined, the Data Reporters can be assigned to their roles. Select nodes from the nodes pane, and activate their roles by clicking on the Data Reporter icon.



When the role selection is finished, the hierarchy pane graphically shows the hierarchy.

Rather than selecting nodes individually from the actions associated to the pulldown Action button, the hierarchy could also be defined by selecting the **Create using Ethernet** button. All options available through the command line interface are available in the `perfmon` Perspectives application as well.

Also, after the hierarchy is defined, you can use the Actions button to start the collecting of data, the archiving of the statistics, or both.

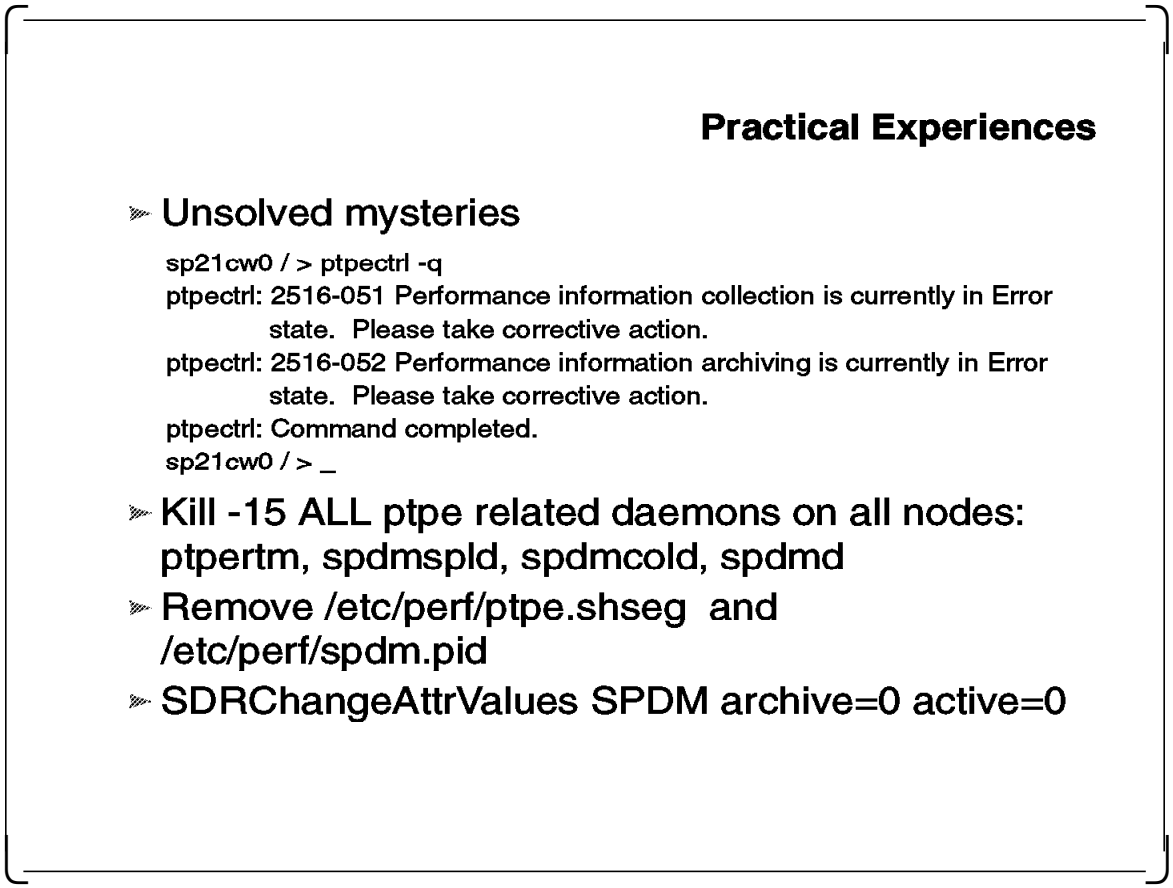


The **Window** button contains the actions for saving the configuration into the SDR.

Objects in the respective panes can be presented in several ways. The **View** button allows actions to change the objects' appearance in the panes. Window styles, like colors and fonts, can now be selected. This is not possible from the Tcl-based spmon application. These window styles can be saved and retrieved later, when the perfmon Perspectives application is invoked again.

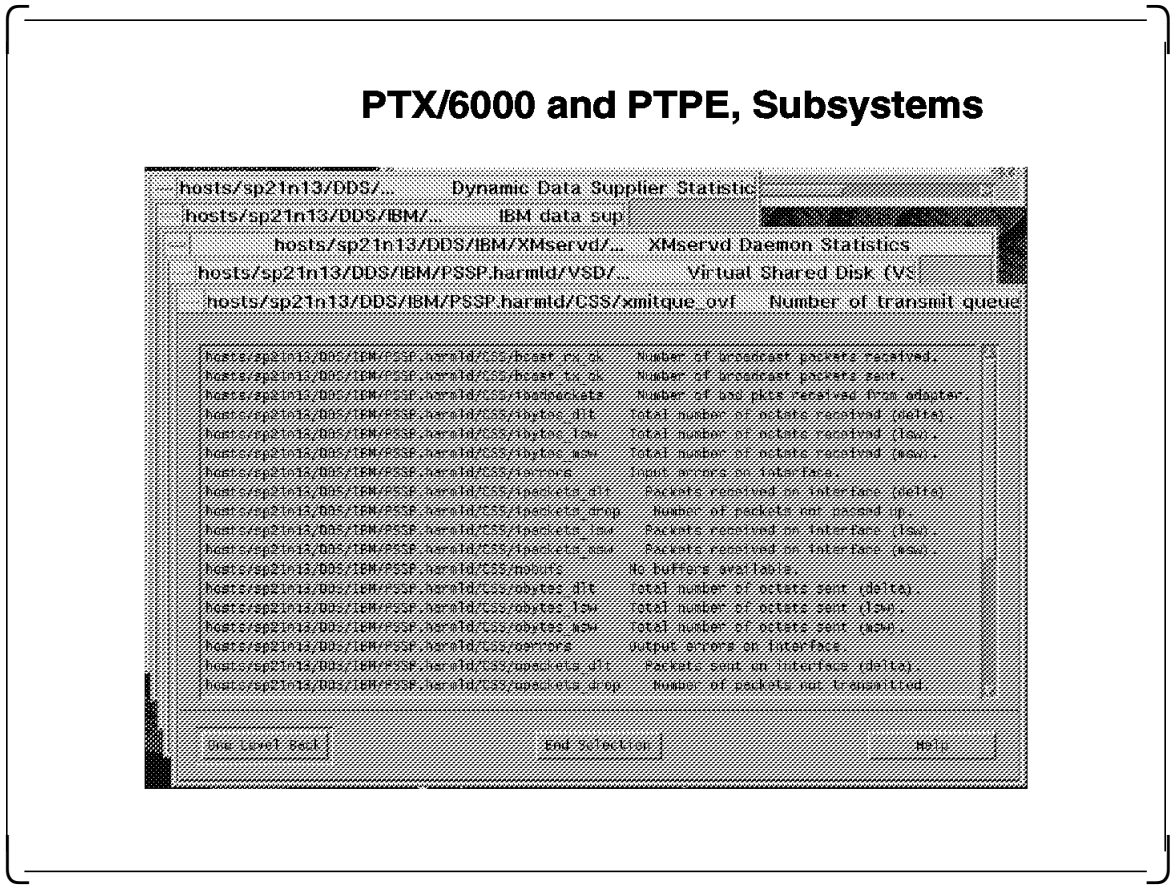
Finally, as expected from all X/Motif applications, extensive help is available through the **Help** button.

8.9 Practical Experiences



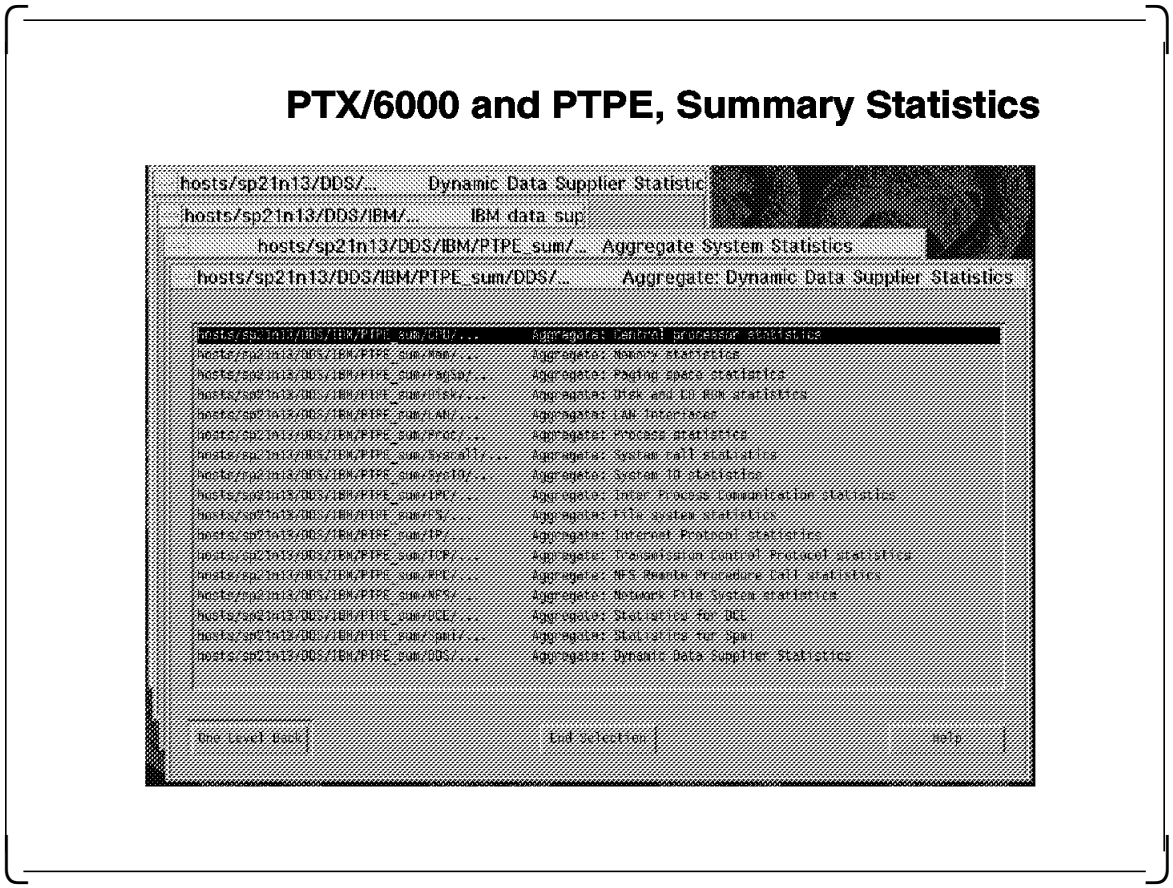
If for any reason, a situation like the one depicted in the foil occurs, the solution shown is the best way to recover. Other solutions may lead to inconsistencies in the data or configuration.

8.10 PTX/6000 and PTPE, Monitoring Subsystems



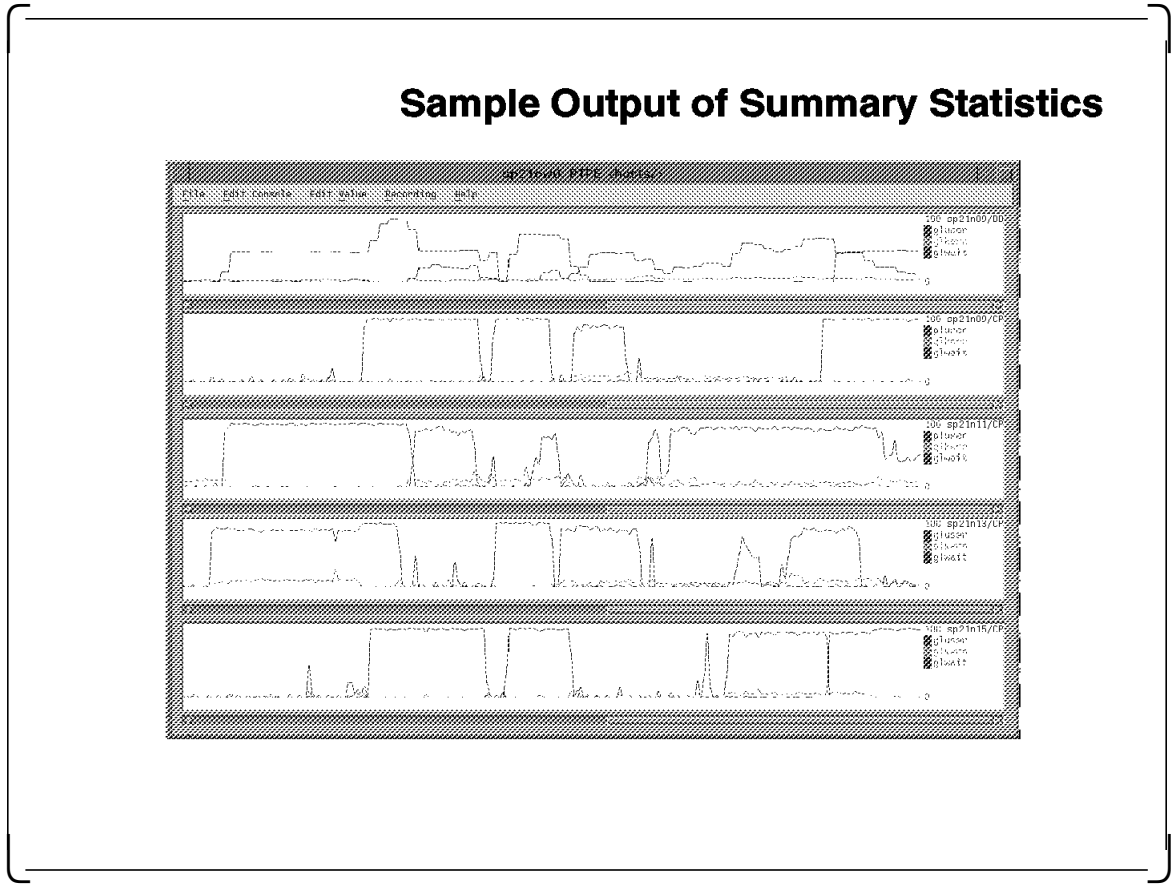
This figure shows the path an operator has to follow from the Performance Manager software to add statistics to a remote or local console while monitoring RS/6000 SP subsystem statistics.

8.11 PTX/6000 and PTPE, Summary Statistics



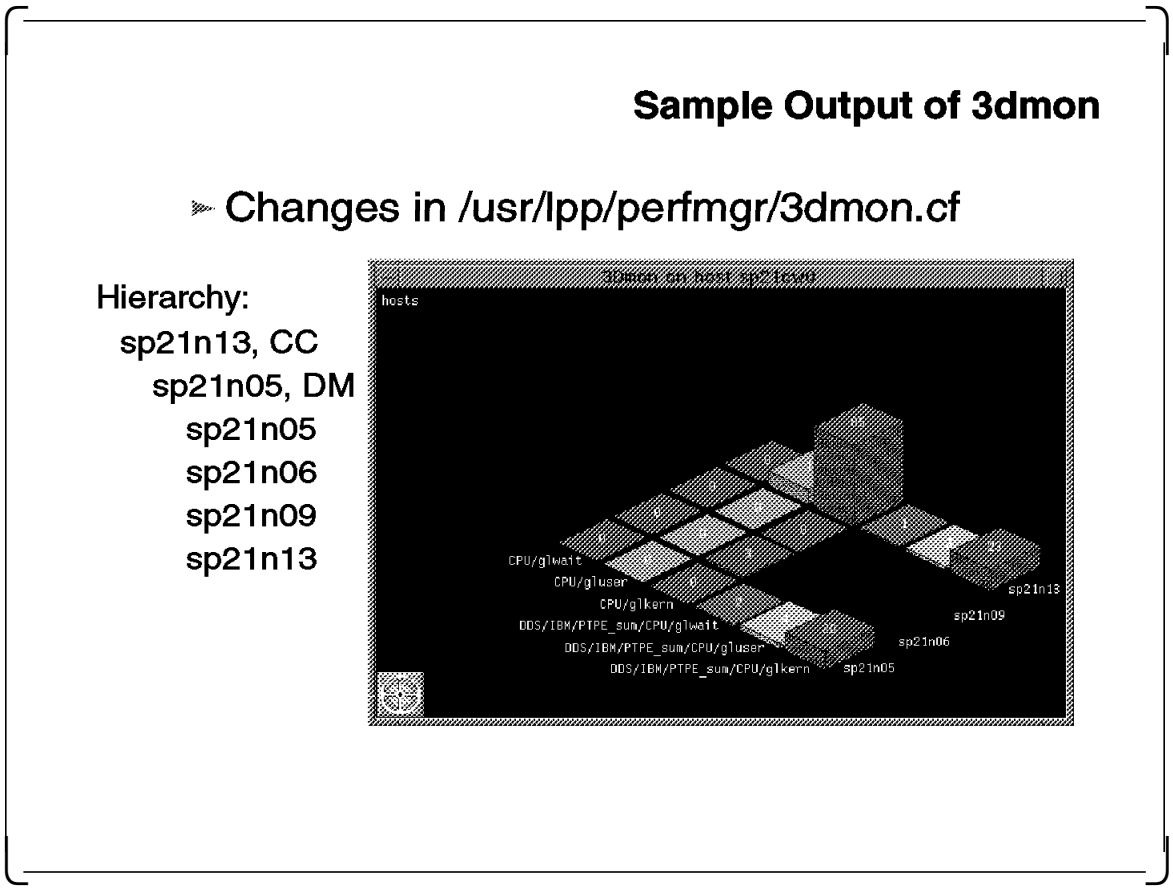
This figure shows the path that can be followed when adding PTPE summary statistics to a local or remote console.

8.11.1 Sample Output of Summary Statistics



This figure shows the CPU statistics output of four nodes and their Data Manager. The CPU statistics measured are CPU/wait, CPU/kernel, and CPU/user. The top node represents the Data Manager, showing the summary statistics of the four node consoles below the Data Manager. The top console provides more information about the four node cluster in terms of overall usage and efficiency than one could retrieve by investigating the node consoles individually.

8.11.2 Sample Output of 3dmon



This output shows an example similar to the one in 8.11.1, “Sample Output of Summary Statistics” on page 287, now displayed from the 3dmon program. In order to make 3dmon understand the statistics provided by the DDS protocol, the /usr/lpp/perfmgr/3dmon.cf file needs to be updated. Depending on the host-set you choose to display, you must apply changes to the selected section in the 3dmon.cf file, for example, in the small host-set or the large host-set. In this example, the small host-set was changed so that it only monitors and displays the CPU and summary CPU statistics. The 3dmon file looks similar to the following:

```
wildcard:    hosts
CPU/glkern
CPU/gluser
CPU/glwait
DDS/IBM/PTPE_sum/CPU/glkern
DDS/IBM/PTPE_sum/CPU/gluser
DDS/IBM/PTPE_sum/CPU/glwait
```

Of course, as mentioned before, if the PTPE_sum statistics are not found on a node, the ptpertm program will set the value for provide to 0, and thus does not show on the 3dmon display.

Appendix A. Resource Class Definition

This appendix contains a list of the Event Management Resource Classes that are defined in the SDR as EM_Resouce_Class.

:

Table 4. Resource Class Definition.. This table lists the Event Management Resource Classes that are defined in the default EM_Resource_Class SDR 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	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

Appendix B. ibmSP MIB

This appendix contains specific object types that are supplied for the ibmSP MIB.

```
-- Licensed Materials - Property of IBM
--
-- 5765-529
--
-- (C) Copyright IBM Corp. 1996 All Rights Reserved.
--
-- US Government Users Restricted Rights - Use, duplication or disclosure
-- restricted by GSA ADP Schedule Contract with IBM Corp.
--
-- Script Name:  ibmSPMIB.my
--
-- Description:
--      definition of the ibmSP mib.
--
-- "@(#)49   1.13   src/ssp/snmp_proxy/ibmSPMIB.my, probmgmt, ssp_rloc,
--           rloct7d6 9/6/96 11:15:01"

IBMSP-MIB  DEFINITIONS ::= BEGIN

IMPORTS
        Counter, Gauge, TimeTicks, IpAddress, DisplayString, enterprises
           FROM RFC1155-SMI
     TRAP-TYPE
          FROM RFC1215;

ibm             OBJECT IDENTIFIER ::= { enterprises 2 }

ibmProd         OBJECT IDENTIFIER ::= { ibm 6 }

ibmSP           OBJECT IDENTIFIER ::= { ibmProd 117 }

-----------------------------------------------------------------
--

--              IBM SP MIB

--

ibmSPConfig     OBJECT IDENTIFIER ::= { ibmSP 1 }

ibmSPhostnodenumber OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
           "A number from 0 to the total number of slots.  It identifies
       the relative node number assigned to this processor."
     ::= { ibmSPConfig 1 }

ibmSPhostpartaddr OBJECT-TYPE
     SYNTAX IpAddress
     ACCESS  read-only
     STATUS mandatory
```

```
      DESCRIPTION
       "ip address assigned to the system partition in which this host
        resides. If this host is acting as a control workstation (i.e.
        the value of ibmSPhostnodenumber.0 is 0), this will be the ip
        address of the default partition."
      ::= { ibmSPConfig 2 }

ibmSPCWScodeversion OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The ssp release that is installed on the control workstation.
        Values are of the form 2.0, 2.1, etc. A NULL value means the
        release is not known."
      ::= { ibmSPConfig 3 }

ibmSPprimaryCWSname OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "hostname of the primary control workstation."
      ::= { ibmSPConfig 4 }

ibmSPprimaryCWSoperstatus OBJECT-TYPE
      SYNTAX INTEGER {
          up(1),
          down(2)
      }
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "operational state of the primary control workstation."
      ::= { ibmSPConfig 5 }

ibmSPbackupCWSname OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "hostname of the backup control workstation."
      ::= { ibmSPConfig 6 }

ibmSPbackupCWSoperstatus OBJECT-TYPE
      SYNTAX INTEGER {
          up(1),
          down(2)
      }
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "operational state of the backup control workstation."
      ::= { ibmSPConfig 7 }

ibmSPSystemTable OBJECT-TYPE
      SYNTAX SEQUENCE OF IbmSPNodeEntry
      ACCESS  not-accessible
      STATUS mandatory
```

```
     DESCRIPTION
      "A list of SPNodeEntrys."
     ::= { ibmSPConfig 8 }

ibmSPNodeEntry OBJECT-TYPE
     SYNTAX IbmSPNodeEntry
     ACCESS  not-accessible
     STATUS mandatory
     DESCRIPTION
      "Identifies a processor node residing in an SP frame."
     INDEX { ibmSPpartitionaddr, ibmSPnodenumber }
     ::= { ibmSPSystemTable 1 }

IbmSPNodeEntry ::=
     SEQUENCE {
          ibmSPpartitionaddr
                     IpAddress,
          ibmSPnodenumber
                     INTEGER,
          ibmSPframenumber
                     INTEGER,
               ibmSPslotnumber
                     INTEGER,
               ibmSPslotsused
                          INTEGER,
          ibmSPinitialhostname
               DisplayString,
          ibmSPreliablehostname
               DisplayString,
          ibmSPsysparname
               DisplayString,
          ibmSPcodeversion
               DisplayString
          }

ibmSPpartitionaddr OBJECT-TYPE
     SYNTAX IpAddress
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "ip address assigned to the partition in which this node resides."
     ::= { ibmSPNodeEntry 1 }

ibmSPnodenumber OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
           "A number from 1 to the total number of slots.  It identifies
           the relative node number assigned to the processor."
     ::= { ibmSPNodeEntry 2 }

ibmSPframenumber OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
           "A number from 1 to the number of frames.  It identifies
           the number of the frame in which the processor node resides."
```

```
     ::= { ibmSPNodeEntry 3 }

ibmSPslotnumber OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
          "A number from 1 to the number of slots in a frame.  It
          dentifies the number of the first slot occupied by the
          processor node within the frame."
     ::= { ibmSPNodeEntry 4 }

ibmSPslotsused OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
          "A number from 1 to two.  It identifies the number of slots
          occupied by the processor."
     ::= { ibmSPNodeEntry 5 }

ibmSPinitialhostname OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
          "The initial hostname assigned to the node during the SP customization
     phase."
     ::= { ibmSPNodeEntry 6 }

ibmSPreliablehostname OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
          "The hostname associated with the SP ethernet."
     ::= { ibmSPNodeEntry 7 }

ibmSPsysparname OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
       "The name of the system partition containing this processor
     node."
     ::= { ibmSPNodeEntry 8}

ibmSPcodeversion OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "The ssp release that is installed on the processor node.
       Values are of the form 2.0, 2.1, etc. A NULL value means the
       release is not known."
     ::= { ibmSPNodeEntry 9 }

    --  The format of traps containing errlog entries whose tmplates
    --  are defined with 'Alert=yes'.
```

```
--  ibmSPErrlogTrap TRAP-TYPE
--      ENTERPRISE ibmSP
--      VARIABLES  { ibmSPellabel, ibmSPelidentifier, ibmSPeldatetime,
--                   ibmSPelsequencenum, ibmSPelmachineid, ibmSPelnodeid,
--                   ibmSPelclass, ibmSPeltype, ibmSPelresource,
--                   ibmSPelrscclass, ibmSPelrsctype, ibmSPellocation,
--                   ibmSPelvpd, ibmSPetdescription, ibmSPetprobcauses,
--                   ibmSPetusercauses, ibmSPetuseraction, ibmSPetinstcauses,
--                   ibmSPetinstaction, ibmSPetfailcauses, ibmSPetfailaction,
--                   ibmSPetdetaildata}
--      DESCRIPTION
--         "These traps contain the contents of errlog entries formatted
--         into objects defining the contents of the errlog entry.
--         Since any single errlog entry does not contain all of the
--         fields defined in the collection of errlog templates, when a
--         object contains a null value, it will not be included in the
--         trap."
--    ::= ibmSPelidentifier.0

ibmSPErrlogVars OBJECT-TYPE
     SYNTAX SEQUENCE OF IbmSPErrlogEntry
     ACCESS  not-accessible
     STATUS mandatory
     DESCRIPTION
      "A single SPNodeErrlogEntry."
     ::= { ibmSP 2 }

IbmSPErrlogEntry ::=
     SEQUENCE {
          ibmSPellabel
                     DisplayString,
          ibmSPelidentifier
                     DisplayString,
          ibmSPeldatetime
                     DisplayString,
               ibmSPelsequencenum
                     DisplayString,
          ibmSPelmachineid
                     DisplayString,
          ibmSPelnodeid
                     DisplayString,
          ibmSPelclass
                     DisplayString,
          ibmSPeltype
                     DisplayString,
          ibmSPelresource
                     DisplayString,
          ibmSPelrscclass
                     DisplayString,
          ibmSPelrsctype
                     DisplayString,
          ibmSPellocation
                     DisplayString,
          ibmSPelvpd
                     DisplayString,
          ibmSPetdescription
                     DisplayString,
          ibmSPetprobcauses
```

```
                    DisplayString,
          ibmSPetusercauses
                    DisplayString,
          ibmSPetuseraction
                    DisplayString,
          ibmSPetinstcauses
                    DisplayString,
          ibmSPetinstaction
                    DisplayString,
          ibmSPetfailcauses
                    DisplayString,
          ibmSPetfailaction
                    DisplayString,
          ibmSPetdetaildata
                    DisplayString
          }

ibmSPellabel OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "The label associated with the error identifier defined by
       the ibmSPelidentifier object."
     ::= { ibmSPErrlogVars 1 }

ibmSPelidentifier OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "A unique identifier defining the type of error entry written
       to the system error log existing on the host from which the
       trap originated."
     ::= { ibmSPErrlogVars 2 }

ibmSPeldatetime OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "A timestamp identifying the time that this error log entry
       was written to the system error log.  It is of the form:
       day   month   day_of_month   hour:min:sec"
     ::= { ibmSPErrlogVars 3 }

ibmSPelsequencenum OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "A decimal number which is the sequence number assigned to this
       entry.  This is the value specified on the -l switch of the
       errpt command used to obtain the trap data on the host from
       which the trap originated."
     ::= { ibmSPErrlogVars 4 }

ibmSPelmachineid OBJECT-TYPE
     SYNTAX DisplayString
```

```
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "A decimal number which is the machine ID of the host on which
        the trap originated. This is the value returned by the AIX
        'uname -m' command when issued on the host from which the trap
         originated."
      ::= { ibmSPErrlogVars 5 }

ibmSPelnodeid OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "An alpha-numeric string which is the node ID of the host on
        which the trap originated.  This is the value returned by the AIX
        'uname -m' command when issued on the host from which the trap
         originated."
      ::= { ibmSPErrlogVars 6 }

ibmSPelclass OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "An alphabetic string which is the error class of this entry:
        H (hardware), S (software), O (errlogger command messages)."
      ::= { ibmSPErrlogVars 7 }

ibmSPeltype OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "An alphabetic string which is the severity of the error entry:
        'PEND' (impending loss of availability), 'PERF' (unacceptable
        performance degradation), 'PERM' (permanent), 'TEMP' (temporary)
        , 'UNKN' (unknown)."
      ::= { ibmSPErrlogVars 8 }

ibmSPelresource OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The resource name associated with the error. For hardware
        errors this is a device name, for software errors this is
        the name of the failing executable, for operator command
        messages this is 'OPERATOR'."

      ::= { ibmSPErrlogVars 9 }

ibmSPelrscclass OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The resource class associated with the error. For hardware
        errors this is a device class (or 'NONE')."
```

```
     ::= { ibmSPErrlogVars 10 }

ibmSPelrsctype OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "The resource type associated with the error.  For hardware
       errors this is a device type (or 'NONE'),  for software errors
       (when specified) this is an LPP."
     ::= { ibmSPErrlogVars 11 }

ibmSPellocation OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "For hardware errors, information about the location of the
       failing device (or 'NONE')."
     ::= { ibmSPErrlogVars 12 }

ibmSPelvpd OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "IBM or user supplied vital product data."
     ::= { ibmSPErrlogVars 13}

ibmSPetdescription OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "Error description."
     ::= { ibmSPErrlogVars 14}

ibmSPetprobcauses OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "Probable causes of the error."
     ::= { ibmSPErrlogVars 15 }

ibmSPetusercauses OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "User actions which may have caused the error."
     ::= { ibmSPErrlogVars 16 }

ibmSPetuseraction OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "Recommended actions the user may take to correct the error."
```

```
     ::= { ibmSPErrlogVars 17 }

ibmSPetinstcauses OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "Installation causes of the error."
     ::= { ibmSPErrlogVars 18 }

ibmSPetinstaction OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "User actions which may have been performed during installation
       to cause the error."
     ::= { ibmSPErrlogVars 19 }

ibmSPetfailcauses OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "A list of candidates which may be the source of the error."
     ::= { ibmSPErrlogVars 20 }

ibmSPetfailaction OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "A list of reccommended actions which may be taken to correct
       the possible failures."
     ::= { ibmSPErrlogVars 21 }

ibmSPeldetaildata OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "Detailed data about this particular error."
     ::= { ibmSPErrlogVars 22 }

ibmSPEMVariables OBJECT IDENTIFIER ::= { ibmSP 3 }

    --  The format of traps containing errlog entries whose tmplates
    --  are defined with 'Alert=yes'.

--  ibmSPEMEventTrap TRAP-TYPE
--      ENTERPRISE ibmSP
--      VARIABLES  { ibmSPEMEventID, ibmSPEMEventFlags, ibmSPEMEventTime,
--                       ibmSPEMEventLocation, ibmSPEMEventPartitionAddress,
--                       ibmSPEMEventVarsTableName, ibmSPEMEventVarsTableInstanceID,
--                   ibmSPEMEventVarName, ibmSPEMEventVarValueInstanceVector,
--              ibmSPEMEventVarValuesTableInstanceID,
--              ibmSPEMEventVarValue, ibmSPEMEventPredicate
--           }
--      DESCRIPTION
```

```
--         "These traps contain the contents of events generated from the
--         PSSP Event Manager.  The events have been formatted
--         into objects defining the contents of the event and where
--         the variable pertaining to the event and its value are located
--         in the ibmSP mib.
--    ::= ibmSPEMEventID.0

ibmSPEMEvent OBJECT-TYPE
      SYNTAX SEQUENCE OF IbmSPEMEventEntry
      ACCESS  not-accessible
      STATUS mandatory
      DESCRIPTION
       "A single IbmSPEMEventEntry."
      ::= { ibmSPEMVariables 1 }

IbmSPEMEventEntry ::=
      SEQUENCE {
           ibmSPEMEventID
                      INTEGER,
           ibmSPEMEventFlags
                      INTEGER,
           ibmSPEMEventTime
                      TimeTicks,
                ibmSPEMEventLocation
                      INTEGER,
           ibmSPEMEventPartitionAddress
                      IpAddress,
           ibmSPEMEventVarsTableName
                      DisplayString,
           ibmSPEMEventVarsTableInstanceID
                      DisplayString,
           ibmSPEMEventVarName
                      DisplayString,
           ibmSPEMEventVarValueInstanceVector
                      DisplayString,
           ibmSPEMEventVarValuesTableInstanceID
                      DisplayString,
           ibmSPEMEventVarValue
                      DisplayString,
           ibmSPEMEventPredicate
                      DisplayString
           }

ibmSPEMEventID OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The trap identifier assigned to the occurence of the Event."
      ::= { ibmSPEMEvent 1}

ibmSPEMEventFlags OBJECT-TYPE
      SYNTAX INTEGER {
          re-arm(1),
          false-predicate(2),
          unregister-event(4)
      }
      ACCESS  read-only
      STATUS mandatory
```

```
      DESCRIPTION
       "EM Flags."
      ::= { ibmSPEMEvent 2}

ibmSPEMEventTime OBJECT-TYPE
      SYNTAX TimeTicks
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "Elapsed time between the activation of the SP proxy sub-agent
        and the occurence of the event."
      ::= { ibmSPEMEvent 3}

ibmSPEMEventLocation OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The number of the SP node from which the event was generated."
      ::= { ibmSPEMEvent 4}

ibmSPEMEventPartitionAddress OBJECT-TYPE
      SYNTAX IpAddress
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The ip address of the SP partition from which the event was
        generated."
      ::= { ibmSPEMEvent 5}

ibmSPEMEventVarsTableName OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name of the table in the SP MIB in which contains the variable
       definition objects."
      ::= { ibmSPEMEvent 6}

ibmSPEMEventVarsTableInstanceID OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The instance identifier used to locate the row in the table named
       by the ibmSPEMEventVarsTableName object value which contains
       further information about the EM Event variable definition."
      ::= { ibmSPEMEvent 7}

ibmSPEMEventVarName OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name of the EM variable  about which the event is recorded."
      ::= { ibmSPEMEvent 8}

ibmSPEMEventVarValueInstanceVector OBJECT-TYPE
      SYNTAX DisplayString
```

```
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The instantiation vector of the EM variable instance that resulted
        in the event."
      ::= { ibmSPEMEvent 9}

ibmSPEMEventVarValuesTableInstanceID OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The instance identifier used to locate the row in the
       ibmSPEMVarValuesTable identifying for which instantiation of the named
       EM variable the value is reported. This is used to obtain the current
       variable value in the table to see if it has changed since the time
       the trap was issued.

       When the value of the ibmSPEMEventLocation object does not match
       the number of the node on which the agent generating the trap is
       running, the value for this object will be null. In this case, the node
       whose node number is in the ibmSPEMEventLocation object must be
       queried to obtain the current value of the variable instance named
       by objects ibmSPEMEventVarName and ibmSPEMEventVarValueInstanceVector.
       This is accomplished by first obtaining the value of the index into
       the ibmSPEMVarValuesTable from the table named in the
       ibmSPEMEventVarsTableName object using the instance value from the
       ibmSPEMEventVarsTableInstanceID. If the value of the
       ibmSPEMEventVarsTableName object is ibmSPEMNodeDepVarsTable', the
       ibmSPEMNodeDepVarCurValueIndex object has the index number.  If the
       value of the ibmSPEMEventVarsTableName is 'ibmSPEMNodeIndepVarsTable',
       the ibmSPEMNodeIndepVarCurValueIndex object has the index number.
       The derived index number is then encoded as an instance, followed by
       the encoded value of the ibmSPEMEventVarValueInstanceVector to obtain
       the current value of the ibmSPEMVarValue object."
      ::= { ibmSPEMEvent 10}

ibmSPEMEventVarValue OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The value of the variable instance at the time of the event."
      ::= { ibmSPEMEvent 11}

ibmSPEMEventPredicate OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The predicate string which caused the event."
      ::= { ibmSPEMEvent 12}

ibmSPEMNodeDepVarsTable OBJECT-TYPE
      SYNTAX SEQUENCE OF IbmSPEMNodeDepVarEntry
      ACCESS  not-accessible
      STATUS mandatory
      DESCRIPTION
       "A table of node-dependent EM Variable attributes. Variables
```

```
        in this table are only instantiated on the node containing the
        resource monitor. See the ibmSPEMNodeDepVarLocator object
        description."
      ::= { ibmSPEMVariables 2 }

ibmSPEMNodeDepVarEntry OBJECT-TYPE
      SYNTAX IbmSPEMNodeDepVarEntry
      ACCESS  not-accessible
      STATUS mandatory
      DESCRIPTION
       "The attributes of an event manager variable.  Each octet of
        the variable-length index string is encoded in a separate
        sub-identifier. The number of the sub-identifiers in the index
        is IMPLIED."
      INDEX { ibmSPEMNodeDepVarName }
      ::= { ibmSPEMNodeDepVarsTable 1 }

IbmSPEMNodeDepVarEntry ::=
      SEQUENCE {
            ibmSPEMNodeDepVarName
                  DisplayString,
            ibmSPEMNodeDepVarDescr
                  DisplayString,
            ibmSPEMNodeDepVarType
                  DisplayString,
            ibmSPEMNodeDepVarDataType
                  DisplayString,
            ibmSPEMNodeDepVarSBSFormat
                  DisplayString,
            ibmSPEMNodeDepVarInitValue
                  DisplayString,
            ibmSPEMNodeDepVarCurValueIndex
                  INTEGER,
            ibmSPEMNodeDepVarClass
                  DisplayString,
            ibmSPEMNodeDepVarVecElDefn
                  DisplayString,
            ibmSPEMNodeDepVarVecElDescr
                  DisplayString,
            ibmSPEMNodeDepVarPTXName
                  DisplayString,
            ibmSPEMNodeDepVarDefPred
                  DisplayString,
            ibmSPEMNodeDepVarEventDescr
                  DisplayString,
            ibmSPEMNodeDepVarLocator
                  DisplayString,
            ibmSPEMNodeDepVarOrderGroup
                  DisplayString
            }

ibmSPEMNodeDepVarName OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "A resource variable name as defined to the Event Manager."
      ::= { ibmSPEMNodeDepVarEntry 1 }
```

```
ibmSPEMNodeDepVarDescr OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "A description of the variable, including its semantics."
      ::= { ibmSPEMNodeDepVarEntry 2 }

ibmSPEMNodeDepVarType OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "One of the strings Counter, Quantity or State."
      ::= { ibmSPEMNodeDepVarEntry 3 }

ibmSPEMNodeDepVarDataType OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "One of the strings long, float or SBS. SBS is only permitted
        for a variable of the type State. If the value of this
        object is SBS, then the definitions of the structured fields
        that comprise the structured byte string contained in the
        ibmSPEMNodeDepVarSBSFormat object."
      ::= { ibmSPEMNodeDepVarEntry 4 }

ibmSPEMNodeDepVarSBSFormat OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "If the ibmSPEMNodeDepVarDataType object with the corresponding
        instance id has a value of SBS, then this object describes the
        structured fields within the variable. Included for each structured
        field is the structured field name, followed by an equal sign,
        followed by the data type for the field. The structured fields
        are defined sequentially beginning with sequence number 0."
      ::= { ibmSPEMNodeDepVarEntry 5 }

ibmSPEMNodeDepVarInitValue OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The initial value of a resource variable before it is observed
        for the first time."
      ::= { ibmSPEMNodeDepVarEntry 6 }

ibmSPEMNodeDepVarCurValueIndex OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "An index into the ibmSPEMVarValuesTable table to locate value
        instances for this variable."
      ::= { ibmSPEMNodeDepVarEntry 7 }
```

```
ibmSPEMNodeDepVarClass OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name of the class to which this resource variable belongs."
      ::= { ibmSPEMNodeDepVarEntry 8 }

ibmSPEMNodeDepVarVecElDefn OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name(s) of instantiation vector elements associated with the
        resource variable."
      ::= { ibmSPEMNodeDepVarEntry 9 }

ibmSPEMNodeDepVarVecElDescr OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "A description of the instantiation vector elements associated with
        the resource variable."
      ::= { ibmSPEMNodeDepVarEntry 10 }

ibmSPEMNodeDepVarPTXName OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name used to read and write the resource variable in the PTX
        shared memory."
      ::= { ibmSPEMNodeDepVarEntry 11 }

ibmSPEMNodeDepVarDefPred OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The default predicate to be applied to the resource variable."
      ::= { ibmSPEMNodeDepVarEntry 12 }

ibmSPEMNodeDepVarEventDescr OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "A description of the event generated by the application of the
        default predicate."
      ::= { ibmSPEMNodeDepVarEntry 13 }

ibmSPEMNodeDepVarLocator OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name of the vector element whose value is the number of the
        node containing the variable instance."
```

```
      ::= { ibmSPEMNodeDepVarEntry 14 }

ibmSPEMNodeDepVarOrderGroup OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name of an Availability group. All events generated by the
        default predicates of resource variables within this order group
        name are guaranteed ordered delivery with respect to one
        another."
      ::= { ibmSPEMNodeDepVarEntry 15 }

ibmSPEMNodeIndepVarsTable OBJECT-TYPE
      SYNTAX SEQUENCE OF IbmSPEMNodeIndepVarEntry
      ACCESS  not-accessible
      STATUS mandatory
      DESCRIPTION
       "A table of node-independent EM Variable attributes.  No EM Locator
        element is defined for these variables so the location of the
        resource monitor is unknown within the partition. Variables
        in this table are only instantiated on the Control Work Station.
        See the ibmSPEMNodeDepVarLocator object description."
      ::= { ibmSPEMVariables 3 }

ibmSPEMNodeIndepVarEntry OBJECT-TYPE
      SYNTAX IbmSPEMNodeIndepVarEntry
      ACCESS  not-accessible
      STATUS mandatory
      DESCRIPTION
       "The attributes of an event manager variable.  Each octet of
        the variable-length index string is encoded in a separate
        sub-identifier. The number of the sub-identifiers in the index
        is IMPLIED."
      INDEX { ibmSPEMNodeIndepPartaddr, ibmSPEMNodeIndepVarName }
      ::= { ibmSPEMNodeIndepVarsTable 1 }

IbmSPEMNodeIndepVarEntry ::=
      SEQUENCE {
           ibmSPEMNodeIndepPartaddr
                IpAddress,
           ibmSPEMNodeIndepVarName
                DisplayString,
           ibmSPEMNodeIndepVarDescr
                DisplayString,
           ibmSPEMNodeIndepVarType
                DisplayString,
           ibmSPEMNodeIndepVarDataType
                DisplayString,
           ibmSPEMNodeIndepVarSBSFormat
                DisplayString,
           ibmSPEMNodeIndepVarInitValue
                DisplayString,
           ibmSPEMNodeIndepVarCurValueIndex
                INTEGER,
           ibmSPEMNodeIndepVarClass
                DisplayString,
           ibmSPEMNodeIndepVarVecElDefn
                DisplayString,
```

```
          ibmSPEMNodeIndepVarVecElDescr
               DisplayString,
          ibmSPEMNodeIndepVarPTXName
               DisplayString,
          ibmSPEMNodeIndepVarDefPred
               DisplayString,
          ibmSPEMNodeIndepVarEventDescr
               DisplayString,
          ibmSPEMNodeIndepVarOrderGroup
               DisplayString
          }

ibmSPEMNodeIndepPartaddr OBJECT-TYPE
     SYNTAX IpAddress
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "ip address assigned to the system partition in which the
       this resource variable resides."
     ::= { ibmSPEMNodeIndepVarEntry 1 }

ibmSPEMNodeIndepVarName OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "A resource variable name as defined to the Event Manager."
     ::= { ibmSPEMNodeIndepVarEntry 2 }

ibmSPEMNodeIndepVarDescr OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "A description of the variable, including its semantics."
     ::= { ibmSPEMNodeIndepVarEntry 3 }

ibmSPEMNodeIndepVarType OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "One of the strings Counter, Quantity or State."
     ::= { ibmSPEMNodeIndepVarEntry 4 }

ibmSPEMNodeIndepVarDataType OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "One of the strings long, float or SBS. SBS is only permitted
       for a variable of the type State. If the value of this
       object is SBS, then the definitions of the structured fields
       that comprise the structured byte string contained in the
       ibmSPEMNodeIndepVarSBSFormat object."
     ::= { ibmSPEMNodeIndepVarEntry 5 }

ibmSPEMNodeIndepVarSBSFormat OBJECT-TYPE
     SYNTAX DisplayString
```

```
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "If the ibmSPEMNodeIndepVarDataType object with the corresponding
        instance id has a value of SBS, then this object describes the
        structured fields within the variable. Included for each structured
        field is the structured field name, followed by an equal sign,
        followed by the data type for the field. The structured fields
        are defined sequentially beginning with sequence number 0."
      ::= { ibmSPEMNodeIndepVarEntry 6 }

ibmSPEMNodeIndepVarInitValue OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The initial value of a resource variable before it is observed
        for the first time."
      ::= { ibmSPEMNodeIndepVarEntry 7 }

ibmSPEMNodeIndepVarCurValueIndex OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "An index into the ibmSPEMVarValuesTable table to locate value
        instances for this variable."
      ::= { ibmSPEMNodeIndepVarEntry 8 }

ibmSPEMNodeIndepVarClass OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name of the class to which this resource variable belongs."
      ::= { ibmSPEMNodeIndepVarEntry 9 }

ibmSPEMNodeIndepVarVecElDefn OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name(s) of instantiation vector elements associated with the
        resource variable."
      ::= { ibmSPEMNodeIndepVarEntry 10 }

ibmSPEMNodeIndepVarVecElDescr OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "A description of the instantiation vector elements associated with
        the resource variable."
      ::= { ibmSPEMNodeIndepVarEntry 11 }

ibmSPEMNodeIndepVarPTXName OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
```

```
      DESCRIPTION
       "The name used to read and write the resource variable in the PTX
        shared memory."
      ::= { ibmSPEMNodeIndepVarEntry 12 }

ibmSPEMNodeIndepVarDefPred OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The default predicate to be applied to the resource variable."
      ::= { ibmSPEMNodeIndepVarEntry 13 }

ibmSPEMNodeIndepVarEventDescr OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "A description of the event generated by the application of the
        default predicate."
      ::= { ibmSPEMNodeIndepVarEntry 14 }

ibmSPEMNodeIndepVarOrderGroup OBJECT-TYPE
      SYNTAX DisplayString
      ACCESS  read-only
      STATUS mandatory
      DESCRIPTION
       "The name of an Availability group. All events generated by the
        default predicates of resource variables within this order group
        name are guaranteed ordered delivery with respect to one
        another."
      ::= { ibmSPEMNodeIndepVarEntry 15 }

ibmSPEMVarValuesTable OBJECT-TYPE
      SYNTAX SEQUENCE OF IbmSPEMVarValuesEntry
      ACCESS  not-accessible
      STATUS mandatory
      DESCRIPTION
       "A table of current EM Variable values. Variables."
      ::= { ibmSPEMVariables 4 }

ibmSPEMVarValuesEntry OBJECT-TYPE
      SYNTAX IbmSPEMVarValuesEntry
      ACCESS  not-accessible
      STATUS mandatory
      DESCRIPTION
       "The current value of an event manager variable instantiation.
        The value of the ibmSPEMVarValueIndex is assigned when the
        SP sub-agent (sp_configd) is initialized; it is contained in the
        ibmSPEMNodeDepVarCurValueIndex object within the ibmSPNodeDepVarsTable
        (if the variable is node-dependent) or in the
        ibmSPEMNodeIndepVarCurValueIndex object within the
        ibmSPEMNodeIndepVarTable (if the variable is node-independent).

        Each octet of the variable-length ibmSPEMVarValueInstanceVector value
        string is encoded in a separate sub-identifier, preceded by its
        length which may be 0 if its value is null. A 0 length indicates
        the function represented by the EM variable is not being monitored."
      INDEX { ibmSPEMVarValueIndex, ibmSPEMVarValueInstanceVector }
```

```
     ::= { ibmSPEMVarValuesTable 1 }

IbmSPEMVarValuesEntry ::=
     SEQUENCE {
          ibmSPEMVarValueIndex
               INTEGER,
          ibmSPEMVarValueInstanceVector
               DisplayString,
          ibmSPEMVarValuePartaddr
               IpAddress,
          ibmSPEMVarValueName
               DisplayString,
          ibmSPEMVarValue
               DisplayString
          }

ibmSPEMVarValueIndex OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "An index used as part of the instance id to identify the
       object instance containing the current value of an EM
       resource variable instance "
     ::= { ibmSPEMVarValuesEntry 1}

ibmSPEMVarValueInstanceVector OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "The instantiation vector elements associated with the
       resource variable."
     ::= { ibmSPEMVarValuesEntry 2}

ibmSPEMVarValuePartaddr OBJECT-TYPE
     SYNTAX IpAddress
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "ip address assigned to the system partition in which the
       this resource variable resides."
     ::= { ibmSPEMVarValuesEntry 3}

ibmSPEMVarValueName OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "A resource variable name as defined to the Event Manager."
     ::= { ibmSPEMVarValuesEntry 4}

ibmSPEMVarValue OBJECT-TYPE
     SYNTAX DisplayString
     ACCESS  read-only
     STATUS mandatory
     DESCRIPTION
      "The current value of a resource variable."
     ::= { ibmSPEMVarValuesEntry 5}
```

```
END
```

Appendix C. Special Notices

This publication is intended to help IBM customers, Business Partners, and IBM System Engineers and other RS/6000 SP specialists who are involved in Parallel System Support Programs (PSSP) Version 2 Release 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 Parallel System Support Programs. See the PUBLICATIONS section of the IBM Programming Announcement for PSSP Version 2 Release 2 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 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.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AIX/6000
AIXwindows	DB2/6000
HACMP/6000	IBM
LoadLeveler	NetView
POWER Architecture	Power PC 604
POWERparallel	PowerPC 604
POWER2 Architecture	RS/6000
Scalable POWERparallel Systems	SP
SP2	9076 SP2

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, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Other trademarks are trademarks of their respective companies.

Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 317.

- *PSSP Version 2 Technical Presentation*, SG24-4542
- *RS/6000 SMP Servers Architecture*, SG24-2583
- *RS/6000 SP PSSP 2.2 Technical Presentation*, SG24-4868

D.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
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

D.3 Other Publications

These publications are also relevant as further information sources:

- *PSSP Installation and Migration Guide*, GC23-3898
- *PSSP Diagnosis and Messages Guide*, GC23-3899
- *PSSP Command and Technical Reference*, GC23-3900
- *Group Services Programming Guide and Reference*, GC28-1675 (to be available by year end 1996)
- *PSSP System Planning Guide*, GC23-3902

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 WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  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://w3.itso.ibm.com/redbooks

- **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	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (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

- **Please put me on the mailing list for updated versions of the IBM Redbook Catalog.**

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	***TCP/IP***	Transmission Control Protocol / Internet Protocol
SRC	System Resource Controller	***UDP***	User Datagram Protocol
SSI	Single System Image	***VSD***	Virtual Shared Disk
TS	Topology Services	***VSM***	Visual System Management

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

X

Printed in U.S.A.