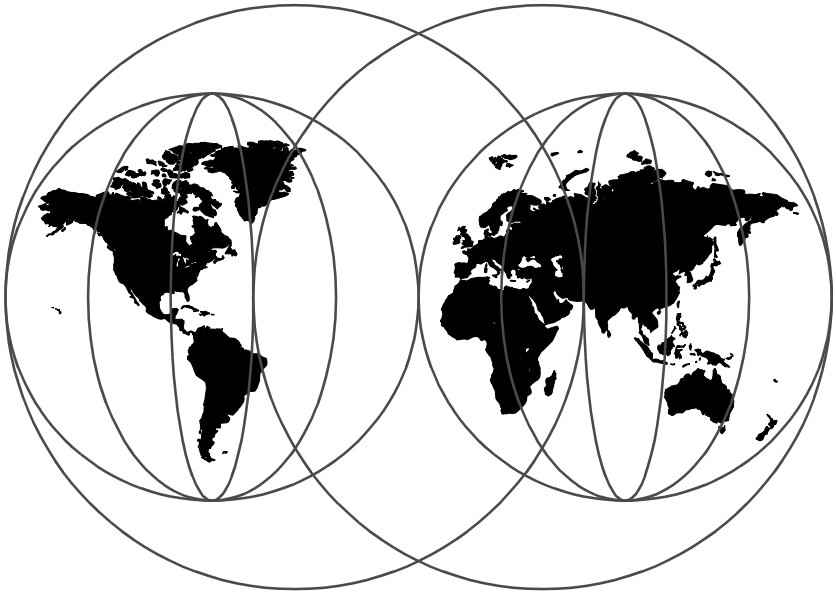
HACMP Enhanced Scalability: User-Defined Events

Yoshimichi Kosuge, John Easton



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5327-00

SG24-5327-00

International Technical Support Organization

HACMP Enhanced Scalability: User-Defined Events

November 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special Notices” on page 185.

First Edition (November 1998)

This edition applies to Version 4.2.1 and later of High Availability Cluster Multiprocessing Enhanced Scalability (HACMP ES) and Version 2.3 and later of Parallel System Support Programs (PSSP) for use with the AIX 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

Contents

Figures

Tables

Preface

This redbook describes the various methods in which High Availability Cluster Multi-Processing for AIX (HACMP for AIX) clusters can be extended to handle problem conditions other than those handled by the base product. The majority of these methods are enabled through the use of RS/6000 Cluster Technology (RSCT). System implementors and system administrators should use this book as a guide to defining, configuring, and maintaining user events within their system using AIX and IBM Parallel System Support Programs for AIX (PSSP) facilities and the IBM High Availability Cluster Multi-Processing for AIX Enhanced Scalability (HACMP/ES) product.

The redbook is of value to system administrators and system implementors who wish to understand and utilize RSCT in a HACMP/ES environment. Many practical examples are presented as implementations of the techniques described.

An introduction is given to the automatic operation of a HACMP for AIX cluster. The benefits of automated operations are looked at, and an introduction to the key concepts are described in detail.

The concept of an event and the issues involved with event monitoring are discussed, and the different actions that may be taken when an event occurs are described. In addition, the book assists in determining the best methodology for monitoring a given resource.

The AIX error logging mechanism is described along with the various methods of writing error log entries and how error notification may be used to trigger user events. This is illustrated with a series of examples and operational scenarios.

The Event Management subsystem and SP Perspectives are also discussed. The Event Management capabilities of Perspectives are examined, and the book describes how to configure the system by using the GUI to define important user events. User-defined events are the actions that should be triggered when the event occurs.

Furthermore, the integrated user event functions of the HACMP/ES product are introduced, and the configuration of the rules.hacmprd file and recovery programs are discussed.

Other utilities and functions that are useful within clustered environments for either detecting the occurrence of events or running recovery actions are discussed, and the book shows how Event Management can be used to

perform common monitoring tasks. As each task is introduced, the resource variables that are used for monitoring are explained and examples of their use are given.

Common actions that may be taken when an event occurs are provided, and guidance is given for using these actions.

Finally, the redbook describes some of the system components that may run on an AIX system and provides examples of the events that should be monitored for and sample recovery actions, should these events occur.

The Team That Wrote This Redbook

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

Yoshimichi Kosuge is an IBM RS/6000 SP project leader at the International Technical Support Organization, Poughkeepsie Center. He joined IBM Japan in 1982 and has worked in the following areas: LSI design, S/390 CP microcode, VM, MVS, OS/2, and AIX. Since joining the ITSO in 1998, he has been involved in writing redbooks and teaching IBM classes worldwide on all areas of RS/6000 SP.

John Easton has worked for IBM for 12 years in a variety of UNIX technical roles. He worked in Austin during the development of the RISC System/6000 and AIX Version 3 and holds several computer security patents. Since 1991, he has focussed on high availability and clustered systems and is currently Technical Consultant for the IBM High Availability Cluster Competency Centre, based in the UK. He has developed parts of the IBM HACMP for AIX product and is part of the team responsible for shaping IBM clustering and high availability strategies and directions.

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

Mike Coffey
Simon Marchese
Chris Owen
Mike Schmidt
Dave Spurway
Tom Weaver

Comments Welcome

Your comments are important to us

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

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 203 to the fax number shown on the form.
- Use the electronic evaluation form found on the redbook Web sites:

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

- Send us a note at the following address:

  `redbook@us.ibm.com`

Part 1. Automated Operations of an HA Cluster

Chapter 1. High Availability (HA) Clusters in the Real World

This chapter provides an introduction to the automatic operation of a HACMP for AIX cluster. It looks at the benefits of automated operations and introduces the key concepts that are described in detail in the subsequent chapters.

1.1 System Downtime

Modern businesses rely upon the availability of their computer systems. It is becoming increasingly common that if the computer systems that run the business fail, then a company can only function properly for a short period of time, if at all. As a result, there is a great effort made to keep the systems available. The key to doing this is to reduce system downtime.

System downtime can be either planned or unplanned. *Planned downtime* is the time that a system is down for applying maintenance updates, taking backups, and so on. In most environments, planned downtime accounts for over 90% of the time a computer is unavailable.

Unplanned downtime is the time that a system is down because something unexpected has happened or because a component has failed; it accounts for the remaining downtime.

Whether planned or unplanned, the four most likely causes of system downtime, according to many industry surveys, are:

- Software failures
- Systems management activity
- Operator errors
- Hardware failures

A similar list created ten years ago would probably have placed hardware failures at the top of the list (or at least it certainly would be higher than its fourth place position today). This is because significant effort has been expended on the design and manufacture of modern hardware components, which makes them much more reliable than earlier versions. This trend looks like it will continue for the foreseeable future.

Software, on the other hand, has become much more complex. People now are trying to address more complex problems because the performance of the underlying hardware has improved to the point whereby it becomes feasible to attempt to address these problems. In addition, modern software

engineering techniques tend to produce software with much more function than was traditionally supplied.

The increasing use of code generators and 4GL environments also increases the volumes of code generated, and while they may significantly reduce the time taken to create new software functions, the level of redundant or unused code is very high. This adds dramatically to the complexity of the software and increases the likelihood of failure.

Just as the complexity and functionality of the application software has increased significantly, the complexity of the operating systems and other system software has also increased. Partly this is due to the increasing complexity and novel designs now being utilized in the hardware, but mostly it is due to operating systems having to provide more and more integrated functions.

A few years ago, communications or utility software would have been an additional cost add-on to the operating system. It is now included in the package. Whereas previously a system administrator required significant skill in the systems hardware, software now accounts for the vast majority of training that system administrators and operators require in order to carry out their jobs.

As systems become more complex, the time and skills needed to manage them also increase. Firstly, as the operating system and applications acquire more functionality, there are more tasks that need to be performed to keep them running. This takes up additional time that previously was not required because the functions did not exist. Secondly, as the complexity increases, there is more need for training in the new functionality or potentially. You could say there was more chance of making a mistake.

Software failures will continue to be the major cause of system downtime because of this increasing complexity. Allied with this will be an increasing amount of time required to perform systems management tasks. In an already crowded day, this increased pressure will ensure that mistakes and errors will continue to be made.

1.2 High Availability vs. Concurrent Operations

The HACMP software allows you to build a highly available environment that ensures that the critical applications at your site are available to your end users. However, high availability is only part of the equation that needs to be solved. Continuous availability is the goal towards which the majority of end users would like to see their systems move. *Continuous availability* is the

combination of the two components of continuous operations and high availability. *Continuous operations* provides freedom from planned downtime: the time taken to perform systems administration and operational tasks. *High availability,* on the other hand, can be regarded as providing freedom from unplanned downtime, such as hardware failures.

At the start, it is worthwhile setting some expectations about what a HACMP system can and cannot do. While much of this is limited only by the skills of the implementer, it is important to understand what a HACMP cluster actually provides in terms of addressing the high availability and continuous operations aspects of a continuous availability system.

1.2.1 The High Availability Capabilities

1. If the HACMP software is merely installed on a system, it will not provide any high availability functions. These are only available after customizing.
2. If the TCP/IP and LVM prerequisite steps have been performed as documented in the *HACMP Installation Guide* Chapters 4 through 7, and if the HACMP SMIT panels have been completed correctly (see Chapter 11 and 12 of the *HACMP Installation Guide*), and if no extra shell scripts or programs are added to the system, the cluster can provide protection against:
   - TCP/IP LAN network interface/adapter failure

     HACMP performs this by reconfiguring the IP address (and in certain configurations a hardware address to a standby network adapter).
   - Server node (CPU) failure

     HACMP performs this by reconfiguring another cluster node to take over any shared disks and the IP address (and in certain configurations, a hardware address of the failed node).
   - TCP/IP failure

     This is the sole partial AIX failure that HACMP can protect against. Should TCP/IP fail, keepalive traffic across an RS232 line or a SCSI target mode heartbeat (if implemented) will prevent either processor from trying to assume control of the other's resources.
3. After performing step 2, and adding some additional code (for example, application server start and stop scripts), the cluster can, in addition to the failures documented, provide for restart of an application following a server node failure. It may also be able to provide protection against:
   - TCP/IP LAN network failure

HACMP performs this by moving a service onto a secondary network should the primary network fail. This is only possible if you have two physical networks with network adapters on each.

Beyond these basic functions, the software provides little availability functionality. Other functions are certainly possible with a HACMP cluster, but some of them may require some significant effort to achieve. Almost any function is possible, depending upon the skill of the implementer.

Note that the following failures are not handled by the HACMP software:

- Disk and/or disk adapter failures

  Handling these failures is a function of the AIX Logical Volume Manager, or of a device such as a RAID disk array to implement mirroring or another data protection model.

- Failures of other devices

  Handling these failures is a function that is left to the implementer of the cluster to provide by using AIX facilities such as error notification, and so on.

- Application failures

  Handling these failures is a function that is left to the implementer of the cluster to provide.

1.2.2 The Concurrent Operations Capabilities

When an HACMP cluster has been correctly implemented, it can also provide certain concurrent operations capabilities, in addition to the previously mentioned high availability functions. The AIX operating system provides many facilities, such as the Logical Volume Manager and its dynamically pageable kernel that allow potential downtime to be avoided. HACMP allows you to continue providing a service to the end users of the system while a cluster node is taken down for maintenance (such as the application of fixes to the system, the upgrading of software and hardware, and so on). This eliminates a portion of the time that a system will be down for planned maintenance.

In effect, this ability is provided through a controlled failure. In a *controlled failure*, a node is taken down, the workload is automatically moved to a surviving node, and users continue working. When the maintenance process completes, the node may be reintegrated into the cluster and the workload restored to its original location. This process may then be repeated on the other cluster nodes in turn.

It is important to differentiate between the service provided by a node, which during maintenance activities may be provided by another cluster node, and that provided by the cluster as a whole. Using HACMP, all of the highly available services provided by the individual nodes may be provided to the end users during a maintenance period.

HACMP provides a basic mechanism that allows a cluster to provide a certain amount of continuous availability function. Given a suitable configuration, the high availability functions can give complete freedom from some, but not all, unplanned outages. Likewise, the concurrent operations facilities give some freedom from planned outages.

If the facilities provided by the AIX operating system itself are taken into account, both of these areas are covered to a greater extent, but there are still a number of other areas that need to be addressed. Before we address these, however, it is worth putting this into perspective by considering just what tasks need to be performed to run a computer system successfully.

1.3 Systems Management Disciplines

The management of a computer system falls into six main disciplines. These disciplines are:

- Business Management

  These are functions that support the business and its administration.

- Change Management

  These are functions that manage and control changes within the computer system.

- Configuration Management

  These are functions that allow for the planning, maintenance, and development of the resources that make up the computer system.

- Operations Management

  These are functions to support the computer system.

- Performance Management

  These are functions to measure and address the effectiveness of the system in meeting the business needs it was designed to address.

- Problem Management

  These are functions to manage the process from detection to resolution of problems with the system.

The six disciplines may be subdivided into sets of tasks or processes. The absolute definitions of these, and indeed the assignment of a given task to a given discipline, is very much dependent upon the model within which you are working. However, a non-exhaustive breakdown might look something like this:

- Business Management

  Inventory Control
  Financial Services
  Budget Planning
  Service Agreement
  Project Management
  Security Management
  Business Strategy Planning
  Services Marketing
  Architecture Definition
  Management System Definition
  Organization Planning and Control
  Staff Performance
  Education and Training

- Change Management

  Change Entry
  Assess and Approve
  Plan
  Schedule
  Distribute
  Synchronize
  Install
  Activate
  Test

- Configuration Management

  Configuration Design
  Environmental Planning
  Configuration Creation
  Updating Configuration Information
  Accessing Configuration Information

- Operations Management

  Workload Planning
  Operations Planning
  Workload Control
  Operations Control

- Performance Management

  Capacity Planning
  Performance Policy
  Performance Control
  Performance Execution and Measurement

- Problem Management

  Problem Policy Planning and Definition
  Problem Process Planning and Tracking
  Problem Correlation and Determination
  Problem Analysis and Diagnosis
  Problem Bypass and Recovery
  Problem Assignment
  Problem Fix Determination
  Problem Escalation
  Problem Resolution and Verification

If we are completely parochial, from a continuous availability perspective, almost all of these key tasks have some bearing on the availability of the system. The first step in increasing the availability of a system is to provide or perform systems management for it. But, as computer systems get larger and more complex, they become correspondingly harder to manage. The clustering of systems, rather than making this easier, actually increases this difficulty. Ten independent systems that are networked together are significantly easier to manage than ten systems which are clustered together because of the interdependencies between the systems in the clustered configuration. Consequently, tools are needed to assist with their management.

For true mission-critical computing, there should also be as little reliance on external resources as possible; the system should handle as much function as possible. This allows much swifter action to be taken and also increases the chance of the action being correct.

Consider, for example, the case of a component failure in a computer system and the associated recovery. When the component fails, the system will, in the vast majority of cases, be able to detect the failure significantly faster than an operator. The detection time for the system is a function of how often the component is used, sampled, or monitored for correct operation. The detection time for operators is a much more complex combination of whether they happen to be in front of the screen when the alert or error message comes in, and of what tools they have available to assist in isolating the error and/or linking the error to the particular failing component. This is also very

much dependent upon the skill of the operator in question. Even the most skilled operator, with intimate knowledge of the system, will be potentially several orders of magnitude slower than the system itself in detecting a failure. And until the problem has been detected and the failing component identified, no recovery action can occur.

When it comes to the recovery from the failure, the system again has an advantage in terms of speed, although this may not be as important as its advantage in correctness.

The skilled operator will know what action to take from experience and will promptly start the correct recovery procedure. But what if the operator is new to the job or is unfamiliar with this particular system? What if this recovery procedure has not been invoked for several months or even years? What if there is no a recovery procedure for this failure defined within the organization? In the worst case, an incorrect recovery action could be invoked that actually causes more trouble than the original problem.

Whatever the situation, additional time will be taken to recover from the failure, time that the system is potentially unavailable for doing what it was designed to do, to assist with the running of the business.

1.4 RS/6000 Cluster Technology (RSCT)

RS/6000 Cluster Technology (RSCT) is the name given to a set of software facilities that are designed to make management of a cluster easier. These have been added to the HACMP for AIX product to create HACMP ES. The RSCT software provides facilities to detect outages, failures, and impending failure conditions. It also allows software components, executing on different cluster nodes, to react to the failure conditions and coordinate recovery actions with each other.

The main tools provided by RS/6000 Cluster Technology are:

- Event Management
- Group Services

These two components work together to provide the infrastructure needed to address the issues of software failures and operational errors.

1.4.1 Event Management

The function of the Event Management subsystem is to monitor the status of system resources. Through this, it can detect unplanned outages and failures. Also, by using monitoring data over a period of time, it is possible to anticipate

potential problems before they become failures. The Event Management subsystem can:

- Notify software when an unplanned outage or failure occurs in the cluster.
- Notify when the status of a cluster resource meets a set of predefined conditions to allow anticipation of problems before they become failures.
- Provide status information about cluster resources.

Event Management provides an application programming interface to allow software to add information about its own resources that will enable it to be monitored using the Event Management subsystem. The software can be alerted, not only to failures in the cluster, but also to potential resource problems. This permits the software to react and remain available. For example, by monitoring memory resources, the software can modify its use of memory to head off a memory shortage before it cripples the application. Alternatively, the software could transfer workload from a node that is beginning to thrash, instead of waiting for a failure and then scrambling to pick up the pieces.

For more information on Event Management and the facilities it provides, refer to *Event Management Programming Guide and Reference*.

1.4.2 Group Services

Group Services provides a set of utilities that helps software to coordinate recovery and non-recovery activities between its own components and with other applications. It also provides notification when a software component or basic hardware service fails. Group Services helps in both the design and implementation of highly available software, and in the consistent recovery of multiple applications, by providing:

- Coordination for applications in formulating and executing failure recovery plans.
- Coordination for your applications and others in managing any change in responsibilities between the components of your distributed software.
- Notification to your software when an unplanned outage or failure occurs in one of your software's components or in some of the basic hardware services.

Group Services provides the basic system infrastructure that software needs to be highly available. It provides tools to coordinate distributed components in deciding how to respond to failures and to synchronize them in carrying out this response. Software will be notified of failures in key cluster hardware,

such as node or network adapter failures, as well as of failures in key components of your software.

But Group Services does more than provide recovery and synchronization capability; it permits your software to monitor the recovery actions taken by other Group Services applications. This allows your software to compensate when another Group Services application that you depend upon executes its failure recovery. This feature can greatly help applications that work closely together and are equally impacted if a failure occurs.

If your software already has an infrastructure for high availability, your application can make use of Group Services to coordinate itself with other Group Services applications and continue to make use of your own infrastructure. Group Services does not force you to abandon any existing infrastructure that you may already have; you simply make use of Group Services to coordinate with other applications that do not make use of your infrastructure.

For more information on the Group Services subsystem and the facilities it provides, refer to *Group Services Programming Guide and Reference*.

1.5 HACMP and RS/6000 Cluster Technology (RSCT)

As we saw earlier, HACMP provides a fairly large set of basic high availability functions and some continuous operations functions. From a high availability perspective, RSCT provides the additional mechanisms to fill the remaining gaps in the high availability functions. As previously mentioned, the main limitations in the capabilities of the HACMP software were its inability to handle:

- Disk and/or disk adapter failures
- Failures of other devices
- Application failures

RSCT provides the necessary facilities that allow HACMP to address these failures. The majority of the rest of the book shows how this can be done.

From a concurrent operations perspective, RSCT provides facilities that allow system conditions to be handled automatically before they necessarily require maintenance. Monitoring trends in the state of various system components as they change over time, or against a threshold, allows rectification of a potential situation before it becomes absolute. By acting upon these transitional state changes, it is quite possible to avert an absolute failure.

The end result of this is that there is a perceptible increase in the availability of the system because the recovery action for a resource in a non-critical state is likely to take a shorter time than that for a resource in a critical or failed state. There will still be a need to take the system down to perform certain tasks; however, the application of RSCT allows for significantly reduced planned downtime periods.

HACMP and RSCT can be regarded as providing System Intelligence. Once configured, the facilities they provide resemble those of a skilled operator who is always on duty and never makes mistakes. This allows a system to react swiftly to conditions as they arise, to select the correct course of action to handle a particular occurrence, and to run this action in the most suitable way to maintain the availability of the system. If the action itself fails, RSCT can even determine a secondary approach to handle the condition and use that instead.

Chapter 2. Events and Actions

This chapter introduces the concept of an event and the issues involved with event monitoring. It then introduces the different sorts of actions that may be taken when an event occurs.

2.1 What is an Event?

There are many possible definitions of “event”, but for the purpose of this discussion, an *event* is the occurrence of a change in state. The manifestation of the state change may be such that the event appears to be occurring for a given and defined resource. For example, a disk or a network adapter. Alternatively, it may manifest itself in such a fashion that the actual underlying cause of the state change is unknown or is a result of multiple state changes for a set of potentially unrelated components. The key criteria that we are really concerned about is that we can monitor for the occurrence of the event in some fashion.

Traditionally, event monitoring has only been concerned with major changes of state, for example, a hard disk in a computer system will either be in a working state or a failed state. There is no concept of transition within a model such as this. Events such as the following are fairly easy to detect; the resource is either functioning or not.

- Disk hdisk1 is failed.
- Node alpha has been powered off.
- Software program C has just abended.

But a monitoring scheme that is only concerned with absolute state is very limited. Futhermore, when an absolute failure occurs and is consequently detected, the time to recover from it may be significant.

The Event Management subsystem provides a mechanism by which transitional changes in state may be measured. By monitoring the changes in state for a resource over time, it is possible to detect a trend that may lead to a failure in the future. By acting upon the transitional state changes, it may be possible to avert the absolute failure. This means that there is a perceptible increase in the availability of the system because the recovery action for a resource in a non-critical state is most likely to take a shorter time than that of a resource in a critical or failed state. The recovery action is also likely to be less invasive.

As well as monitoring state transition trends over time, it is possible to set a threshold for a given measurable quantity. The concept of a threshold is completely alien to a traditional event monitoring model where the resource will either be working or not. It cannot be 95% working.

The transitional model, by providing the ability to measure state changes against a threshold value, provides more flexibility. Consider the following examples:

- Free space in /tmp is less than 5%.
- CPU utilization is greater than 85%.
- The volume of network traffic received on the tok0 adapter has just doubled but is still less than 50% of the maximum possible throughput.

In each case, the detection of these situations, by monitoring trends in state change over time or against a threshold, can allow us to rectify a potential failure before it becomes an absolute failure.

2.2 Event Monitoring

Event monitoring by nature is invasive. Even though a significant effort may be made to avoid this, there will always be some impact, somewhere on the system resulting from the event monitoring activities. For example, if you were to undertake monitoring of every single file on an average AIX system for any possible change, the system would be spending all of its time performing the monitoring tasks and very little time, if any, actually doing what it was designed to do. This may seem an extreme example, but it illustrates the main problem of event monitoring. Given all of the advanced facilities provided, how should event monitoring be configured to provide useful information, while at the same time minimize system impact? This is really a combination of three different items: *how often* to monitor (sampling frequency), *how to* monitor (types of monitoring), and *what to* monitor (critical resources).

2.2.1 Sampling Frequency

Given a mechanism that is monitoring for changes in state, the frequency with which measurements are taken has a major bearing on the system impact of the monitor. Measuring a value once every minute will consume significantly less system resource than making the same measurement once every second.

The resource that is being monitored very much determines the sampling frequency. Resources that change frequently need much shorter time

intervals between each measurement or observation; whereas, those resources that change at a slower rate can get away with longer measurement intervals without running the risk of missing important changes.

Once the measurement has been taken, the time within which it is reported is also of consideration. Again, resources changing at a faster rate require faster reporting intervals.

Example observation intervals (in seconds) for different types of resources might be:

- CPU 15
- Disk 80
- Filesystem 60
- LAN 40
- Memory 15
- Process 60

If the monitoring is being provided by an Event Management Resource Monitor, the observation and reporting intervals may be found in the EM_Resource_Class of the SDR.

2.2.2 Types of Monitoring

The impact of monitoring a resource depends upon how the monitoring is done. This in turn depends upon the resource being monitored. While there are alternative monitoring methods, the options are very much the same as the ways of implementing a resource monitor for Event Management. These are:

- Through a daemon
- Through a command
- Through code added to an existing program

The choice of mechanism very much depends upon your starting point. If you are coming from “nowhere,” then you have more options potentially than if you already have an existing resource monitor that you wish to modify.

For more information on the different methods of implementing resource monitors and the implications of each type, refer to *Event Management Programming Guide and Reference*.

Monitoring a resource internally normally requires either an internal monitoring facility to have been provided by the application developer or

access to the source code of the application that allows you to add monitoring capabilities.

Monitoring a resource using an external resource monitor is more susceptible to error and interference than a resource monitor that is implemented within the resource or its controlling software. This is due to the requirements of communication between the resource and monitor. By externalizing the monitor from the resource, an additional potential point of failure is also introduced. Given a monitor that is checking for the existence and correct function of a given device, if communications between the two components fails, how will this manifest itself? Is there the possibility that this will be perceived, albeit incorrectly, as a device failure? These issues need to be balanced against the fact that an external monitor is potentially easier to write. All of these factors need to be taken into consideration when determining where to monitor from.

Monitoring via an external monitor will use more system resources in terms of the additional process(es) of the monitor and communications between the monitor and resource. Monitoring provided internally will not require this, but may result in a possible increase in system utilization by the resource itself. In the majority of cases, the incremental increase in resource utilization is likely to be small; however, the cumulative effect of multiple monitors should not be underestimated.

2.2.2.1 Active Monitors vs. Passive Monitors

Having determined the type of monitoring required, a decision should be made as to whether to monitor in an active or passive mode. An *active monitor* is more invasive than a passive monitor because it has to physically query the resource or its controlling software. This may interrupt any activity that is currently occurring with the resource or, if there is a delay in which data is returned to the monitor because the resource is busy, a false reading may be taken. There are various techniques to minimize the effect of this, such as issuing a non-blocking read against a disk, but some interruption is unavoidable.

A *passive monitor*, on the other hand, is much more inclined to wait until something happens, say an error is returned, before doing anything. These monitors are much less invasive but potentially less accurate.

If a monitor samples a resource at a given frequency, there is a good chance that it is of the active type.

2.2.3 Critical Resources

To make the most effective use of monitoring facilities, you need to give careful consideration to how the resources of the system are to be monitored and, if active monitoring is to occur, at what rate the results will be gathered. These decisions will have some effect on the level of invasion that monitoring places on the system. The most effect, however, will result the selective choice of what resources should or should not have monitoring enabled for them.

Not all of the resources in a system are critical to its correct operation. In fact, in many cases, the failure of a given resource will effect no one other than the people directly using it at the time of failure. The failure of a user's shell process effects no one but that user. The system is still available. Consequently, monitoring for a failure of this shell process is unlikely to derive any real benefit. On the other hand, should a critical filesystem fill up or a system disk fail, there is more likelihood of serious system performance degradation. Monitoring in this situation shows measurable benefits.

The technique by which critical resources are identified, and the effect of their loss quantified, is known as Component Failure Impact Analysis (CFIA). The basic stages of this type of analysis involve:

- Describing the system design in terms of the resources.
- Describing the services that the system is supposed to provide.
- Defining resource chains, so that the interdependencies between different resources and services may be determined.
- Analyzing these resource chains to identify critical resources.
- Identifying the effect that the failure of/reduction in availability of the critical resources will have on the services.
- Defining a mechanism by which failure of a resource can be recovered or averted such that the services are maintained.

2.2.3.1 Critical Resource Categories

Critical Resources generally fall into four categories. These are dependent upon where, in the system as a whole, a backup resource exists.

The first category consists of those resources that have no backup resource or recovery mechanism. The failure of a resource in this category is likely to stop the system from performing useful work. In a system which has been designed for high availability, there should be no resources in this category. If there are, then there is something wrong with either the design or the implementation.

Monitoring for this first category provides little benefit other than to inform the system operator when a problem has occurred because no recovery action is possible.

The second and third categories consist of resources that have a backup. This backup may either be another resource of the same type that is capable of taking over the workload (for example, an Ethernet service adapter failing over to an Ethernet standby adapter), or a resource of a different type that is capable of providing the service, albeit potentially in a different or reduced form. An example of this might be the failover of an NFS file server from the SP switch to an FDDI backup network. The service will continue to be provided but in a potentially slightly modified form and at a lower bandwidth.

The difference between these categories relates to where the backup resource is physically located. The second category consists of those resources that have a backup resource within the same cluster node. The third category consists of those resources that have no backup resource within the same cluster node. In the case of the third category, a recovery following a failure is performed by moving the effected services to another cluster node.

In the vast majority of cases, monitoring for these resources should be carried out by the node containing the resources. For example, monitoring for the failure of a critical adapter. In a very few cases, another cluster node may provide monitoring in the case where the critical resource is the node itself or a component whose failure the containing node cannot detect.

The final category of resources consists of those that have no backup resource but have a recovery mechanism. These resources are often those that do not fail in the absolute sense but become constrained. For example, a filesystem filling up, or a system starting to thrash.

Monitoring for these resources should be carried out on the nodes containing the resources.

2.3 Event Monitoring Example

Consider the simple two-node cluster shown in Figure 1 on page 21. However, note that a cluster configuration like this is not designed for no single point of failure and, as such, should not be used in a production environment. The design is used here purely as an illustration.

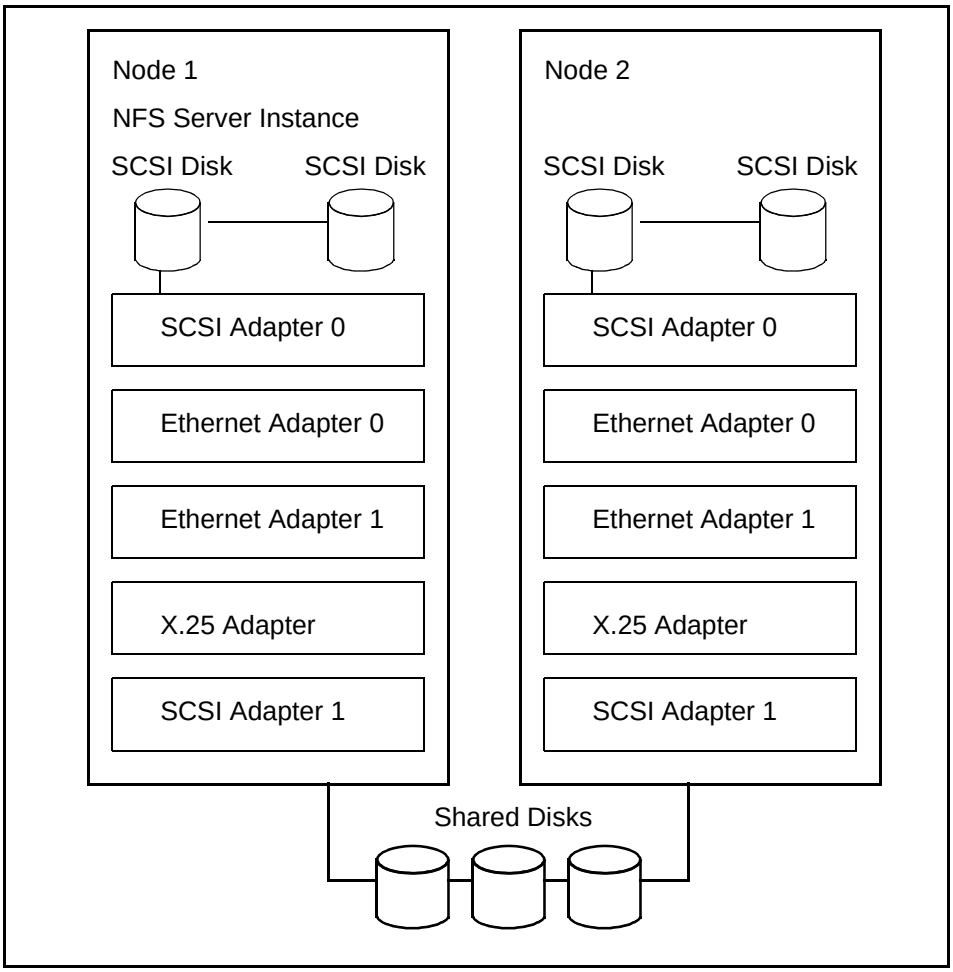


Figure 1. Simple Two-Node Cluster Configuration

Node 1 is an NFS server serving the file system held on the shared disks. Node 2 is a backup for Node 1. Each node contains two Ethernet adapters for client attachment, one in a service role, one in a standby role. All disks are mirrored. The X.25 adapter provides connection to a remote site for software updates.

For the NFS server instance to function correctly, the shared disks must be available. These are attached through a SCSI Adapter 1. The operating system should be running. This is provided by the SCSI disks attached to SCSI Adapter 0.

The critical resources for this node are:

- The two SCSI adapters

  If the SCSI adapter attached to the external disks fails, the NFS server cannot access the data it is serving. Monitoring for failure of this adapter can only be carried out on Node 1. If the SCSI adapter attached to the internal disks fails, the operating system disks (containing, for example, paging spaces) becomes inaccessible. Monitoring for the failure of this adapter can only be carried out on Node 1. Sampling for these resources should occur once every 30 seconds.

- The two SCSI buses

  If the SCSI bus attaching Node 1 to the external disks fails, the server cannot access the data it is serving. Monitoring for failure of this bus can be carried out by any system attached to the external SCSI bus. It is most likely to be performed by Node 1, as this is the system using the disks. If the SCSI bus attaching the internal disks fails, the operating system disks becomes inaccessible. Monitoring for the failure of this resource can only be carried out on Node 1. Sampling for these resources should occur once every 30 seconds.

- The service Ethernet adapter

  If the service Ethernet adapter fails, client traffic cannot be handled. Monitoring for this failure can be carried out on this node, or on the other cluster nodes, or from one of the NFS clients. Assuming HACMP is not monitoring the system, sampling for this resource should occur once every 40 seconds.

- The X.25 adapter

  If the X.25 card fails, the connection to the remote system(s) will fail. Monitoring for failure of the X.25 adapter does not bring much benefit other than notification of the failure because there is no possible backup.

- Node 1 itself

  If Node 1 fails, the service will fail. Monitoring for failure of the node can only be performed on another system. Sampling for this resource should occur once every 30 seconds.

The critical service is:

- The NFS server

  Total failure of the NFS server, or the processes that make up the server, will cause the service to stop. Monitoring for the failure can either occur on the server or on a client. As the service would most likely be monitored by

checking for the life of the processes that comprise the instance, this would typically be performed on the node currently running the NFS server. Sampling for service availability should occur about once every 30 seconds.

2.4 Classes of Actions

When the occurrence of an event has been detected, it is possible for the system to automatically initiate an action. This action (which is most likely, though not necessarily, a shell script) can cause other activities to occur.

There are four basic classes of action. Each class of action has a specific role in system recovery and will only be suitable for specific situations.

The four classes of actions are:

Class I	Do nothing
Class II	Notification actions
Class III	Simple recovery actions
Class IV	Complex recovery actions

Each action class builds upon the former, in that a simple recovery action is also likely to provide notification functionality.

2.4.1 Class I Action

It may seem at first glance that the unit or “do nothing” action is defined more for completeness than any other purpose. In fact, most actions defined to the system will be Class 1 actions, in that the act of not specifically defining an action is equivalent to defining an action that does nothing.

Apart from the unintentional use of this action, there are very few occasions where it would typically be used in a recovery sense.

2.4.2 Class II Action

Notification actions are the most common actions that are defined on a system. The basic purpose of a Class II action is to tell someone that something happened. It does not attempt to recover the event or error.

Some examples of Class II actions might be:

- Write a message in a log file.
- Write a message to the console.

- Send mail to a user ID (typically root).
- Notify another reporting mechanism (such as Tivoli).

When creating a notification action, there are several basic pieces of information that need to be provided to make the notification useful.

The first is when the event occurred; in other words, some form of time stamp. The time at which the event occurred is important because it helps to establish a sequence of events, and hence, the interrelationship between one event and another. If you know that component A failed at time x, then you might expect component B, which relies upon A for its correct operation, to fail at a time of x+y. If B were to fail at a time of x-y (that is, *before* the A component failed), the failure of B would, in this case, most likely be due to a different cause.

A cluster adds some complexity over a single system in that there must be a mechanism for coordinating time within the cluster of machines. Many problems may arise should the system times of the various cluster nodes differ widely. When reporting the time of an event to a notification action, you should ensure that you are reporting a "cluster time" rather than a local "machine time" because this allows a sequence of events to be established between the machines in the cluster, as well as between processes on the local machine.

The second component of a successful notification is where the event actually occurred— that is, on which machine in the cluster. The notification action is normally run by the system that physically detected the event. In the vast majority of cases, though not necessarily, this will be the system where the event occurred. HACMP provides facilities to swap IP addresses between network adapters. Consequently, care should be taken to ensure that the identifier that is returned is related to the physical cluster node rather than something that is related to the IP address, which may only temporarily have been swapped there. In some circumstances, it is also useful to know which system in the cluster detected the event.

The final piece of information is an identifier that can allow the event to either be uniquely identified or to point to more detailed information. The less additional information required by the reader to determine the cause of the event, the better. However, care should be taken to ensure that a reasonable level of threshold is provided. If the message written is several pages long, while complete in all details, there is probably too much information being notified. What needs to be determined is a sensible minimum for a system operator to react to, or for the log record to be meaningful. The other obvious

drawback of large messages being logged is that you increase the risk of filling log files or the directories holding them.

Consider the following example. A simple notification action might look like:

```
#!/bin/ksh
TIME=‘/usr/bin/date‘
NODE=‘/usr/sbin/cluster/utilities/get_local_nodename‘
/usr/bin/echo $TIME $NODE volume group sharevg is in an error state
exit 0
```

The output from this action would be:

```
Mon Oct 20 12:01:39 BST 1997 node1 volume group sharevg is in an error
state
```

Compare this with:

```
Mon Oct 20 12:01:39 BST 1997 node1 disk hdisk11 in volume group sharevg
has failed with error DISK_ERR1, SEQNO=559
```

The second output provides much more useful information, yet takes very little extra space. From it you can determine not only what disk has failed but also how it failed and where to go to find more information about the failure.

Notification output is always more useful when it is logged to a file. It can also be sent to the console or mailed to a user, but then there is a risk that it will either be missed or ignored until later. The first example output is less useful because it forces the user to search elsewhere to determine what actually happened. Any notification being sent directly to a user, or a systems management application like Tivoli, needs to provide more information than this basic set to be useful.

2.4.3 Class III Action

A simple recovery action is one that runs on a single cluster node that attempts to rectify the situation that caused the event. These actions are normally used where the event is caused by a situation that is specific to one node and has no interrelated actions that are required on the other cluster nodes. Such events generally involve resources internal to the cluster node or resources that are external to the node and not shared with any other. As soon as the recovery action involves activity on the other nodes, a more complex Class IV action is needed.

Remember that the Class III action builds upon the Class II action, in that it should provide logging and/or other forms of notification.

Consider the following example. The event that is anticipated occurs when the /tmp filesystem is over 90% full. The /tmp filesystem is part of rootvg, so it is internal to the cluster node in question. As a result, a Class III action may be invoked to handle this condition because there are unlikely to be any dependencies from the other cluster nodes. A possible, albeit drastic, recovery action might look something like:

```
#!/bin/ksh
DATE=‘/usr/bin/date‘
NODE=‘/usr/sbin/cluster/utilities/get_local_nodename‘
echo $DATE $NODE "/tmp filesystem over 90% utilized - cleanup invoked" \
>> /var/local/logfile
/usr/bin/rm -rf /tmp/* /.*
if (($? != 0))
then
echo "/tmp cleanup - failed " >> /var/local/logfile
exit -1
else
echo "/tmp cleanup - succeeded" >> /var/local/logfile
exit 0
fi
echo $DATE $NODE \
"/tmp filesystem over 90% utilized - cleanup completed" \
>> /var/local/logfile
```

This will remove the contents of the /tmp directory, and hence, cure the situation that generated the original event. The main issue with such a recovery action is that the very act of running it, while fixing the original problem, may well cause more problems in the future. A recovery action should always perform the minimum activity required to recover the situation in order to reduce the risk of the recovery action itself giving rise to problems. A better version of the preceding script might, for example, remove files older than one week.

2.4.4 Class IV Action

The complexity of a cluster arises from the interdependencies between the cluster nodes. In many cases, an event is handled by taking one action on one cluster node and different, though related, actions on some, or all, of the other cluster nodes.

Writing a traditional shell script that executes separate functions on different nodes is fairly uncommon due to the lack of a suitable mechanism to tie all of the components back together to provide synchronized actions. This is especially relevant when the processing power of the different cluster nodes varies widely and, as a result, timing issues become much more significant.

The addition of function such as dsh on the RS/6000 SP, or the recovery program model within HACMP itself provides synchronization mechanisms that facilitate the complex recovery models needed for Class IV actions.

The simplest form of a Class IV action is actually a Class III action that causes the node upon which it is run to fail. The node failure is detected by HACMP in the usual fashion, and takeover processing occurs. This involves separate actions being run on the other cluster nodes, hence, it becomes a complex action by definition.

Most Class IV actions will be managed using recovery programs provided by HACMP for AIX. For more information on using recovery programs, refer to Chapter 6, “HACMP Recovery Programs” on page 95.

Complex recovery actions are typically invoked for failures involving resources that are either shared or connected to multiple systems. Consider the following situation when a disk attached to multiple systems fails.

2.4.4.1 Complex Recovery Sample

1. The node that has the volume group containing the failed disk detects the disk failure.
2. This node runs a recovery action that:
   - Automatically removes the disk from the volume group.
   - Adds a new disk to the volume group.
   - Remakes and resynchronizes the mirrors held on the failed disk onto the new disk.
3. Having completed these activities, this node informs the other nodes of the event and invokes a recovery action on the other nodes that:
   - Changes the definition of the disk in the local ODM to now contain the PVID of the new disk.

This functionality provides a "hot-spare" type facility for mirrored shared disks. The second stage of the recovery activity is necessary because the ODMs on the other cluster nodes will have the PVID of the failed disk rather than that of the new disk. If these ODMs are not updated, when the varyon of the shared volume group is attempted, it will fail because it cannot find the disk with the PVID it has a record for. By changing the PVID of the failed disk to that of the new disk, this problem does not arise.

Recent releases of HACMP (since Version 4.2.0) provide functionality to update the ODMs of the other cluster nodes automatically during a failover by performing an exportvg/importvg. This facility is known as *lazy update*.

The example in 2.4.4.1, “Complex Recovery Sample” on page 27 illustrates one action being run on one node in the cluster, with different actions being run on the other cluster nodes. In this case, the actions run on the other nodes are likely to be (though are not necessarily) identical. If so, there are only two components that need to be invoked, one for the node detecting the event, and one for the other cluster nodes.

More complexity is required when the actions run on the nodes are all different; in which case, it is necessary to determine where the event occurred, where the action is running, and potentially perform other tests to be able to determine what component of the action to run and with what parameters.

The action invoked is, to a certain extent, limited by the mechanism by which it was detected. The recovery action depends upon the location (node) where the action is run and the degree to which the script exploits any intelligence it might have. Referring again to the example in 2.4.4.1, “Complex Recovery Sample” on page 27, if all cluster nodes see an identical view of the shared disks, that is, if the device name hdisk5 refers to the same physical device on all nodes, then assuming that it is this disk that has failed, the update of the ODM needs to be performed against hdisk5. If the disks are named differently on each cluster node, additional logic needs to be provided (for example, by using the PVID of the failed disk) to determine which hdisk record of the ODM to change on which system.

As is the case with most programs, there is a trade-off between functionality, ease of creation, and ease of maintenance. A Class I action is likely to be simple to write and maintain. Actions that consist of multiple components are distributed across multiple cluster nodes or provide additional intelligent logic that is much more powerful, yet are correspondingly harder to create and maintain.

The choice of which approach to take is very much defined by the skills available to create and maintain the system. In the majority of situations, it is simpler to create and maintain multiple small programs rather than a few large, more complex, ones.

Chapter 3. Defining and Configuring Event/Action Pairs

This chapter helps you determine the best methodology for monitoring a given resource.

3.1 Determining the Monitoring Mechanism

There are various techniques for monitoring a resource within a cluster. Some of these techniques only allow for monitoring and logging, while others provide mechanisms by which recovery actions may be triggered. Assuming that a recovery action is required to be run, it is necessary to determine the best mechanism to use. To perform this, there are two questions that should be asked:

- How is the event detected?
- Where does the recovery action need to run?

The following sections look at some of the issues involved in answering these questions.

3.1.1 How Can the Event Be Detected?

There are four mechanisms for detecting the occurrence of an event:

- AIX error log
- syslog
- User monitor (shell script or program)
- Resource monitor

The interaction between these four mechanisms can be fairly complex, as Figure 2 on page 30 shows. For example, syslog and both types of user monitors can write to the AIX error log. A program-based user monitor can pass information to a resource monitor or may indeed be a resource monitor itself. Just to add to the complexity, resource monitors can write to the AIX error log and vice versa.

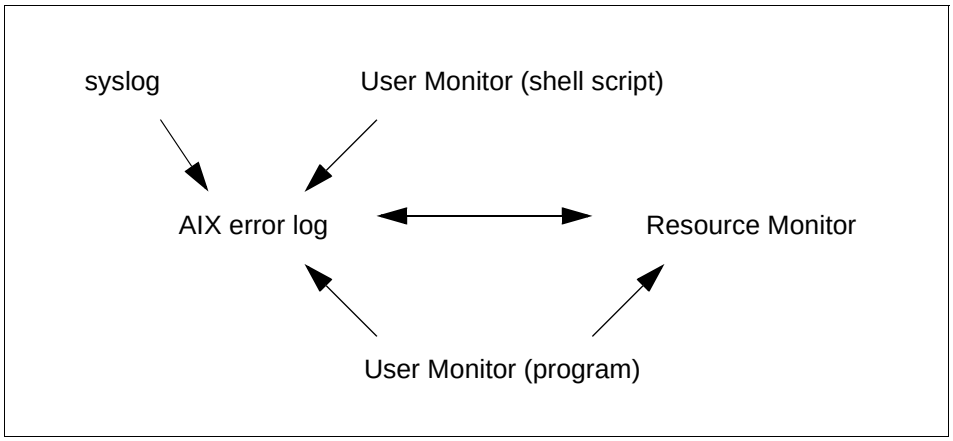


Figure 2. Relationship between Event Detection Mechanisms

In reality, this then leaves two main detection mechanisms: the AIX error log, which may be fed from other sources, and a resource monitor.

3.1.1.1 Using the AIX Error Log

The flow between the various components that can write to the AIX error log, and the actions that can be triggered from the AIX error log, are shown in Figure 3 on page 31.

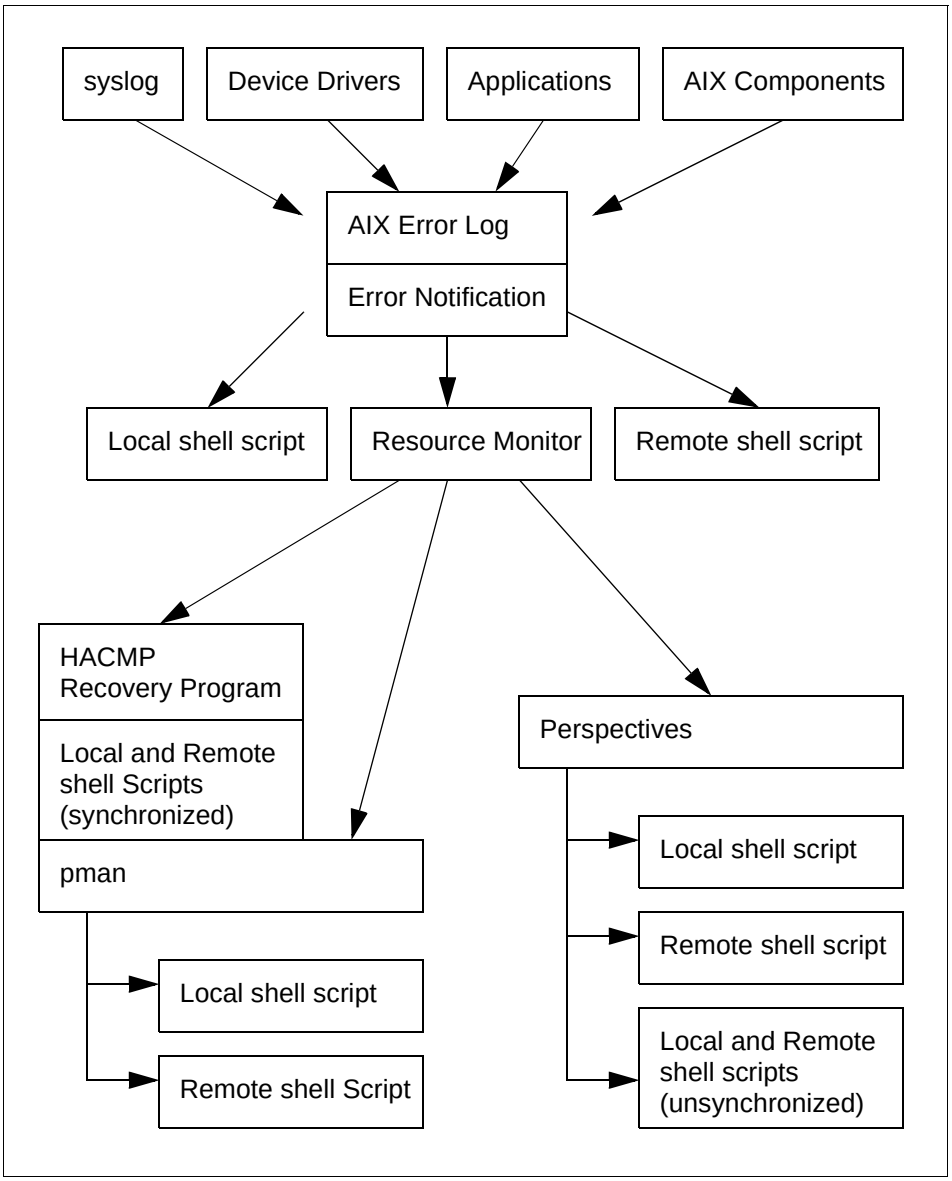


Figure 3. AIX Error Log Event Detection Mechanism

Each entry in the error log has an error label. In order to trigger an action by using an error notification object, this error label needs to be known. Certain error labels can only be generated by a specific device. For example, TOK_ADAP_CHK by a Token Ring adapter. There are also generic error

labels that can be generated by different types of devices. The DISK_ERR1 error, for example, could be generated by a CD-ROM drive, a R/W optical drive, or a disk drive. There are error labels generated by software and error labels, such as SYSLOG or OPMSG.

Determining the correct error label for a given failure or problem is, in many cases, not as straightforward as it could be. Refer to Chapter 4, “The AIX Error Log and Error Notification” on page 47 for more information on determining the correct error label.

3.1.1.2 Using a Resource Monitor

Use of a resource monitor provides increased flexibility over and above use of the AIX error log as a detection mechanism. Whereas the AIX error log provides a much more traditional monitoring mechanism, showing that a device is either in a failed state or not, a resource monitor introduces the concept of defining an *event predicate*, such that the event is triggered when a value reaches a certain level (for example, paging space on a cluster node reaches 90% utilization).

Despite this flexibility, there is still a limitation, in that a resource monitor is required to provide the information about the resource, and this monitor has to be capable of detecting the measurable quantity that you are using as a trigger. For example, the harmpd resource monitor provides, among other things, measurement facilities for the IBM.PSSP.Prog.pcount resource variable. This resource variable allows monitoring of processes to see whether they are running a specific process or not. You cannot use this resource variable to monitor, for example, the amount of memory used by a given process. It is necessary to ensure that the resource monitor you are using is capable of providing the data you need.

The flow between the various components that can write to a resource monitor, and the actions that can be triggered from this monitor, are shown in Figure 4 on page 33.

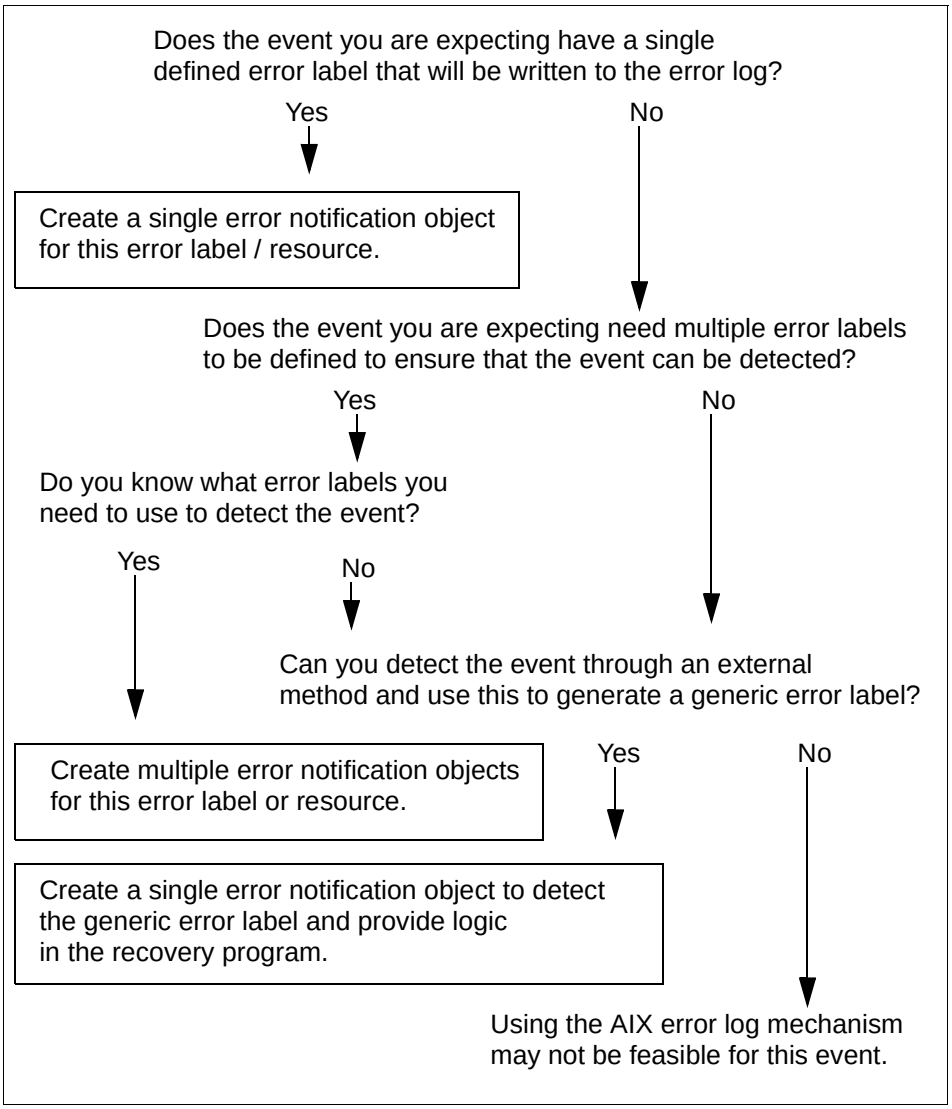


Figure 4. Decision Tree to Create Error Notification

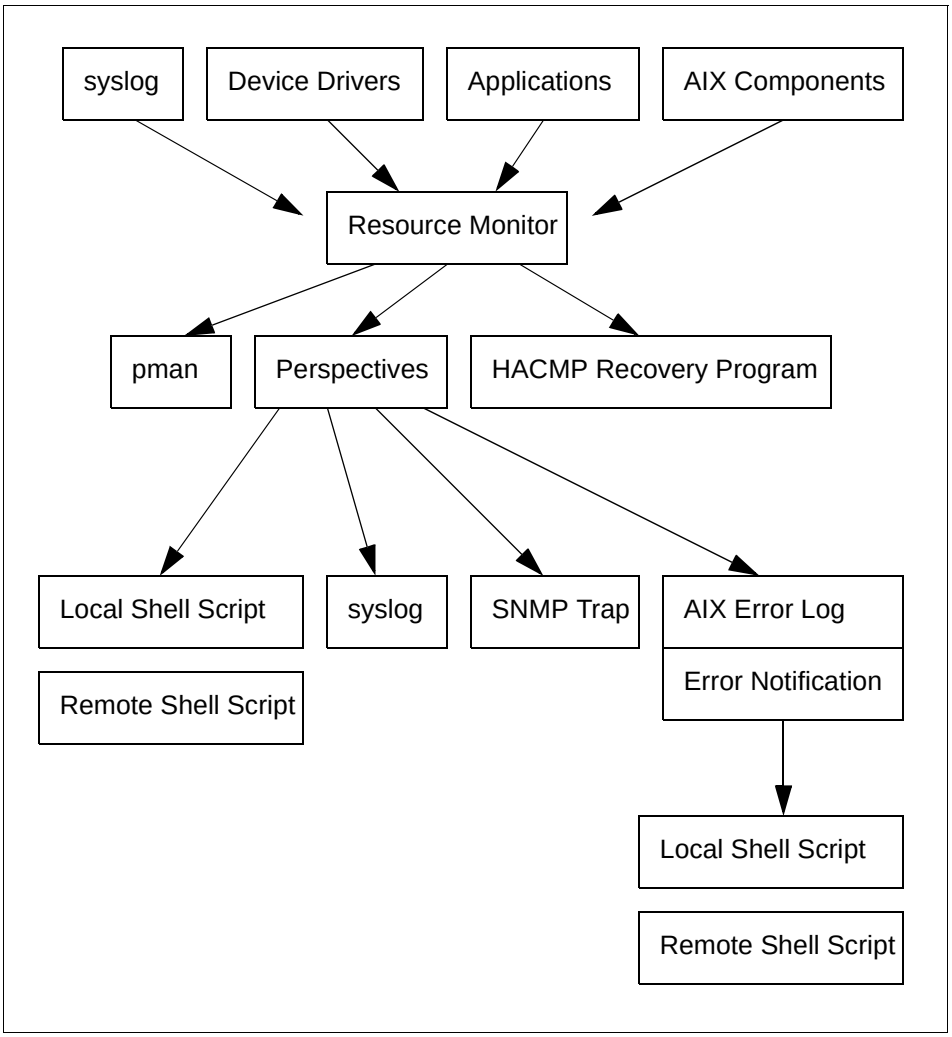


Figure 5. Resource Monitor Event Detection Mechanism

The resource monitors that are provided as standard allow for most of the typical monitoring functions to be performed. Determining whether the resource monitor provides the necessary functionality, or whether the resource variables exist to correctly identify the resource property, is the responsibility of the system administrator.

Refer to Chapter 8, “Common Monitoring Tasks” on page 123 for more information on common monitoring tasks. Refer to Chapter 11, “Sample

Events and Actions” on page 161 for more information on monitoring various common software subsystems.

3.1.2 Where Does the Recovery Action Need to Run?

A recovery action may need to be invoked either on a single node or on multiple nodes. In the single node case, this may be on the node where the event occurs, or on the node that detected the event (these may in fact be the same), or on one other node in the cluster (for example, a recovery action on the control workstation).

In the multiple node situation, the event may need to run on all of the cluster nodes or on a subset. All nodes may be required to run the same recovery action or different recovery actions. The recovery actions may or may not need to be synchronized to run.

Figure 6 on page 36 shows a decision tree illustrating how to select recovery actions.

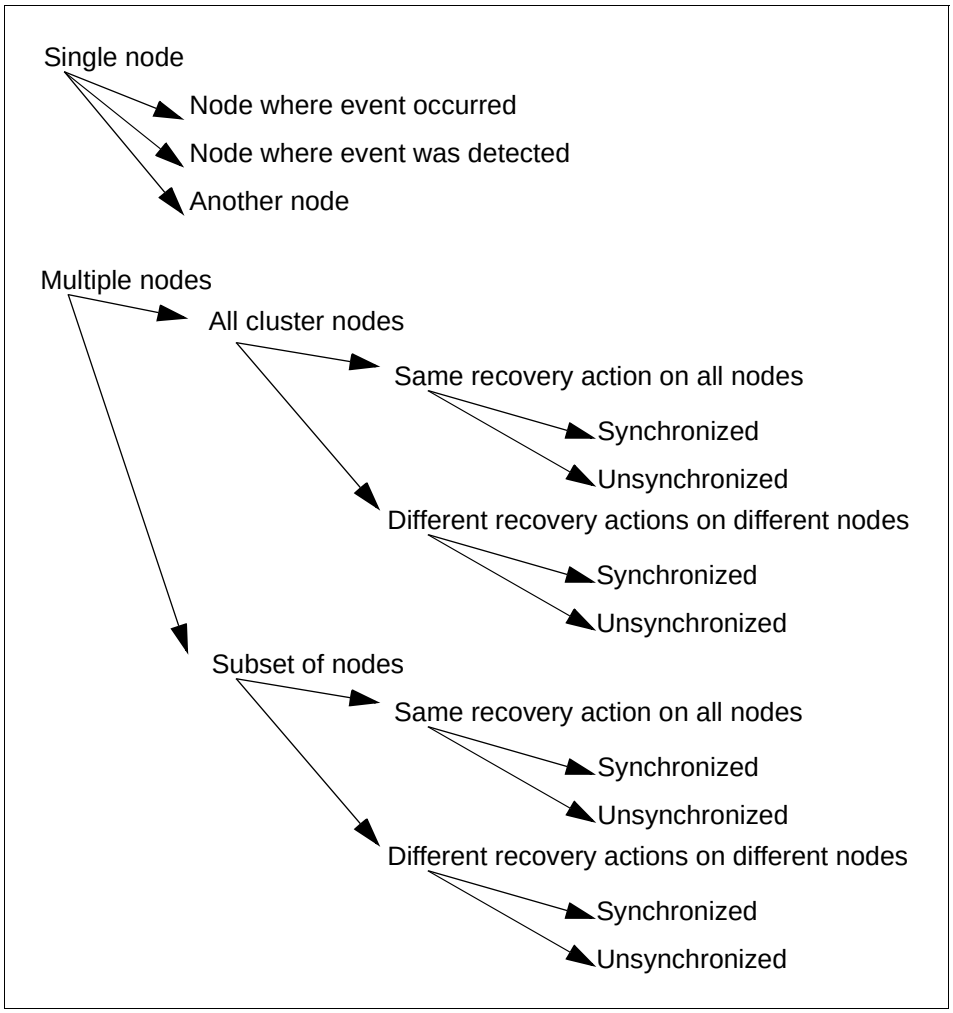


Figure 6. Recovery Action Decision Tree

3.1.2.1 Single Node Recovery Actions

Execution of a recovery action on a single node is relatively simple to perform. An event detected through the AIX error log, when it matches an existing error notification object, can cause a recovery method to be run. This recovery action can either run on the detecting node or can cause (by rsh or a similar function) the recovery action to run on a different cluster node, security permissions permitting.

Creating an error notification object to execute on the detecting node is simple. Executing on remote nodes is also easy but becomes complicated should the node on which the method has to execute change based upon the error.

Refer to Chapter 4, “The AIX Error Log and Error Notification” on page 47 for more information and examples of error notification methods.

The definition of a recovery action from an event detected by a resource monitor may be performed using the Event Perspective or by a Problem Management subsystem (pman). An event notification object may be created in the usual fashion. This defines the instance vectors for the resource variable.

If you need to define new predicates, these are defined by creating or modifying a condition. Remember that in Event Perspective a predicate is referred to as an *expression*. Event Perspective also allows an action to be invoked on a node different from that where the event was detected.

For more information on using the Event Perspective, refer to Chapter 5, “Event Management and Perspectives” on page 73. For more information on using the Problem Management subsystem, refer to Chapter 7, “Other Event Utilities and Functions” on page 105.

3.1.2.2 Multiple Node Recovery Actions

Multiple node recovery actions are more complex to define. There are three main considerations:

- Which nodes from the set of cluster nodes need to run an action?
- What action should the individual cluster nodes run?
- Do you need synchronization between the actions running on cluster nodes?

There are three main collections of nodes that may be called upon to run an event:

- The node where the event occurred runs the command.
- All nodes *except* the one where the event occurred run the command.
- All cluster nodes run the command.

These correspond to the three options offered by an HACMP ES recovery program. In the vast majority of situations, these node collections will provide all of the required functionality.

An HACMP ES recovery program will provide the necessary facilities to allow an action to be run on any of the node collections. The action of running the same action on both the “event” and “other” collections simultaneously is the same as running the action on the “all” collection. The reasoning behind this breakdown is that it allows for different actions to be run on different collections. There is no need to provide barrier synchronization if it is not required.

There is one other case of a node collection. This is where the nodes, upon which the recovery action is run, are independent of where the event occurred. Consider the following two examples. The first demonstrates the all/event/other approach. This is the node_down recovery program from HACMP ES.

```
event "node_down"  0 NULL
#
barrier
#
other “node_down” 0 NULL
#
barrier
#
all "node_down_complete"  X NULL
```

The node where the event occurs runs the node_down script. When this completes successfully, the other nodes (that is, all those except where the event occurred) run the node_down script. Where this might initially appear to be the same as running node_down against the "all" collective, timing is important in this case. Also, the node_down script provides its own logic to determine whether to run the node_down_local or node_down_remote script on the cluster node in question. All cluster nodes then run node_down_complete. This recovery program could potentially be rewritten as:

```
event "node_down_local"  0 NULL
#
barrier
#
other "node_down_remote"  0 NULL
#
barrier
#
all "node_down_complete"  X NULL
```

However, consider the cluster environment shown in Figure 7 on page 39.

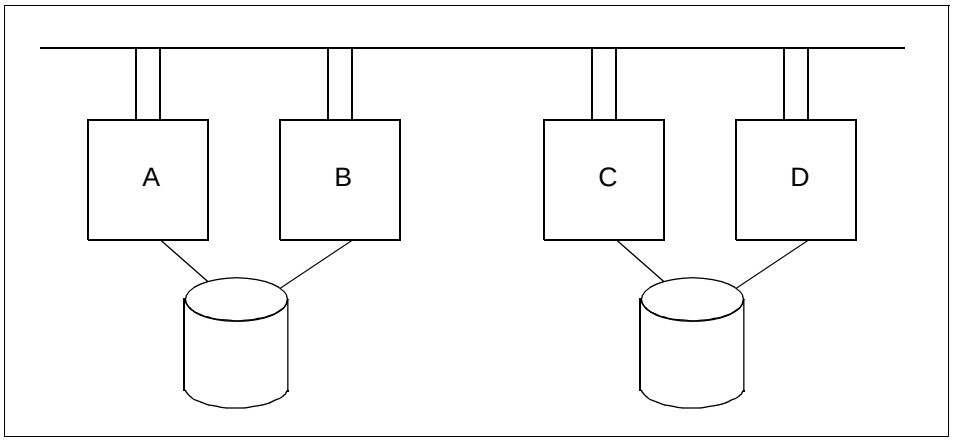


Figure 7. Cluster Environment Sample

For one of the applications, node C is the “server”. Nodes A and B are clients. Node C is protected by Node D. When Node C fails, Node D will have to run one action to take over the server functions. Nodes A and B will have to reconnect to the server. This process is shown in Figure 8 on page 40.

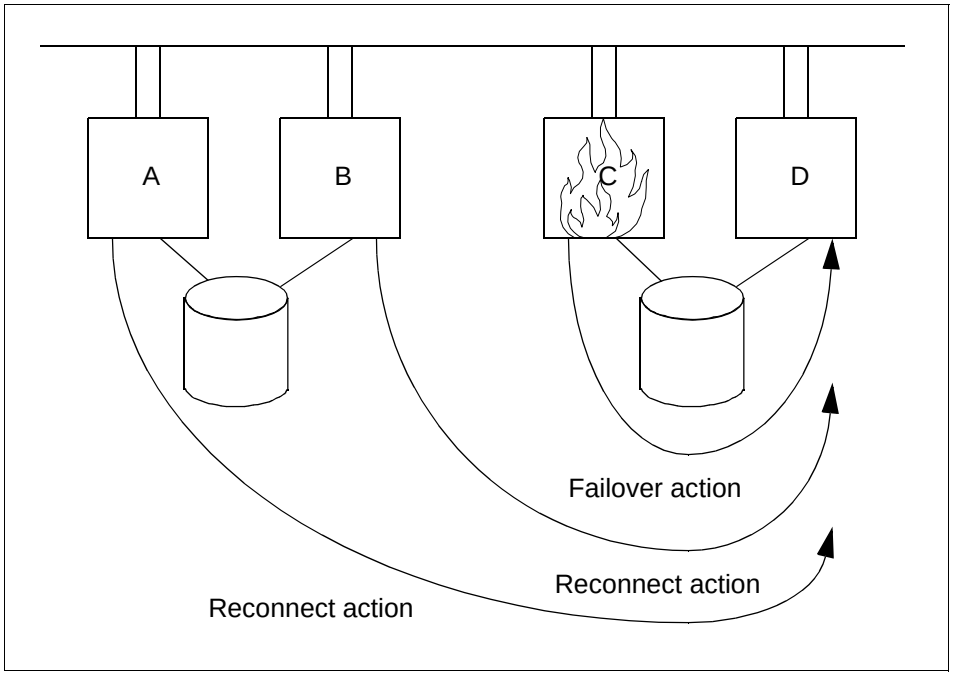


Figure 8. Cluster Environment Sample Recovery Action

In this environment, the event/other/all model falls down. It would be perfectly possible to force the recovery program mechanism to deal with this. One method of doing this might be to create different actions with the same name on Nodes A, B, and D. However, this might potentially lead to systems management confusion in the future.

Another alternative could be to provide additional logic in the action script to determine whether to run either a failover or a reconnect, depending upon where in the cluster the action was run. This might be complex to do, given the environment. A possible solution would be to add the concept of a “list” node collection--this would run the action on the nodes in the list.

Cluster Single Point of Control (CSPOC) provides functionality to execute a script on a list of nodes. Hence, the “recovery program” in this case could be provided by CSPOC. This does not provide synchronization between nodes, but in this case, this is not necessary. The pseudo code for the CSPOC recovery might look like:

```
%try_serial nodeD
    run_failover_actions
```

```
wait for failover to complete
%try_parallel nodeA nodeB
    run reconnect actions
```

In practice, if nodes A and B are using TCP/IP to communicate with nodes C and D, this problem should be resolvable using IP address takeover. If the communications between the two systems are not IP based, a mechanism such as this would be required.

For more information on using CSPOC, see the *HACMP for AIX Administration Guide*.

There is no simple way of determining which action to run where in a cluster. To do this, you need to understand the implications of a given event on each of the nodes. This should guide you into determining the relevant actions that need to run within each node. Some basic guidelines to follow are:

1. Identify the services you need to provide and the resources on which these services depend.
2. Identify the users of these services and their physical location.
3. Identify the steps that need to be followed to get the resource back into a functional state following a failure.
   - An event occurring against a device or resource that is only used by a single node only needs a recovery action to run on that node.
   - An event occurring against a device or resource that is used or accessed by other cluster nodes needs a "recovery" action that runs on the node holding it and "reconnect" actions to run on the other (using) cluster nodes.

3.2 Choosing Event or Action Mechanisms

Table 1 through Table 4 on page 43 and Figure 9 on page 44 may be of use when determining what mechanism to use to detect an error and trigger an action.

1. Table 1 applies to cases where you use defined or known AIX error labels to detect events.

Table 1. Using Defined or Known AIX Error Labels

Required Recovery Action	Mechanism
Recovery action on the detecting node	AIX error notification object

Required Recovery Action	Mechanism
Recovery action on one other node	AIX error notification object (remote execution of recovery command)
Unsynchronized recovery action on multiple nodes	Event Perspective (triggered via errlog_rm notification object) CSPOC triggered from error notification object
Synchronized recovery action on multiple nodes	HACMP Recovery Program (triggered by errlog_rm notification object)

2. Table 2 applies to cases where you use a resource monitor to detect events.

Table 2. Using Resource Monitor

Required Recovery Action	Mechanism
Recovery action on the detecting node	Event Perspective
Recovery action on one other node	Event Perspective
Unsynchronized recovery action on multiple nodes	Event Perspective
Synchronized recovery action on multiple nodes	HACMP Recovery Program

3. Table 3 applies to cases where you use shell scripts or programs to detect events. Use errlogger/errlog()/logger to write to the AIX error log.

Table 3. Using Shell Script or Programs

Required Recovery Action	Mechanism
Recovery action on the detecting node	AIX error notification object
Recovery action on one other node	AIX error notification object (remote execution of recovery command)
Unsynchronized recovery action on multiple nodes	Event Perspective (triggered by errlog_rm notification object) CSPOC triggered from error notification object
Synchronized recovery action on multiple nodes	HACMP Recovery Program (triggered by errlog_rm notification object)

4. Table 4 applies to the case where you use syslog to detect events. Use syslog redirection to write to the AIX error log.

Table 4. Using syslog

Required Recovery Action	Mechanism
Recovery action on the detecting node	AIX error notification object
Recovery action on one other node	AIX error notification object (remote execution of recovery command)
Unsynchronized recovery action on multiple nodes	Perspectives (triggered by errlog_rm notification object) CSPOC triggered from error notification object
Synchronized recovery action on multiple nodes	HACMP Recovery Program (triggered by errlog_rm notification object)

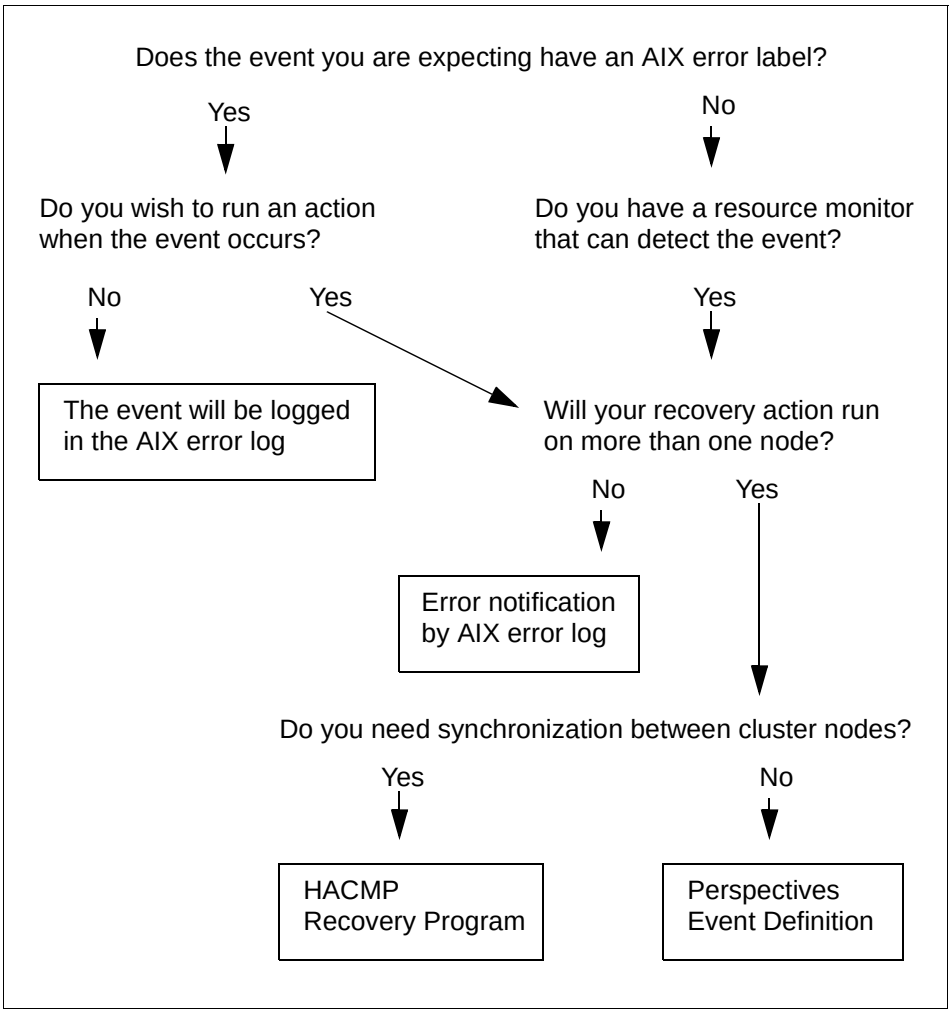


Figure 9. Decision Tree to Select Event Detection Mechanism

Part 2. Tools and Utilities

Chapter 4. The AIX Error Log and Error Notification

This chapter describes the use of the AIX system error logging facilities. It discusses how the error logging mechanism works, provides several mechanisms for writing to the log, and describes how to automatically react to errors written there.

4.1 The Error Logging Process and Error Notification

The AIX Error Log provides a very powerful facility to allow an event to automatically trigger a recovery action. The main function this facility provides is to allow a device driver to record software or hardware failures. These may be recorded for informational purposes or to assist with fault detection and problem determination. Although the term “error” is used, in many cases the information written to the error log is not necessarily the result of a failure. As a result, and because the error logging mechanism is easily extensible, it can be exploited to address many areas beyond its original scope.

When an error or fault is detected, the detecting software module creates an error record that is written to /dev/error. To do this, it will use either the errlog() subroutine or the errsave() kernel service, depending upon whether the detecting module is running in user or kernel space, respectively. The error record is time stamped, and the error daemon (errdemon) takes this error record and creates an error log entry from it. Before the entry is written to the error log, the errdemon compares the label sent by the detecting module to the contents of the error record template repository. The daemon retrieves the appropriate template from the repository, the resource name of the entity that caused the error, and any available detail data. If the error signifies a hardware-related problem and hardware Vital Product Data (VPD) exists, the daemon retrieves the VPD from the Object Data Manager (ODM).

The error logging mechanisms provide a way to record the occurrence of an event. This does nothing in itself to trigger an action. To do this, the second component, *error notification*, is invoked. Each time an error is logged, the error daemon determines if the error log entry matches the selection criteria of any of the error notification objects. If matches exist, the daemon runs the programmed action, called a *notify method*, for each matched object. The selection criteria for an event and the associated notify method are stored in an Error Notification object. This is held in the Error Notification object class of the ODM, /etc/objrepos/errnotify.

Error Notification objects may be created in one of two ways. The first is to use an editor to create a stanza file and then use the `odmadd` command to take

this file as input to the error notification object class. The format of the stanza file for an error notification object is:

```
errnotify:
        en_pid =
        en_name = ""
        en_persistenceflg =
        en_label = ""
        en_crcid =
        en_class = ""
        en_type = ""
        en_alertflg = ""
        en_resource = ""
        en_rtype = ""
        en_rclass = ""
        en_symptom = ""
        en_method = ""
```

For more information about the error notification object class, the descriptors, and the possible values, refer to *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*, SC23-4128.

If the stanza file is /tmp/error1.stz, the error notification object is created using:

```
odmadd /tmp/error1.stz
```

Alternatively, if you have HACMP installed on your system, you can use SMIT to create the error notification object. The fastpath to the SMIT panel for managing error notification objects is:

```
smit cm_EN_menu
```

SMIT will display the following menu:

```
                              Error Notification

Move cursor to desired item and press Enter.

  Add a Notify Method
  Change/Show a Notify Method
  Delete a Notify Method

F1=Help             F2=Refresh          F3=Cancel           Esc+8=Image
Esc+9=Shell         Esc+0=Exit          Enter=Do
```

Selecting **Add a Notify Method** will present the following screen:

```
                           Add a Notify Method

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
 * Notify Object Name                                    []
 * Persist across system restart?                      No        +
   Process ID for use by Notify Method                  []      +#
   Select Error Class                                 None      +
   Select Error Type                                  None      +
   Match Alertable errors?                            None      +
   Select Error Label                                 []        +
   Resource Name                                        All     +
   Resource Type                                        All     +
   Resource Class                                       All     +
 * Notify Method                                      []


F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         Esc+6=Command       Esc+7=Edit          Esc+8=Image
```

The appropriate data should be inserted in the relevant fields. Pressing the Enter key will add the error notification object to the ODM. For more information about the use of SMIT in defining error notification objects, refer to *HACMP for AIX Installation Guide*, SC23-1940. For information on the possible values that can be used in the entry fields, see *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*, SC23-4128.

On systems with PSSP installed, a similar menu may be found using the fastpath:

```
smit padd_en
```

SMIT will display the following menu:

```
                              Add a Notification Object

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
  Add Object to all Nodes in Partition                 no                      +
  Add Object to Hosts                                 []
* Notify Object Name                                  []
  Process ID for use by Notify Method                 []                       #
  Persist across system restart                        no                      +
  Error Label                                         []                       +
  Error Class                                          All                     +
  Error Type                                           All                     +
  Match Alertable Errors                               All                     +
  Resource Name                                       []
  Resource Type                                       []
  Resource Class                                      []
* Notification Method                                 []


F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         Esc+6=Command       Esc+7=Edit          Esc+8=Image
Esc+9=Shell         Esc+0=Exit          Enter=Do
```

The only difference here is the ability to add an error notification object on multiple nodes.

Error Notification provides a powerful facility to automate recovery actions should a problem arise. The RS/6000 hardware and AIX Operating system are the main sources for errors that are logged to the error log. However, it is easy to log your own errors to the error log to take advantage of this functionality within your environment.

The following section discusses how to write information to the error log and how to define error notification objects and methods to deal with the errors you are writing. Examples of how you might use this facility in conjunction with the HACMP for AIX software are also provided.

4.2 Logging Errors to the Error Log

This section describes several methods to log errors to the error log.

4.2.1 Logging Errors from a Shell Script

The simplest method of writing to the error log, and one that is easily implementable in a shell script, is to use the `errlogger` command. This allows an error log entry to be written that can contain a message of up to 230 bytes in length and is primarily used by the system operator to write information to the error log. The entry in the error log is of type OPMSG. Running the command:

```
errlogger "This is an error message"
```

will result in an entry similar to the following being written into the error log:

```
# errpt -aJ OPMSG
---------------------------------------------------------------------------
LABEL:          OPMSG
IDENTIFIER:     AA8AB241

Date/Time:       Sun May 10 09:45:50
Sequence Number: 687
Machine Id:      000081007000
Node Id:         sp2en0
Class:           O
Type:            TEMP
Resource Name:   OPERATOR

Description
OPERATOR NOTIFICATION

User Causes
ERRLOGGER COMMAND

        Recommended Actions
        REVIEW DETAILED DATA

Detail Data
MESSAGE FROM ERRLOGGER COMMAND
```

```
This is an error message
```

This is of limited use for the purpose of triggering an action directly because the error log entry will have the same error label (OPMSG) regardless of the content of the error message. In order to make use of this message, the error message (or, more accurately, the detail data) needs to be extracted by the error notification method and then potentially a second script needs to be invoked to handle the error condition.

A generic error notification mechanism that uses this method to write error log entries might look something like the following:

The shell scripts that call the `errlogger` command do so with one of the following patterns:

```
errlogger A1
errlogger B1
errlogger C1
```

These are trapped by an error notification object that looks like:

```
errnotify:
        en_pid = 0
        en_name = "my_script_errs"
        en_persistenceflg = 1
        en_label = "OPMSG"
        en_crcid = 0
        en_class = "O"
        en_type = "TEMP"
        en_alertflg = ""
        en_resource = ""
        en_rtype = ""
        en_rclass = ""
        en_symptom = ""
        en_method = "/usr/local/err_method/script_errs $1"
```

The `/usr/local/err_method/script_errs` notify method that is invoked looks like:

```
#!/bin/ksh
ERR=‘/usr/bin/errpt -a -l $1 | tail -1‘
case $ERR in
    A1) exec /usr/local/scripts/A1_recovery_script;;
    B1) exec /usr/local/scripts/B1_recovery_script;;
    C1) exec /usr/local/scripts/C1_recovery_script;;
esac
exit 0
```

4.2.2 Redirecting syslog Messages to the AIX Error Log

Some operating system components and applications use the syslog facility for logging errors and other events. User applications can also use these facilities. Where syslog provides its own logging mechanism, notification of these types of events can also be redirected to the error log. For instance, it may be desirable to be able to list error log messages and syslog messages in a single report. This facility also allows an error notification object to be triggered upon receipt of a syslog event. The entry in the error log for a redirected syslog message is of type SYSLOG.

To redirect syslog messages to the error log, you should specify errlog as the destination in the syslog configuration file, */*etc/syslog.conf.

For more information on the syslog facility, see the `syslogd` daemon.

By default, HACMP for AIX informational messages are redirected by syslog to /var/adm/cluster.log. If you wish to redirect these messages to the error log, the entry in /etc/syslog.conf should be changed to:

```
# HACMP for AIX Informational Messages from HACMP for AIX
local0.info errlog
```

Should an HACMP event occur, the error log will contain entries similar to:

```
LABEL:                SYSLOG
IDENTIFIER:           C6ACA566

Date/Time:            Wed 24 Sep 05:38:26
Sequence Number:      201
Machine Id:           002119924800
Node Id:              HAserver2_en0
Class:
Type:                 UNKN
Resource Name:        syslog

Description
MESSAGE REDIRECTED FROM SYSLOG

User Causes
OPERATOR REDIRECTED SYSLOG MESSAGES TO ERROR LOG
                      Recommended Actions
                      REVIEW DETAILED DATA

Detail Data
SYSLOG MESSAGE
<13>Sep 24 05:38:26 HACMP for AIX: EVENT COMPLETED: node_down_complete
server1
```

As with the use of the `errlogger` command, this change is of limited value for directly triggering an action because all error log entries that have been redirected by syslog will have the same error label (SYSLOG) regardless of the original source of the event. Once again, in order to make use of this message, the detail data needs to be extracted by the error notification method and then a second script needs to be invoked to handle the condition.

A generic mechanism for handling messages redirected from syslogd could resemble the following. First, the redirected syslog event is matched with an error notification object that looks like:

```
errnotify:
        en_pid = 0
        en_name = "syslog_msgs"
        en_persistenceflg = 1
        en_label = "SYSLOG"
        en_crcid = 0
        en_class = "S"
        en_type = "UNKN"
        en_alertflg = ""
        en_resource = ""
        en_rtype = ""
        en_rclass = ""
        en_symptom = ""
        en_method = "/usr/local/bin/syslog_msg_handler $1"
```

The notify method is dependent on the format of the message written by the application or operating system function that you are trying to monitor. You will need a mechanism to extract a unique identifier from the message before you can act upon it. Assuming an application that writes messages to syslog in the format:

```
error_number    information_message
```

the `/usr/local/bin/syslog_msg_handler` script that is invoked might look like:

```
#!/bin/ksh
MSG=‘/usr/bin/errpt -a -l $1 | tail -1 | awk ’{print $5}’‘
case $MSG in
    "01-234"|"01-456") /usr/local/scripts/app_restart;;
    "02-456") /usr/local/scripts/disk_space_recovery;;
    "03-123"|"03-987"|"03-675") /usr/local/scripts/reauth_handler;;
     *)  /usr/bin/true;;    # This does nothing as all other errors are
not serious
esac
exit 0
```

4.2.3 Generating syslog Messages from a Shell Script

In addition to application use of syslog, it is possible for user error messages to be written to syslog. The `logger` command provides a mechanism to write to the syslog facility from the command line or a shell script. The message passed to the `logger` command is written to the system log. If /etc/syslog.conf is configured to allow it, this message will then be placed in the error log where it can be used to trigger an error notification method.

The functionality provided by the `logger` command is similar to that provided by the `errlogger` command, with two major exceptions:

- The `logger` command may be run by any user, whereas `errlogger` can only be run by root. This is particularly useful when a shell script has to run with a given non-root user ID.
- The `logger` command allows other data to be passed to syslog, and hence to the error log, such as a tag. This tag may be useful when it comes to uniquely identifying the message parsed by the error notification method.

For more information on the use of the `logger` command, refer to *AIX Version 4.3 Commands Reference*, SBOF-1877.

4.2.4 Writing to the AIX Error Log from an Application

A full description of the steps required to write to the error log from an application may be found in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts*, SC23-4125. Despite the title of that manual, the methods are also applicable to standard applications. The steps required to do this are briefly reviewed in “Error Message Definitions” on page 55.

4.2.4.1 Error Message Definitions

Error messages are composed of a type, codepoints, and data. *Codepoints* are hexadecimal numbers representing messages. Most of them are reserved for system use, but some are available for user-defined messages.

The text that appears in the error report is defined through an error template. This is a combination of type and codepoint data and is identified by a unique error number.

When an error occurs, *detail data* is formatted according to the template to produce an entry in the error log.

The process of enabling an application to write to the AIX error log consists of three steps:

1. Define what messages need to be logged. If suitable messages are not already defined to the system, the new ones need to be defined to the system as codepoints.
2. Codepoints are then combined to produce the template.
3. The application needs to have a subroutine call added to invoke the template and log the message.

4.2.4.2 Adding Codepoints

There are many codepoints defined to AIX. You may be able to reuse an existing codepoint rather than define your own. The existing ones are divided among a number of message sets that relate to possible causes for the error. The codepoints may be listed using the `errmsg` command.

If you cannot find a suitable message (codepoint) in existence already, then you will need to add one. A file containing the text for the codepoint should be created. An example of such a file might be:

```
SET R
F001 "New Codepoint Text"
```

This can then be added to the system using the `errinstall` command. The errinstall operation will create a file called file_name.undo, which can be used to remove the codepoint if required.

4.2.4.3 Creating an Error Template

Having added the codepoint, the next step is to create an error template. Use an editor to create a template definition file. The file will look something like this:

```
+MY_APP_ERR1:
Comment         = "This application has failed"
Class           = S
Log             = True
Report          = True
Alert           = True
Err_type        = PERM
Err_desc        = 2000
Prob_causes     = 0000
User_causes     = 0000
User_actions    = F001
Inst_causes     = 0000
Inst_actions    = 0000
Fail_causes     = 0000
Fail_actions    = 0000
```

The template can then be added to the system using the errupdate command:

```
errupdate -h file_name
```

The **-**h flag tells errupdate to produce the C language header file, which can then be used when writing applications. The header file created as a result of the errupdate command will look similar to:

```
# define ERRID_MY_APP_ERR1 0x7ca84f9e /* This application has failed */
```

4.2.4.4 Adding Code to the Application to Log Errors

Having added your error message and template to the system, you are now ready to use it with the errlog() system call. An example of code to invoke the template and message created above is shown in “Creating an Error Template” on page 56.

```
#include <sys/errids.h>
#include <sys/err_rec.h>
#include <memory.h>
#include <string.h>
#include "errdesc.h"
int main(void)
{
    ERR_REC(1) errbuf;
    errbuf.error_id=ERRID_MY_APP_ERR1;
    strcpy(errbuf.resource_name,"My Application");
    return errlog(&errbuf,sizeof(errbuf));
}
```

When this is invoked, the summary output from errpt will show:

```
ERROR_ID TIMESTAMP T CL RESOURCE_NAME ERROR_DESCRIPTION
7CA84F9E 0719122895 P S My Application SOFTWARE PROGRAM ABNORMALLY
TERMINATED
```

with the more detailed output from errpt -a giving:

```
ERROR LABEL:          MY_APP_ERR1
ERROR ID:             7CA84F9E

Date/Time:            Wed 19 Jul 12:28:59
Sequence Number:      76319
Machine Id:           000029111000
Node Id:              node2
Error Class:          S
Error Type:           PERM
Resource Name:        My Application

Error Description
```

```
SOFTWARE PROGRAM ABNORMALLY TERMINATED

Probable Causes
PROCESSOR

User Causes
0000
                        Recommended Actions
                        New Codepoint Text

Install Causes
0000
                        Recommended Actions
                        PERFORM PROBLEM DETERMINATION PROCEDURES
Failure Causes
PROCESSOR
                        Recommended Actions
                        PERFORM PROBLEM DETERMINATION PROCEDURES
```

Having enabled your application to write to the error log, you can now create error notification objects to cause recovery actions to run should specific errors be generated.

4.3 Implementing Error Notification in the Real World

This section describes the method to implement error notification.

4.3.1 Implications of Error Logging on System Performance

The error logging functionality provided by AIX is a very powerful facility, but, as with any facility like this, it may be subject to misuse. The vast majority of entries in the AIX error log will be derived from the underlying hardware, the AIX Operating System, or other system software. There is little that can be done to control the number of events logged from these resources. The AIX Operating System and its components have been architected, for the most part, to provide suitable thresholding of these.

However, if you add large numbers of error notification objects, this will affect the behavior of the system. Each time an error is logged, the error daemon determines if the error log entry matches the selection criteria of any of the error notification objects. If there are large numbers of objects to be checked, the time taken to perform the checking will increase. Similarly, the use of system resources to perform this task will also increase. In the majority of systems, this will not be a major issue; however, systems that are already resource-constrained may run into problems.

When adding your own facilities to write to the AIX error log, three key factors should be considered when deciding what information should be written to the error log.

4.3.1.1 The Importance of the Event

The first step is to determine whether a particular event should be logged or not. The general rule is to not log information that is either unimportant or confusing to its intended audience. However, a worse mistake is to not log an error that merits logging.

There are no hard and fast rules as to what should and should not be logged, but typically, you should only log critical or unrecoverable failures. In those cases where it may be useful to also log informational messages or events that were recovered internally, for problem determination purposes, you should work with the software developer, if possible, to identify a mechanism to turn on this logging only when it is really required rather than having these items logged by default.

The correct execution of a piece of software should not be logged. To do so not only causes unnecessary use of system resources, but also leads to a tendency of the operator to ignore these messages, vast numbers of them being irrelevant. This runs the risk of missing a critical failure that has been written to the error log quite legitimately.

4.3.1.2 The Text Written to the Log

Having determined that an event should be logged, the next step is to determine the text to be written. Messages should be short and informative rather than long and cryptic. If you are automating recovery following the logging of an error by using error notification, remember that it may be necessary to use a short tag or error number that is easier for programmers to test for rather than trying to parse a unique string from several lines of text. The needs of automated recovery have to be balanced against the clarity of the message, or a combination of the two may be required. For example, messages similar to:

```
123-456:  A critical software failure has occurred
123-768:  Critical failure in module auth_check
```

would be easier for an error notification method to handle than:

```
A critical software failure has occurred
Critical failure in module auth_check
```

Yet they still retain the same degree of legibility to an operator. This is mainly of concern if you are using a facility such as errlogger from a shell script or

the errlog system call from a program. If you are using the `logger` command to write to the error log via syslog, you can use the -t flag to specify a tag that can be separate from the message itself.

4.3.1.3 Levels of Threshold

It is also important to determine the correct level of threshold and ensure that the error logging mechanisms you are using keep within the guidelines you have set. Each error to be logged, regardless of whether it is a software or hardware error, should be limited by threshold to avoid filling the error log with duplicate information. Just as you should determine what errors are important to log, the mechanism by which the log entry is written should ensure that you log an error once rather than multiple times.

The effects of excessive error logging include overwriting existing error log entries and potentially missing or losing errors. This can also be unduly alarming to an end user. The end user or service person can perceive a situation as more serious or pervasive than it is if they see hundreds of identical or nearly identical error entries where one or two might have sufficed.

Where it is important to set reasonable threshold levels, you can mitigate the effects of a poor threshold somewhat by ensuring that the sizes of the logs and buffers are adequate for the system. It is possible to configure the sizes of the in-memory buffer used by the error log device, and the size of the error log itself, by using flags on the `errdemon` command. It is important to ensure that these parameters are set to the correct size for your system. In practice, you should leave them at their default sizes until a problem occurs.

The in-memory buffer is pinned, and hence, if the buffer is made excessively large, performance of other processes will be impacted. If you make the buffer too small, however, the buffer can become full if error entries arrive faster than they can be read from the buffer and put into the log file. When the buffer fills, new entries are discarded until space becomes available in the buffer. You will know that this situation has occurred because the errdemon will create an error log entry to inform you. However, you may have lost an error that you wished to trap, and the required error notification object will not have been run.

The error log held on disk needs to be set to the correct size. When the error log fills, the log will wrap and existing information will be overwritten. This may lead to inaccurate diagnostic analysis should an important error message be lost.

At the same time, if the error log is made excessively large, it will consume disk resource. The default system error log in the /var file system also contains subdirectories and data files that are used by many busy applications, such as accounting, mail, and the print spooler. If applications on your system use the /var file system extensively, and you have set a large maximum size for the error log, you may fill the file system, which could cause other operational problems on the system.

4.3.2 Which Events Should Have Error Notification Objects?

Consider the example of a two-node cluster shown in Figure 10:

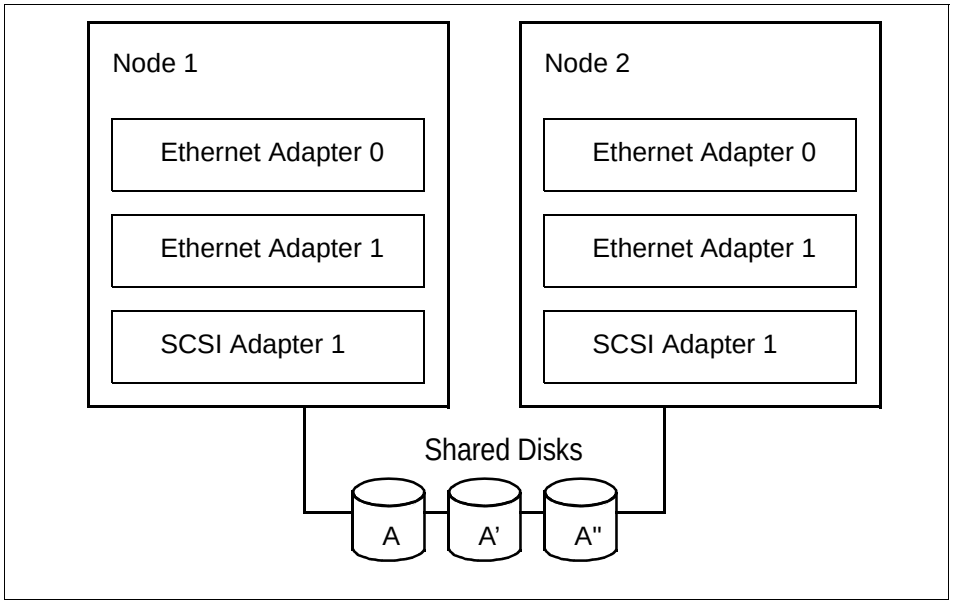


Figure 10. Example of a Two-Node Cluster

Each cluster node in this configuration has a single adapter to attach it to the shared disk subsystem that contains a database. Node 1 is active and running the database application. Node 2 is a standby node. If the disk adapter in node A fails, there is no way for it to access the database. Consequently, the system is unable to run the application.

However, from HACMP's perspective, the node will still be functioning correctly because heartbeat packets (keepalives) will still be being sent to or received from the other cluster node. No failover or recovery action will occur.

This is a classic situation where error notification would be used to detect the failure of the disk adapter and force a failover to the standby node. The failure of the adapter is detected when errors are logged and a recovery action (error notification method) is then initiated, which enables failover to occur.

The data in the external subsystem exists in a three-copy mirror, each copy denoted by A, A' and A”. The failure of a single mirror will not cause Node 1 to be unable to access the data and, as such, an error notification object to force a failure similar to that in the case of the adapter failure is unnecessary. However, it would be useful to know that a mirror had failed.

The subsystem has a single power cord. Failure of the power cord would cause loss of access to the entire database. Once again, an error notification object that forced failover to Node 2 would be of little use because Node 2 would be unable to access the disks, and the failover would fail.

There are no hard and fast rules as to what components should have error notification objects because this is very much subject to the design of the cluster. In a cluster configured for no single point of failure, there may be no need for error notification to be used. Typically though, any components that are key to the correct operation of the system, be they hardware or software, which do not have backup within a cluster node, will require error notification objects to force a failover to another system. Those critical components that do have a backup within a system may also require error notification objects, although in these cases, the notification method will control failover to the backup resource in the same node.

To determine the need for error notification, you should refer to the cluster diagram created during the planning phase of your implementation and identify those components without backup or that require special recovery processing.

Having determined which of the components need error notification objects, it is necessary to work out what events or errors the component is capable of generating.

If the component is software that has been developed locally, you should know which events can be generated and which need to be handled. If the component is hardware or software that has not been developed locally, you should follow the following procedure.

4.3.3 Identifying the Correct Error Templates

Using the `errpt` command with the `-t flag` lists the error record templates that have been installed on the system. These are all of the events or errors

that it is possible to generate. To get those events that a particular class of device is capable of generating, you will need to search through this list.

For a list of the categories of error identifiers written to the error log, refer to *AIX Version 4.3 Problem Solving Guide and Reference,* SC23-4123 .

For example, to get a list of all of the errors generated by SCSI devices, enter the command

```
errpt -t | grep SCSI
```

The resulting output will look similar to:

```
038F2580  SCSI_ERR7    UNKN    H   UNDETERMINED ERROR
0502F666  SCSI_ERR1    PERM    H   ADAPTER ERROR
0BA49C99  SCSI_ERR10   TEMP    H   SCSI BUS ERROR
13881423  SCSI_ERR4    TEMP    H   MICROCODE PROGRAM ERROR
4EDEF5A1  SCSI_ERR5    PERM    S   SOFTWARE PROGRAM ERROR
52DB7218  SCSI_ERR6    TEMP    S   SOFTWARE PROGRAM ERROR
54E423ED  SCSI_ERR9    PERM    H   POTENTIAL DATA LOSS CONDITION
5CC986A0  SCSI_ERR3    PERM    H   MICROCODE PROGRAM ERROR
C14C511C  SCSI_ERR2    TEMP    H   ADAPTER ERROR
```

There are six error types that identify the severity of an error log entry. They are:

- INFO Informational Entries
- PEND Impending loss of Availability
- PERM Permanent
- PERF Unacceptable performance degradation
- TEMP Temporary
- UNKN Unknown

A permanent (PERM) error is generated when the component or its controlling software is unable to recover from the causing condition. Usually, receipt of this type of error implies a hardware or software defect. Error types other than PERM usually do not indicate a defect but are recorded so that they can be analyzed by the diagnostics programs. For example, temporary (TEMP) errors are normally generated when a condition has arisen that has been recovered after several attempts.

A good starting point would therefore be to create error notification objects for all of the permanent errors defined for the critical components identified earlier. This will provide increased protection for your environment. Error notification objects for other error types may be useful or required, depending upon your operational environment.

Note that in addition to listing the error type for an error, the `errpt -t` command also shows the error class in the fourth column of the output. This is used when creating the error notification object.

See *AIX Version 4.3 Problem Solving Guide and Reference,* SC23-4123 or the documentation supplied with the component for more information about the errors that can be generated and their respective severities.

Remember that errors can be generated that are not those that you necessarily expect. The interaction between hardware and software is often complex and tightly linked. This can potentially lead to one or more unexpected errors being generated when a device fails. When a disk fails, it is quite possible, for example, to generate any one of, or any combination of, errors from the Logical Volume Manager, from the generic disk device driver (that is HDISK errors), or from the specific disk device driver (that is SCSI errors). This is in addition to any adapter or subsystem errors that might be generated.

So, when trying to determine which error notification objects are required for a given component, remember to allow for these complex interactions. Similarly, when testing the system, test the environment thoroughly to ensure that you see as wide a range of potential errors as possible.

Remember also that the generation of an error log entry is often dependent upon the activity of the component. It is quite possible to power off a device and for no error to be generated until some system activity attempts to access or use the device. For example, a disk will not be marked as failed until you attempt to write to it. A very common mistake is to create an error notification object for a disk, and then power off the disk, and then wonder why the error notification object is not triggered. Devices which are infrequently used will be difficult to monitor passively.

4.3.4 Error Notification Methods

The error notification method is typically a shell script or a command string. This is run when an error matching the selection criteria of the Error Notification object is logged. The most important thing to know about this, and a frequent cause of errors when implementing error notification objects, is that the error notification daemon uses the `sh -c` command to execute the notify method. When creating and testing error notification methods, it is important to test that the command can be invoked correctly with `sh -c`. The error notification daemon provides the following arguments to the notify method.

- $1 - Sequence number from the error log entry

- $2 - Error ID from the error log entry
- $3 - Class from the error log entry
- $4 - Type from the error log entry
- $5 - Alert flags value from the error log entry
- $6 - Resource name from the error log entry
- $7 - Resource type from the error log entry
- $8 - Resource class from the error log entry
- $9 - Error label from the error log entry

Invoking an error notification method on the detecting node is simple. The shell script or command string is simply defined as part of the error notification object. When invoking a method on a different cluster node, this is also simple, provided the node on which the method is invoked does not change. If this is the case, the notification method can include the relevant `rsh` commands. Consider the following examples:

The error notification object looks like:

```
errnotify:
        en_pid = 0
        en_name = "disk_error"
        en_persistenceflg = 1
        en_label = ""
        en_crcid =
        en_class = "H"
        en_type = "PERM"
        en_alertflg = ""
        en_resource = ""
        en_rtype = ""
        en_rclass = "disk"
        en_symptom = ""
        en_method = "/usr/local/bin/disk_err_handler $9"
```

This will run the notification method, `/usr/local/bin/disk_err_handler`, each time a PERM type error is logged against a disk resource. Note that the error label is passed to the error notification method with $9.

In general, it is better to invoke the recovery commands from a shell script rather than directly from the error notification method. This is because some commands do not function correctly when invoked by `sh-c`. Note that `/usr/local/bin/disk_err_handler` might look like this when the method is to be run on the detecting node only:

```
#!/bin/ksh
#
case $1 in
```

```
    DISK_ERR1) /usr/local/bin/do_something;;
    DISK_ERR2) /usr/local/bin/do_something_else;;
    DISK_ERR3) /usr/local/bin/do_something_different;;
esac
```

When execution on a single remote node is required, this shell script might look like:

```
#!/bin/ksh
#
case $1 in
    DISK_ERR1) /usr/bin/rsh ctrl_ws /usr/local/bin/do_something;;
    DISK_ERR2) /usr/bin/rsh ctrl_ws /usr/local/bin/do_something_else;;
    DISK_ERR3) /usr/bin/rsh ctrl_ws /usr/local/bin/do_another_thing;;
esac
```

If the remote node changes depending upon the error, the script changes to:

```
#!/bin/ksh
#
case $1 in
    DISK_ERR1) /usr/bin/rsh node1 /usr/local/bin/do_something;;
    DISK_ERR2) /usr/bin/rsh node2 /usr/local/bin/do_something_else;;
    DISK_ERR3) /usr/bin/rsh node2 /usr/local/bin/do_another_thing;;
esac
```

4.4 Testing Error Notification Objects

It can be difficult to test an error notification object. The first problems arise in actually generating the particular error in the first place. Most hardware errors are generated when a piece of hardware fails or functions incorrectly. This is complicated by the fact that certain errors are generated when the hardware fails in a certain way. Short of physically breaking the hardware, and potentially breaking it in a specific way, how are you going to generate the error to test your error notification object?

Assuming that you have a mechanism to fail a piece of hardware, for example by powering off a disk, you should not have too much trouble in generating a range of errors. For those errors that are harder to generate, it is necessary to have a mechanism for simulating errors using software.

The following source code allows you to generate errors for testing:

```
#include <sys/errids.h>
#include <sys/err_rec.h>
#include <string.h>
#include <stdio.h>
```

```
int main(int argc, char *argv[])
{
    int err_code;
    ERR_REC(1) errbuf;
    if (argc != 3)
    {
        printf("Usage: err_test <error_id> <resource_name>");
        exit (1);
    }
    sscanf(argv[1], "%x", &err_code);
    errbuf.error_id=err_code;
    strcpy(errbuf.resource_name, argv[2]);
    return errlog(&errbuf, sizeof(errbuf));
}
```

The `err_test` command, generated from the preceding code, allows you to write an entry to the error log. This can be used to trigger, and hence test, an error notification object. For example, to test an error notification object that is designed to react to loss of quorum for a volume group, that is, triggering upon receipt of an error log entry with an error label of `LVM_SA_QUORCLOSE`, you would perform the following steps:

1. Determine the error ID for the `LVM_SA_QUORCLOSE` error:

   ```
   errpt -t | grep LVM_SA_QUORCLOSE
   ```

This will extract the error template as follows:

```
91F9700D LVM_SA_QUORCLOSE    UNKN H  QUORUM LOST, VOLUME GROUP CLOSING
```

2. The `err_test` command is invoked:

   ```
   err_test 91F9700D sharevg
   ```

This will write the following error into the error log:

```
LABEL:              LVM_SA_QUORCLOSE
IDENTIFIER:         91F9700D

Date/Time:          Tue 23 Sep 16:38:10
Sequence Number:    34
Machine Id:         003540094100
Node Id:            node3
Class:              H
Type:               UNKN
Resource Name:      sharevg
Resource Class:     NONE
Resource Type:      NONE
Location:           NONE
```

```
Description
QUORUM LOST, VOLUME GROUP CLOSING

Probable Causes
PHYSICAL VOLUME UNAVAILABLE

Detail Data
MAJOR/MINOR DEVICE NUMBER
0000 0000
QUORUM COUNT
ACTIVE COUNT
```

If the error notification object is correctly defined, it will trigger upon receipt of this error log entry.

4.5 Error Logging and Event Management

The AIX error logging facility is very powerful in its own right; however, its scope and function can be further extended by utilizing Event Management functionality such as Perspectives.

4.5.1 Informing Event Management That an Error Has Been Logged

The base AIX error notification facility allows an error notification method to be invoked when an error occurs. This notification method will run on the system in which the error was logged. In many cases, this is sufficient to handle the condition that generated the error. For example, should an error occur that can only be recovered by failing over to another cluster node, the notification method need only halt the system or run a graceful stop of the cluster manager with takeover. HACMP can then provide recovery in the usual fashion. Assume, however, that the failure condition was more complex, and that recovery actions needed to be run on a different node or on multiple nodes in the cluster. This capability can be provided by the Event Management subsystem.

Event Management can be informed that an error has occurred using the errlog_rm notification method (this, more accurately, is a resource monitor). This method is physically located in the /usr/lpp/ssp/bin directory and is invoked by an error notification object like this:

```
errnotify:
        en_pid = 0
        en_name = ""
        en_persistenceflg = 1
        en_label = ""
```

```
        en_crcid = 0
        en_class = ""
        en_type = ""
        en_alertflg = ""
        en_resource = ""
        en_rtype = ""
        en_rclass = ""
        en_symptom = ""
        en_method = "/usr/lpp/ssp/bin/errlog_rm $1 $2 $3 $4 $5 $6 $7 $8
$9"
```

The `errlog_rm` program updates the IBM.PSSP.pm.Errlog resource variable. This allows error log entry parameters to be passed to Event Management. Event Management clients can then register to be notified on an event based on this resource variable. One example of this in use might be using the Event Management Perspectives GUI to monitor the error log.

Consider the following error that was written to the error log following a core dump:

```
LABEL:            CORE_DUMP
IDENTIFIER:       DE0A8DC4

Date/Time:        Mon Nov 17 11:40:41
Sequence Number:  626
Machine Id:       000022718000
Node Id:          deneb
Class:            S
Type:             PERM
Resource Name:    SYSPROC

Description
SOFTWARE PROGRAM ABNORMALLY TERMINATED

Probable Causes
SOFTWARE PROGRAM

User Causes
UNKNOWN
                  Recommended Actions
                  CORRECT THEN RETRY

Failure Causes
SOFTWARE PROGRAM
                  Recommended Actions
                  RERUN THE APPLICATION PROGRAM
                  IF PROBLEM PERSISTS THEN DO THE FOLLOWING
```

```
                    CONTACT APPROPRIATE SERVICE REPRESENTATIVE

Detail Data
SIGNAL NUMBER
                    11
USER'S PROCESS ID:
                    13206
FILE SYSTEM SERIAL NUMBER
                    57
INODE NUMBER
                    6160
PROGRAM NAME
notes
```

On the monitoring system, the Perspectives Event Management interface has an Event Definition that monitors changes in the IBM.PSSP.pm.Errlog resource variable. The predicate used in this case is `X@0 != X@1`; in other words, the event triggers whenever a new error is logged to the AIX error log. Viewing the event notification shows the following:

```
Event Definition Name: error_event
Condition
        Name:                   errtest1
        Description:            An error log entry has beenwritten
        Resource Variable:IBM.PSSP.pm.Errlog
        Expression:             X@0!=X@1

Event Status
        Resource Variable Value:
                Field:  0
                name:   sequenceNumber
                value:  626
                type:   char
                Field:  1
                name:   errorID
                value:  DE0A8DC4
                type:   char
                Field:  2
                name:   errorClass
                value:  S
                type:   char
                Field:  3
                name:   errorType
                value:  PERM
                type:   char
                Field:  4
                name:   alertFlagValue
```

```
                value:  0
                type:   char
                Field:  5
                name:   resourceName
                value:  SYSPROC
                type:   char
                Field:  6
                name:   resourceType
                value:  NONE
                type:   char
                Field:  7
                name:   resourceClass
                value:  NONE
                type:   char
                Field:  8
                name:   errorLabel
                value:  CORE_DUMP
                type:   char
        Time:   Mon Nov 17 11:40:41 1997
        Resource:       NodeNum=1
```

The main advantage of monitoring the error log in this fashion is that monitoring may take place, and/or an action may be invoked on a different cluster node from that where the error occurred. This would be harder to achieve using the standard AIX error notification mechanism.

4.5.2 Logging Events Detected to the AIX Error Log

In the same way that it is possible to tell Event Management about an error that has been logged to the AIX error log, it is possible for Event Management to write an entry to the AIX error log to the effect that an event has occurred. Events defined using the Event Management Perspectives interface (spevent) allow for an entry to be written to the error log. This can be used to trigger an event notification method as before.

Once again, the error label will be the same regardless of the event that has occurred (HA_PMAN_EVENT_ON), so additional logic would have to be provided to assist in isolating the absolute cause of the error log entry to allow the correct recovery action to be taken.

The following is an example of the format of an entry written to the AIX error log by this mechanism. Note that the Event Management resource variable, instance vector, and predicate are all provided in the error log entry.

```
LABEL:              HA_PMAN_EVENT_ON
IDENTIFIER:         E460E36E
```

```
Date/Time:           Wed Oct  8 04:38:55
Sequence Number:     1005
Machine Id:          00034173A000
Node Id:             ctrl_ws
Class:               S
Type:                UNKN
Resource Name:       pmand

Description
MONITORED SITUATION EXISTS

Probable Causes
MONITORED EVENT

Failure Causes
MONITORED EVENT
                     Recommended Actions
                     REVIEW DETAILED DATA

Detail Data
NAME
tmperr1
Node Number
1
RESOURCE TYPE
IBM.PSSP.aixos.FS.%totused
RESOURCE NAME
VG=rootvg;LV=hd3;NodeNum=1
SURPASSED THRESHOLD VALUE
X>90
DESCRIPTION
TMPERR1-TMP FULL EVENT HAS OCCURRED!
```

Chapter 5. Event Management and Perspectives

Perspectives is a feature of the IBM PSSP software that is used on the RS/6000 SP. If you are not using an SP system, you will not have this functionality available to you.

This chapter introduces the Event Management subsystem and the Perspectives Graphical Systems Management tool. It looks at the Event Management capabilities of Perspectives and describes how to configure the system by using the GUI to define important user events. User-defined events are the actions that should be triggered when an event occurs.

5.1 A Brief Overview of the Event Management Subsystem

Event Management is a distributed subsystem that ensures that status information related to one component of the system can be delivered to a second component, such that the latter can act upon this information. Event Management looks after resources. A resource can be anything in the system that we care to monitor, though typically the resources that are of interest are those that provide some important or critical service to a system user, for example, CPU usage, file system capacity, and so on.

The various resources that make up the system are monitored by entities called Resource Monitors. These monitors are usually programs that keep track of what the resource is up to and also supply information about the resource. Rather than provide all of the possible information about the resource, certain key attributes are monitored. Information about the status of these key attributes is provided to Event Management through an application programming interface known as the Resource Monitor Application Programming Interface (RMAPI).

Collecting information about these key resources is only part of the story. The information is of little use unless it can be acted upon, and Event Management clients perform this role. An Event Management client is an application that expresses an interest in one or more of the attributes of a resource being monitored by a Resource monitor.

Event Management clients interface with the Event Management subsystem through a second API known as the Event Management Application Programming Interface (EMAPI). When a client starts, it registers an interest in a given set of attributes for one or a number of resources. For each attribute, a condition is defined. Event Management keeps track of both the information being provided to it from the Resource monitors and the

information requested from it by its clients. When an attribute that Event Management is receiving information about matches a condition in which a client has expressed an interest, Event Management generates an event and notifies the client.

Event Management subsystems on different systems communicate such that a client does not have to be on the same machine as the resource in which it is interested. This also allows a single view of these events to be provided to the entire set of machines. In Figure 11, client C on Node 2 may register for, and will receive, the same event information about the resource monitored by Resource Monitor 1 as client A.

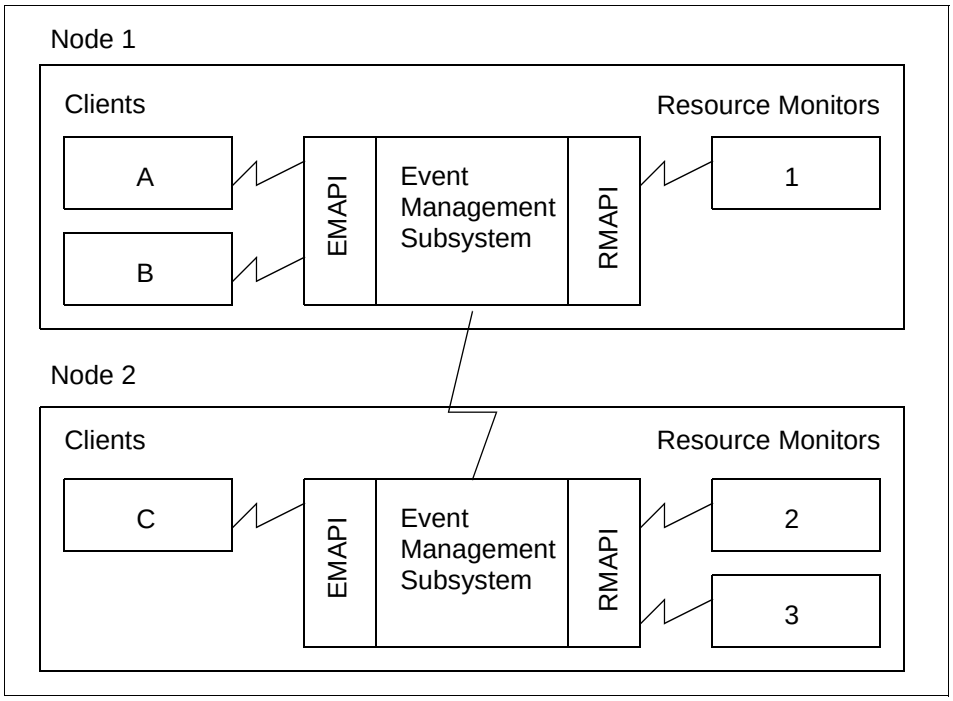


Figure 11. Event Management Subsystem

The ability to specify a condition of interest, plus the ability to monitor a given resource from any machine within the cluster, makes Event Management an ideal tool for improving the availability of a system. This is achieved by setting the conditions such that you anticipate problems and can hence compensate for them before they become actual failures.

Event Management is extensible through its two APIs, so you not only can automatically react to problems elsewhere in the system, but also create your own resource monitors to manage the status of your applications.

For more information on the Event Management subsystem, on writing applications and resource monitors and sample code, refer to *Event Management Programming Guide and Reference*.

5.2 Event Management Using the Perspectives GUI

A good introduction to the concepts of Event Management can be obtained by using the Perspectives interface. Perspectives is a systems management tool with an iconic interface. The main functions it provides are:

- Hardware control and monitoring
- Performance monitoring
- VSD management
- Event monitoring

For more information on Perspectives and how to use it, refer to the *IBM Parallel System Support Programs for AIX: Administration Guide*, GC23-3897 or the Perspectives online help facility.

The Event Perspective interface may be invoked from the main Perspectives window. To invoke the Perspectives interface, type the following on the command line:

```
perspectives &
```

This assumes that /usr/lpp/ssp/bin is in your PATH. The window shown in Figure 12 will be displayed:

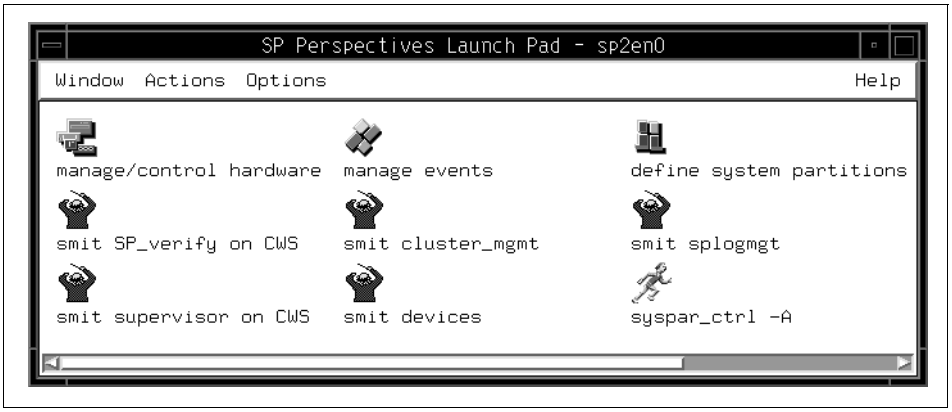
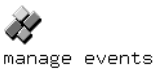


Figure 12. SP Perspectives Launch Pad

Double-click the manage events icon:



The Event Perspective window, shown in Figure 13, will be displayed:

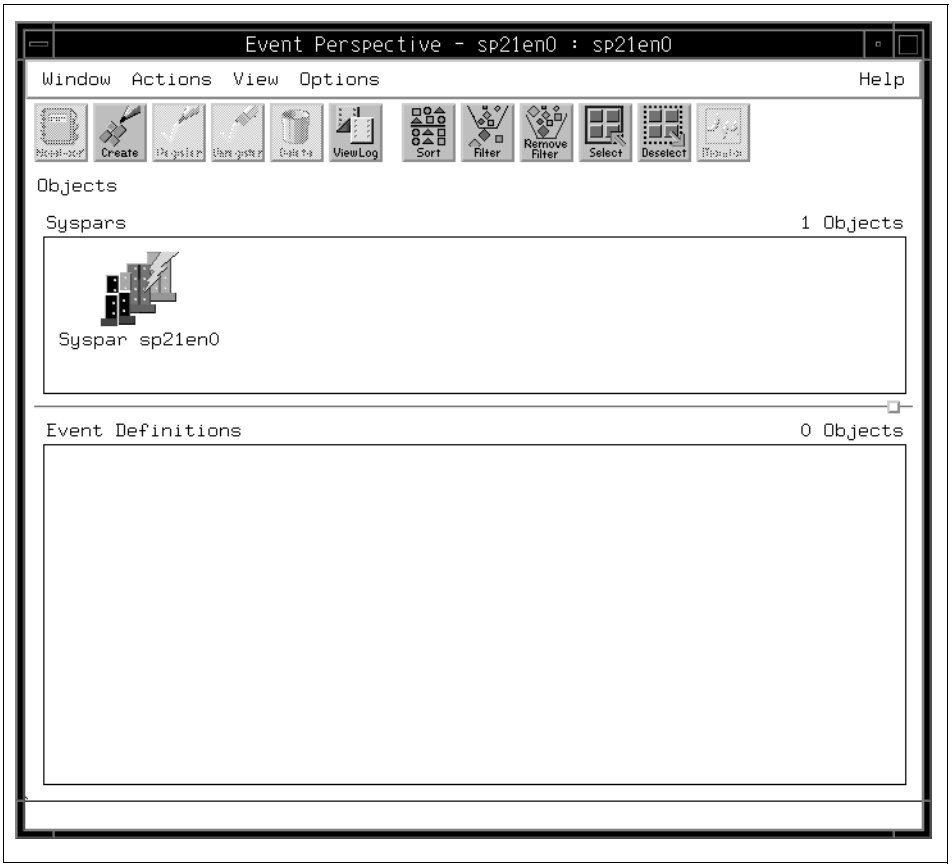


Figure 13. Event Perspective

Alternatively, you can invoke the Event Perspective directly from a window by typing the following on the command line:

```
spevent &
```

Once again, this assumes that /usr/lpp/ssp/bin is in your PATH. The Event Perspective window will appear as before.

5.3 Event Notification Using Perspectives

This section describes how to use the Perspectives.

5.3.1 Creating a Perspectives Object for Event Notification

To create a Perspectives Object Using Perspectives, do the following:

1. If you have not already started the Event Perspective, do so now by following the instructions in 5.2, “Event Management Using the Perspectives GUI” on page 75.
2. If the Event Definitions pane is not already highlighted, click the left mouse button in the pane to highlight it. This enables the Create button near the top of the window.
3. Click **Create** . The Create Event Definition Window shown in Figure 14 on page 78 appears.

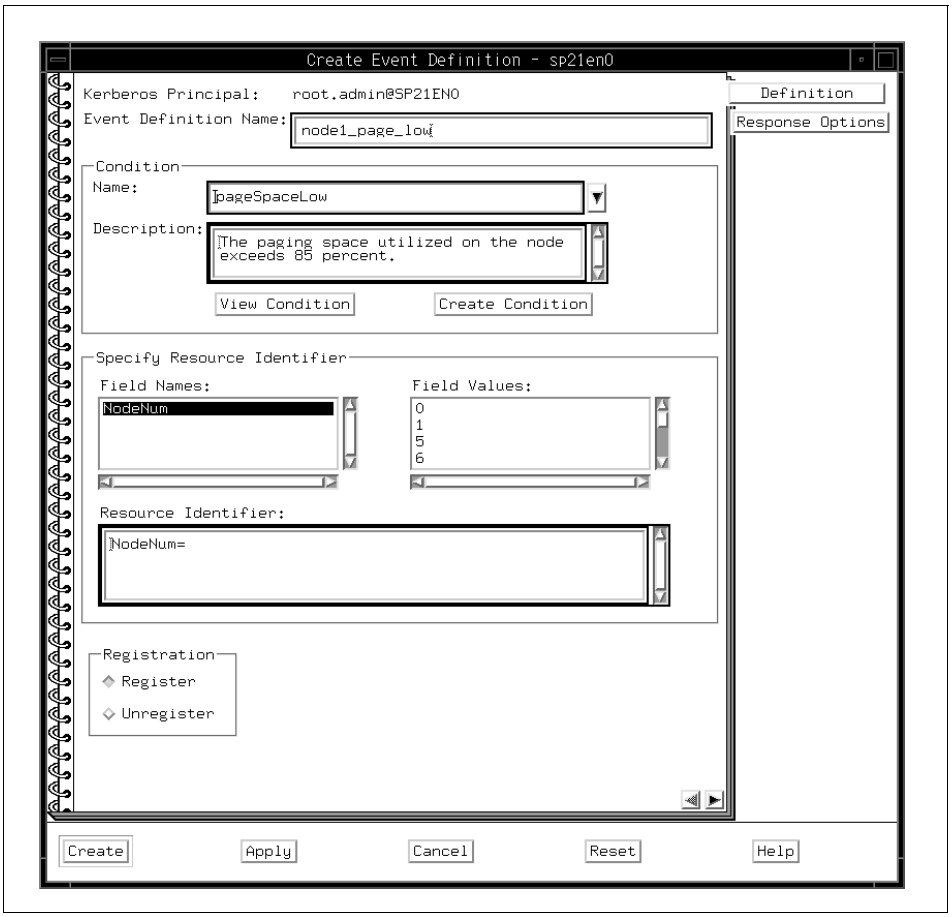


Figure 14. Create Event Definition

4. Select the Condition that you wish to monitor.

   If you do not know the name of the condition, press the down arrow to display the list of known conditions. Use the scroll bar to scroll through the list.

   Select the condition, for example **pageSpaceLow**, from the list by clicking on it to highlight it. The Description box will show a description of the condition. Verify that the description describes that which you wish to monitor for.

   If you cannot find a condition that meets your requirements, see 5.5, “Creating a New Condition” on page 85.

5. Specify the resource that you wish to monitor for the condition you defined. For example, NodeNum (the node number). The Resource Identifier box will show the identifiers that you have selected.
6. Select **Register** to register the event with Event Management.
7. Finally, type the name that you wish the event to be known in the Event Definition Name text entry box. Always select a meaningful name that will assist, rather than hinder, systems management activities. For example: node1_pgsp_low.
8. Press the Create button near the bottom of the window to create an Event Definition object that will allow you to monitor the condition specified in the definition. An icon for the event definition will appear in the Event Definition pane. It will look like one of the icons shown in Figure 15:

If the event definition is in error, you will need to modify your definition until it is correct. To change the event definition, double click on the event definition icon, and the Event Definition window will open.

Figure 15. Event Icons

5.3.2 Monitoring for Event Notifications Using Perspectives

When the Perspectives interface is running, the Event Definitions will be monitored provided they are registered with Event Management. When the defined conditions of an Event Definition are met, the registered icon will change to look like this:

To receive information about the event, double click on the changed icon. The window shown in Figure 16 on page 80 will display:

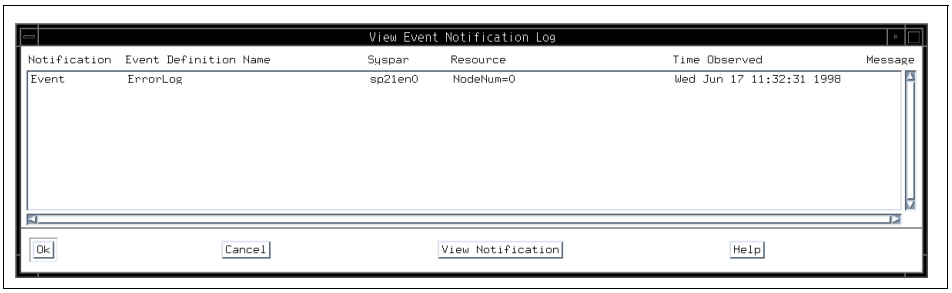


Figure 16. View Event Notification Log

This is the Event Notification Log. To receive information about the specific event, double-click the line related to the event definition, and the Event Notification screen shown in Figure 17 on page 81 will display.

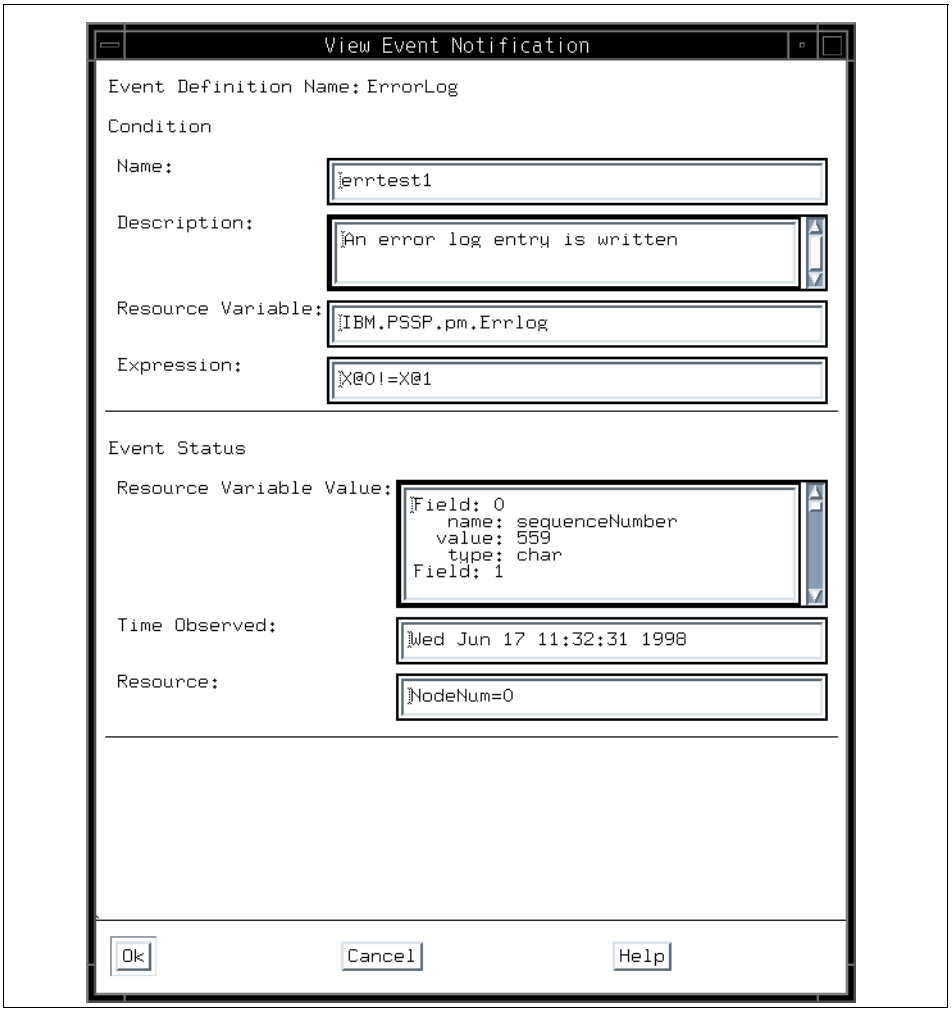


Figure 17. View Event Notification

Notification that an event has occurred is only the starting point. To be useful for automating operations, it has to be possible to trigger an action when an event occurs.

5.4 Triggering Actions from Events Using Perspectives

This section describes how to create a Perspectives object to react to an event.

5.4.1 Creating a Perspectives Object to React to an Event

1. If you have not already started the Event Perspective, do so now by following the instructions in 5.3, “Event Notification Using Perspectives” on page 77.
2. If the Event Definitions pane is not already highlighted, click the left mouse button in the pane to highlight it. This enables the Create button near the top of the window.
3. Press the Create button. The Create Event Definition Window shown in Figure 14 on page 78 appears.
4. Select the Condition that you wish to monitor.

   If you do not know the name of the condition, press the button with the down arrow to display the list of known conditions. Use the scroll bar to scroll through the list.

   Select the condition, for example, **pageSpaceLow**, from the list by clicking on it to highlight it. The Description box will show a description of the condition. Verify that the description describes what you wish to monitor for.

   If you cannot find a condition that meets your requirements, see 5.5, “Creating a New Condition” on page 85.
5. Specify the resource you wish to monitor the condition you defined. For example: NodeNum (the node number). The Resource Identifier box will show the identifiers that you have selected.
6. Select **Register** to register the event with Event Management.
7. Finally, type the name by which you wish the event to be known in the Event Definition Name text entry box. Always select a meaningful name that will assist, rather than hinder, systems management activities. For example: node1_pgsp_low.
8. To define the actions to be run when the event conditions are met, select the **Response Options** tag near the top of the window on the right hand side. The second page of the Create Event Definition screen shown in Figure 18 on page 83 is presented.

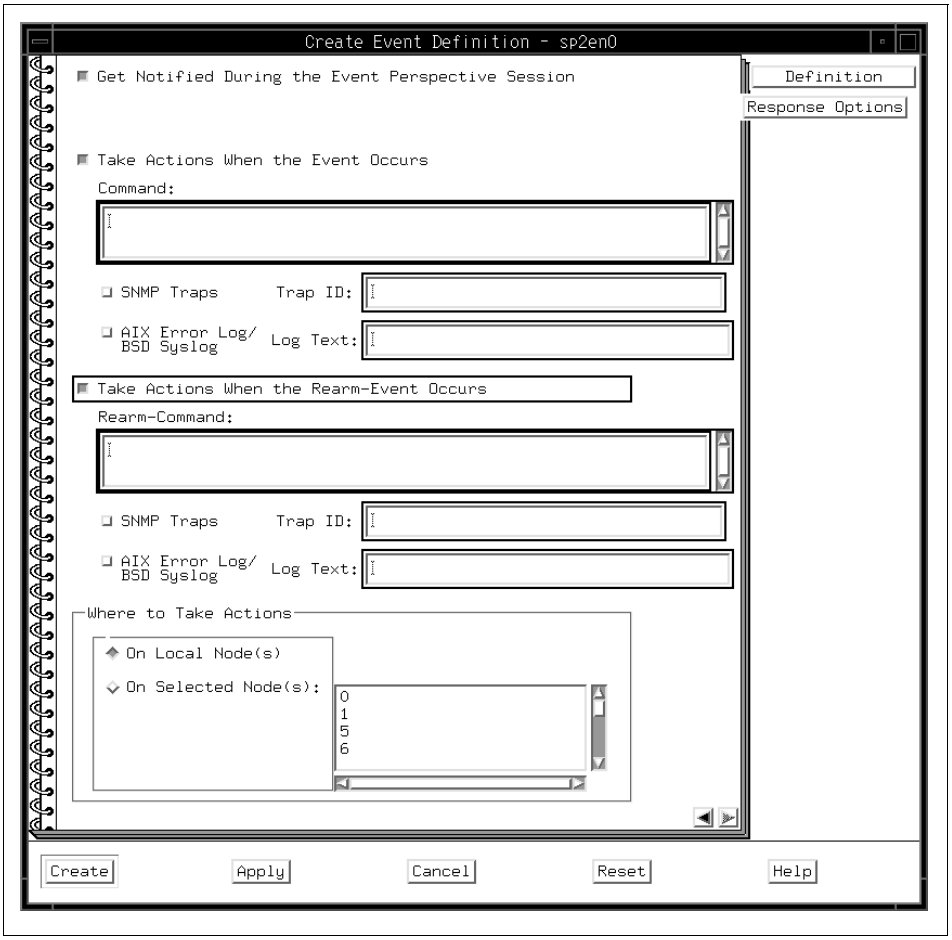


Figure 18. Response Options

9. Define the actions you wish to run when the event occurs:
   - If you simply wish to be notified of the occurrence of the event where you are running the Perspectives interface, you do not need to do anything more than ensure that the "Get Notified During the Event Perspective Session" button is on. It is on by default.
   - If you wish an action to be run when the event occurs, select **Take Actions When the Event Occurs** . The Command box is enabled, and you can enter the command to invoke the recovery action in this box. This is typically, though not necessarily, the full pathname of a shell script. For example, if we wanted to stop the AppA application when the event

occurs, the Command box would contain something like /usr/local/bin/stop_AppA, which is the command to stop the application.

- If you wish an action to be run when the rearm event occurs, select **Take Actions When the Rearm Event Occurs** . The Rearm-Command box is enabled, and you can enter the command to invoke the rearm action in this box. This is typically, though not necessarily, the full pathname of a shell script. For example, if we wanted to restart the AppA application when the event occurs, the Command box would contains something like /usr/local/bin/start_AppA, which is the command to start the application.
- You can also specify whether you want SNMP traps, or AIX error log, or syslog entries to be written by selecting the relevant buttons and entering the required information into the relevant fields.

  For example, if you enable AIX error logging or syslogging of errors and type the following text in the panel:

  ```
  TMPFULL1-TMP FULL EVENT HAS OCCURRED!
  ```

  then when the error occurs, the following will be logged in the error log:

  ```
  LABEL:HA_PMAN_EVENT_ON
  IDENTIFIER:      E460E36E

  Date/Time:       Wed Oct  8 04:38:55
  Sequence Number: 1005
  Machine Id:      00034173A000
  Node Id:         ctrl_ws
  Class:           S
  Type:            UNKN
  Resource Name:   pmand

  Description
  MONITORED SITUATION EXISTS

  Probable Causes
  MONITORED EVENT

  Failure Causes
  MONITORED EVENT
                   Recommended Actions
                   REVIEW DETAILED DATA

  Detail Data
  NAME
  tmperr1
  Node Number
  1
  ```

```
RESOURCE TYPE
IBM.PSSP.aixos.FS.%totused
RESOURCE NAME
VG=rootvg;LV=hd3;NodeNum=1
SURPASSED THRESHOLD VALUE
X>90
DESCRIPTION
TMPERR1-TMP FULL EVENT HAS OCCURRED!
```

Simultaneously, the following will be logged by syslog. The actual location where this message has been sent will depend upon the definitions in your /etc/syslog.conf file.

```
Nov 21 17:29:56 ctrl_ws pmand[14088]:
SP Problem Mgmt: Monitored Situation Exists:
Name=tmperr1
Node Number=0Resource Variable=IBM.PSSP.aixos.FS.%totused
Instance Vector=VG=rootvg;LV=hd3;NodeNum=0
Predicate=X>90
Description=TMPERR1-TMP FULL EVENT HAS OCCURRED!
```

- Finally, you should select the cluster node where these actions should be run. For example, to run the actions on the control workstation (the node with a node number of 0), select **On Selected Nodes** and highlight **0** in the list of nodes.

10. Pressing the Create button will now add the event with its associated actions to the configuration and register it with Event Management. Pressing the Create button near the bottom of the window will create an Event Definition object. An icon for the event definition will appear in the Event Definition pane.

5.5 Creating a New Condition

Many event conditions have been defined on the system. However, if the event condition you wish to monitor does not exist, a new condition can easily be created. From the main spevent window, highlight the **Event Definitions** pane and press **Create** to create a new event. The Create Event Definition window appears. From this window, select **Create Condition** . The Create Condition window shown in Figure 19 on page 86 appears:

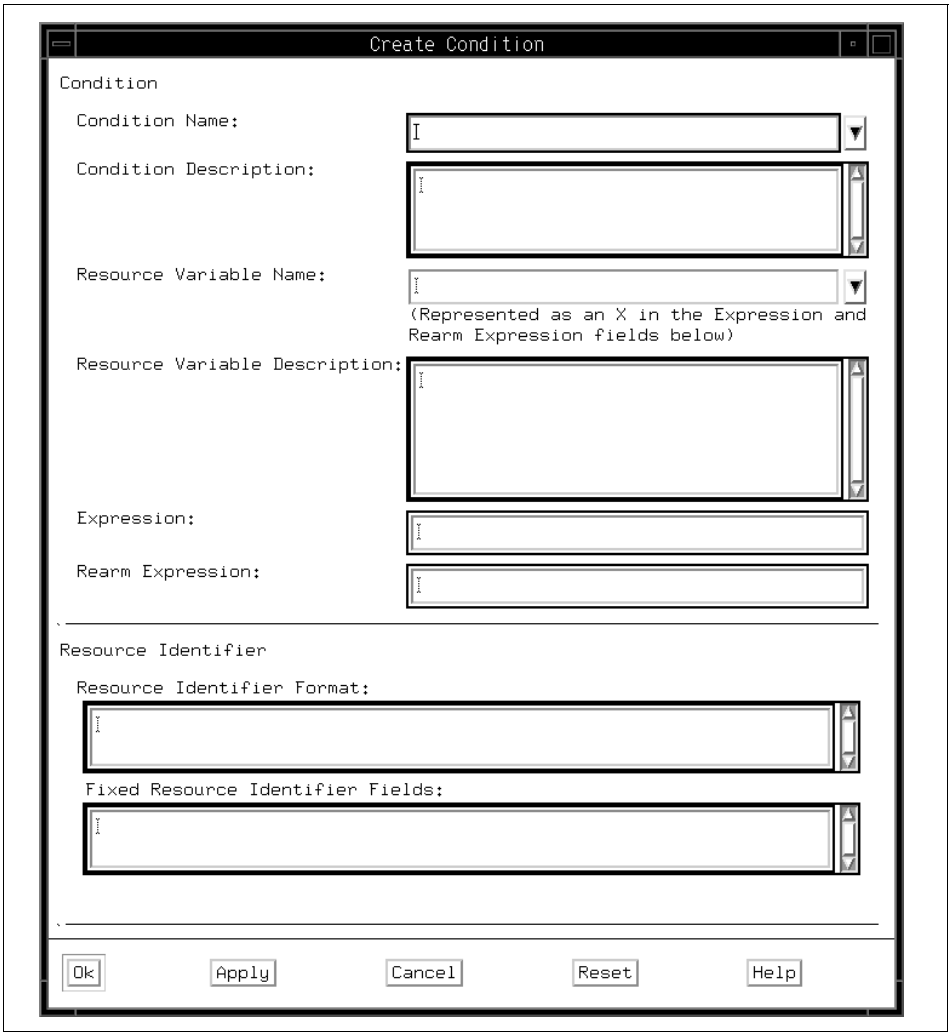


Figure 19. Create Condition

To create a new condition, perform the following steps:

1. Enter the Resource Variable Name, for example IBM.PSSP.SampleCmdMon.state

   If you do not know the name of the resource variable or cannot remember the complete resource variable name, pressing the down arrow will reveal a selection list. This may be viewed using the scroll bar. Alternatively, typing the individual characters in the Resource Variable Name field will

cause the highlight value to change, allowing you to find the resource variable if you know part of its name.

The Resource Variable Description field will contain a description of the resource variable. This should describe the condition you are defining. When the Resource Variable is selected, other fields, including the Resource Identifier fields, will now be filled in using information from the SDR if it is available. This can be changed if it is incorrect.

2. Enter the Expressions that describe the values of the Resource Variable to trigger the event and rearm the monitor. Note that Perspectives uses the term "expression" where the rest of Event Management uses the term "predicate". For more information on predicates, refer to *IBM Parallel System Support Programs for AIX: Event Management Programming Guide and Reference,* SC23-3996.
3. Type the name by which the condition will be known in the Condition Name field. This will be used to select conditions from the Event Definition screen, so it should be descriptive of the function of the condition.
4. Type some descriptive text in the Condition Description box to describe the function of the condition.
5. Press OK to create the condition. The create condition window disappears, and you are returned to the Create Event Definition window.

For more information on using the Create Condition window, see the Perspectives online help.

5.6 Using the Event Perspective for Notification and Recovery

Perspectives provides a simple-to-use interface for both defining error conditions and creating error definition objects that apply these conditions to specific resources. The error definition objects allow for the detection of events and allow actions to be triggered upon receipt of an event. These actions allow for scripts or programs to be run on one or more nodes of the cluster. They also allow for logging of errors to the AIX error log and/or syslog and for SNMP traps to be generated.

This may seem to provide all of the required functionality, and indeed in many cases this will be true. However, there are a few areas where Perspectives may not be the best tool for the job because it does not provide all of the necessary functionality, as described in the following sections.

5.6.1 Event Monitor or Event Configuration Tool

The first decision that should be taken regarding the Event Perspective is how it is intended to be used within your environment. You can use spevent as a systems management interface for events, as a configuration tool for configuring event or action pairs, or as a combination of the two. Whichever approach you decide upon, the number of event definitions should be kept reasonably low. This is not because of any underlying restriction in its capabilities, but rather because too many event definitions make the interface overly complex to manage and use. As the number of event icons increases, it becomes increasingly difficult to find the particular event definitions you are looking for.

This is especially true if you intend to be using the Event Perspective within a larger systems management framework or as a standalone event monitor. If this is the case, the number of event definitions should be limited only to those key events that are critical to the correct operation of your system.

A potentially better use of Perspectives is as a simple configuration tool for user events. In this environment, the number of event definitions matters less because they are not being actively managed. However, as before, when it comes to changing or updating an existing event definition, having a large number of them will result in the desired one being harder to find.

5.6.2 Using the Event Perspective as an Event Monitor

If you intend to use spevent primarily as an Event Monitor, you should ensure that each event definition has a rearm predicate in addition to the event predicate. If you do not do this, the event definition is very limited in function. Once triggered, the event definition cannot process further changes in the state of a resource until it is reset, either by restarting the spevent monitor or, potentially, following a system reboot or similar restart.

Besides making the spevent unmanageable, a very large number of event definitions will also impact the speed with which event notifications are received. This is especially true if there are multiple instance vectors in existence for a single resource variable.

The behavior of the spevent perspective is not necessarily intuitive when it comes to handling an event. When an event is triggered, the event definition icon changes, and the Event Notification Log window appears.

If there is a highlight bar in this window, this is not necessarily placed on the event that has just triggered. Consequently, it may mislead an untrained operator. The most recent event is to be found at the top of the screen.

In complex environments, however, where there is a lot of event activity, the event that you think has caused the Event Notification Log to be displayed may not in fact be at the top, having been superseded by subsequent events.

If this occurs frequently, a review needs to be done of the number of events defined or the conditions used to trigger notification, because the values chosen make the use of Perspectives impractical within environments such as these.

Double-clicking on the icon of a triggered event definition will display the Create Event Definition screen and will not take you to the Event Notification log, as might be expected.

5.6.3 Using the Event Perspective as an Event Configuration Tool

While the Event Perspective can perform a useful systems management role in certain systems, its use as an event and condition configuration tool is probably appropriate.

Events configured with Perspectives are only monitored where a Perspectives session is running. If Perspectives will not be running at all times, or if event definitions are required to be monitored when Perspectives is inoperative, the event definitions should be configured using the SP problem management subsystem.

The first thing that has to be understood is that the Event Perspective relies on resource monitors. If you do not have a resource monitor that can provide the necessary information to allow you to monitor your critical resources, then the Event Perspective interface loses some of its appeal. That said, there are a number of resource monitors provided that cover a large proportion of the common functions you are likely to need. In addition, although it requires additional effort, you can create resource monitors yourself that provide any additional function required within your environment.

When creating conditions and event definitions, there are several alternatives. These relate to where the instance vectors are defined.

Consider the following example. To measure how much space has been used in the */*tmp file system, the IBM.PSSP.aixos.FS.%totused resource variable would be used. This could be configured in a couple of different ways. The first would be to perform most of the configuration in the condition. In this case, you might configure a condition, tmpFull, with the Fixed Resource Identifiers thus:

```
LV=hd3;VG=rootvg
```

In this case, the Event Definition only has to identify the nodes of interest; that is:

```
NodeNum=3
```

Contrast this with the creation of an fsFull condition with no Fixed Resource Identifiers. The Event Definition in this case has to provide all of the necessary information for the instance vector.

```
NodeNum=3;LV=hd3;VG=rootvg
```

While at first sight there might appear to be no difference between these two alternatives, the former provides for greater control and manageability than the latter. There is less chance of making mistakes when predefining most of the instance vector within the condition. Consider also a situation where you have a different set of predicates for different types of file systems. For example, 90% for /tmp, 80% for /var, and 80% for /. In this environment, creating three different conditions, tmpFull, varFull, and rootFull, where each has the correct predicate, is more easily understandable than having three variants on fsFull with different predicate settings.

Finally, although the Event Perspective can run recovery events on multiple nodes, there is no simple mechanism for synchronizing the actions of these scripts. This effectively limits Perspectives to simple multinode (Class III) events. If you need this level of synchronization, then you will need to use HACMP and its recovery programs, which provide barrier synchronization protocols.

5.6.4 Invoking a Recovery Action from an AIX Error Log Entry

As discussed in Chapter 5.6.3, “Using the Event Perspective as an Event Configuration Tool” on page 89, the Event Perspective relies on a resource monitor to act as a trigger. If no resource monitor exists for the condition you wish to monitor, then it can be difficult to harness the power of the Perspectives interface.

The AIX error notification facility allows an error notification method to be invoked when an error occurs. Event Management, and hence Perspectives, can be informed that this error has occurred using the errlog_rm notification method (this, more accurately, is a resource monitor). Using this mechanism, a Perspectives recovery action can be triggered from an error being written to the AIX error log.

You can monitor errlog_rm by using the IBM.PSSP.pm.Errlog resource variable. This has a single instance vector, NodeNum, which specifies the cluster node that you wish to monitor.

If monitoring of any error log activity is required, a condition needs to be defined using the spevent perspective with an expression (predicate) of:

```
X@0!=X@P0
```

This is unlikely to be of any use unless spevent is being used solely as a monitor for these events. If an error definition was to be created using a condition based upon this predicate, any action associated with this error definition would be run whenever an event was written to the error log.

A better method would be to define a predicate in the condition that triggers only when a specific event that is of interest occurs. A Predicate providing this would look like the following:

```
X@0!=X@P0 && X@1=="0x91f9700d"
```

This predicate illustrates several interesting things. First, the `X@0!=X@P0` part of the predicate ensures that an event is triggered only when a new event arrives. In this case, `X@0` refers to a field in the Structured Byte String (SBS) that contains the sequence number of the error that occurred, and `X@1` refers to the field of the SBS that contains the error ID of the error, while `91F9700D` is the error ID for the error `LVM_SA_QUORCLOSE`, indicating a loss of quorum for a volume group.

Having defined the condition with this event predicate, the condition can then be used to define an event definition.

This predicate could also have been written as:

```
X@0!=X@P0 && X@8=="LVM_SA_QUORCLOSE"
```

where X@8 refers to the field in the SBS that contains the error label of the error that was generated. To identify the error label or error ID for a particular error, use the `errpt` command with the **`-t`** flag. This lists the error record templates that have been installed on the system. For example, to get a list of all of the errors generated by SCSI devices, enter the command:

```
errpt -t | grep SCSI
```

The resulting output will look similar to:

```
038F2580  SCSI_ERR7   UNKN  H   UNDETERMINED ERROR
0502F666  SCSI_ERR1   PERM  H   ADAPTER ERROR
0BA49C99  SCSI_ERR10  TEMP  H   SCSI BUS ERROR
13881423  SCSI_ERR4   TEMP  H   MICROCODE PROGRAM ERROR
4EDEF5A1  SCSI_ERR5   PERM  S   SOFTWARE PROGRAM ERROR
52DB7218  SCSI_ERR6   TEMP  S   SOFTWARE PROGRAM ERROR
54E423ED  SCSI_ERR9   PERM  H   POTENTIAL DATA LOSS CONDITION
```

```
5CC986A0  SCSI_ERR3   PERM  H   MICROCODE PROGRAM ERROR
C14C511C  SCSI_ERR2   TEMP  H   ADAPTER ERROR
```

The error ID for a failure is the first field in a line entry. The error label is the second field.

For more information on using the IBM.PSSP.pm.Errlog resource variable, see Chapter 8, “Common Monitoring Tasks” on page 123 .

5.6.5 Invoking a Recovery Action from a Shell Script

Just as a Perspectives recovery action may be invoked from an AIX error log entry, this same technique can be used to trigger an event via Perspectives from a shell script by invoking the errlog_rm notification method (resource monitor) directly. An error notification object that invokes errlog_rm has a definition that looks like this:

```
errnotify:
        en_pid = 0
        en_name = ""
        en_persistenceflg = 1
        en_label = ""
        en_crcid = 0
        en_class = ""
        en_type = ""
        en_alertflg = ""
        en_resource = ""
        en_rtype = ""
        en_rclass = ""
        en_symptom = ""
        en_method = "/usr/lpp/ssp/bin/errlog_rm $1 $2 $3 $4 $5 $6 $7 $8
$9"
```

The errlog_rm resource monitor can also be invoked on the command line or from a shell script. Hence, it is perfectly possible to create a monitoring shell script that looks like this:

```
#!/bin/ksh
#
# monitoring shell script
#
while true
do
    /usr/local/bin/test_device
    if (($? != 0))
    then
        /usr/lpp/ssp/bin/errlog_rm ‘/bin/date‘ 1
```

```
        fi
        sleep 60
    done
```

This shell script is rather crude and serves only to illustrate the point rather than being a reasonable example of a monitor. In the Perspectives condition, a predicate of:

```
X@0!=X@P0 X@1=="1"
```

could be used. This is then used within an event definition. The use of the date command to create the first field ensures a unique identifier. `X@0!=X@P0` ensures that the same error will not be retrieved multiple times.

Chapter 6. HACMP Recovery Programs

This chapter introduces the integrated user event functions of the HACMP Enhanced Scalability product. It discusses the configuration of the rules.hacmprd file and introduces recovery programs.

6.1 HACMP and Events

An HACMP cluster is event-driven. The software was initially configured so that the cluster can detect and provide recovery actions for the following cluster topology events:

node_up	A cluster node joins the cluster.
node_down	A cluster node leaves the cluster, either due to a failure or in a controlled fashion.
network_up	A network has become available.
network_down	A network has failed.
swap_adapter	A service adapter has failed.

In addition to these, some reconfiguration events are also defined. These occur when the cluster topology has been changed. Each event has an associated completion event, which is run after the event itself. For example, when the node_down event has finished running, the node_down_complete event is started. These predefined events allow the cluster manager to react to major changes, such as node joins or network failures. However, by themselves, they do not allow the cluster to react to other events such as disk or software component failures. To do this, the cluster requires additional configuration. Traditionally, this has been provided through AIX functions such as error notification. However, it is now possible to use RS/6000 Cluster Technology (RSCT) to provide these functions.

HACMP ES uses RSCT to handle user-defined events. It uses the Event Management subsystem to perform and provide event detection and Group Services to provide synchronization and control facilities for the recovery mechanisms. As was discussed earlier, Event Management receives information about a resource from a resource monitor. HACMP ES is an Event Management client in that it registers with Event Management to receive information about the resource conditions that it is interested in. When the event predicate evaluation matches those conditions that have been defined, the Event Management subsystem generates an event. This is passed to HACMP, which then invokes the appropriate recovery program for

that event. If the event program requires actions to be performed on multiple nodes in the cluster, Group Services provides a barrier synchronization protocol for the recovery programs.

6.2 The /usr/sbin/cluster/events/rules.hacmprd File

Events are mapped to their respective recovery programs in the rules.hacmprd file. This is located in the /usr/sbin/cluster/events directory. If the event is predefined to HACMP, notification will come from Group Services. If the event is user-defined, then notification will come from the Event Management subsystem. In both cases, the entry in the rules.hacmprd file is of the same format as follows:

1. Name
2. State (qualifier)
3. Recovery program path
4. Recovery type (reserved for future use)
5. Recovery level (reserved for future use)
6. Resource variable name (used for Event Manager events)
7. Instance vector (used for Event Manager events)
8. Predicate (used for Event Manager events)
9. Rearm predicate (used for Event Manager events)

All event definitions that are added to the rules.hacmprd file must consist of exactly nine lines. Comments (lines beginning with a #) are not counted. If there is no value for a line, then the line should be left blank rather than removing or ignoring it. If the event definition is not in this format, the system will hang when the cluster manager is started, and error messages similar to the following will be written to the console:

```
Oct 13 05:15:19 node1 clstrmgr[8576]: Mon Oct 13 05:15:19 RdInit: Bad
format in rules file at line 223
Oct 13 05:15:19 node1 clstrmgr[8576]: Mon Oct 13 05:15:19 RdInit error
Oct 13 05:15:19 node1 clstrmgr[8576]: Mon Oct 13 05:15:19 hacmprd on
node 0 is exiting with code 3
```

6.2.1 Predefined Events

Entries for the following predefined events (those that are triggered from Group Services) exist in the rules.hacmprd file by default:

- node_up

- node_down
- network_up
- network_down
- swap_adapter
- join_standby
- fail_standby
- reconfig_topology
- reconfig_resource

A typical event definition that is triggered by a Group Services notification might look like this:

```
##### Beginning of Event Definition  Node Up ################
#
TE_JOIN_NODE
0
/usr/sbin/cluster/events/node_up.rp
2
0l
#6) resource variable name (Used for Event Manager events)

#7) Instance vector, only used for event manager events

#8) Predicate, only used for event manager events

#9) Rearm predicate, only used for event manager events

##### End of Event Definition       Node Up  ######
```

These predefined events provide the necessary functionality to allow HACMP to protect against node, network and network adapter failures, as well as allowing dynamic reconfiguration to function.

6.2.2 User-Defined Events

There are no user-defined events in the rules.hacmprd file as it is shipped. Any events found in this file that are not on the predefined list above were added by someone. User events have the same format, but a different layout, because they are triggered from the Event Management subsystem.

A typical event definition triggered by Event Management might look like this:

```
##### Beginning of Event Definition  Temp Directory Full ######
#
```

```
TEMP_FULL
0
/usr/local/events/tmp_full.rp
2
0
IBM.PSSP.aixos.FS.%totused
NodeNum=*;LV=hd3;VG=rootvg
X>90
X<80
##### End of Event Definition  Temp Directory Full ######
```

The difference between events triggered by Event Management and those triggered by Group Services is the additional definition of a resource variable and a set of conditions.

Note that in both of the above sample event definitions, there are exactly nine lines between the lines defining the beginning and the end of the event definition.

6.2.3 Event Handling by HACMP

When an event occurs, the following steps are followed:

1. Group Services or Event Management notifies the cluster manager. Group Services handles the predefined events, such as node or network events, whereas Event Management takes care of user-defined events.
2. The cluster manager reads the rules.hacmprd file, which maps recovery actions to events.
3. The cluster manager runs the recovery program.
4. The recovery program in turn runs a series of recovery commands. These are potentially run simultaneously on multiple cluster nodes and kept synchronized by barriers.
5. The cluster manager receives return status from the recovery commands.

6.3 Recovery Programs

This section describes recovery programs.

6.3.1 Recovery Command Specifications

The third line of the event definition specifies the recovery program that is to be run when the user-defined event is triggered. The recovery program is not a shell script but a sequence of recovery command specifications. Each specification is of the following format:

node_set recovery_command expected_status NULL

where the values on the line are separated by spaces.

The nodeset specifies which nodes should run the recovery command. This can take one of three values:

all All cluster nodes run the recovery command.

event The node where the event occurred runs the recovery command.

other All nodes *except* the one where the event occurred run the recovery command.

recovery_command is a quote-delimited string containing the full pathname of the command to be executed.

expected_status is the successful return code of recovery_command.

When recovery_command is run, the return code is compared to the expected_status value. If the values do not match, recovery_command has failed, and the cluster manager generates an event error. The handling of the event will then hang until fixed by manual intervention. The node on which recovery_command has failed will not hit the barrier, and consequently, all other nodes that are participating in running the event wait at the barrier. If the return code is not important, or you do not wish to make a comparison, specify an X as the value.

NULL is, at this time, a reserved field for future use. The word NULL should appear at the end of the line.

6.3.2 Barriers

Recovery program specifications may be separated by barriers. These are denoted by the keyword "barrier". They provide synchronization points for all of the commands prior to the barrier. When a node hits the barrier, the cluster manager on the node initiates the barrier protocol. When all nodes have met at the barrier and have voted to approve the protocol, the next phase of the recovery program is executed.

A recovery program might look like this:

```
# Recovery Program 1
#
event "/usr/local/bin/restart_server" 0 NULL
other "/usr/local/bin/reconnect_clnt" 0 NULL
#
```

```
barrier
#
all "/usr/local/bin/restart" 0 NULL
```

In this example, when the event occurs, the node that detected the event runs the /usr/local/bin/restart_server script. At the same time, the other cluster nodes run the /usr/local/bin/reconnect_clnt command. Both of these commands have a zero return code when they successfully complete. If a non-zero return code is received from any of the cluster nodes, the recovery program will hang and, after a period of time, the config_too_long event will be triggered by the cluster manager. Upon the successful completion of the /usr/local/bin/reconnect_clnt commands, the other nodes wait at the barrier until the /usr/local/bin/restart_server command, which takes much longer to execute, also completes. Once all cluster nodes have reached the barrier, after voting to approve the barrier protocol, they then all simultaneously execute the **/**usr/local/bin/restart command.

6.3.3 Passing Information to Recovery Commands

When a recovery program runs, the following environment variables are set for the recovery_command:

- **EVRVNAME** - the resource variable of the event, that is, IBM.PSSP.Prog.xpcount
- **FIELD_2**=
- **FIELD_1**=1
- **FIELD_0**=0
- **EVNAME** - the name of the event (from the rules.hacmprd file, UE1)
- **EVENT_NODE** -=1
- **EVRV**=SBS
- **COORDINATOR**=1
- **MEMBERSHIP** - the cluster membership when the event was generated; that is, 1 2
- **EVLOCATION** - the node where the event was generated; that is, 1
- **PWD**=/
- **TIMESTAMP** - the time when the event was generated; that is, Mon Oct 13 11:37:02 1997

These variables can be used to pass information about the event to the file system recovery command to allow more complex recovery to occur.

6.4 Adding User Events to a HACMP Configuration

This section describes how to add user events to a HACMP configuration.

6.4.1 User Events That Have a Specific Resource Monitor

Many events can be monitored by a defined resource monitor. It is simple to add these events to the rules.hacmprd file, and hence, to the HACMP configuration. To register the cluster manager for interest in one of these events, you need only to edit the rules.hacmprd file to include an entry for the event. This defines the resource variable, the instance vector for the resource, and the event and rearm predicates.

For example, assume that you wish to monitor the amount of free space in the root file system on node 2 of the cluster and take an action when this gets less than 10%. In this case, the event that you are interested in can be monitored by a specific resource monitor. The resource variable is IBM.PSSP.aixos.FS.%totused, and the instance vector for the / file system on cluster node 2 is NodeNum=2;LV=hd4;VG=rootvg. This then gives an event definition added to the rules.hacmprd file that looks like this:

```
##### Beginning of Event Definition  node2root Directory Full ######
#
NODE_2_ROOT_FULL
0
/usr/local/events/root_full.rp
2
0
IBM.PSSP.aixos.FS.%totused
NodeNum=2;LV=hd4;VG=rootvg
X>90
X<80
##### End of Event Definition  node2root Directory Full ######
```

Other entries for other resources/resource monitors can be added in a similar fashion.

6.4.2 User Events That Do Not Have a Specific Resource Monitor

Adding new events that do not have specific resource monitors is more complex. The recovery program can only be triggered by a resource monitor; and therefore, there needs to be an alternative mechanism to detect the event. For example, through the AIX error log or syslog, and then use this to trigger the recovery program through the IBM.PSSP.pm.Errlog resource monitor.

Assuming that you have a mechanism to write to the error log, whatever that might be, you can trigger a HACMP recovery program on this event using errlog_rm.

The first example is where a recovery program is triggered directly from an error received by a device driver. Consider a cluster where each cluster node has a single X.25 adapter providing a communications service. When this adapter fails, the service needs to fail over to another system. The error that we wish to trap can be found using `errpt -t`. The error id is:

```
57797644
```

The entry in the rules.hacmprd file will look like this:

```
##### Beginning of Event Definition x25_failure ######
#
X25_ADAPTER_FAIL
0
/usr/local/cfg/events/kill_node.rp
2
0
IBM.PSSP.pm.Errlog
NodeNum=*
X@0!=X@P0 && X@1=="0x57797644"

##### End of Event Definition  x25_failure ######
```

When an error with an error id of 57797644 is logged to the error log on any cluster node, the `kill_node Recovery Program` will be run. This recovery program looks like this:

```
# kill_node Recovery Program
#
event "/etc/halt -q" 0 NULL
```

The second example is one where the recovery program is again triggered from the errlog_rm resource monitor, but rather than the error coming directly from a device driver or application, the resource monitor is invoked directly by the shell script performing the monitoring function. The cluster consists of two nodes, with the active node running an application. Because of the way the application functions, it cannot be restarted on the same node when it fails. Consequently, the service needs to fail over to the other system in the cluster. The application is monitored by the following shell script, which is run periodically by cron:

```
#!/bin/ksh
# Appmonitor
#
```

```
/bin/ipcs -s| grep suppsvr > /dev/null
if (($? != 0))
then
    /usr/lpp/ssp/bin/errlog_rm AA BB CC
    exit 0
fi
exit 0
```

The corresponding entry in the rules.hacmprd file that triggers the recovery program looks like this:

```
##### Beginning of Event Definition suppsvr_failure ######
#
SUPPSVR_FAIL
0
/usr/local/cfg/events/kill_node.rp
2
0
IBM.PSSP.pm.Errlog
NodeNum=*
X@0!=X@P0 && X@1=="BB"

##### End of Event Definition  suppsvr_failure ######
```

The `kill_node Recovery Program` looks like this:

```
# kill_node Recovery Program
#
event "/etc/halt -q" 0 NULL
```

Chapter 7. Other Event Utilities and Functions

SP Problem Management is a function of the IBM PSSP software that is used on the RS/6000 SP. If you are not using an SP system, you will not have this functionality available to you.

This chapter describes the other utilities and functions that are useful within clustered environments for either detecting the occurrence of events or running recovery actions.

In addition to the major functions discussed previously, there are several other utilities or functions that can be utilized within a clustered environment to provide either enhanced error detection or recovery functions. While the facilities offered are not necessarily as key as those provided, for example, by AIX with its error logging functionality, those provided by PSSP with the Event Perspective, or HACMP and its recovery programs, they may still have an important role to play in certain clusters.

This chapter covers the following tools and utilities:

- Cluster Single Point of Control (CSPOC)
- The SP Problem Management Subsystem (pman)
- Using IBM.PSSP.pm.User_State resource variables

7.1 The Role of CSPOC in Multisystem Event Recovery

The Cluster Single Point of Control (CSPOC) facility is provided to assist with the administration of a HACMP cluster from a single point. When it is invoked, CSPOC automatically performs administrative tasks on the nodes in a cluster. If CSPOC were not present, a system administrator would have to execute each task separately on each node in the cluster. This causes additional administrative overhead and can lead to inconsistencies within the cluster.

By automatically propagating system administration tasks, CSPOC eases the administration of clusters by providing a set of cluster administration commands. The main area of interest from a recovery action perspective, however, is CSPOC's extensibility.

First, there are a few terms that need to be understood. In CSPOC terms, the *source node* is the node on which the CSPOC command is invoked, and the *destination node* is the node on which the CSPOC command executes any underlying AIX commands. Each time any remote execution is invoked, a list of destination nodes is used to determine where to execute the CSPOC

commands. The nodes in these destination lists must be all, or a subset of, the cluster nodes.

7.1.1 Remote Execution Using the CSPOC Execution Language

Any CSPOC command has a set of responsibilities to control the execution of commands on multiple nodes, to collect status and log information, to respond to errors, and so on. Written in a traditional programming or scripting language, there is a large degree of code duplication and significant potential for errors. To resolve this dilemma, CSPOC commands are defined as execution plans in a language called CSPOC Extension Language (CEL), which contains the necessary constructs to handle common tasks with a minimum of programming input.

The two primary constructs used for remote execution of commands that are defined by CEL preprocessor (celpp) are try_serial and try_parallel. The main difference between them manifests itself in the order in which commands are executed and return codes are checked. When using try_serial, the commands in the script are executed on one node at a time, and the checking of return codes, and so on, is interleaved with the command execution. The control flow is as follows:

```
for each node
    initiate command on node
    wait for command to complete
    check status for node and execute except/others clauses
end
```

On the other hand, try_parallel allows commands to be executing on multiple nodes simultaneously, but consequently limits the opportunities to check for errors on one node before executing commands on others. In this case, the control flow is:

```
for each node
    initiate command on each node
end
wait for all commands to complete
for each node
    check status for node and execute except/others clauses
end
```

Use of try_parallel can result in significantly lower command execution times compared to try_serial, but at the cost of requiring more sophisticated error handling and recovery logic.

7.1.2 CSPOC Multinode and Single Node "Recovery Programs"

The following examples show how CSPOC may be used to provide functions similar to a HACMP recovery program, but without the barrier synchronization protocols. The example shows the execution plans for the equivalent of the node_down.rp recovery program. These execution plans could either be prepared in advance, or, in a more advanced environment, the CEL could be processed dynamically. This would avoid the need to maintain multiple, very similar shell scripts on the systems. As a comparison, here is the original recovery program:

```
event "node_down"  0 NULL
#
barrier
#
other "node_down"  0 NULL
#
barrier
#
all "node_down_complete"  X NULL
```

The first execution plan handles the local node. This is assumed to be the node where the event has occurred. This script might be triggered by an error notification method:

```
# Init CSPOC
%include cl_init.cel

THISNODE=‘/usr/sbin/cluster/utilities/get_local_nodename‘
_TARGET_NODES=$THISNODE

%try_serial _NODE _TARGET_NODES
    /usr/local/cspoc/node_down
%end
exit 0
```

The second execution plan handles the remote cluster nodes:

```
# Init CSPOC
%include cl_init.cel

THISNODE=‘/usr/sbin/cluster/utilities/get_local_nodename‘
_TARGET_NODES=$(echo $_TARGET_NODES | sed "s/,*$THISNODE,*//")

%try_parallel _NODE _TARGET_NODES
    /usr/local/cspoc/node_down
%end
exit 0
```

In both of these execution plans, the /usr/local/cspoc/node_down script that is called looks like:

```
#!/bin/ksh
#
# Set up environment
LOCALNODENAME=‘/usr/sbin/cluster/utilities/get_local_nodename‘
export LOCALNODENAME
for param in ‘/usr/sbin/cluster/utilities/cllsparam -n $LOCALNODENAME
do
    export $param
done

# Run event
/usr/sbin/cluster/events/cmd/clcallev node_down graceful
exit $?
```

Finally, this execution plan handles the node_down_complete functions for all cluster nodes:

```
# Init CSPOC
%include cl_init.cel

%try_parallel _NODE _TARGET_NODES
    /usr/local/cspoc/node_down_complete
%end
exit 0
```

where /usr/local/cspoc/node_down_complete looks like:

```
#!/bin/ksh
#
# Set up environment
LOCALNODENAME=‘/usr/sbin/cluster/utilities/get_local_nodename‘
export LOCALNODENAME
for param in ‘/usr/sbin/cluster/utilities/cllsparam -n $LOCALNODENAME
do
    export $param
done

# Run event
/usr/sbin/cluster/events/cmd/clcallev node_down_complete graceful
exit $?
```

7.1.3 Considerations for Using CSPOC

CSPOC offers a set of facilities that, in certain situations, may be of use when implementing recovery actions. It provides a relatively simple mechanism for

executing commands on multiple systems in serial and parallel, plus mechanisms for returning errors and messages to a single point. However, there are a few restrictions that may make it unsuitable for implementing recovery plans in your environment. Some of these are discussed in the following sections.

7.1.3.1 Security Aspects of CSPOC

CSPOC uses the `rsh` command to remotely execute commands on the cluster nodes. Hence, it requires rsh access to the nodes in the cluster. If running without the enhanced security features of HACMP that use Kerberos, the /.rhosts, $HOME/.rhosts, and /etc/hosts.equiv files must be configured appropriately so that the user has the permissions required to execute commands on all the cluster nodes. In addition, the CSPOC commands require any privileges needed to run the underlying AIX system administration tasks. If running the underlying AIX commands requires the user to be root, the CSPOC command will also require the user to be root.

If your environment does not permit the use of rsh, and you are not utilizing the enhanced security features of HACMP, you will not be able to use CSPOC-based recovery actions.

7.1.3.2 Activity Logging and Error Reporting

All warning and error messages are reported in the CSPOC command's standard error and the CSPOC log file. The output from a CSPOC command resides on the node in which the command is invoked. The output consists of the output of the underlying administrative tasks grouped by cluster node. For a CSPOC command executed across three nodes, the output format would be:

```
Node 1:   1st output line from AIX command run on node 1
...
Node 1:   nth output line from AIX command run on node 1
Node 2:   1st output line from AIX command run on node 2
...
Node 2:   nth output line from AIX command run on node 2
Node 3:   1st output line from AIX command run on node 3
...
Node 3:   nth output line from AIX command run on node 3
```

CSPOC also maintains a log file that records information about the execution of each CSPOC command. Again, this file resides on the source node. If a CSPOC recovery command fails, you need a mechanism for parsing any error messages, either out of the log file or from stdout, in order to initiate other recovery actions.

Before executing the underlying AIX commands, a CSPOC command verifies that the following are true:

- All cluster nodes are available, that is, the command is able to rsh to all of the cluster nodes.
- All cluster nodes are installed with HACMP version 4.2 or higher.
- When creating a new object, that the object does not already exist on any of the cluster nodes.

The user has the option of passing a force flag to the CSPOC command, which causes it to skip this verification. However, if the force flag is not set, and any of these conditions exist, the CSPOC stops execution, and the appropriate error message is reported. In a failure state, where potentially one or more cluster nodes on a destination list are unavailable or inaccessible, this verification may cause a CSPOC recovery action to fail. If the force flag is set, this may cause a cluster inconsistency to arise. This could potentially lead to serious problems in the future if left unresolved for any length of time.

If an error occurs because an underlying AIX command fails, the CSPOC command may either issue a warning message and continue execution or issue an error message and stop execution. CSPOC does not attempt to roll back any changes already made to the system. This may leave the cluster in an inconsistent state. It is up to the system administrator to consult the CSPOC log and output in order to determine and perform the appropriate cleanup. This may cause problems where there are complex interactions between cluster nodes.

CSPOC does not prevent more than one CSPOC command from being run at the same time across the cluster. As long as the user has the proper system administration privileges, CSPOC will attempt to execute the command. This can again cause potential inconsistencies.

7.1.4 CSPOC Summary

CSPOC provides useful facilities for running a command on multiple machines; however, there are limitations as to its effectiveness for recovery actions. If a simple command needs to be run on a number of cluster nodes and these commands are independent of one another, CSPOC may offer the facilities that are required. CSPOC does not provide the more comprehensive functions, such as barrier synchronization; hence, where complex interactions occur between nodes, CSPOC is unsuitable for use.

CSPOC also does not provide any form of rollback facilities; whereas the facilities provided by Group Services allow for alternative recovery actions to be followed should an initial attempt at recovery fail. In summary, using CSPOC to drive recovery actions is only advised where the commands being run are simple and independent of each other.

7.2 The SP Problem Management Subsystem (pman)

This section describes the SP Problem Management subsystem.

7.2.1 Basic pman Operations

Just as the Perspectives interface provides a graphical tool for configuring actions that should occur when a given event happens, the set of commands and utilities provided by the SP Problem Management subsystem provides a similar function from the command line. The problem management subsystem is an Event Management client and subscribes to receive notification of an event. When this event occurs, various actions can be run. As with Perspectives, these include:

- Running a command or script
- Issuing an SNMP trap
- Logging the event to syslog/AIX error log

The vehicle by which Problem Management functions is the pmand daemon. Because pmand is a daemon, the events that it has subscribed to receive notifications about are processed, even though the user who subscribed to them originally has logged off. This is in contrast to the event perspective that only monitors for the configured events where Perspectives itself is running.

Events are configured using the `pmandef` command. The following are examples of the `pmandef` command in use:

To configure an event definition that will run a command when the event evaluates

```
pmandef -s error_def1 \
-e "IBM.PSSP.aixos.FS.%totused:NodeNum=1;LV=hd4;VG=rootvg:X>90" \
-c "/etc/chfs -a size=+1 /" -n 1
```

For more information and the full syntax of the `pmandef` command, refer to *PSSP Commands and Technical Reference*.

In the preceding example, when the root file system on node1 exceeds 90% utilization, the `chfs` command is run on node1 to extend the file system size

by one physical partition. Notice that the resource variable, the instance vector, and the predicate are all supplied on the command line. error_def1 is just the name by which the error definition can be managed.

Similarly, if instead of running a command it was determined that an SNMP trap should be generated, the preceding event definition command would look like this:

```
pmandef -s error_def1 \
-e "IBM.PSSP.aixos.FS.%totused:NodeNum=1;LV=hd4;VG=rootvg:X>90" \
-t 12345
```

In this case, when the event occurs a SNMP trap of ID 12345 will be generated.

Finally, if logging of the event to syslog and the AIX error log was required, the `pmandef` command would look like this:

```
pmandef -s error_def1 \
-e "IBM.PSSP.aixos.FS.%totused:NodeNum=1;LV=hd4;VG=rootvg:X>90" \
-i "The root filesystem on node1 is over 90% full"
```

The entry written to the error log would look like:

```
LABEL:                  HA_PMAN_EVENT_ON
IDENTIFIER:             E460E36E

Date/Time:              Mon Dec 15 18:02:43
Sequence Number:        1115
Machine Id:             00034173A000
Node Id:                ctrl_ws
Class:                  S
Type:                   UNKN
Resource Name:          pmand

Description
MONITORED SITUATION EXISTS

Probable Causes
MONITORED EVENT

Failure Causes
MONITORED EVENT
                        Recommended Actions
                        REVIEW DETAILED DATA

Detail Data
NAME
```

```
error_def1
Node Number
1
RESOURCE TYPE
IBM.PSSP.aixos.FS.%totused
RESOURCE NAME
VG=rootvg;LV=hd4;NodeNum=1
SURPASSED THRESHOLD VALUE
X>90
DESCRIPTION
The root filesystem on node1 is over 90% full
```

The Problem Management subsystem relies on the Event Management subsystem, which itself is distributed. As a result, monitoring does not have to be performed on the node containing the resource.

When `pmandef` is run, the event definition is created and stored in the pmandConfig class of the SDR. Once there, it may be queried and manipulated using the standard SDR commands.

7.2.2 HACMP and pman Interactions

The Problem Management subsystem can support execution of recovery actions on multiple nodes. As with a Perspectives error definition, the execution of the recovery actions will not be synchronized between the different cluster nodes unless a synchronization mechanism is provided. If there is a requirement to trigger, for example, a HACMP recovery program from a pman error definition, this may be performed by defining an entry in the rules.hacmprd file to trigger a recovery program based upon error written to the error log.

In practice, however, this would never need to be done. Being driven by a resource monitor, the Problem Management subsystem is driven in the same way as a HACMP recovery program. Consequently, creating a pman definition that writes to the AIX error log, and then using this to trigger a HACMP recovery program, is adding unnecessary complexity. It would be simpler to trigger the recovery program directly from the same resource monitor/instance vector/predicate combination.

For more information on the SP Problem Management subsystem, refer to *IBM Parallel System Support Programs for AIX: Administration Guide,* GC23-3897.

7.3 Using IBM.PSSP.pm.User_State Resource Variables

After using the Event Management system for a while, it will become clear that its real strength lies in its extensibility. However, writing a program to either provide information to Event Management through the Resource Monitor API (RMAPI), or to subscribe to event notifications using the Event Management API (EMAPI), is a fairly major undertaking. Often all that is required is the ability to interact with the Event Management subsystem from a shell script. The Problem Management subsystem provides this functionality through the pmanrmd daemon and its associated set of sixteen resource variables.

The pmanrmd daemon, or Problem Management Resource Monitor daemon, to give it its full name, provides a set of resource variables: IBM.PSSP.pm.User_state1 through IBM.PSSP.pm.User_state16. These resource variables are of type State, in that each fluctuation in the value of the resource variable is significant and must be observed. The data type of the State is a structured byte string (SBS). This has a single field that can contain a character string. The single instance vector for the resource variable is NodeNum, the node number.

The value of the resource variable is set using the `pmanrminput` command. For example, to set the value of the IBM.PSSP.pm.User_state1 resource variable to FAILED, the following command should be issued:

```
pmanrminput -s pman -a "IBM.PSSP.pm.User_state1+FAILED+"
```

This can be detected in any of the usual ways and used as a trigger for a recovery action. For example, to use the SP Problem Management subsystem to detect and trigger an action, the pman error definition would be created by the following command:

```
pmandef -s user_state1 \
-e "IBM.PSSP.pm.User_state1:NodeNum=1:X@0==FAILED" \
-c "/usr/bin/banner failed > /dev/console"
```

If the user_state1 pman error definition object is defined using the preceding `pman` command, then whenever the preceding `pmanrminput` command is run, the word "failed" will be displayed on the console using the `banner` command. This is a trivial example, but serves to illustrate the power of the `pmanrminput` command.

It is also worth explaining these two commands. In the `pmanrminput` command, the two plus symbols (+) are delimiters. In the `pmandef` command, X@0 is

used as a predicate operand because the data type is a structured byte string, zero referring to the first (and only) field in the SBS.

Just as a pman error definition can be triggered from a `pmanrminput` command, a HACMP recovery program may also be triggered in a similar fashion. To trigger a node_down action when the preceding `pmanrminput` command is run, the following definition should be added to the rules.hacmprd file:

```
##### Beginning of Event Definition user_state1_failure ######
#
U_STATE1
0
/usr/sbin/cluster/events/node_down.rp
2
0
IBM.PSSP.pm.User_state1
NodeNum=*
X@0=="FAILED"

##### End of Event Definition  user_state_1_failure ######
```

For more information on the `pmanrminput` command and the SP Problem Management subsystem, refer to *IBM Parallel System Support Programs for AIX: Administration Guide,* GC23-3897.

7.3.1 Creating Script-Based Resource Monitors

Using the mechanisms provided by SP Problem Management, it is a relatively easy task to create simple script-based resource monitors for monitoring complex environments. This is much easier than creating the corresponding C programs to perform the same task.

There are limitations with this approach, chiefly that you are limited to sixteen resource variables of the IBM.PSSP.pm.User_state type, so they should be used appropriately. Also, you are limited to passing character strings, though obviously a character string can contain a numeric value. However, having only a single field in the SBS for each resource variable, if more than one piece of information needs to be passed, you will need to utilize multiple resource variables to do it.

Consider the following example:

In a cluster, a database application instance consists of five processes. These are named:

DB_writer

DB_logger

DB_recov

DB_reader

DB_ctrl

If you were to attempt to monitor this using the IBM.PSSP.Prog.pcount or IBM.PSSP.Prog.xpcount, you would need a set of five instance vectors. These, assuming that we are monitoring on node 5 of the cluster, would look like this:

```
ProgName=DB_writer;UserName=root;NodeNum=5
ProgName=DB_logger;UserName=root;NodeNum=5
ProgName=DB_recov;UserName=root;NodeNum=5
ProgName=DB_reader;UserName=root;NodeNum=5
ProgName=DB_ctrl;UserName=root;NodeNum=5
```

Each of these would need a predicate similar to:

```
X@0 == 0
```

Because the processes interact, monitoring of the entire database subsystem requires a mechanism to link these five predicates together, such that the combination in order to take an action would be:

$X@0_{DB\_writer}==0 \;||\; X@0_{DB\_logger}==0 \;||\; X@0_{DB\_reader}==0 \;||\; X@0_{DB\_ctrl}==0 \;||\; X@0_{DB\_recov}==0$

Event Management does not provide a method for combining the predicates of different resource variables, so this would have to be provided by the implementer. As you can see, this is becoming increasingly complex. Compare this, then, with the following simple shell script:

```
#!/bin/ksh
# Simple DB monitor
#
# count the DB_processes
COUNT=‘ps -ef | egrep DB_ | grep -v egrep | wc -l‘
/usr/lpp/ssp/bin/pmanrminput -s pman -a
"IBM.PSSP.pm.User_state1+$COUNT+"
exit 0
```

This can be run every 5 minutes from cron. Assume that the action that we wish to take when a process dies is to cleanly stop and restart the database. This could be handled on the local node by a Perspectives object, or more likely, a pman object. This would be created using:

```
pmandef -s dbmon \
-e "IBM.PSSP.pm.User_state1:NodeNum=5:X@0!=5" \
-c "/usr/DB/cycle"
```

where `/usr/BD/cycle` is the script that stops the database cleanly and restarts it.

If, on the other hand, the required action was to perform a HACMP node_down on the node and to fail over to another, this could also be achieved by using the preceding script in conjunction with the following entry in the rules.hacmprd file:

```
##### Beginning of Event Definition DB_proc_failure ######
#
U_STATE1
0
/usr/sbin/cluster/events/node_down.rp
2
0
IBM.PSSP.pm.User_state1
NodeNum=5
X@0!="5"

##### End of Event Definition  DB_proc_failure ######
```

This second mechanism will not be as accurate, or as immediate in detecting a failure of a process in the database instance, as creating a mechanism using the IBM.PSSP.prog.pcount resource variable. It will also not handle the situation where a process hangs but remains in the process table. However, for the vast majority of situations, it will provide sufficient accuracy. The ease with which it is created is its main strength.

Consider this second example, which is more complex, but still a trivial, example to illustrate the points under discussion. This uses a single User_state resource variable to monitor multiple different devices and relies on multiple pman objects to handle the different outputs. Once again, this script would be likely to be run periodically by cron:

```
#!/bin/ksh
# Multiple monitor
#
# Check that /usr/local is mounted
```

```
/etc/mount | grep "/usr/local" > /dev/null
if (($? != 0))
then
    /usr/lpp/ssp/bin/pmanrminput -s pman -a
"IBM.PSSP.pm.User_state1+LOCAL_UNMOUNTED+"
fi
# Check that all print queues are available
for test in ‘/usr/bin/lpstat | grep -v Status | grep -v  |
awk"/usr/local"‘
do
    if [[ $test != "READY" || $test != "RUNNING" ]]
    then
        /usr/lpp/ssp/bin/pmanrminput -s pman -a
"IBM.PSSP.pm.User_state1+PRINTQ_DOWN+"
    fi
done
# Check NFS connections to clients
remotes=‘/usr/etc/showmount | wc -l‘
if (($remotes < 5))
then
    /usr/lpp/ssp/bin/pmanrminput -s pman -a
"IBM.PSSP.pm.User_state1+CLNT_NFS_DOWN+"
fi
exit 0
```

This script is then matched to three different pman objects created through the following commands:

```
pmandef -s local_mount \
-e "IBM.PSSP.pm.User_state1:NodeNum=5:X@0!=LOCAL_UNMOUNTED" \
-c "/etc/mount /usr/local"

pmandef -s pqmon \
-e "IBM.PSSP.pm.User_state1:NodeNum=5:X@0!=PRINTQ_DOWN" \
-c "/usr/local/bin/restart_printq"

pmandef -s nfsmountmon \
-e "IBM.PSSP.pm.User_state1:NodeNum=5:X@0!=CLNT_NFS_DOWN" \
-c "/usr/local/bin/client_check"
```

The third, and final example, shows how multiple script based User_state-type resource monitors may be combined in a single monitoring tool to invoke a HACMP fail over by another User_state resource monitor. The first component represents the two script monitors and their corresponding pman object definitions:

```
#!/bin/ksh
```

```
# Multiple resource monitor example - 1
#
# Check that all print queues are available
for test in ‘/usr/bin/lpstat | grep -v Status | grep -v  |
awk"/usr/local"‘
do
    if [[ $test != "READY" || $test != "RUNNING" ]]
    then
        /usr/lpp/ssp/bin/pmanrminput -s pman -a
"IBM.PSSP.pm.User_state1+PRINTQ_DOWN+"
    fi
done
exit 0
```

This is matched with the pman object created using the following:

```
pmandef -s pqmon \
-e "IBM.PSSP.pm.User_state1:NodeNum=5:X@0!=PRINTQ_DOWN" \
-c "/usr/bin/touch /tmp/pqfail"
```

This creates a file marker in /tmp when the first resource monitor evaluates as true.

This is the second script/pman object combination:

```
#!/bin/ksh
# Multiple resource monitor example - 2
#
# Check that all printers are available
count=‘/etc/lsdev -Ccprinter | grep Available | wc -l‘
if (($count < 5))
then
    /usr/lpp/ssp/bin/pmanrminput -s pman -a
"IBM.PSSP.pm.User_state2+PRINTERDOWN+"
fi
done
exit 0
```

This is matched with the pman object created using the following:

```
pmandef -s pqmon \
-e "IBM.PSSP.pm.User_state2:NodeNum=5:X@0!=PRINTERDOWN" \
-c "/usr/bin/touch /tmp/printerfail"
```

Finally, these are tied together with a third monitor that is run periodically by cron. This looks like:

```
#!/bin/ksh
# Basic Print Subsystem monitor
```

```
#
# Check for marker files
if [[ -f /tmp/pqfail && -f /tmp/printerfail ]]
then
    /usr/lpp/ssp/bin/pmanrminput -s pman -a
"IBM.PSSP.pm.User_state3+FAILED+"
fi
exit 0
```

This then triggers a HACMP recovery program through:

```
##### Beginning of Event Definition print subsys failure ######
#
PRINT_SUBSYS_FAIL
0
/usr/sbin/cluster/events/node_down.rp
2
0
IBM.PSSP.pm.User_state3
NodeNum=5
X@0!="FAILED"

##### End of Event Definition  print subsys failure ######
```

Part 3. How Do You Monitor This?

Chapter 8. Common Monitoring Tasks

This chapter shows how Event Management can be used to perform common monitoring tasks. As each task is introduced, the resource variables that are used for monitoring are explained and examples of their use are given. These examples may be used with Perspectives or the SP Problem Management (pman) subsystem.

8.1 Monitoring Processes

There are two event management resource variables that are used for monitoring *processes*. They provide very similar functions, and care should be taken to ensure that the correct one is used if incorrect results are to be avoided. They are:

- IBM.PSSP.Prog.pcount, which represents all processes running a program, regardless of why they are running it.
- IBM.PSSP.Prog.xpcount, which only represents those processes that are running a specified program as a result of having called one of the exec**()** routines.

Typically, a process runs a program because it called an exec() routine specifying the program. It is possible, however, that a process may be running a program because it inherited the program from its parent process, and hence, never called an exec() routine. Some daemons do this. For example, the following ps output shows that only the process whose PID is 16706 called an exec() routine to run the biod program. All the other processes inherited the biod program from their parent process--the process with PID 16706:

```
# ps -ef | grep biod
root 15942 16706   0   Oct 12      -  0:16 /usr/sbin/biod 6
root 16706  2344   0   Oct 12      -  0:15 /usr/sbin/biod 6
root 16972 16706   0   Oct 12      -  0:15 /usr/sbin/biod 6
root 17224 16706   0   Oct 12      -  0:15 /usr/sbin/biod 6
root 17486 16706   0   Oct 12      -  0:15 /usr/sbin/biod 6
root 17744 16706   0   Oct 12      -  0:15 /usr/sbin/biod 6
```

To monitor processes that inherit programs, the IBM.PSSP.Prog.pcount resource variable should be used.

To only monitor processes that explicitly call an exec() routine to run a program, the IBM.PSSP.Prog.xpcount resource variable should be used.

Using IBM.PSSP.Prog.pcount to monitor certain classes of programs, such as those that create child processes to run other programs, may lead to unintended events. Consider inetd as an example; inetd's function is to spawn other daemons. As the services of a daemon it controls are requested, inetd calls fork() to create a child process. That child process starts out running the inetd program but quickly calls an exec() routine to run the appropriate daemon, such as telnetd. Therefore, for brief periods of time, more than one process will be running the inetd program.

If IBM.PSSP.Prog.pcount was used to monitor inetd, events might be generated showing these child processes running inetd for small periods of time. To avoid these "false" events, it would be better to use IBM.PSSP.Prog.xpcount.

As can be seen, some knowledge about how a program operates may be needed before deciding how best to monitor it. For most cases, it is probably appropriate to use the IBM.PSSP.Prog.xpcount resource variable to monitor a program. However, if the program to be monitored is inherited by long-running processes that do not call an exec() routine, IBM.PSSP.Prog.pcount might be appropriate.

Both IBM.PSSP.Prog.pcount and IBM.PSSP.Prog.xpcount have instance vectors that specify the program name, the user name, and the node number of interest. The ProgName and UserName instance vector elements cannot be wildcarded. The NodeNum instance vector element may be wildcarded.

Table 5 shows the structured byte string (SBS) included in the resource variable's value.

Table 5. SBS for IBM.PSSP.Prog.pcount and IBM.PSSP.Prog.xpcount

Field Serial Number	Data Type	Description
0	long	The current number of processes running the program for the user.
1	long	The previous number of processes that had been running the program for the user.
2	char	A comma-separated list of the PIDs of the processes currently running.

If we wish to monitor the inetd process on node 5, the instance vector would be:

```
ProgName=inetd;UserName=root;NodeNum=5
```

A change in the value of an IBM.PSSP.Prog.xpcount or IBM.PSSP.Prog.pcount instance either indicates that fewer processes *or* more processes are running the program. Which condition is indicated can be determined by comparing the value of the current process count with the value of the previous process count. This is done with the predicate.

If the event you are monitoring for is a change in the number of processes running a program, the predicate would be:

```
X@0 != X@1
```

This indicates when the value of the SBS Field serial number 0 is different from SBS Field serial number 1. This will not tell you whether additional processes have started or existing processes have stopped or died, only that the number of processes has changed. If you wish to monitor for a reduction in the number of processes, that is, the stop or death of a process, the predicate would be:

```
X@0 < X@1
```

This would be used to monitor a set of processes, such as the biod daemons in the preceding example, with the same name.

If you are monitoring for process death, that is, when no processes are running the program, the predicate would be:

```
X@0 == 0
```

This indicates when the value of the SBS Field serial number 0 (the current number of processes running the program) equals zero. This would also be used to monitor for the existence of a single important process.

Both the IBM.PSSP.Prog.pcount and IBM.PSSP.Prog.xpcount resource variables are designed to be used to monitor programs that are expected to have long lifetimes. If they are used to monitor a program that runs for only a few seconds, not all processes that run the program may be detected.

8.1.1 Recovery Actions for Process Death Events

The choice of a recovery action to handle a process death event has to be made with care. Where it may seem that the obvious course of action is to restart the process, this can often lead to further problems. Consider the examples in the following section.

8.1.1.1 Process Dies Because of Resource Unavailability

In many situations, the fact that the process has failed is not due to a fault in the code that makes up the process, but rather to a problem elsewhere in the

system. One activity that a process may undertake is to log its activity to a file. When the file system that contains the log file fills up, the process is unable to log and, therefore, abends. In this situation, a simple restart of the process will have no effect whatsoever because the process will simply die again. Only by first rectifying the underlying cause of the process death can the process be made to restart successfully.

8.1.1.2 Controlled Stop of a Process vs Process Death

When a process is stopped in a controlled fashion, it will often perform cleanup activities. If a process dies, there is no cleanup as a result, and the attempted restart of the process will fail.

Consider an application which runs as a single process. The process ID of this is held in file /tmp/app.PID. This file exists to prevent multiple instances of the process from running on the system at the same time. Should the app process fail, the attempted restart also fails because of the existence of the app.PID file. This makes the startup routine believe that an instance of app is already running on the system. Only by first cleaning up the remains of the previous app instance can the restart of the process succeed.

8.1.1.3 Failure of a Process of Multiple Interrelated Processes

Many applications consist of multiple interrelated processes that make up a single application instance. Often, these processes will be communicating through some defined mechanism. For example, an area of shared memory. In these situations, the uncoordinated restart of a single process within the instance may cause additional problems. One possibility is that the newly restarted process will simply be unable to communicate with those remaining from the original application invocation. Another possibility might be that the other processes fail. When running an instance that consists of multiple processes, you should always restart the *entire* instance if a single process fails rather than attempt to start only the failed process, unless you have fully verified that a process within an instance may be restarted independently of the others.

8.1.1.4 4. Controlled Process Death

It is difficult for the Event Management subsystem to determine the difference between a real failure of a process and a controlled stop, particularly if the process is stopped quite legitimately, albeit incorrectly, by another user. This then introduces a requirement on the recovery action to be able to determine the difference between the two.

To illustrate this, consider the following example where we have a process called critical_process that we are monitoring for any abnormal termination.

The recovery action looks for the presence of a marker to determine whether this is a controlled or abnormal stop. The *critical_process* recovery action hence might resemble:

```
#!/bin/ksh
# recovery action for critical_process
#
if [[ -f /tmp/crit_proc_marker ]]
then
    # Abnormal stop - restart critical_process
    /usr/local/bin/critical_process
    exit 0
else
    # Normal stop - do nothing
    exit 0
fi
```

This is a simple example but illustrates the problem that we face. Having now used a marker to determine whether this is a normal or abnormal stop, additional effort must be applied to be made to the mechanism for starting and stopping the critical_process process. Whereas before it might simply have been sufficient to run critical_process on the command line, this approach implies the use of start and stop scripts to create and destroy the marker as appropriate.

The start script, start_crit_proc, might look like:

```
#!/bin/ksh
# start script for critical_process
touch /tmp/crit_proc_marker
/usr/local/bin/critical_process
exit 0
```

The corresponding stop script, stop_crit_proc, would then look like:

```
#!/bin/ksh
# stop script for critical_process
#
rm /tmp/crit_proc_marker
/usr/local/bin/kill_critical_process
exit 0
```

8.1.2 Rearm Considerations for Process Death Events

The rearm event for a process death notification is not always required. To a large extent this depends upon the original event that occurred. If the system is deemed to be working correctly only when a given number of processes are running, for example 6, the rearm predicate would be:

```
X@0 == 6
```

However, if a single instance of a process is all that is needed, the rearm predicate would be:

```
X@0 >= 0
```

8.2 File System Space

There are two event management resource variables that are used for monitoring *file systems*. They are:

- IBM.PSSP.aixos.FS.%totused, which is very useful because it provides information about the utilization of a file system in percentage terms.

  The instance vector for the IBM.PSSP.aixos.FS.%totused resource variable comprises three values: the node in which the file system resides (NodeNum), the name of the logical volume containing the file system (LV), and the name of the volume group containing the logical volume (VG).

  The NodeNum instance vector element may be wildcarded.

- IBM.PSSP.aixos.FS.%nodesused, which provides information about inode utilization within a file system, is less useful except in environments where very large numbers of small files are required to be stored in a single file system.

The resource variable's value is a quantity of data type float containing the percentage utilization of the file system.

If we wish to monitor the root file system (/) on node 3, the instance vector would be:

```
VG=rootvg;LV=hd4;NodeNum=3
```

The logical volume associated with a file system can be identified using the `lsfs` command. Once the logical volume name is known, the volume group can be located. Under normal circumstances, an AIX file system should not be more than 90% full if optimum performance is to be maintained. An event may be triggered when this threshold is reached using the predicate:

```
X > 90
```

In certain environments, for example for DFS and AFS cache file systems, the utilization should be lower. Monitoring in these situations requires a lower value.

If you expect the utilization of a file system to remain approximately constant and wish to be informed if the utilization suddenly changes, the predicate would be:

```
X >= X@P + 5
```

This predicate will evaluate as true if the utilization of the file system has increased by more than 5% since its previous observation. File system observations are made, by default, every 60 seconds. If the resource variable is used to monitor file systems that change rapidly over time, the resource variable may not provide totally accurate information.

8.2.1 Recovery Actions for File System Full Events

The choice of a recovery action to handle a file system full condition, or a file system hitting the limit predefined by the event condition predicate, must be selected with care if any data loss is to be avoided. The seemingly obvious course of action is to remove the files within the file system, but this is almost guaranteed to introduce future problems.

The contents of a file system, and the volatility of the data stored there, vary widely among systems. While it is impossible to give a set of guidelines that will hold in every environment, the following general concepts tend to hold true:

(*/*) The root file system (***/***) is the top of the file tree. It is usually small and contains mount points for many other file systems. However, it also contains the files and directories critical for system operation, including the device directory, the ODM, and programs for booting the system. If ***/*** fills up, the system configuration stands a very good chance of becoming corrupted. Consequently, the root file system filling up should be avoided at all costs.

/home The /home file system contains user home directories. Disk space within this file system can be managed through techniques such disk quotas to ensure that users do not use too much space. In most systems, however, this file system is the most likely to fill up over time as users store increasing volumes of data.

/var The /var file system tends to fill up because it contains subdirectories and data files that are used by busy applications such as accounting, mail, and the print spooling subsystem. The majority of data in /var is transient; for

example, files are held here before printing. Once the file has been successfully sent to the printer, the disk space used by it is freed. If /var is full, functions such as mail, printing, and so on will stop functioning.

/tmp The /tmp file system is normally intended to be used to hold files on a temporary basis. In reality, however, it often tends to fill up because users use it as a pool of additional disk space. It also often contains temporary files created by applications during the course of their operation such as temporary files created by compilers. If /tmp is full, these applications may cease to function correctly.

/usr The /usr file system contains executable files that can be shared among machines. The majority of files in this file system do not change. As a result, /usr should not change on a day-to-day basis unless new applications or software maintenance is being applied. Good system management practices should avoid the majority of situations that result in /usr suddenly filling up.

/export Some systems have a **/**export file system that contains the server files exported to clients, such as diskless or dataless systems. The /export file system typically contains executables, paging space for diskless clients, and root file systems for diskless or dataless systems. As with the /usr file system, the majority of files in this file system do not change, and disk space allocated for paging space and so on is already allocated. As a result, on a day-to-day basis, file system space should not be an issue. Good system management should avoid the majority of situations that result in it filling up.

Given the preceding general guidelines for which file systems run the risk of filling, and the results of such an event, consider the examples of recovery mechanisms for file system full events given in the following section.

8.2.1.1 Invoke Backup or Archive Mechanism

All systems should have some backup mechanism; typically, it is to tape. One possibility for recovering space in a file system is to back up either all of the files, or the least recently used files, to tape or to another backup medium. This will allow file system space to be recovered but may result in initial complaints from users that they have "lost" data. Having been backed up, of course, the data can be recovered. Should this occur, it is often a good time

to then convince users to perform their own housekeeping to release space occupied by files that they are no longer using.

Use of a hierarchical storage management (HSM) system can also alleviate file system space problems. Most HSM systems migrate data from fast storage devices, predominantly magnetic disks, to slower storage, such as optical or tape devices. Typically this is performed when objects within a file system reach a certain age or have not been accessed for a period of time, say two months. When migration occurs, the object is copied to the slower storage medium, and the copy held in the file system is removed. A pointer remains to the object. If the user attempts to access the object, it is retrieved from the slower storage and brought back onto magnetic media. The user simply perceives a longer time before the object is available for use.

Most HSM environments provide mechanisms to move data from fast storage to slower storage. One possibility would be to temporarily reduce the time-to-live value for objects held on magnetic media. The effect of this would be to move data to slower storage, and hence, reacquire file system space. Other HSM systems provide facilities to archive the entire contents of a file system to slower storage.

8.2.1.2 Increasing the Size of a File System

The AIX logical volume manager provides the capability to dynamically extend a file system. One possible recovery action to invoke, should a file system hit its preset utilization threshold, could be to extend a file system.

As with any recovery action, however, this has its potential advantages and disadvantages. In a disk-rich environment where there is a lot of spare capacity, adding additional space is a quick, easy, and non-disruptive change to the system. In a disk-poor environment, this is unlikely to be an option. Even in the disk-rich environments, however, this is not always an ideal solution. Disk space is allocated in logical partition-sized chunks. In environments with large capacity physical disks, these logical partitions may be quite large; consequently, more disk space may be allocated than intended if care is not taken.

Another consequence of logical partitions is that they are allocated from the within the volume group. This may mean that they are allocated on a different physical device. In some environments, this may lead to performance issues because the logical partitions are not contiguous, or because they are located on the relatively slow "edges" of the disk platters.

A final consideration with automatically increasing file system sizes is, that if each time the predefined limit is reached, a new logical partition is added.

This will also tend to fill up over time. User data tends to expand to fill the available space. This is simply putting off the issue until it can no longer be addressed non-disruptively. Where automatically increasing the file system size is a powerful tool, it has to be matched by careful systems management and corresponding planning.

8.2.1.3 Deleting File System Contents

Making a conscious decision to delete some or all of the contents of a full file system is a "last resort" option and is not one that should be undertaken without careful consideration beforehand of the implications of this action. By its very nature, a recovery action such as this is disruptive, and because it deletes files, its results are not predictable.

That said, there are often many files present in a file system that are either obsolete or unneeded. These include objects such as core files, a.out files, temporary or checkpoint files created by editors, and so on.

Should you decide to follow this course of action, a good starting point for a recovery script is the skulker program. Take a copy of this and modify it to fit your environment because it has much of the necessary logic in place already to deal with obsolete or unneeded files. A recovery action script to do this for a single file system based upon skulker might look like this:

```
#!/usr/bin/bsh
# fs_clean - Script to reclaim file system space
#
# WARNING: THE WHOLE PURPOSE OF THIS SCRIPT IS TO REMOVE FILES,
# SO IT HAS THE POTENTIAL FOR UNEXPECTED RESULTS.

remove_file()
{
    if [ -z "‘/usr/sbin/fuser $1 2>/dev/null‘" ]; then
        /usr/bin/rm -f $1
    fi
}

date=‘/usr/bin/date‘
uname=‘/usr/bin/uname -nm‘
echo "$0 cleaned the $1 file system at $date on $uname"

/usr/bin/find $1 \
\( \( \( -name "*.bak" -o -name core -o -name a.out -o \
-name "...*" -o -name ".*.bak" -o -name ed.hup -o \
-name smit.script -o -name smit.log \) \
-atime +1 -mtime +1 -type f \
\) \
```

```
-o \
\( \( -name proof -o -name galley \) \
-atime +1 -mtime +1 -type f ! -perm -0200 \
\) \
\) -xdev -print | \
while read FILE2REM
do
    remove_file $FILE2REM
done
```

To invoke this against the /home directory, the following command would be used:

```
/usr/local/bin/fs_clean /home
```

8.2.2 Rearm Considerations for File System Space Events

The rearm predicate for a file system space event is typically the opposite of the original event predicate. If the event is triggered when free space in the file system is less than 10% of the total space, that is, when the predicate is:

```
X > 90
```

the rearm predicate for the event would be:

```
X < 90
```

8.3 Error Log Entries

The event management resource variable that is used for monitoring the AIX error log is:

- IBM.PSSP.pm.Errlog

IBM.PSSP.pm.Errlog contains information from an entry written to the AIX error log. The sole instance vector for the resource variable is NodeNum. This specifies the number of the node in which the error log that you wish to monitor resides. The NodeNum instance vector element may be wildcarded.

The resource variable's value is a structured byte string that contains strings of data from the AIX error notification daemon. For more information, refer to Table 6.

Table 6. SBS for IBM.PSSP.pm.errlog

Field Serial Number	Data Type	Description
0	cstring	The sequence number of the error log entry.
1	cstring	The error ID of the error log entry. For example, 00530EA6.
2	cstring	The class of the error log entry. For example, H=hardware, S=software, and so on.
3	cstring	The type of the error log entry. This identifies the severity of the error log entry. For example, PERM, TEMP, PEND, and so on.
4	cstring	The alert flags value of the error log entry. This identifies whether the error is alertable.
5	cstring	The resource name of the error log entry. For a hardware error, this is the device name.
6	cstring	The resource type of the error log entry. For hardware errors, this is the device type of the resource from the devices object class.
7	cstring	The resource class of the error log entry. For hardware errors, this is the device class. It is not applicable for software errors.
8	cstring	The error label from the error log entry. For example, X25_ALERT25.

If we wish to monitor the AIX error log on node 1 of the cluster, the instance vector would be:

```
NodeNum=1
```

A change in the value of an IBM.PSSP.pm.Errlog instance indicates that a new entry has been written to the error log or logs that are being monitored. Monitoring solely for changes, for example, new log entries, can be achieved using the following predicate:

```
X@0 != X@P0
```

This indicates when the value of the SBS Field serial number 0, the sequence number of the error, is different from its previous value. Each entry in the AIX error log has a sequence number. These are assigned in increasing order. Creating a condition with a predicate like this will tell you only that an error has occurred and has been logged. It will not tell you anything about the

importance of the error. To monitor the error log for a specific error or set of errors, a predicate similar to the following should be used:

```
X@0!=X@P0 && X@1=="0x476b351d"
```

In this predicate, X@1 refers to SBS Field serial number 1. This is the error ID associated with the error. In this case, 476B351D is the error ID for the TAPE_ERR2 error, a permanent failure of a tape drive attached to the system. This predicate could also have been written:

```
X@0!=X@P0 && X@8=="TAPE_ERR2"
```

where X@8 refers to the field in the SBS that contains the error label of the error that was generated. To identify the error label or error ID for a particular error, use the `errpt` command with the `-t flag`. The error ID for a failure is the first field, and the error label is the second field in the resulting output. If used with pman, SBS field 8 contents will need to be enclosed in double quotation marks.

8.3.1 Recovery Actions for Error Log Entry Events

The recovery action used to handle an error log entry event is dependent upon: how critical the resource is within the system; whether recovery is possible; if so, where that recovery should occur. There are various generic actions that may be taken. These include:

- Restart of the failed resource locally
- Failover of the resource to another local resource
- Failover of the resource to another cluster node

This discussion is no different from the case where the errors have been detected by the AIX error log itself, and the event is triggered through error notification. See Chapter 11, “Sample Events and Actions” on page 161 for more detail.

8.3.2 Rearm Considerations for Error Log Entry Events

The rearm predicate for an error log entry event is very dependent upon the recovery action. In general, however, because the initial trigger of the resource monitor occurs based on a "one-off" event rather than, for example, comparing a value against a threshold, there is no valid concept of a rearm event for these occurrences, and hence, no valid rearm predicate.

Chapter 9. Common Recovery Actions

This chapter provides some sample actions that may be used in response to an event. As each action is introduced, guidelines for the use of the action are given.

9.1 Freeing Up File System Space

This shell script, which is loosely based upon the logic in /etc/skulker, frees up file system space by deleting old files. For the most part, only files older than seven days old are deleted. This can be changed by changing the numbers associated with the `-atime` and `-mtime` flags in the `find` command. In a well-managed system, a script like this should not be a problem to run, there being a week in which to process any outstanding issues that have been brought to light through logfile output.

The script is invoked with the sole parameter being the file system that is to be cleaned. That is:

```
filesystem_clean /tmp
```

As with any shell script that deletes files, there is an inherent risk of losing data when running this script:

```
#!/usr/bin/ksh
#
# Filesystem cleanup script
#
function general_cleanup
{
    DIR=$1
    /usr/bin/find $DIR \
    \( \( \( -name "*.bak" -o -name ".*.bak" -o \
    -name "*.BAK" -o -name ".*.BAK" -o -name "...*" -o \
    -name "*.CKP" -o -name ".*.CKP" -o -name core -o \
    -name a.out -o -name ed.hup -o -name core.last -o \
    -name "*~" -o -name .sh_history -o -name smit.script -o \
    -name smit.log \) \
    -atime +7 -mtime +7 -type f \
    \) \
    -o \
    \( \( -name proof -o -name galley \) \
    -atime +7 -mtime +7 -type f ! -perm -0200 \
    \) \
    \) \
    -xdev -print | \
```

```
while read FILE
    do
        clear_file $FILE
    done
}
function clear_file
{
    if [ -z "‘/usr/sbin/fuser $1 2>/dev/null‘" ]
    then
        # remove it if no-one is using it
        /usr/bin/rm -f $1
    else
        # clear it if something is holding it open
        > $1
    fi
}
# Log what we are about to do
DATE=‘/usr/bin/date‘
UNAME=‘/usr/bin/uname -n‘
/usr/bin/echo "Filesystem cleanup of $1 run at $DATE on $UNAME" | mail
root
case $1 in
    "/") # root directory
        general_cleanup /
        # root specific cleanup
        /usr/bin/find $1 \
        \( \( -name image.data -o -name "ntp.conf0*" -o \
        -name "ntp.conf1*" \) \
        -atime +7 -mtime +7 -type f \
        \) \
        -xdev -print | \
        while read FILE
        do
            clear_file $FILE
        done;;
    "/tmp") # tmp directory
        general_cleanup /tmp
        # tmp specific cleanup
        /usr/bin/find $1 \
        \( \( -name "*" -o ! -name "tkt*" \) \
        -atime +7 -mtime +7 -type f \
        \) \
        -xdev -print | \
        while read FILE
        do
            clear_file $FILE
        done;;
```

```
"/var") # var directory
    general_cleanup /var
    # SP var specific cleanup
    /usr/bin/find $1 \
    \( \( -name "hmlogfile.*" -o -name amd.log -o \
    -name "pmand.*.log*" -o -name "pmanrmd.*.log*" -o \
    -name sysctld.log -o -name kerberos.log -o \
    -name "sdrd.*" -o -name admin_server.syslog -o \
    -name SPdaemon.log -o -name SDR_test.log -o \
    -name spmon_itest.log -o \
    -name jm_install_verify.log -o \
    -name SYSMAN_test.log -o \
    -name auto.log -o -name "haem*" -o \
    -name "hats*" -o -name "hags*"  \) \
    -atime +7 -mtime +7 -type f \
    \) \
    -xdev -print | \
    while read FILE
    do
        clear_file $FILE
    done
    # other var specific cleanup
    /usr/bin/errclear 7

    /usr/bin/find $1 \
    \( \( -name INEd.FATAL.LOG -o -name trcfile -o \
    -name sulog -o -name log \) \
    -atime +7 -mtime +7 -type f \
    \) \
    -xdev -print | \
    while read FILE
    do
        clear_file $FILE
    done
    for VARDIR in /var/spool/qdaemon /var/spool/lpd/qdir \
    /var/spool/uucp
    do
        if [[ -d $VARDIR ]]
        then
            /usr/bin/find $VARDIR -mtime +7 HR> -type f -print | \
            while read FILE
            do
                clear_file $FILE
            done
        fi
    done
    if [[ -d /var/spool/qftp ]]
```

```
        then
            /usr/bin/find /var/spool/qftp \
            \( -name 'tmp*' -o -name '[0-9]*' \) \
            -mtime +7 -print | \
            while read FILE
            do
                clear_file $FILE
            done
        fi
        if [[ -d /var/tmp ]]
        then
            /usr/bin/find /var/tmp -atime +7 \
            -mtime +7 -type f -print | \
            while read FILE
            do
                clear_file $FILE
            done
        fi
        if [[ -d /var/news ]]
        then
            /usr/bin/find /var/news -mtime +45 \
            -type f -print | \
            while read FILE
            do
                clear_file $FILE
            done
        fi
        if [[ -d /var/adm/nim ]]
        then
            /usr/bin/find /var/adm/nim -mtime +7 \
            -type f -print | \
            while read FILE
            do
                /usr/bin/rm -rf $FILE
            done
        fi;;
    *) # Any other directory
        general_cleanup $1 ;;
esac
```

9.2 Killing Processes

This shell script kills the processes that are the top ten users of CPU resource or memory as measured by the ps command. In order to ensure that key processes on the system are not inadvertently killed, the script uses an *exclusion list.* This contains the names of those processes that are not to be

considered for termination. The script is invoked with a single parameter, `m` or `c`, that determines whether the processes to be killed are memory or CPU users. There is also a "safety catch" to ensure that if the script is inadvertently invoked, no processes will be killed. To kill the top ten memory-using processes that are not on the exclusion list, the script would be invoked thus:

```
proc_clean m Y
```

The exclusion list is a separate file. Its location is defined within the script and may be changed as required. The format of the file consists of a single entry per line, as such:

```
hatsd
swapper
init
syncd
portmap
```

As with any shell script that kills processes, the behavior of this script has an inherent risk of causing unforeseen problems when run. This is compounded by the fact that the processes that may be killed by it will vary depending upon when the script is run:

```
#!/bin/ksh
#
# Process cleaning script
#

EXCLUSION_LIST=/tmp/exclusion_list

# Safety catch
if [[ $2 != "Y" ]]
then
    echo "proc_clean was not confirmed"
    echo "exiting..."
    exit 0
fi
# Log what we are about to do
DATE=‘/usr/bin/date‘
UNAME=‘/usr/bin/uname -n‘
/usr/bin/echo "Process cleanup run at $DATE on $UNAME" | mail root
case $1 in
    m) # Top 10 memory using processes
    for PID in ‘ps auxw | grep -v kproc | grep -vf $EXCLUSION_LIST |
tail +2 | sort -k 1.21,1.25nr|head | awk ’{print $2}’‘
    do
        kill -15 $PID
        ps -ef | awk ’{print $2}’ | grep $PID
```

```
        if (($? == 0))
        then
            kill -9 $PID
        fi
    done;;
    c) # Top 10 CPU using processes
    for PID in ‘ps auxw | grep -v kproc | grep -vf $EXCLUSION_LIST |
tail +2 | sort -k 1.16,1.20nr|head | awk ’{print $2}’‘
    do
        kill -15 $PID
        ps -ef | awk ’{print $2}’ | grep $PID
        if (($? == 0))
        then
            kill -9 $PID
        fi
    done;;
    *) # Error - no parameter passed
    echo "Error: no parameter passed to proc_clean";;
esac
exit 0
```

Chapter 10. The Top Ten Events and Actions

This chapter discusses the major events or occurrences that every system should monitor for. It provides both mechanisms for monitoring for these events and sample recovery actions should these events occur.

10.1 Introduction

This section identifies the top ten events that should be monitored for. Occurrence of any of these events is likely to lead to problems with the system in the near future if action is not taken to resolve the underlying problem or provides some other sort of remedial action. Note that the events are not given in any particular order of severity.

For each event, a selection of the problems that might arise if it is not fixed is given along with mechanisms for detecting these events and potential recovery actions.

Providing a list of the "top ten" events is much harder than it might seem. To begin with, the list has to be generic enough to apply in more than a few specific systems. Secondly, there is no way to easily assume what is right or wrong for a particular environment. In many cases, a CPU which is 100% utilized would appear to be a problem. Yet if this same CPU is performing all of the tasks which it is called upon to do, is there actually a problem? Similarly, large amounts of paging may be perceived in some situations to be an issue. Yet if the perceived performance of the system as a whole is deemed to be OK, again, just where is the problem in this event?

What this list tries to provide is a set of ten situations where problems will arise--in some cases, immediately after the event occurs--in others, after a longer period of time. They may not all apply on every single system in existence, but the vast majority of systems will suffer when these events occur.

Finally, the AIX Operating System, and the underlying hardware, is very robust. Many devices and operating system components provide their own recovery mechanisms should a failure occur: bad blocks on disks may be relocated; failed daemons may be respawned; or more complex recovery will occur. For example, within a device driver.

Therefore, for most systems, it will be the application software that causes most problems, and hence it is this software that needs to be monitored. This redbook does not discuss the intricacies of monitoring specific applications. If

you think it is difficult to identify the top ten AIX events, just think about trying to identify the top ten applications.

In many cases, it is not possible to provide a generic recovery action. For those examples where file system space is an issue, or for when the death of a process has occurred, refer to Chapter 8, “Common Monitoring Tasks” on page 123 for general guidelines on recovery actions. See also Chapter 9, “Common Recovery Actions” on page 137.

10.2 File System Full Events

Three of the top ten events and actions are related to the file system full condition.

10.2.1 The root (/) File System Filling Up

The root file system contains many critical components, any one of which may become corrupted or inaccessible should the file system fill up. Among others, the following problems may arise if this occurs:

- Corruption of the Object Data Manager (ODM)

The ODM holds a large amount of AIX configuration information that is not held in flat files. This data can be impossible to recreate without restoring from a backup. If lost, it contains information about device configurations, logical volume information, installed software, and so on. Loss of access to this data will result in system failures.

- Corruption of system configuration files in /etc

There are many flat configuration files held in the */*etc directory in the root file system. These contain other configuration details, such as information about users and groups, file system configurations, network names and addresses, and other network configuration data. Loss of access to this data will most likely result in subsystem, and potentially entire system, failures.

File system space may be monitored using the following Event Management Resource Variable:

```
IBM.PSSP.aixos.FS.%totused
```

The normal instance vector for the root file system will be:

```
NodeNum=*;LV=hd4;VG=rootvg
```

In most systems, the root file system is deliberately kept small but fairly full.

Consequently, a typical predicate for the condition might be:

```
X>95
```

In most environments, there is very little that can be deleted from the root file system. It may be that there are leftover files from editing sessions or other processes that have failed or ended abnormally. The following command might be run to attempt to remove these files:

```
find /  -name "*.bak" -o -name core -o -name a.out -o \
-name "...*" -o -name ".*.bak" -o -name ed.hup \
-atime +1 -mtime +1 -type f -print | xargs -e rm -f
```

In many cases, the root file system is also the home directory of the root user. This may mean that there are SMIT logfiles that are held here. These are only useful to keep if you intend doing something with them in the future. These can be very large if not cleaned up regularly; consequently, removing these logfiles may also free up space in */*.

```
rm -f /smit.script /smit.log
```

As a last resort, if there is still no space free in the root file system, it is worth considering an automatic extension of the file system. The following command will add a single logical partition to the */* file system:

```
chfs -a size=+1 /
```

10.2.2 The /var File System Filling Up

The */*var file system contains working space used by many subsystems. These will cease to function correctly if the file system fills up. The following subsystems are likely to exhibit problems if the */*var file system runs out of space:

- Accounting
- Error logging
- cron
- mail
- printing
- uucp

File system space may be monitored using the following Event Management Resource Variable:

```
IBM.PSSP.aixos.FS.%totused
```

The normal instance vector for the /var file system will be:

```
NodeNum=*;LV=hd9var;VG=rootvg
```

The **/**var file system is typically fairly large in an average AIX system, and the amount of freespace may vary quite significantly over time. It is therefore important to set a predicate that is likely to be triggered only when there is a problem rather than being triggered under normal operational conditions, which would be quite possible if the value of the right operand was fairly low. A typical predicate for the condition might be:

```
X>90
```

In most environments, there are many logfiles that can be deleted from the **/**var file system. These may contain important data to assist with problem determination; therefore, a backup followed by a removal is a preferred option if this is possible. However, this is not always the case, and consequently, the following procedure to remove logfiles may be followed should the key consideration be keeping the system running:

- Remove printer log files:

```
rm -f /var/adm/lp-log
rm -f /var/adm/lw-log
```

- Remove uucp log files:

```
rm -f /var/spool/uucp/LOGFILE
rm -f /var/spool/uucp/SYSLOG
rm -f /var/spool/uucp/ERRLOG
```

- Remove older files in /var/tmp:

```
find /var/tmp -type f -atime +7 -exec rm -f {}
```

Delete /var/adm/wtmp and /var/adm/acct if you do not need the files for accounting purposes.

The following cleanup operations are more invasive and should be used as last resort options. Use of these may cause loss of data that you wish to keep.

Firstly, the AIX error log may be of a significant size. This can be cleared using:

```
errclear 0
```

Remove any files in the /var/preserve directory. This holds preserved data from interrupted edit sessions:

```
rm -f /var/preserve/*
```

Also check for the existence of tracefiles. These are kept by default in /var/adm/ras/trcfile.

10.2.3 The /tmp File System Filling Up

The /tmp file system contains working space used by some subsystems. It is also frequently (and potentially incorrectly) used as extra space by users of the system. Those subsystems that use **/**tmp will cease to function correctly if the file system fills up. Where the given effects of /tmp filling up are specific to the software installed on a particular system, the following components are most likely to exhibit problems if the **/**tmp file system runs out of space:

- compilers
- editors

File system space may be monitored using the following Event Management Resource Variable:

```
IBM.PSSP.aixos.FS.%totused
```

The normal instance vector for the /tmp file system will be:

```
NodeNum=*;LV=hd3;VG=rootvg
```

The **/**tmp file system is typically fairly large in an average AIX system and the amount of freespace may vary quite significantly over time. It is therefore important to set a predicate that is likely to be triggered only when there is a problem rather than being triggered under normal operational conditions, which would be quite possible if the value of the right operand was fairly low. A typical predicate for the condition might be:

```
X>90
```

It is important that users understand the site policy for the **/**tmp file system. In many system environments, data placed in **/**tmp is held there at the user's own risk, and if a situation should arise that requires **/**tmp to be cleared out, any files which have been placed there may be lost. This is a sensible policy to follow because, strictly speaking, the **/**tmp directory is really intended for system-generated temporary files rather than being a spare area of disk space for users. If a policy such as this is implemented, the recovery action for **/**tmp filling up would be fairly simple—that is:

```
rm -rf /tmp/*
```

Be aware that in RS/6000 SP environments, the existence of kerberos ticket files, for example, /tmp/tkt0, will be effected by such a removal exercise. The tickets associated with these files should be destroyed and regenerated if this policy is followed.

If this policy is not implemented at your site, then you are once again forced into determining the relative importance of the files held in /tmp. A possible solution might be to remove older files, as such:

```
find /tmp -type f -atime +7 -exec rm -f {}
```

10.3 Process Death Events

Two of the top ten events and actions are related to process death.

10.3.1 Failure of Domain Nameserver (named) Process

Name resolution is of surprising importance to the day-to-day operation of a computer system within a networked environment. Where it is perfectly feasible for each computer to have its own */*etc/hosts file containing all of the hostnames and IP addresses that it will be required to communicate with, the management of such an environment rapidly becomes time consuming. The implementation of a nameserver, a single repository of this information, is the obvious solution once the number of machines reaches a certain point.

Many people assume that a nameserver is used only when you wish to communicate with another system. However, once nameserving is implemented in a networked environment, reference will be made to the nameserver even for local tasks. Many more functions require name lookup or address resolution than would immediately spring to mind. If the nameserver fails, in addition to the more obvious TCP/IP communications problems that will arise, many, or all, of the following will cease to function or function incorrectly:

- fonts
- fileserving
- X windows
- mail

The named process can be monitored using the following Event Management Resource Variable:

```
IBM.PSSP.Prog.pcount
```

The normal instance vector for the named daemon will be similar to:

```
NodeNum=4;ProgName=named;UserName=root
```

Notice that rather than wildcarding the NodeNum component of the instance vector, because there will only be a single node running named, there is no

point in monitoring all nodes. A typical predicate for the condition describing the death of the named process would be:

```
X@0==0
```

The obvious solution to the nameserver being a single point of failure is to have two of them; in other words, to implement a secondary nameserver for the domain. This is the preferred long-term fix for named problems. However, if this has not been done, it is important to get the named process running again as soon as possible.

The named daemon can be restarted as required. Assuming that the configuration files were correct beforehand and the daemon was running normally, the recovery action to restart the daemon will consist of the following command:

```
startsrc -s named
```

This command assumes that the named daemon has its configuration files in the default location, /etc/named.boot, /etc/named.data, and so on. If the files were held in another location, in /usr/local/dns for example, the command would be:

```
startsrc -s named -a '-b /usr/local/dns/named.boot'
```

This solution assumes that the configuration was fine and was working correctly. If the failure has arisen because of a change in the configuration, however, this will need to be resolved before the named daemon will restart.

If you are using SCCS to manage changes to your system files, the rmdel command will allow you to back out a delta (change) from a file. This may allow you to return to a working configuration.

If you are not using SCCS, and the named process fails repeatedly after a change, there is little automation that can be performed to return the system into a working configuration. Instead, this will require manual intervention.

10.3.2 Failure of Portmapper Process

As with name resolution, as discussed in 10.3.1, “Failure of Domain Nameserver (named) Process” on page 148, the portmap daemon provides services to many operating system components. Many strange errors can arise that are seemingly unrelated if the portmap daemon fails. For example, it is possible to generate the following error if the portmapper is not running:

```
Fatal error: Invalid Shared Memory operation
```

The /etc/rpc file contains the list of RPC program numbers. The contents of this file will give some feel for those functions of your system that would fail if the portmapper were to cease functioning:

```
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
etherstatd      100010  etherstat
rquotad         100011  rquotaprog quota rquota
sprayd          100012  spray
3270_mapper     100013
rje_mapper      100014
selection_svc   100015  selnsvc
database_svc    100016
rexd            100017  rex
alis            100018
sched           100019
llockmgr        100020
nlockmgr        100021
x25.inr         100022
statmon         100023
status          100024
bootparam       100026
ypupdated       100028  ypupdate
keyserv         100029  keyserver
sunlink_mapper  100033
tfsd            100037
nsed            100038
nsemntd         100039
showfhd         100043  showfh
cmsd            100068  dtcalendar
ypxfrd          100069  ypxfr
pcnfsd          150001
ttdbserver      100083  tooltalk
```

The portmap process can be monitored using the following Event Management Resource Variable:

```
IBM.PSSP.Prog.pcount
```

The normal instance vector for the portmap daemon will be similar to:

```
NodeNum=*;ProgName=portmap;UserName=root
```

A typical predicate for the condition describing the death of the process would be:

```
X@0==0
```

The portmap daemon may be restarted as required. However, if it terminated abnormally, all the RPC servers on the system that were using it should also be restarted. Fortunately, a large number of these restarts can be performed very easily. If the primary user of portmap services on the system was NFS and NIS, a possible recovery action for a portmap failure would be:

```
# Stop NFS
sh /etc/nfs.clean
sleep 10
# Restart portmapper
startsrc -s portmap
# Restart NFS
/etc/rc.nfs
```

Other users of the portmap subsystem, such as the various CDE components, rpc.cmsd, rpc.ttdbserver, and so on, are normally invoked automatically when the user starts the relevant CDE application. For example, rpc.cmsd is started when the dtcm (CDE calendar agent) process is run. A solution for these might be to kill any existing calendar agent processes (dtcm) so that the next user to invoke dtcm will also restart the rpc.cmsd RPC server.

10.4 Virtual Memory Events

One of the top ten events and actions is related to virtual memory.

10.4.1 Running Out of Paging Space

AIX, being a virtual memory operating system, is reliant on its paging space(s) having sufficient capacity to be able to support its processes. The initial default paging space size is determined during the system customization phase of AIX installation according to the following standards:

- Paging space can use no less than 16 MB.
- Paging space can use no more than 20% of total disk space.
- If real memory is less than 32 MB, paging space is two times real memory.
- If real memory is greater than, or equal to, 32 MB, paging space is real memory plus 16 MB.

In certain environments, for example those that run with massive executables or datasets, there may be a requirement to have up to twice the amount of physical memory available as page spaces. Therefore, there is very often a need to increase the amount of paging space from that initially configured when the system was installed.

If paging space runs low, processes may be lost. If paging space runs out, the system may "panic". The system monitors the number of free paging space blocks and detects when a paging space shortage exists.

When the number of free paging space blocks falls below a threshold known as the *paging space warning level*, the system informs all processes (except kprocs) of the low condition by sending the SIGDANGER signal.

If the shortage continues and falls below a second threshold known as the *paging space kill level*, the system sends the SIGKILL signal to processes that are the major users of paging space and that do not have a signal handler for the SIGDANGER signal (the default action for the SIGDANGER signal is to ignore the signal). The system continues sending SIGKILL signals until the number of free paging space blocks is above the paging space kill level.

If you do not wish to have processes killed in this fashion because they do not handle SIGDANGER appropriately, then you need a warning mechanism. This may be obtained by using the Event Management resource variable:

```
IBM.PSSP.aixos.PagSp.totalfree
```

By default, the SIGDANGER signal will start being sent when the number of free paging space pages falls below 512, although this may have been changed using the vmtune command. The predicate for the resource variable therefore needs to be triggered before this limit is reached. For example, by using:

```
X<600
```

Hopefully, capacity planning for the system will ensure that this event does not occur. However, if it does, additional paging space should be defined when the paging space low condition is detected. This may be performed using the chps command, as such:

```
chps -s1 hd6
```

This will add one logical partition of disk space to the hd6 page space. This is equivalent to 1024 extra paging space frames if the logical partition size for

the volume group is 4 MB. This should be sufficient to alleviate the current crisis.

The downside of allocating additional page space in this fashion is that performance of the system may deteriorate because the two paging spaces (the original hd6 plus its recently allocated logical partition) are unlikely to be contiguous on the disk. The addition will thus keep the system running but is likely to require some systems management activity in the near future to move partitions around on the disk to make the paging space contiguous again.

While it would be perfectly feasible to create a script to perform this task at the same time as increasing the paging space size, this is not advised as the system would already be under stress at the time, and the additional activity may push the system into the danger area once again or potentially push it into the paging space kill area.

An alternative would be to use this threshold as a trigger for systems management activity to manually clean up processes that are excessively using paging space.

10.5 Performance Events

Two of the top ten events and actions are related to performance.

10.5.1 Excessive Paging Activity

As was stated earlier, any performance measurement is likely to be very much system-specific, and as such, difficult to define. However, excessive paging activity is likely to be a cause of concern. If this occurs, then the performance of the system will be reduced. There are two potential mechanisms to determine that paging is excessive. The first uses the Event Management resource variable:

```
IBM.PSSP.aixos.Mem.Virt.pgspout
```

This provides a measure of the number of pages written to paging space. This is equivalent to the po column in vmstat output; that is:

```
r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 41218 103830   0   0   0   0    0   0 134  471  51  1  1 98  0
 0  0 41218 103830   0   0   0   0    0   0 130  415  43  0  0 99  0
```

In a well-balanced system, this value should be zero. If this is the case, there is no contention for the available memory, and hence, no paging of memory used by running processes out to page space occurs. However, as this

number increases, there is more and more time spent paging memory than running the processes. In just about any system, there will be occasions when paging does occur, so triggering an event when paging occurs would be an invalid thing to do. A better measurement of excessive paging would be to review vmstat output over a period of time and then determine a suitable value for paging out that you deem to be excessive. For example:

```
X>80
```

Another method of determining overcommitment of memory is to look at the number of pages in the free list. This shows the number of free pages of memory that are available to accommodate page faults. As memory becomes more and more utilized, the number of free pages decreases. Eventually, pages will be reclaimed by the operating system to ensure that the system does not come to a grinding halt. A situation where there are no free pages for a long period of time is a cause for concern. The number of pages on the free list can be monitored using the following resource variable:

```
IBM.PSSP.aixos.Mem.Real.numfrb
```

The predicate for this needs to not only reflect the current value, but also a *trend* in values, such that the event is triggered when the problem has been occurring for a while rather than at its first occurrence. Again, the absolute values will depend on your observations of the system, but a predicate to do this might be:

```
X<=50 && X<=X@P
```

Memory overcommitment may be a short-term problem caused by the running of a number of memory-intensive processes, or may be a long-term problem where the system is underconfigured for its workload. The only real solution in the latter case is to purchase and install more memory.

Short-term solutions involve changing the way that a system utilizes virtual memory. These can be made using the `vmtune` command. The advantage of vmtune is that any changes made to the operational parameters of the Virtual Memory Manager only last until the next reboot of the system. If the changes you make only serve to make the situation worse, a reboot of the cluster node will restore the system to its original state.

For more information on the use of the `vmtune` command, see *AIX Version 4.3 Commands Reference,* SBOF-1877. For more information on tuning VMM page replacement, see *AIX Versions 3.2 and 4 Performance Tuning Guide,* SC23-2365.

10.5.2 Memory Leakage in an Application

A so-called "memory leak" occurs when a program has a bug in it. This bug normally takes the form of repeatedly allocating memory, using it, and then neglecting to free it. A memory leak in a long-running program is a serious problem because it can result in memory fragmentation and the accumulation of large numbers of mostly garbage-filled pages in real memory and page space. Systems have been known to run out of page space because of a memory leak in a single program.

A memory leak can be detected by looking for processes whose working segment continually grows in size. To get detailed information about the size of the working segment, the `svmon` command may be used. Not all systems may have the `svmon` command installed however, but a reasonable approximation may be achieved by looking at the output of the `ps v` command, as follows:

```
PID   TTY STAT TIME PGIN SIZE  RSS  LIM  TSIZ  TRS %CPU %MEM COMMAND
31740 pts/1 A  0:00   0  232  468 32768  101  176  0.0  0.0 tn
34272 pts/0 A  0:00   9  232  468 32768  101  176  0.0  0.0 tn
34312 pts/2 A  0:00  15  172  436 32768  190  228  0.0  0.0 ksh
35292 pts/2 A  0:00   1  208  300 32768   46   60  0.0  0.0 ps v
```

The columns of interest are those that are memory-related: RSS, SIZE, TRS, and TSIZ. The sum of these columns gives a quick approximation of the memory utilization of the process. If you wish to monitor a particular process that you suspect of having a memory leak, you could use a mechanism similar to the following. Here, the process ID of the process in question is 8404.

```
ps v 8404 | grep -v PID | awk '{print $6+$7+$9+$10}'
```

This returns the total memory usage (RSS+SIZE+TRS+TSIZ) as:

```
950
```

It is possible to run a shell script based upon this command periodically over time (using `cron`) and place the value received into one of the User_state resource variables thus:

```
#!/bin/ksh
PROCID=‘ps -ef | egrep DBappl | grep -v egrep | awk '{print $2}’‘
MEMUSE=‘ps v $PROCID | grep -v PID | awk '{print $6+$7+$9+$10}’‘
/usr/lpp/ssp/bin/pmanrminput -s pman \
-a "IBM.PSSP.pm.User_state3+$MEMUSE+"
exit 0
```

This places the value of the memory usage of the DBappl process into the IBM.PSSP.pm.User_state3 resource variable. This can then be monitored using a predicate like:

```
X@0!=X@P0 && X@0>=X@P0+30
```

If the memory usage of the process increases between observations by more than 30 pages, this predicate will evaluate as true. The absolute value of the delta in such a case is system-and process-dependent.

If a process develops a memory leak, the simplest action is to stop it running. This prevents it from using up more system resources, and hence, potentially putting the entire system at risk. However, in many situations, this may not be practical. In this case, the policy for handling rogue processes at your site should be invoked.

10.6 Time Events

One of the top ten events and actions is related to time.

10.6.1 System Clock Wandering

The time at which a system clock is set in a stand-alone system can have several interesting implications for applications and the operating system. Within a clustered environment, where there is a much greater interdependence between the cluster nodes, time--or rather the time on a node that is out of step with the others--can cause serious conditions to arise. These include:

- Security failures

The timestamp on a kerberos key may result in it being invalid on one system, even though it was only recently granted or created.

- Application start-up

It is not uncommon, following a failure, for an application such as a database to refuse to start because the latest timestamp on the database log is after the current time on the local system. Variations on this can occur when the timestamp and the current system time, even though in the correct temporal order (that is, the log timestamp is earlier than the system time), are widely different, or when a node attempts to participate in a parallel database and is refused because its system time is widely different from the existing cluster nodes.

- User access

It is possible to specify the time periods that users may use a system. An incorrect system time may cause user access to be denied.

There is no resource variable for monitoring system time or the degree to which one clock is different from another. In a networked environment, and especially in a clustered environment, time synchronization is important, but if the mechanisms that have been put in place to ensure synchronization fail, or if the clocks wander nevertheless, there needs to be a mechanism to detect and handle this.

The most obvious preventative mechanism is to use the following command to monitor either timed or xntpd, whichever you are using in your environment:

IBM.PSSP.Prog.pcount

The normal instance vector for the timed daemon will be similar to:

```
NodeNum=*;ProgName=timed;UserName=root
```

The predicate for the death of the process would be:

```
X@0==0
```

Note that xntpd tends to run in a central location, for example, in the control workstation in a RS/6000 SP environment. Monitoring for it should therefore not be performed with a wildcarded NodeNum.

Even though the timed or xntpd processes may be running, system clocks may still wander. This can be detected using the `timedc` command. For example, running the following command on svrnode2:

```
timedc clockdiff svrnode1
```

will provide input similar to this:

```
time on svrnode1 is 2983 ms. behind time on svrnode2
```

This command can be placed into a periodically run shell script and used to feed information into a User_state resource variable, as such:

```
#!/bin/ksh
TIMESERVER=ctrl_ws
TIMESHIFT=‘/usr/sbin/timedc clockdiff $TIMESERVER | awk ’{print $5}’‘
/usr/lpp/ssp/bin/pmanrminput -s pman \
-a "IBM.PSSP.pm.User_state10+$TIMESHIFT+"
exit 0
```

This places the value of the time difference between this system and the timeserver into the IBM.PSSP.pm.User_state10 resource variable. This can then be monitored using a predicate like:

```
X@0>"900000"
```

If the time difference between this node and the timeserver is greater than 900000 milliseconds (15 minutes), this predicate will evaluate as true. The absolute value of the delta is system-dependent and, potentially, application-dependent.

Should this predicate trigger, the obvious action to take is to resynchronize the clock of the node that is falling out of step with that of the master time reference. While there are various techniques that can be used to do this, the simplest method is to use the `date` command to reset the time.

10.7 Hardware Failures

One of the top ten events and actions is related to hardware.

10.7.1 Failure of the scsi0 Adapter

It may seem strange to have this item as an entry in the "top ten" events, but it is here for good reason. The scsi0 in the title is a generic descriptor for the SCSI adapter that is attached to the internal (operating system) disks. In most systems, this will be /dev/scsi0.

A common perception is that a failure of this adapter will always cause a failure of the operating system, which ought to manifest itself as a complete system crash. This is not always true. In fact, it is very rarely the case.

The basic problem is the unpredictable state of the system as a result of the failure of this adapter. It is possible to observe many different effects, ranging from the "expected" system crash, through various degrees of operating system "hang," to some systems with very large memory sizes seeming not to "notice" this failure.

Under normal operating conditions, a large proportion of AIX programs and data will already be resident in memory because they will have been used earlier. Consequently, there is little need to use the disk or disks attached to the scsi0 adapter. The system will continue to function correctly until some attempt is made to access the disks connected to the failed adapter. The session, or process, attempting this will then hang. However, the system itself is not hung, and it is quite feasible to go to another terminal and continue

working (or for another process to continue working) until this also attempts to access the disks and consequently hangs too.

The standard solution to a problem such as this would be to use error notification to detect the failure and then trigger an error notification method or use another Event Management technique, such as the SP Problem Management subsystem, to trigger a recovery action. This is designed to handle such failures as these.

However, the single SCSI adapter has an effect here too. Consider the following scenario where an error notification object is configured such that, should a catastrophic failure of the SCSI adapter occur, a shell script is to be run that halts the machine.

When the SCSI adapter fails, an error is generated by the device driver. This error is written to the error log that is held in memory, as would normally occur. However, the error cannot be written to the disk error log because the disk is no longer accessible. The converse of this is also true: just as information cannot be written to the disk, the script to be run to halt the system (the error notification method or the action triggered from Event Management) is also inaccessible.

The scdsk_mon program monitors the SCSI adapter that is driving the internal disks. Should a failure of this adapter occur, then the system is halted. The other cluster nodes see this as a crash and take normal action to take over the workload that was running on this system.

The way that scdsk_mon works is to issue non-blocking reads against each of the disks that are attached to the adapter. The read is issued against block 0 of the raw device, /dev/rhdiskN, so that the interference that the monitor will cause is kept to a minimum. As each read is issued, an independent timer is set running.

As we have previously seen, the failure of the SCSI adapter causes the process attempting to access the disk to hang. The timeout periods specified as part of the SCSI standard are too long to be useful in this case. The independent timer gives us the freedom to ignore the built-in timeouts.

If the disk read fails, the timer runs down, and the disk is marked as being potentially unavailable. Each disk attached to the adapter has a non-blocking read request issued against it in turn. The monitor then sleeps for a while before "polling" the disks again.

Because a single disk may well be busy performing other I/O, and hence, fail to respond to a single non-blocking read request, a disk is not marked as

permanently unavailable unless access to it fails on multiple consecutive attempts. As long as a single disk returns a successful read, the adapter and bus are assumed to be good. Should *all* disks on the adapter fail to return successful reads during a single polling period, the adapter is marked as bad, and the system is halted.

In order to be able to handle different system environments, the choice of adapter to monitor, the polling frequency, and the disk read timeout are configurable. The defaults as supplied are suitable for the majority of systems and will detect a failure and halt a typical system in less than 1 minute. The usage of the `scdsk_mon` command is as follows:

```
scdsk_mon -l logical_device_name [-t timeout] [-p polling_interval]
```

where:

- -l stands for the logical device name.

  The device name of the adapter that you wish to monitor, that is, scsi0. Do not put a /dev prefix onto this name.

- -t stands for the timeout for the disk read request in seconds.

  This is the value that the timer waits for before marking the disk as potentially unavailable. The default value for this is 10 seconds.

- -p stands for the polling interval in seconds.

  This value determines how frequently the disk and adapter are monitored. The default value is 30 seconds.

For example, to monitor the internal adapter scsi0 using scdsk_mon, issue:

```
scdsk_mon -l scsi0 &
```

The source code for scdsk_mon can be found in Appendix A, “scdsk_mon” on page 175.

Chapter 11. Sample Events and Actions

This chapter discusses some of the system components that may run on an AIX system. It also provides examples of the events that should be monitored for and sample recovery actions should these events occur.

11.1 Components, Events and Possible Responses

This section ties together the monitoring techniques discussed in Chapter 10 with typical events that can occur. It explains the effects, or implications, of these events, how to detect them, and possible recovery actions.

Each section in this chapter has the following format:

- Name of Entity to be monitored.

  Comments about what the normal situation should be, the conditions that should be monitored for, or the implications of the event.

  ```
  Example Resource Variable
  Example Predicate
  ```

  Where appropriate, a possible recovery action or comments about recovering from this situation.

In many cases, it is not possible to provide a generic recovery action. For those examples where file system space is an issue, or for when the death of a process has occurred, refer to Chapter 8, “Common Monitoring Tasks” on page 123 for general guidelines on recovery actions. See also Chapter 9, “Common Recovery Actions” on page 137.

11.1.1 Accounting

Key files and directories:

The files and directories that are used depend upon the types of accounting being performed. If there is no space in these directories, no accounting records can be written, and consequently, the data will be incorrect. For each type of accounting, the following files or directories may be used:

- Connect-Time Accounting

  /var/adm/wtmp

- Process Accounting

  /var/adm/pacct

- Disk-Usage Accounting

/var/adm/acct/nite/dacct

- Printer-Usage Accounting

  /var/adm/qacct

```
IBM.PSSP.aixos.FS.%totused
X>90
```

11.1.2 AFS Client

Processes:

- afsd

The number of afsd daemons and the options with which they run are site dependent. Typically between 6 and 10 afsd processes should be running. Failure of these processes will cause the client to function incorrectly.

```
IBM.PSSP.Prog.pcount
X@0==8
```

This assumes that your site normally runs with 8 afsd processes.

- inetd.afs

The inetd.afs process is optionally run. This allows for functions such as AFS authenticated r-commands. Failure of this process will cause authenticated r-commands to cease to function.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Key files and directories:

- /usr/vice/cache - AFS cache

The AFS cache should be configured on a separate logical volume. At all times, there should be a minimum of 10% of the LV/file system size available as free space. If the cache fills beyond this, the performance of the client may fall to unacceptable levels.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

The AFS cache may be flushed to free space. This forces unwritten data in the cache to be written back to the server or written data to be discarded. To flush data from selected directories, use the `fs flush` command. To force all cache data to be invalidated, use the `fs flushvolume` command. After the cache has been flushed, subsequent access to this data will retrieve those accessed pages from the server. This access will initially be slower than if the

page were locally cached. If this event is triggered frequently, the cache size should be reviewed and potentially increased.

11.1.3 AFS Server

Processes:

- afsd - see AFS client in 11.1.2
- bosserver
- buserver
- fs
- kaserver
- ptserver
- vlserver

Failure of any of these processes will cause the server to function incorrectly or cease functioning as a server. You will need multiple instance vectors (one for each process) to monitor these processes.

```
IBM.PSSP.Prog.pcount
X@0==0
upserver - optional
```

The upserver process is optionally run.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Key files and directories:

- /vicepa etc

The directories hold the data from the AFS file systems that are served to the clients. If these directories fill up, no more data can be saved on the server.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

11.1.4 Auditing

Process:

If you are using the BIN collection mode for auditing records:

- auditbin

Failure of this process will cause the system to be unable to process audit records. This may result in the system being shutdown, as security may potentially be compromised.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Key files and directories:

If you are using the BIN collection mode for auditing records:

- File systems containing audit records

These directories contain the audit records detected on your system. The location of these directories is specific to your environment. Typically they will exist in a separate file system named /audit (or some similar name). In many environments, if these directories fill up, you will not be able to store any new audit records. In this case, it is often a predetermined policy to shut down the system.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

11.1.5 BNU/UUCP

Key files and directories:

- /var/spool/uucppublic

This is the uucp spool directory. Free space should exist here to hold files for transfer to other systems.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

The commands `uuclean, uucleanup` and `uudemon.cleanu` are provided to clear out files and logfiles from the uucp spooling and logging directories. These may be used to free up space before more drastic measures are taken.

11.1.6 cron Daemon

Process:

- cron

The cron daemon should be respawned automatically following a failure. If the cron daemon has failed totally–that is, if it cannot be respawned automatically, this is likely to be due to another cause.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.7 DCE/DFS Clients and Servers

Process:

The different processes that will run on the system are dependent upon the role the system should perform. The number of each type of process is site-dependent; see Table 7.

Table 7. DCE/DFS Sites and Processes

	dced	cdsadv	cdsclerk	dtsd	secd	cdsd	bosserver	upserver	flserver	ftserver	fxd	dfsd	dfsbind
Security Client	X												
CDS Client	X	X	X										
DTS Client	X	X	X	X									
Security Server	X	X	X		X								
CDS Server	X	X	X			X							
DTS Server	X	X	X	X									
DFS SCM	X	X	X				X	X					
DFS FLDB Server	X	X	X				X	X	X				
DFS Fileset Server	X	X	X				X			X	X		
DFS Client	X	X	X									X	X

Failure of any of these processes will cause the system to function incorrectly or cease functioning. You will need multiple instance vectors (one for each type of process) to monitor these processes.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Key files and directories:

- /var/dce/adm/dfs/cache

This is the DFS cache, which should be kept on a separate logical volume. The normal recommendation for a DFS cache is that at least 15% of the total logical volume size should be available as free space.

```
IBM.PSSP.aixos.FS.%totused
X>85
```

- /var/dce/adm/directory/cds

This contains the CDS clerk cache. The amount of free space in the CDS clerk cache should be at least the size of the CDS database. This is site-dependent, but is typically less than 2MB in size.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

This assumes that the CDS clerk cache is held in its own file system.

- /var/dce/security/rgy_data

This holds the DCE Security Registry. The amount of free space in the security registry should be at least the size of the security registry.

```
IBM.PSSP.aixos.FS.%totused
X>50
```

This assumes that the security registry is held in its own file system.

- logical volumes used to hold DFS aggregates/filesets

These directories hold the data from the DFS file systems that are served to the clients. If these directories fill up, no more data can be saved on the server. Their location is dependent upon the configuration at your site.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

11.1.8 DHCP Client

Process:

- dhcpd

Failure of this process will cause the system to be unable to request a dynamic IP address, and hence, be unable to connect to the network. If the system has already obtained an IP address, it will be unable to renew it.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.9 DHCP Server

Process:

- dhcpsd

Failure of this process will cause the system to be unable to function as a DHCP server. Clients will not be able to discover IP addresses, and hence, will be unable to connect to the network.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.10 Domain Name Server

Processes:

- named

Failure of this process will cause the system to be unable to function as a nameserver. This may cause many other, seemingly unrelated, system functions to fail.

```
IBM.PSSP.Prog.pcount
X@0==0
```

The named daemon can be restarted automatically if it should fail by using the `startsrc` command with appropriate parameters for your environment. If this event triggers repeatedly, the problem most likely is not with the process, but lies elsewhere in the named configuration files.

11.1.11 iFOR/LS

Processes:

- llbd
- glbd
- netlsd

Failure of any of these processes will cause the system to function incorrectly or cease functioning. You will need multiple instance vectors (one for each type of process) to monitor these processes.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.12 Mail

Process:

- sendmail

Failure of this process will cause the system to function incorrectly.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Key files and directories:

- /var/spool/mqueue

By default, this directory contains the temporary files and the log file associated with the messages in the outbound mail queue. This directory, or its contents, may be in a different location in your environment.

- /var/spool/mail

By default, this directory contains the inbound mail for users on the local system. This directory, or its contents, may be different in your environment. If this directory fills up, mail for users on this system may not be able to be delivered.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

11.1.13 NCS

Processes:

- llbd
- glbd

Failure of any of these processes will cause the system to function incorrectly or cease functioning. You will need multiple instance vectors (one for each type of process) to monitor these processes.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.14 NFS Client

Processes:

- biod
- rpc.statd
- rpc.lockd

Failure of any of these processes will cause the system to function incorrectly or cease functioning. You will need multiple instance vectors (one for each type of process) to monitor these processes. On earlier versions of AIX, multiple biod daemons appear in the process table. Later versions have a single biod daemon entry. Your predicate should reflect the situation on your system.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.15 NFS Server

Processes:

- rpc.lockd
- rpc.statd
- nfsd
- rpc.mountd

If the server environment is required to support automounting of file systems:

- amd

or

- automount

Failure of any of these processes will cause the system to function incorrectly or cease functioning. You will need multiple instance vectors (one for each type of process) to monitor these processes. On earlier versions of AIX, multiple biod daemons appear in the process table. Later versions have a single biod daemon entry. Your predicate should reflect the situation on your system.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Key files and directories:

- Exported data directories

These directories hold the data from the NFS file systems that are served to the clients. If these directories fill up, no more data can be saved on the server. Their location is dependent upon the configuration at your site.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

11.1.16 NIM Server

Process:

- nimesis

Failure of this process will cause the system to be unable to function as a NIM server.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Key files and directories:

- File systems containing NIM objects

These directories contain the NIM objects such as lpp_source, SPOT and so on. The location of these directories are specific to your environment. If these directories fill up, you will not be able to store any additional NIM objects.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

11.1.17 NIS Client

Process:

- ypbind

Failure of this process will cause the client to be unable to contact the server. The system will cease to function correctly.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.18 NIS Server (master or slave)

Processes:

- ypserv

If the NIS server is a master server, it also runs:

- ypupdated
- yppasswdd

Failure of any of these processes will cause the system to function incorrectly or cease functioning. You will need multiple instance vectors (one for each type of process) to monitor these processes.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.19 Portmapper

Process:

- portmap

The portmap daemon is needed to map RPC program requests to Internet port numbers. If it fails, many system functions will cease to work.

```
IBM.PSSP.Prog.pcount
X@0==0
```

The portmapper may be restarted as required. However, when the portmapper is restarted, all services that were using its services should also be restarted.

11.1.20 Printing

Processes:

- qdaemon

The qdaemon is responsible for handling requests from print commands, scheduling jobs to the appropriate queue, and so on. Failure of this process will cause the system to be unable to process new print jobs. The jobs will normally be queued but not printed. This potentially increases the risk of the /var file systems filling up.

```
IBM.PSSP.Prog.pcount
X@0==0
```

If the qdaemon process fails, it should be respawned automatically. If this respawn fails, it is likely to be due to a configuration problem, such as a missing /etc/qconfig file.

If the system is acting as a network print server, it will also be running:

- lpd

The lpd process accepts print requests from foreign hosts. Failure of this process will cause the system to be unable to accept print requests from other systems.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Key files and directories:

- /var/spool/lpd/qdir

This directory contains pointers to the files being printed. If this directory fills up, no more print jobs will be able to be submitted to the system.

```
IBM.PSSP.aixos.FS.%totused
X>90
```

- /var/spool/qdaemon

This directory contains the files themselves that are being printed. If this directory fills up, no more print jobs will be able to be submitted to the system.

```
IBM.PSSP.aixos.FS.%totused
```

```
X>90
```

11.1.21 Secure NFS - see NIS

Refer to 11.1.17, “NIS Client” on page 170

11.1.22 syslog daemon

Process:

- syslogd

Failure of this process will cause the system to be unable to perform logging of some system messages.

```
IBM.PSSP.Prog.pcount
X@0==0
```

The writesrv daemon may be started as required.

11.1.23 TCP/IP

The majority of occurrences of TCP/IP failure will be detected by a loss of HACMP heartbeat. This is likely only to occur in the event of a total TCP/IP failure. The failure of individual processes or daemons may be detected by monitoring.

Processes:

- inetd
- snmpd
- gated
- routed

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.24 TIME Client (timed)

Processes:

- timed

Failure of this process will cause the client to be unable to contact the timeserver. The clocks of the systems may wander.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.25 TIME Server (master) (NTP)

Processes:

- xntpd

Failure of this process will cause the system to be unable to function as a timeserver. The clocks of the systems may wander.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.26 TIME Server (master or submaster) (timed)

Process:

- timed -M

Failure of this process will cause the system to be unable to function as a timeserver. The clocks of the systems may wander.

```
IBM.PSSP.Prog.pcount
X@0==0
```

11.1.27 writesrv

Process:

- writesrv

Failure of this process will cause the system to be unable to handle write requests.

```
IBM.PSSP.Prog.pcount
X@0==0
```

The writesrv daemon may be started as required. However, if it terminated abnormally, you must manually clean out the /var/spool/writesrv directory in order to remove any files left behind by the old writesrv daemon before you restart it.

11.1.28 XStation Server

Process:

- x_st_mgrd

Failure of this process will cause the system to be unable to function as an Xstation server.

```
IBM.PSSP.Prog.pcount
X@0==0
```

Appendix A. scdsk_mon

This appendix provides the source code for the scdsk_mon monitor described in Chapter 10, “The Top Ten Events and Actions” on page 143.

scdsk_mon consists of three files:

- cfgodm.c
- cfgodm.h
- scdsk_mon.c

To compile the executable, perform the following steps:

```
cc -c cfgodm.c
cc -c scdsk_mon.c
cc -O -c scdsk_mon cfgodm.o scdsk_mon.o -lodm
```

A.1 cfgodm.c

```
#include "cfgodm.h"
static struct ClassElem PdDv_ClassElem[] =  {
 { "type",ODM_CHAR, 12,16, NULL,NULL,0,NULL ,-1,0},
 { "class",ODM_CHAR, 28,16, NULL,NULL,0,NULL ,-1,0},
 { "subclass",ODM_CHAR, 44,16, NULL,NULL,0,NULL ,-1,0},
 { "prefix",ODM_CHAR, 60,16, NULL,NULL,0,NULL ,-1,0},
 { "devid",ODM_CHAR, 76,16, NULL,NULL,0,NULL ,-1,0},
 { "base",ODM_SHORT, 92, 2, NULL,NULL,0,NULL ,-1,0},
 { "has_vpd",ODM_SHORT, 94, 2, NULL,NULL,0,NULL ,-1,0},
 { "detectable",ODM_SHORT, 96, 2, NULL,NULL,0,NULL ,-1,0},
 { "chgstatus",ODM_SHORT, 98, 2, NULL,NULL,0,NULL ,-1,0},
 { "bus_ext",ODM_SHORT, 100, 2, NULL,NULL,0,NULL ,-1,0},
 { "fru",ODM_SHORT, 102, 2, NULL,NULL,0,NULL ,-1,0},
 { "led",ODM_SHORT, 104, 2, NULL,NULL,0,NULL ,-1,0},
 { "setno",ODM_SHORT, 106, 2, NULL,NULL,0,NULL ,-1,0},
 { "msgno",ODM_SHORT, 108, 2, NULL,NULL,0,NULL ,-1,0},
 { "catalog",ODM_CHAR, 110,16, NULL,NULL,0,NULL ,-1,0},
 { "DvDr",ODM_CHAR, 126,16, NULL,NULL,0,NULL ,-1,0},
 { "Define",ODM_METHOD, 142,256, NULL,NULL,0,NULL ,-1,0},
 { "Configure",ODM_METHOD, 398,256, NULL,NULL,0,NULL ,-1,0},
 { "Change",ODM_METHOD, 654,256, NULL,NULL,0,NULL ,-1,0},
 { "Unconfigure",ODM_METHOD, 910,256, NULL,NULL,0,NULL ,-1,0},
 { "Undefine",ODM_METHOD, 1166,256, NULL,NULL,0,NULL ,-1,0},
 { "Start",ODM_METHOD, 1422,256, NULL,NULL,0,NULL ,-1,0},
 { "Stop",ODM_METHOD, 1678,256, NULL,NULL,0,NULL ,-1,0},
 { "inventory_only",ODM_SHORT, 1934, 2, NULL,NULL,0,NULL ,-1,0},
 { "uniquetype",ODM_CHAR, 1936,48, NULL,NULL,0,NULL ,-1,0},
```

```
 };
struct Class PdDv_CLASS[] = {
 ODMI_MAGIC, "PdDv", sizeof(struct PdDv), PdDv_Descs, PdDv_ClassElem,
NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
static struct ClassElem PdCn_ClassElem[] =  {
 { "uniquetype",ODM_CHAR, 12,48, NULL,NULL,0,NULL ,-1,0},
 { "connkey",ODM_CHAR, 60,16, NULL,NULL,0,NULL ,-1,0},
 { "connwhere",ODM_CHAR, 76,16, NULL,NULL,0,NULL ,-1,0},
 };
struct Class PdCn_CLASS[] = {
 ODMI_MAGIC, "PdCn", sizeof(struct PdCn), PdCn_Descs, PdCn_ClassElem,
NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
static struct ClassElem PdAt_ClassElem[] =  {
 { "uniquetype",ODM_CHAR, 12,48, NULL,NULL,0,NULL ,-1,0},
 { "attribute",ODM_CHAR, 60,16, NULL,NULL,0,NULL ,-1,0},
 { "deflt",ODM_CHAR, 76,256, NULL,NULL,0,NULL ,-1,0},
 { "values",ODM_CHAR, 332,256, NULL,NULL,0,NULL ,-1,0},
 { "width",ODM_CHAR, 588,16, NULL,NULL,0,NULL ,-1,0},
 { "type",ODM_CHAR, 604,8, NULL,NULL,0,NULL ,-1,0},
 { "generic",ODM_CHAR, 612,8, NULL,NULL,0,NULL ,-1,0},
 { "rep",ODM_CHAR, 620,8, NULL,NULL,0,NULL ,-1,0},
 { "nls_index",ODM_SHORT, 628, 2, NULL,NULL,0,NULL ,-1,0},
 };
struct Class PdAt_CLASS[] = {
 ODMI_MAGIC, "PdAt", sizeof(struct PdAt), PdAt_Descs, PdAt_ClassElem,
NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
static struct ClassElem Config_Rules_ClassElem[] =  {
 { "phase",ODM_SHORT, 12, 2, NULL,NULL,0,NULL ,-1,0},
 { "seq",ODM_SHORT, 14, 2, NULL,NULL,0,NULL ,-1,0},
 { "boot_mask",ODM_LONG, 16, 4, NULL,NULL,0,NULL ,-1,0},
 { "rule",ODM_CHAR, 20,256, NULL,NULL,0,NULL ,-1,0},
 };
struct Class Config_Rules_CLASS[] = {
 ODMI_MAGIC, "Config_Rules", sizeof(struct Config_Rules),
Config_Rules_Descs, Config_Rules_ClassElem,
NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
static struct ClassElem CuDv_ClassElem[] =  {
 { "name",ODM_CHAR, 12,16, NULL,NULL,0,NULL ,-1,0},
 { "status",ODM_SHORT, 28, 2, NULL,NULL,0,NULL ,-1,0},
 { "chgstatus",ODM_SHORT, 30, 2, NULL,NULL,0,NULL ,-1,0},
 { "ddins",ODM_CHAR, 32,16, NULL,NULL,0,NULL ,-1,0},
 { "location",ODM_CHAR, 48,16, NULL,NULL,0,NULL ,-1,0},
 { "parent",ODM_CHAR, 64,16, NULL,NULL,0,NULL ,-1,0},
```

```
 { "connwhere",ODM_CHAR, 80,16, NULL,NULL,0,NULL ,-1,0},
 { "PdDvLn",ODM_LINK, 96 ,48, PdDv_CLASS,"uniquetype",0,NULL ,-1,0},
 };
struct Class CuDv_CLASS[] = {
 ODMI_MAGIC, "CuDv", sizeof(struct CuDv), CuDv_Descs, CuDv_ClassElem,
NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
static struct ClassElem CuDep_ClassElem[] =  {
 { "name",ODM_CHAR, 12,16, NULL,NULL,0,NULL ,-1,0},
 { "dependency",ODM_CHAR, 28,16, NULL,NULL,0,NULL ,-1,0},
 };
struct Class CuDep_CLASS[] = {
 ODMI_MAGIC, "CuDep", sizeof(struct CuDep), CuDep_Descs, CuDep_ClassElem,
NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
static struct ClassElem CuAt_ClassElem[] =  {
 { "name",ODM_CHAR, 12,16, NULL,NULL,0,NULL ,-1,0},
 { "attribute",ODM_CHAR, 28,16, NULL,NULL,0,NULL ,-1,0},
 { "value",ODM_CHAR, 44,256, NULL,NULL,0,NULL ,-1,0},
 { "type",ODM_CHAR, 300,8, NULL,NULL,0,NULL ,-1,0},
 { "generic",ODM_CHAR, 308,8, NULL,NULL,0,NULL ,-1,0},
 { "rep",ODM_CHAR, 316,8, NULL,NULL,0,NULL ,-1,0},
 { "nls_index",ODM_SHORT, 324, 2, NULL,NULL,0,NULL ,-1,0},
 };
struct Class CuAt_CLASS[] = {
 ODMI_MAGIC, "CuAt", sizeof(struct CuAt), CuAt_Descs, CuAt_ClassElem,
NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
static struct ClassElem CuDvDr_ClassElem[] =  {
 { "resource",ODM_CHAR, 12,12, NULL,NULL,0,NULL ,-1,0},
 { "value1",ODM_CHAR, 24,20, NULL,NULL,0,NULL ,-1,0},
 { "value2",ODM_CHAR, 44,20, NULL,NULL,0,NULL ,-1,0},
 { "value3",ODM_CHAR, 64,20, NULL,NULL,0,NULL ,-1,0},
 };
struct Class CuDvDr_CLASS[] = {
 ODMI_MAGIC, "CuDvDr", sizeof(struct CuDvDr), CuDvDr_Descs,
CuDvDr_ClassElem, NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
static struct ClassElem CuVPD_ClassElem[] =  {
 { "name",ODM_CHAR, 12,16, NULL,NULL,0,NULL ,-1,0},
 { "vpd_type",ODM_SHORT, 28, 2, NULL,NULL,0,NULL ,-1,0},
 { "vpd",ODM_LONGCHAR, 30,512, NULL,NULL,0,NULL ,-1,0},
 };
struct Class CuVPD_CLASS[] = {
 ODMI_MAGIC, "CuVPD", sizeof(struct CuVPD), CuVPD_Descs, CuVPD_ClassElem,
NULL,FALSE,NULL,NULL,0,0,NULL,0,"", 0,-ODMI_MAGIC
 };
```

A.2 cfgodm.h

```
#include <odmi.h>
struct PdDv {
long _id;
long _reserved;
long _scratch;
char type[16];
char class[16];
char subclass[16];
char prefix[16];
char devid[16];
short base;
short has_vpd;
short detectable;
short chgstatus;
short bus_ext;
short fru;
short led;
short setno;
short msgno;
char catalog[16];
char DvDr[16];
char Define[256];/* method */
char Configure[256];/* method */
char Change[256];/* method */
char Unconfigure[256];/* method */
char Undefine[256];/* method */
char Start[256];/* method */
char Stop[256];/* method */
short inventory_only;
char uniquetype[48];
};
#define PdDv_Descs 25
extern struct Class PdDv_CLASS[];
#define get_PdDv_list(a,b,c,d,e) (struct PdDv * )odm_get_list(a,b,c,d,e)
struct PdCn {
long _id;
long _reserved;
long _scratch;
char uniquetype[48];
char connkey[16];
char connwhere[16];
};
#define PdCn_Descs 3
extern struct Class PdCn_CLASS[];
```

```
#define get_PdCn_list(a,b,c,d,e) (struct PdCn * )odm_get_list(a,b,c,d,e)
struct PdAt {
long _id;
long _reserved;
long _scratch;
char uniquetype[48];
char attribute[16];
char deflt[256];
char values[256];
char width[16];
char type[8];
char generic[8];
char rep[8];
short nls_index;
};
#define PdAt_Descs 9
extern struct Class PdAt_CLASS[];
#define get_PdAt_list(a,b,c,d,e) (struct PdAt * )odm_get_list(a,b,c,d,e)
struct Config_Rules {
long _id;
long _reserved;
long _scratch;
short phase;
short seq;
long boot_mask;
char rule[256];
};
#define Config_Rules_Descs 4
extern struct Class Config_Rules_CLASS[];
#define get_Config_Rules_list(a,b,c,d,e) (struct Config_Rules *
)odm_get_list(a,b,c,d,e)
struct CuDv {
long _id;
long _reserved;
long _scratch;
char name[16];
short status;
short chgstatus;
char ddins[16];
char location[16];
char parent[16];
char connwhere[16];
struct PdDv *PdDvLn;/* link */
struct listinfo *PdDvLn_info;/* link */
char PdDvLn_Lvalue[48];/* link */
};
#define CuDv_Descs 8
```

```
extern struct Class CuDv_CLASS[];
#define get_CuDv_list(a,b,c,d,e) (struct CuDv * )odm_get_list(a,b,c,d,e)
struct CuDep {
long _id;
long _reserved;
long _scratch;
char name[16];
char dependency[16];
};
#define CuDep_Descs 2
extern struct Class CuDep_CLASS[];
#define get_CuDep_list(a,b,c,d,e) (struct CuDep * )odm_get_list(a,b,c,d,e)
struct CuAt {
long _id;
long _reserved;
long _scratch;
char name[16];
char attribute[16];
char value[256];
char type[8];
char generic[8];
char rep[8];
short nls_index;
};
#define CuAt_Descs 7
extern struct Class CuAt_CLASS[];
#define get_CuAt_list(a,b,c,d,e) (struct CuAt * )odm_get_list(a,b,c,d,e)
struct CuDvDr {
long _id;
long _reserved;
long _scratch;
char resource[12];
char value1[20];
char value2[20];
char value3[20];
};
#define CuDvDr_Descs 4
extern struct Class CuDvDr_CLASS[];
#define get_CuDvDr_list(a,b,c,d,e) (struct CuDvDr *
)odm_get_list(a,b,c,d,e)
struct CuVPD {
long _id;
long _reserved;
long _scratch;
char name[16];
short vpd_type;
char vpd[512];
```

```
};
#define CuVPD_Descs 3
extern struct Class CuVPD_CLASS[];
#define get_CuVPD_list(a,b,c,d,e) (struct CuVPD * )odm_get_list(a,b,c,d,e)
```

A.3 scdsk_mon.c

```
#include <sys/errno.h>
#include <sys/types.h>
#include <sys/cfgodm.h>
#include <odmi.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
/* #include "Dv.h" */
#include <sys/signal.h>
#define  RECORDSIZE     512
externinterrno;
externintDflag;
/* extern struct Class CuDv_CLASS[]; */
salrm()         /* signal handler for receiving SIGALRM */
{
printf("scdsk_mon: SIGALRM received - potential disk failure.");
}
main(int argc, char **argv)
{
char *Device;
int timeout = 10;
int poll = 30;
int opt;
extern int optind;
extern char *optarg;
 struct sigvec sig_struct;
struct CuDv device;
struct listinfo list_info;
struct CuDv *class;
charsstr[256],*p;
intsleeptime,numdisks,numfailed,c,i,rc;
char    rawname;
char rawdev[128];
int diskstate[16];
        long fildes;
        char buf[RECORDSIZE];

/*Parse command line arguments */
```

```
while((opt = getopt(argc, argv, "t:p:l:")) != EOF)
{
switch(opt)
{
case 't':
timeout = atoi(optarg);
break;
case 'p':
poll = atoi(optarg);
                        break;
case 'l':
Device = optarg;
                        break;
}
}
if (optind != argc)
{
usage();
}
if (! *Device)
{
        usage();
}

/* loop to invoke the test process and then sleep */
for(;;)
{
/* initialise diskstate array */
for (i=0; i < 16; i++)
        {
                diskstate[i] = 0;
        }
/* initialise ODM */
rc = (int)odm_initialize();
if (rc == -1)
{
printf("scdsk_mon: Failed to initialise ODM");
exit(-1);
}
/* get customized object for device */
sprintf(sstr,"parent = '%s' AND name LIKE hdisk*",Device);
class = odm_get_list(CuDv_CLASS,sstr,&list_info,16,1);
if (class == 0 || class == -1)
{
printf("scdsk_mon: Failed to get ODM objects for %s",Device);
exit(-1);
}
```

```
numdisks = list_info.num;
for (i = 0; i < list_info.num; i++)
{
        if( signal(SIGALRM, salrm) == -1 )
        {
                exit(1);
        }
/* get objects with odm_get_next */
class = odm_get_next(CuDv_CLASS,sstr);
/* convert logical name to raw device name */
        sprintf(rawdev,"/dev/r%s",class->name);
sigvec(SIGALRM, 0, &sig_struct);
/* Set alarm timer */
       sleeptime = timeout + 2;
alarm(timeout);
        /* Open the file */
        if( (fildes=open(rawdev, O_RDONLY, 0))== -1)
        {
                sleep(sleeptime);
        }
        /* Read the record from the file */
        if( (rc = read(fildes, &buf[0], RECORDSIZE)) == -1 )
        {
                sleep(sleeptime);
        }
        /* Close the file and terminate */
        close(fildes);
sigvec(SIGALRM, 0, &sig_struct);
if (sig_struct.sv_handler == 0)
{
diskstate[i] = 1;
numfailed = 0;
for (c=0; c < 16; ++c)
{
numfailed = numfailed + diskstate[c];
}
if (numfailed == numdisks)
{
printf("scdsk_mon: TOTAL FAILURE OF DISKS / ADAPTER");
system("cat /dev/kmem > /dev/kmem");
}
}
numfailed = 0;
        alarm(0);
}
sleep(poll);
}
```

```
}
usage()
{
fprintf(stderr,
"Usage: scdsk_mon [-t timeout] [-p poll frequency] -l devicename");
exit(-1);
}
```

Appendix B. Special Notices

This publication is intended to help system administrators and system implementors of HACMP clusters to take advantage of the facilities offered by RISC System Cluster Technology (RSCT). These facilities allow for a wide range of problem conditions to be protected against, and in addition, allow for the automation of many day-to-day systems management tasks. The information in this publication is not intended as the specification of any programming interfaces that are provided by either HACMP or PSSP. See the PUBLICATIONS section of the *IBM Programming Announcements for HACMP and PSSP* 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.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	HACMP/6000
IBM ®	POWERparallel
RS/6000	Scalable POWERparallel Systems
SP	

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix C. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 189.

- *RS/6000 SP High Availability Infrastructure*, SG24-4838
- *HACMP Enhanced Scalability*, SG24-2081

C.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
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

C.3 Other Publications

These publications are also relevant as further information sources:

- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Concepts and Facilities*, SC23-1938
- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Planning Guide*, SC23-1939

- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Installation Guide*, SC23-1940
- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Administration Guide*, SC23-1941
- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Troubleshooting Guide*, SC23-1942
- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Programming Locking Applications*, SC23-1943
- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Programming Client Applications*, SC23-1944
- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Master Index and Glossary*, SC23-1945
- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: HANFS for AIX Installation and Administration Guide*, SC23-1946
- *High Availability Cluster MultiProcessing for AIX, Version 4.2.2: Enhanced Scalability Installation and Administration Guide*, SC23-1972
- *IBM Parallel System Support Programs for AIX: Administration Guide*, GC23-3897
- *IBM Parallel System Support Programs for AIX: Installation and Migration Guide*, GC23-3898
- *IBM Parallel System Support Programs for AIX: Diagnosis and Messages Guide*, GC23-3899
- *IBM Parallel System Support Programs for AIX: Command and Technical Reference*, GC23-3900
- *IBM Parallel System Support Programs for AIX: Event Management Programming Guide and Reference*, SC23-3996
- *IBM Parallel System Support Programs for AIX: Group Services Programming Guide and Reference*, SC28-1675
- *IBM AIX Version 4 General Programming Concepts*, SC23-2610
- *IBM AIX Version 4 Problem Solving Guide and Reference*, SC23-2606

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 `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:

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/`

- **PUBORDER** – to order hardcopies in the United States
- **Tools Disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

- **REDBOOKS Category on INEWS**
- **Online** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

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** – 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	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** – ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

B

Barrier A means of coordinating the activities of a group of cluster nodes. Each cluster node must reach the synchronization point (barrier) before any may continue.

C

Command Execution Language A set of language definitions that defines how a CSPOC program should execute on multiple systems.

Component Failure Impact Analysis A technique by which a system may be evaluated such that the impact of a component failure on the entire system may be determined.

Cluster Single Point Of Control A set of facilities that allows a cluster to be managed from a single cluster node.

E

Enhanced Scalability A feature of the HACMP software that allows up to 32 nodes in a cluster. Non-ES HACMP supports up to 8 nodes in a cluster.

Error Notification Object An object in the ODM that is matched with an error log entry. When a match occurs, the action defined in the error notification object is run.

Event **(1)** An action that takes place in the operation of a system.

(2) Event Management, the notification that a predicate evaluated to true.

Event Management Application Programming Interface A set of routines by which a client program can obtain information about resources monitored by the Event Management subsystem.

Expression A term used in Perspectives. This is equivalent to the Event Management Predicate.

H

High Availability Cluster MultiProcessing An IBM licensed program product that allows RS /6000 systems to be clustered together to provide enhanced availability through eliminating single points of failure and scalability by allowing the activities of multiple systems to be coordinated together.

I

Instance Vector In Event Management, a set of elements which uniquely identify a copy of a resource.

L

Logging The writing of information to persistant storage to allow subsequent analysis.

P

Predicate In Event Management, a relational expression between a resource variable and other elements.

R

Rearm Expression In Perspectives, the equivalent of an Event Management rearm predicate.

Rearm Predicate A predicate that is typically used to indicate the reverse of a predicate. Typically, when a problem condition occurs, the predicate triggers. When the problem condition is fixed, the rearm predicate triggers.

Resource Monitor Application Programming Interface A set of routines by which a program monitoring a resource can provide information about the resource to the Event Management subsystem.

Resource Variable In Event Management, the representation of an attribute of a resource.

RS/6000 Cluster Technology The collective name given to the set of cluster components comprising Topology Services, Group Services, and Event Management.

Recoverable Virtual Shared Disk A function that allows application programs on different cluster nodes to access a raw logical volume as if it were local to the system. In the event of a

failure of a primary access path, a backup takes over providing access to the disk.

S

State In Event Management, the value type of a resource variable whose value changes over time.

Structured Byte String A string of bytes consisting of an SBS length field followed by one or more data fields.

System Data Repository A central collection of information that defines how an SP system is configured and should be run.

List of Abbreviations

AFS	Andrew File System
AIX	Advanced Interactive eXecutive
API	Application Programming Interface
BNU	Basic Networking Utilities
CEL	Command Execution Language
CFIA	Component Failure Impact Analysis
CPU	Central Processing Unit
CSPOC	Cluster Single Point Of Control
DCE	Distributed Computing Environment
DFS	Distributed File System
DHCP	Dynamic Host Configuration Protocol
EMAPI	Event Management Application Programming Interface
ES	Enhanced Scalability
GUI	Graphical User Interface
HACMP	High Availability Cluster MultiProcessing
HSM	Hierarchical Storage Manager/Hierarchical Storage Management
IBM	International Business Machines
IP	Internet Protocol
ITSO	International Technical Support Organization
LVM	Logical Volume Manager
NCS	Network Computing System
NFS	Network File System
NIM	Network Install Manager
NIS	Network Information System
NTP	Network Time Protocol
ODM	Object Data Manager
PSSP	Parallel System Support Program
PVID	Physical Volume IDentifier
RISC	Reduced Instruction Set Computer
RMAPI	Resource Monitor Application Programming Interface
RSCT	RISC System Cluster Technology
RVSD	Recoverable Virtual Shared Disk
SBS	Structured Byte String
SCSI	Small Computer Systems Interface
SDR	System Data Repository
SP	Scalable POWERparallel
TCP	Transmission Control Protocol
VPD	Vital Product Data
VSD	Virtual Shared Disk

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

S

T

U

V

W

X

ITSO Redbook Evaluation

HACMP Enhanced Scalability: User-Defined Events
SG24-5327-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ **Customer** _ **Business Partner** _ **Independent Software Vendor** _ **IBM employee**
_ **None of the above**

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

SG24-5327-00
Printed in the U.S.A.